Università degli Studi di Padova

Facoltà di Ingegneria Industriale

Corso di Laurea Magistrale in
Ingegneria Aerospaziale

Graph-Based Simultaneous Localization And Mapping Using a Stereo Camera

Lorenzo Salvucci

Relatore:
Prof. Stefano Debei

Supervisore:
Marco Pertile

Anno accademico 2014-2015

Contents

Introduction

The ability of a robot to localize itself while simultaneously building a map of its surroundings is a fundamental characteristic required for autonomous operation in unknown environments when external referencing systems such as GPS are absent. This so-called Simultaneous Localization And Mapping (SLAM) problem has been one of the most popular research subjects in mobile robotics for the last two decades, and despite significant progress in this area, it still poses great challenges. At present, robust methods exist for mapping environments that are static, structured, and limited in size, while mapping unstructured, dynamic, or large scale environments remains an open research problem.

The interest in the SLAM field derives from the apparent advantage that the utilization of robots with SLAM capabilities would bring with respect to the safety, costs and feasibility of a wide spectrum of applications ranging from the inspection of unsafe areas in emergency situations to planetary exploration. Published approaches are also employed in unmanned aerial vehicles, autonomous underwater vehicles, self-driving cars, industrial and domestic robots, and even inside the human body [1].

The work of this thesis was aimed at implementing a vision-based SLAM algorithm for a robot equipped with a stereo camera. Vision systems are an attractive choice of sensor and have increased in popularity for SLAM in

recent years; they not only have become much cheaper and compact than traditional SLAM sensors such as laser range finders and radar systems, but also provide more information per sample and work with much higher data rates. Chapter 1 will describe the characteristics of the Simultaneous Localization And Mapping problem and the approaches developed to solve it from a general point of view, while chapter 2 will give some brief remarks on the peculiarities of the current implementation. The optimization algorithm used to numerically compute a solution to the problem at hand will be described in chapter 3, while chapter 4 will discuss the utilization of the stereo camera as a sensor. Finally, the performances of the developed algorithm will be presented in chapter 5.

Chapter 1

Simultaneous Localization And Mapping

1.1 Introduction

Consider a robot roaming an unknown environment, equipped with sensors to observe its surroundings. In such a scenario, one will likely be interested in keeping track of the robot's motion within the unknown setting or in obtaining a spatial map of the environment itself. If no information is provided from the outside, however, the problem presents a chicken-and-egg situation: precise localization is required to build an accurate map, and an accurate map is necessary to locate the robot precisely. It is therefore clear that solving either the localization or the mapping problem requires in all cases solving both at the same time. This chapter will discuss the main aspects that are involved in this type of situation (sections 1.2, 1.3) and give an overview of the various methods developed to date to address the problem (section 1.4).

1.2 Problem Overview

Let the term *pose* denote the combination of position and orientation necessary to define the configuration of a rigid body in 2D or 3D space. Then, the motion of the robot in the unknown environment can be described by a sequence of poses

$$\boldsymbol{x}_{0:T} = \{\boldsymbol{x}_0, .., \boldsymbol{x}_T\}.$$

The initial pose $\boldsymbol{x}_0$ is assumed to be known (it can be chosen arbitrarily), while the others cannot be sensed directly. While moving, the robot acquires a sequence of *odometry* measurements that provide information about the relative displacement between two consecutive locations. Such data might be obtained from the robot's wheel encoders, from the controls given to the motors, from an IMU, etc. Let $\boldsymbol{u}_t$ denote the odometry measurement that characterizes the motion from pose $\boldsymbol{x}_{t-1}$ to $\boldsymbol{x}_t$; then the sequence

$$\boldsymbol{u}_{1:T} = \{\boldsymbol{u}_1, ..., \boldsymbol{u}_T\}$$

describes step by step the motion of the robot along the full path. For noise-free motion, this information would be sufficient to recover the trajectory $\boldsymbol{x}_{1:T}$ from the initial location $\boldsymbol{x}_0$. However, odometry measurements are noisy, and path-integration techniques inevitably diverge from the truth.

Finally, let m denote the *true* map of the environment. The environment may be comprised of landmarks, objects, etc., and m describes their locations. Along its path, the robot senses its surroundings with some kind of instrument, acquiring a set of observations of the environment

$$\boldsymbol{z}_{0:T} = \{\boldsymbol{z}_0, ..., \boldsymbol{z}_T\}$$

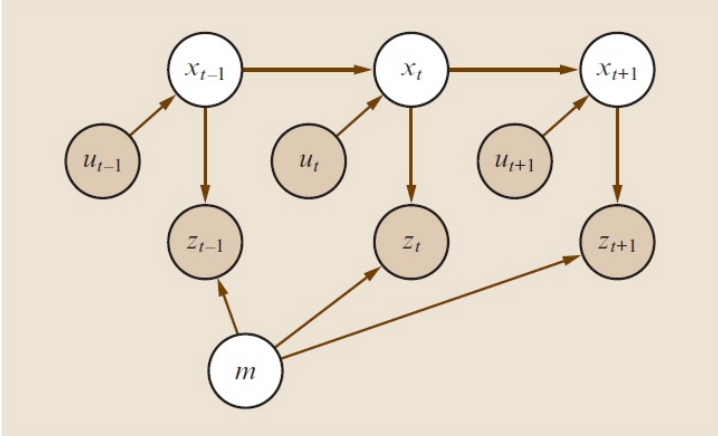

Figure 1.1: Graphical model of the SLAM problem. Arcs indicate causal relationships, and shaded nodes are directly observable to the robot. Through these quantities, we want to estimate the map of the environment and the path of the robot.

that establish information between the robot poses $\boldsymbol{x}_t$ and the elements of the map m.

The SLAM problem then consists in recovering the map of the world m and the path $\boldsymbol{x}_{1:T}$ followed by the robot, given the initial position $\boldsymbol{x}_0$, the odometry measurements $\boldsymbol{u}_{1:T}$ and sensor observations $\boldsymbol{z}_{0:T}$. Figure 1.1 illustrates the variables involved in the problem. If the robot path were known and sensor readings perfect, registering the observations $\boldsymbol{z}_{0:T}$ acquired from the various poses into a common coordinate system would be sufficient to create a unique global map. Unfortunately, two main problems arise:

1. As discussed above, any mobile robot's self-localization system suffers from imprecision, hence the positions from which the observations of the environment are taken are not known exactly.

2. Sensor measurements are affected by noise, and therefore the observations of the environment will not be perfectly consistent, either.

Therefore, given the uncertain nature of the quantities at stake, the SLAM problem is usually described by means of probabilistic tools [2, 3]. The problem is thus reformulated as estimating the posterior probability distribution over the robot's trajectory and the map of the environment, given all the measurements plus the initial position:

$$p\,(\,\boldsymbol{x}_{1:T},\ m \mid \boldsymbol{z}_{0:T},\ \boldsymbol{u}_{1:T},\ \boldsymbol{x}_0\,).$$

To solve this problem, the robot needs to be endowed with two more mathematical tools: a motion model relating odometry measurements $\boldsymbol{u}_t$ to robot locations $\boldsymbol{x}_{t-1}$ and $\boldsymbol{x}_t$, and a measurement model describing the working of sensors (i.e. relating measurements $\boldsymbol{z}_t$ to the map m and the robot location $\boldsymbol{x}_t$). In SLAM, it is common to think of those models as probability distributions as well: $p\,(\,\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1},\, \boldsymbol{u}_t\,)$ characterizes the probability distribution over the location $\boldsymbol{x}_t$ assuming that the robot started from $\boldsymbol{x}_{t-1}$ and measured odometry data $\boldsymbol{u}_t$ (motion model), and likewise $p\,(\,\boldsymbol{z}_t \mid \boldsymbol{x}_t,\, m\,)$ is the probability distribution for a measurement $\boldsymbol{z}_t$ taken at a given location $\boldsymbol{x}_t$ in a known environment m (sensor model). These relationships corresponds to the arcs in Fig. 1.1.

Provided that these models are known, various paradigms have been developed to tackle the SLAM problem. The choice of the type of algorithm to use will depend on the peculiarities of the application at hand and on the desired properties; the following section will give more insight on this subject.

1.3 Taxonomy of the SLAM Problem

SLAM algorithms can be classified along a number of different dimensions. As it will be clear, there is no single best solution to the SLAM problem. The method chosen will depend on a number of factors, such as the desired map resolution, the update time, the nature of the environment, the type of sensor the robot is equipped with, and so on. In the following the main distinctions of SLAM algorithms will be briefly discussed; more details can be found in [3].

Online Versus Offline

The first distinction that can be made is whether the SLAM algorithm processes data in real time (online) or not (offline). While offline methods are often batch (they process all data at the same time), online algorithms are usually incremental and are needed when some sort of real-time decision is to be made (e.g. when a robot must control its next motion step, or another process is waiting for input). To allow computation in real time, online methods must privilege speed and efficiency over accuracy; on the other hand, if the algorithm is passive (see below), real-time processing may not be indispensable, and therefore an offline method may be preferred to compute a more accurate solution.

Volumetric Versus Feature-Based

In volumetric SLAM, the map is sampled at a resolution high enough to allow for photorealistic reconstruction of the environment. The resulting map is dense; therefore the computation can become quite involved. Typical representations include occupancy grids [2], voxel grids, surface maps and octrees

[3]. On the other hand, feature-based algorithms only extract individual elements (landmarks) from the sensor stream. This results in a sparse representation of the environment that is easier to handle but also less satisfactory as far as structure reconstruction is concerned. Feature-based algorithms tend to be more efficient, but their results may be inferior to volumetric approaches due to the fact that the extraction of features discards information from the measurements. The choice of a particular map representation depends on the sensors used, on the characteristics of the environment, and on the estimation algorithm. Landmark maps are often preferred in environments where locally distinguishable features can be identified and especially when cameras are used. In contrast, dense representations are typically used in conjunction with range sensors (laser scanners, lidars, etc.).

Active Versus Passive

In passive implementations, some other entity controls the robot, and the SLAM algorithm is purely observing. The vast majority of algorithms are of this type; they give the robot designer the freedom to implement arbitrary motion controllers and pursue arbitrary motion objectives. In active SLAM, the robot actively explores the environment in pursuit of an accurate map. This yields more accurate maps in less time, but it constrains robot motion.

Static Versus Dynamic

Static algorithms assume that the environment does not change over time, while dynamic methods allow for changes. The vast majority of SLAM literature assumes static environments; dynamic effects are often treated just as measurement outliers. Methods that reason about motion in the environment are more involved, but tend to be more robust in most applications.

1.4 SLAM Paradigms

As the basic taxonomy above suggests, many types of SLAM algorithms are possible. To address all these situations, the main mathematical frameworks developed to date are three: the Extended Kalman Filter, the Particle Filter, and the Graph-Based representation.

1.4.1 Extended Kalman Filter

The Extended Kalman Filter (EKF) formulation of SLAM is historically the earliest [4], and perhaps the most influential. This approach assumes a feature-based environmental representation, in which objects can be effectively represented as points in an appropriate parameter space, and Gaussian noise in odometry and sensor measurements (i.e. the motion model $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_t)$ and the sensor model $p(\boldsymbol{z}_t \mid \boldsymbol{x}_t, m)$ can be represented by normal distributions). Then, a single state vector is used to store the robot poses and a set of landmarks, with an associated error covariance matrix representing the uncertainties in these estimates: the EKF algorithm represents the state of the system by a multivariate Gaussian distribution

$$p(\boldsymbol{x}_{1:t}, m \mid \boldsymbol{z}_{0:t}, \boldsymbol{u}_{1:t}, \boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$$

in which the high-dimensional vector $\boldsymbol{\mu}_t$ contains the robot's best estimate of its own location and the location of the features in the environment, while the matrix $\boldsymbol{\Sigma}_t$ is the covariance of the robot's estimated error in the guess $\boldsymbol{\mu}_t$; details on the implementation can be found in [5]. The covariance matrix is usually distinctly non-sparse, with off-diagonal elements capturing the correlations in the estimates of different variables. These correlations come along because the robot's location is uncertain, and as a result the locations

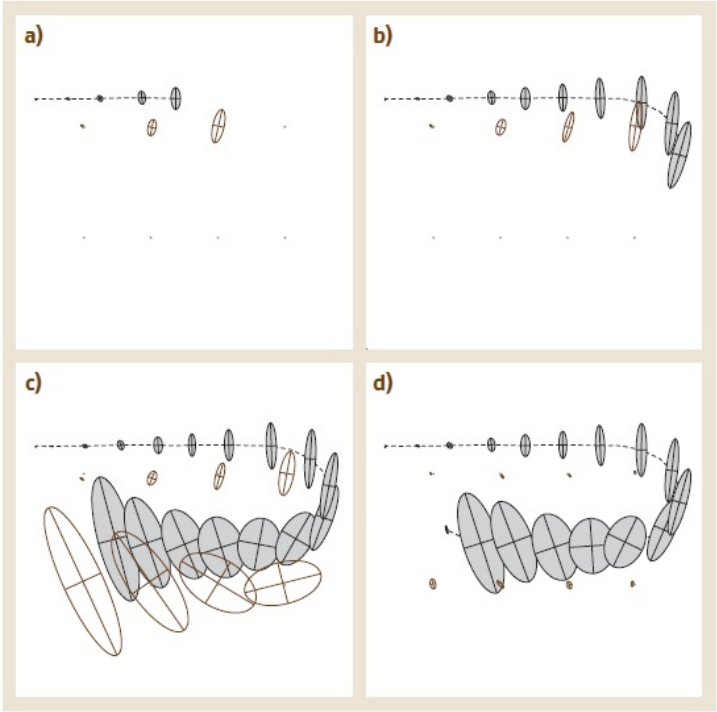

Figure 1.2: EKF applied to the SLAM problem [3]. The robot's path is a dotted line, and its estimates of its own position are shaded ellipses. Eight distinguishable landmarks of unknown location are shown as small dots, and their location estimates are shown as white ellipses.

of the landmarks in the map are uncertain. The effects of this correlation are shown in figure 1.2. The robot navigates from a starting pose that serves as the origin of its coordinate system. As it moves, its own pose uncertainty increases, as indicated by uncertainty ellipses of growing diameter. It also senses nearby landmarks and maps them; landmark uncertainty combines the fixed measurement uncertainty with the increasing pose uncertainty, and therefore grows over time as well. The interesting transition is illustrated in figure 1.2d: the robot observes the landmark it saw in the very beginning of the path, and whose location is relatively well known. Through this observation, the robot's pose error is reduced, and due to the correlations between

variables, this reduces the uncertainty for other landmarks in the map as well. This effect is one of the most important characteristic of the SLAM posterior: information that helps localize the robot is propagated through the map, and as a result the quality of the entire solution is improved.

A key issue of the EKF approach that caused its partial decline in the last few years lies in the quadratic nature of the covariance matrix. As the robot moves and observes new features, new states are added to the system state vector. The size of the covariance matrix grows quadratically with the size of the state vector, and therefore memory consumption and processing time are $O(n^2)$ in the size of the map, which can pose great limitations even for medium scale mapping.

1.4.2 Particle Filters

The first application of particle filtering to the SLAM problem can be tracked back to [6]. Particle filter (or Sequential Monte Carlo) methods represent the posterior through a set of samples (particles); each particle is best thought as a concrete guess as to what the true value of the state may be. By collecting many such guesses, the particle filters capture a representative sample from the posterior distribution. Sampling allows particle filters to represent any kind of probability distribution over the robot poses with ease; as the particle set size goes to infinity, the result has been shown (under some mild conditions) to approach the true posterior distribution.

At any point in time t, the algorithm maintains K particles of the type

$$(\boldsymbol{x}_{0:t}^{[k]}, \quad \boldsymbol{\mu}_1^{[k]} ... \boldsymbol{\mu}_n^{[k]}, \quad \boldsymbol{\Sigma}_1^{[k]} ... \boldsymbol{\Sigma}_n^{[k]})$$

that contain

- a sample path $\boldsymbol{x}_{0:t}^{[k]}$, assumed to be known perfectly. The path uncertainty is not lost, but rather reflected in the presence of other particles holding a different belief of the trajectory.

- a set of N gaussians with means $\boldsymbol{\mu}_i^{[k]}$ and variances $\boldsymbol{\Sigma}_i^{[k]}$, representing the landmarks in the map.

The fact that elements of the map in a particle are stored as single low-dimensional Gaussians instead of being variables of a unique high-dimensional multivariate distribution is one of the advantage of particle filter methods over EKF implementations. The justification for this decomposition arises from a specific conditional independence assumption that can be applied in the context of the single particles. In SLAM, any dependence between multiple landmark estimates comes from the mediation through the uncertain robot path. If the robot poses are assumed to be known (as is in each particle), then all landmark estimates become independent. This implies that if a large Gaussian were used to represent the entire map (one per particle, of course), the off-diagonal elements between different landmarks would remain zero. It is therefore legitimate to implement the map more efficiently, using N small Gaussians, one for each landmark.

The algorithm works as follows. Initialization sets each particle's robot location to its known starting coordinates, and zeroes the map. Then:

- When an odometry reading is received, for each particle new locations are generated stochastically using the probability distribution given by the motion model:

$$\boldsymbol{x}_t^{[k]} \sim p\,(\,\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}^{[k]},\ \boldsymbol{u}_t\,).$$

  $\boldsymbol{x}_{t-1}^{[k]}$ is the previous location, which is part of the particle. This probabilistic sampling step is easily implemented for any robot whose kine-

matics can be computed.

- When a measurement $\boldsymbol{z}_t$ is received, two things happen: first, for each particle k an importance weight is computed that measures how important the particle is in the light of the new sensor measurement. If n is the index of the sensed landmark, the importance weight is computed from the sensor model as

$$w^{[k]} = p\,(\,\boldsymbol{z}_t \mid \boldsymbol{x}_t^{[k]},\ \boldsymbol{\mu}_n^{[k]},\ \boldsymbol{\Sigma}_n^{[k]}).$$

  Next, a step called resampling takes place: a set of new particles is drawn from the set of existing ones, with the probability of drawing a particle being its normalized importance weight. The intuition behind resampling is simple: particles for which the acquired measurement is more congruent with the state have a higher chance of surviving the process, while particle whose state is less likely in the light of the new information will probably be discarded. Resampling is necessary to control the number of particles maintained, that otherwise would scale exponentially with the dimension of the underlying state space. Finally, based on the measurement $\boldsymbol{z}_t$, the means $\boldsymbol{\mu}_n$ and covariances $\boldsymbol{\Sigma}_n$ are updated for the new set of particles, following the standard EKF update rules [5].

As time passes, particles with bad estimates of the state tend to be discarded by the resampling process, while good particles survive to retain a sample of the posterior probability distribution over the robot trajectory and the map.

Particle filter algorithms have some remarkable properties. First, by decorrelating the landmarks they sidestep some of the issues arising from the natural inter-feature correlations in the map, which plagued the EKF. Using advanced tree methods to represent the maps, the update can be performed

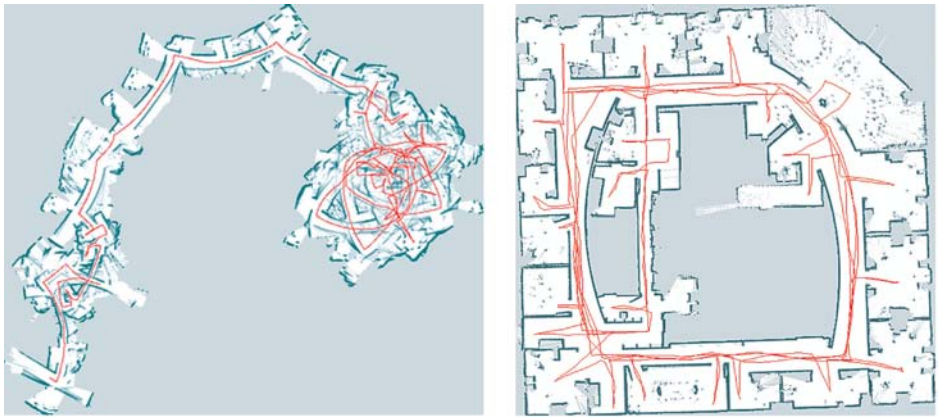

Figure 1.3: Occupancy grid map generated from laser range data, based on pure odometry (left) and resulting from a scan-matching particle filter algorithm (right) [7].

in time logarithmic in the size of the map N, and linear in the number of particles K, while EKF needed quadratic time. Second, they allow for multiple data association hypotheses. The term *data association* denotes the process of identifying the entity a given observation refers to (see section 4.3). In case of an ambiguous situation, it is straightforward to make data association decisions on a per-particle basis, while EKF implementations must commit to the same hypothesis for the entire filter. As a result, while EKF is extremely vulnerable to outliers, this extra degree of freedom grants particle filters a significant improvement in robustness. Furthermore, they can be extended to obtain dense map representations. Figure 1.3 shows a grid-based version in which Gaussians were replaced by occupancy grid maps obtained from scan matching [7]. These properties, along with the relative ease of implementation, have made particle filters a popular choice; on the negative side however, the number of necessary particles can grow very large, especially for robots seeking to map multiple nested loops.

1.4.3 Graph-Based

The Graph-Based method draws its intuition from a graphical representation of the SLAM problem. Landmarks and robot poses can be thought of as nodes in a graph, linked by soft constraints established by odometry measurements and sensor observations. Relaxing these constraints yields the robot's best estimate for the map and the full path. From a theoretical point of view, the graph-based approach has been known for quite a while [8], but only became popular for SLAM in the last few years due to the comparably high complexity of solving this constrained optimization problem using standard techniques. Recent insights into the structure of the SLAM problem and advancements in the field of sparse linear algebra however resulted in efficient approaches to the optimization problem at hand. Consequently, graph-based methods have undergone a renaissance, and currently belong to the state-of-the-art techniques with respect to speed and accuracy.

Referring to figure 1.4, every time an odometry measurement $\boldsymbol{u}_i$ is received, a new pose $\boldsymbol{x}_i$ is created and linked with the previous one with a constraint based on the odometry measurement itself. Each observation of the environment $\boldsymbol{z}_{ij}$ creates a constraint between a landmark j and the pose i from which it is observed. As the robot explores the environment, the graph grows in size as new nodes and constraints are added; nevertheless, the graph is typically sparse, in that each node is only connected to a small number of other nodes. Optimization with respect to the various constraints then yields the map of the environment and the path of the robot that best agree with odometry and sensor data. It can be shown that optimizing the graph with respect to the squared error introduced by the various constraints (section 3.2) is equivalent to computing a Gaussian approximation of the posterior probability distribution over the robot's path and the map [9].

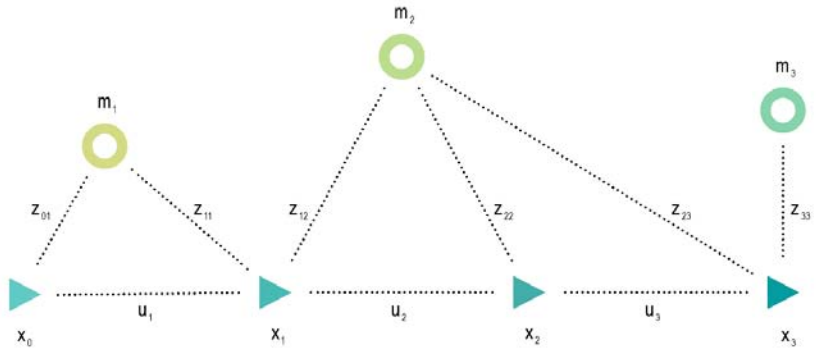

Figure 1.4: Illustration of the graph. $\boldsymbol{x}_i$ denote robot poses, $\boldsymbol{u}_i$ odometry measurements, m_i landmarks, and $\boldsymbol{z}_{ij}$ sensor observations.

This representation of the problem has a number of interesting properties.

- The number of nodes and constraints in the graph is (at worst) linear in the time elapsed, and the graph is sparse. This causes graphical SLAM methods to scale to much higher-dimensional maps than EKF and particle filters. In EKF SLAM, the space and update time required to handle the covariance matrix grow quadratically with the size of the map. Particle filter methods overcome this issue decorrelating features in the map, but the resulting advantage is somewhat reduced by the necessity to store a significant number of particles. In graphical methods, no such limitation exists: memory consumption and time required for sparse optimization are linear in the size of the map.

- The graph is a very convenient representation of the problem. Any additional information on the topology of the scene or on the robot's trajectory can be easily integrated into the graph by adding appropriate constraints between the nodes. Subsets of nodes or even single variables can be fixed by excluding them from the optimization process [9], and nodes can be grouped to simplify the problem [10].

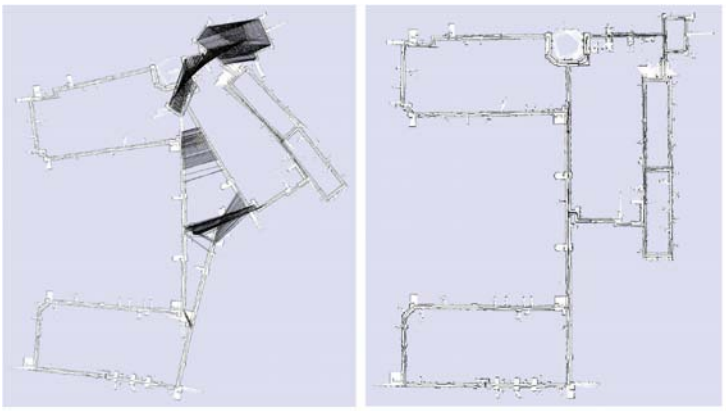

Figure 1.5: Pose-graph corresponding to a data-set recorded at MIT Killian Court, before (left) and after optimization (right) [9].

- The formulation can be extended to dense map representations (figure 1.5). In this case, the graph is only comprised of poses; dense measurements are compared to provide additional information about robot displacement, which is used to further constrain the poses. Once the graph has been optimized, the map is obtained by rendering the dense measurements according to the calculated robot poses.

- Although the original formulation assumes Gaussian probability distributions, the method can be extended to account for multi-modal distributions with a max-mixture model [11].

- Erroneous constraints can be easily identified and managed (see section 3.3.2). This is a crucial requirement for a robust implementation.

- Although most methods in the field optimize over the entire path at each iteration possibly leading to computational requirements too demanding for online application, hierarchical approaches have been developed that overcome this limitation [10]. These algorithms structure the problem in various levels by grouping nodes in the graph (figure

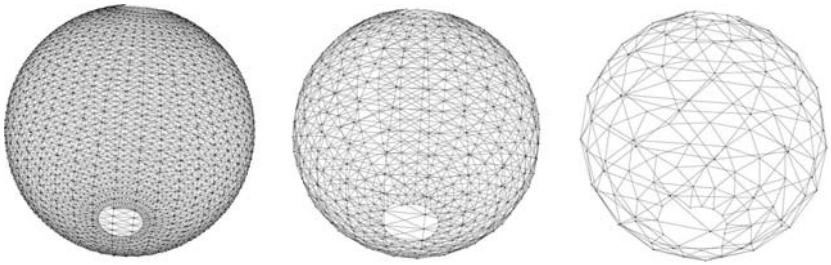

Figure 1.6: Three-level hierarchy for a 2,000 nodes and 8,647 constraints 3D network [10].

1.6). Optimization on the higher levels is faster and provides a working solution for online applications, while the best accuracy can still be retrieved optimizing the full problem offline.

Table 1.1 summarizes the main characteristics of the graph-based method confronted with EKF and particle filters. The advantageous properties of the graph-based approach determined its choice as the method to implement for this work.

	EKF	**Particle Filters**	**Graph-Based**
Complexity	$O(n^2)$	$O(k \log(n))$	$O(n)$
Map	Sparse	Sparse/Dense	Sparse/Dense
Assumed Distributions	Gaussian	Poses: Any (Sampling) Landmarks: Gaussian	Gaussian, Multi Modal
Flexibility/ Robustness	-	Multiple Data Association	Robust to Outliers Multi-Modal Distributions Flexible Graph Handling
Large Scale Mapping	-	Less complex than EKF	Linear Complexity Hierarchical Maps

Table 1.1: Comparison of SLAM paradigms.

Graph-Based Implementation

Chapter 2

Introductive Remarks

2.1 On Graph-Based SLAM

As it was already introduced, graph-based methods represent the SLAM problem through a graph that once optimized yields the robot's path and the map that are more likely given all sensor measurements. Two main tasks can be identified in the process: constructing the graph from raw sensor data and optimizing it.

Front-End: Graph Construction

The part of the algorithm that is responsible for building the graph from the raw measurements is called the SLAM front-end. The graph is constructed according to the following criteria:

- Initialization adds the first pose (the origin) to the empty graph. Additional poses are added to the graph each time an odometry measurement is received. New poses can be initialized to the value predicted by the motion model given the previous pose and the odometry read-

ing. Then, the odometry measurement is used to generate a constraint between the new pose and the previous one.

- From each pose, a number of features of the environment will be observed. For each observation, the algorithm must recognize whether it is relative to a feature already added to the map or not (the so-called data association). Then,
  - If correspondence to a feature of the map is found, a constraint is added between that feature and the current pose based on the observation;
  - If no correspondence is found, a decision must be made whether to add the apparently new feature to the map or not; if the feature is added to the map it is constrained to the current pose.

Clearly, the algorithm needed to perform these tasks will strongly depend on the type of sensor used. The details of the front-end for this implementation will be discussed in chapter 4.

Back-End: Optimization

Once the graph has been built by the front-end, it needs to be optimized. This task involves solving a large error minimization problem and is accomplished by the so-called back-end. The back-end works on an abstract representation of data (the graph) and therefore its implementation is independent of the type of sensor the robot is equipped with. The back-end for this implementation will be discussed in chapter 3.

2.2 On This Implementation

The work of this thesis was aimed at implementing a vision-based SLAM algorithm. For a long time in the history of the SLAM field, much attention was given to sensors such as laser range finders and sonars, for the ease of relating their measurements with the geometry of the environment. Cameras on the other hand capture the world's geometry only indirectly through photometric effects, and for many years it was thought too difficult to turn the sparse sets of features popping out of an image into reliable long-term maps. Nowadays, however, cameras are well-understood, compact, accurate, non-invasive and very cheap, making them an appealing sensor for mobile robotics. Cameras are not only much cheaper than alternative sensors such as laser range finders and radar systems, but they also contain more information per sample and work with much higher data rates. Furthermore, cameras provide a way to identify features. Most SLAM approaches rely on geometric parameters to recognize landmarks: observations are predicted from landmark positions and the current robot pose estimate, and compared to the actual observations. When the errors on some of these positions are large, e.g. when the robot reperceives landmarks after having travelled a long loop trajectory, the association can become ambiguous. This is all the more difficult when the robot is evolving in 3D, the errors in the estimate of the 6 parameters of the robot position having rapidly a drastic influence on the predicted landmark observations. A robust way to solve the data association problem then is to recognize landmarks independently from their position estimate: a good visual feature detection and matching algorithm can provide this ability. The sum of all these proprieties determined a great increase in the popularity of vision-based SLAM systems over the last years.

To use cameras as a sensor, two different setups are possible: stereo and

monocular. With stereo vision, the 3D coordinates of features with respect to the robot are easily obtained by matching and triangulating points in the stereoscopic image pair. If the robot is endowed with a single camera, on the other hand, only the bearings of the features can be observed. A single observation therefore is not sufficient to compute the state of a landmark completely, and a dedicated procedure that integrates several observations over time is required to initialize and update landmarks, which implies a significant complication of the algorithm. A comparison of the results of stereo and monocular approaches is presented in [12]. In this work, the two setups produce similar results as far as accuracy is concerned; therefore, given the greater complexity of monocular systems, stereo vision was chosen as the solution to use.

Referring to the taxonomy outlined in section 1.3, the map representation resulting from the use of a stereo camera is clearly feature-based. Moreover, the algorithm that has been developed is static. Changes in the environment could cause potential outliers and therefore lead to a less accurate result. The system is passive (it does not control the motion of the robot, but is simply observing) and incremental: at each time step, the front end expands the graph and the back end optimizes it, so that at the next time step the graph is expanded from the optimum configuration just obtained. The algorithm thus works continuously interleaving the execution of front end and back end. Being incremental, it is suited for online application if computational requirements are met (see chapter 5).

To keep the implementation as general as possible, no odometry information was assumed to be available. Therefore, no constraint directly links the poses in the graph, which are correlated only through landmark observations. The resulting algorithm works solely on the data obtained from the stereo

camera; a sequence of stereo images with the associated calibration parameters are the only things required for motion and environment reconstruction.

Finally, the algorithm solves the full 3D problem (6 degrees of freedom). If the robot is equipped with an IMU that provides the pitch and roll angles, the algorithm can be sped up by removing these variables from the optimization process (see section 3.4).

Chapter 3

Back End

3.1 Introduction

In graph-based SLAM [3, 9, 10], poses and landmarks are represented by nodes in a graph linked by soft constraints encoding sensor measurements. Such a graph represents an overdetermined problem, because in normal conditions each node will have multiple constraints attached to it (a landmark will have a link to each pose from which it has been seen, and a pose will have a link to each landmark that can be seen from it). Once the graph has been set up from raw sensor data by the front end (chapter 4), optimization with respect to the various constraints yields the optimum configuration of the nodes, i.e. the best estimate of the robot path and the map given all sensor information. There are however many ways of defining optimum, depending upon the criteria chosen to evaluate performance. In graph-based SLAM, the standard optimization procedure is least squares minimization. The reason for this is closely related to probability estimation: optimizing the graph with a least squares scheme is the equivalent of estimating a Gaussian approximation of the posterior distribution over the robot trajectory and the

map [9]. This chapter therefore will give a general overview of the method of least squares (section 3.2) followed by a discussion on its application to the SLAM problem in particular (section 3.3).

3.2 Least Squares Minimization

Let $\boldsymbol{X}$ be a state vector representing the (unknown) configuration of a system and $\{\boldsymbol{z}_i\}_{i=1:n}$ a set of n noisy measurements about the state $\boldsymbol{X}$. Given the n functions $\{f_i(\boldsymbol{X})\}_{i=1:n}$ that map $\boldsymbol{X}$ to the expected measurements $\{\hat{\boldsymbol{z}}_i\}_{i=1:n}$, we want to estimate the state $\boldsymbol{X}^*$ for which the expected measurements $\hat{\boldsymbol{z}}_{1:n}$ best agree with the actual measurements $\boldsymbol{z}_{1:n}$ (figure 3.1).

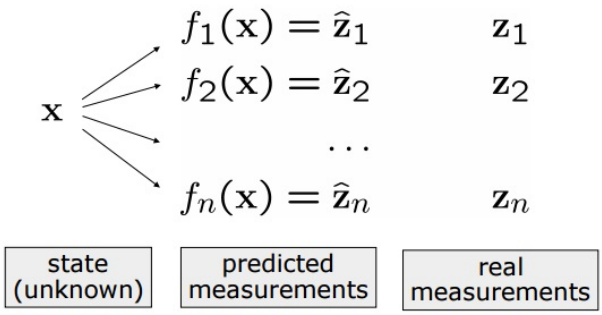

Figure 3.1: Graphic representation of the problem.

The optimum configuration of the system is found minimizing a global error function $F(\boldsymbol{X})$ that quantifies the deviation of the expected observations from the actual ones. This function is the total squared error introduced by the n real measurements with respect to the expected measurements, and can be expressed as follows.

The (non-squared) error $\boldsymbol{e}_i(\boldsymbol{X})$ relative to a single measurement is simply the difference between its expected value and its actual value:

$$\boldsymbol{e}_i(\boldsymbol{X}) = \boldsymbol{z}_i - \hat{\boldsymbol{z}}_i = \boldsymbol{z}_i - f_i(\boldsymbol{X}) \tag{3.1}$$

Since the measurements can be multi-dimensional (e.g. the 3D position of a landmark), the non-squared error in general will be a vector. Furthermore, we consider that sensor observations are affected by noise. Assuming that this noise can be modelled with a Gaussian distribution with mean zero and information matrix $\boldsymbol{\Omega}_i$ [9], the squared error $e_i^2(\boldsymbol{X})$ of a measurement can be written as

$$e_i^2(\boldsymbol{X}) = \boldsymbol{e}_i^T(\boldsymbol{X})\ \boldsymbol{\Omega}_i\ \boldsymbol{e}_i(\boldsymbol{X}),$$

in which the information matrix of the measurement is included to account for its uncertainty. A measure of the information matrix can be given by the inverse of the observation covariance matrix; the higher the accuracy of a measurement, the smaller the covariance matrix and the bigger the information matrix, which means that measurements with higher confidence will have a greater weight in the process. The squared error of a measurement is a scalar and as the non-squared error depends only on the state $\boldsymbol{X}$. The problem is then formalized as finding the state $\boldsymbol{X}^*$ that minimizes the sum of all the squared errors introduced by each measurement:

$$\begin{aligned} \boldsymbol{X}^* &= \underset{\boldsymbol{X}}{\operatorname{argmin}}\ F(\boldsymbol{X}) \\ &= \underset{\boldsymbol{X}}{\operatorname{argmin}}\ \sum_{i=1}^{n} e_i^2(\boldsymbol{X}) \\ &= \underset{\boldsymbol{X}}{\operatorname{argmin}}\ \sum_{i=1}^{n} \boldsymbol{e}_i^T(\boldsymbol{X})\ \boldsymbol{\Omega}_i\ \boldsymbol{e}_i(\boldsymbol{X}) \end{aligned} \tag{3.2}$$

3.2.1 The Gauss-Newton algorithm

A general solution for the problem (3.2) would be to derive $F(\boldsymbol{X})$ and find its nulls. However, if the functions $f_i(\boldsymbol{X})$ (and therefore $\boldsymbol{e}_i(\boldsymbol{X})$) are not linear in all unknowns, the problem can become very complex and in general has no closed form solution; therefore, a numerical approach becomes necessary. Under the assumptions that a good initial guess for the state $\boldsymbol{X}$ is available and that the error functions are smooth in the neighbourhood of the minimum [9], the problem can be solved by iterative local linearizations. The procedure can be summarized as follows:

1. Linearize the error functions $\boldsymbol{e}_i(\boldsymbol{X})$ around the initial guess $\boldsymbol{X}$.
2. Compute the resulting (approximated) global error function $F(\boldsymbol{X})$.
3. Compute its derivative, set it to zero and solve to obtain a new state.
4. Iterate until convergence.

This procedure, also known as Gauss-Newton algorithm, will now be discussed in greater detail. The error functions of the single measurements can be approximated around the initial guess $\check{\boldsymbol{X}}$ by a Taylor expansion:

$$\boldsymbol{e}_i(\check{\boldsymbol{X}} + \boldsymbol{\Delta X}) \simeq \underbrace{\boldsymbol{e}_i(\check{\boldsymbol{X}})}_{\boldsymbol{e}_i} + \underbrace{\boldsymbol{J}_i(\check{\boldsymbol{X}})}_{\boldsymbol{J}_i} \boldsymbol{\Delta X} \tag{3.3}$$

Where $\boldsymbol{J}_i(\check{\boldsymbol{X}})$ is the Jacobian of the $i-$th error function evaluated at $\check{\boldsymbol{X}}$:

$$\boldsymbol{J}_i(\check{\boldsymbol{X}}) = \begin{pmatrix} \frac{\partial e_{i,1}(\check{\boldsymbol{X}})}{\partial X_1} & \frac{\partial e_{i,1}(\check{\boldsymbol{X}})}{\partial X_2} & \cdots & \frac{\partial e_{i,1}(\check{\boldsymbol{X}})}{\partial X_m} \\ \frac{\partial e_{i,2}(\check{\boldsymbol{X}})}{\partial X_1} & \frac{\partial e_{i,2}(\check{\boldsymbol{X}})}{\partial X_2} & \cdots & \frac{\partial e_{i,2}(\check{\boldsymbol{X}})}{\partial X_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{i,l}(\check{\boldsymbol{X}})}{\partial X_1} & \frac{\partial e_{i,l}(\check{\boldsymbol{X}})}{\partial X_2} & \cdots & \frac{\partial e_{i,l}(\check{\boldsymbol{X}})}{\partial X_m} \end{pmatrix} \tag{3.4}$$

With the linearization in (3.3), we can keep $\check{\boldsymbol{X}}$ fixed and carry out the minimization in the increment $\boldsymbol{\Delta X}$. The squared error in the neighbourhood of the initial guess $\check{\boldsymbol{X}}$ becomes

$$\begin{aligned} e_i^2\left(\check{\boldsymbol{X}} + \boldsymbol{\Delta X}\right) &= \left[\boldsymbol{e}_i(\check{\boldsymbol{X}} + \boldsymbol{\Delta X})\right]^T \, \boldsymbol{\Omega}_i \, \left[\boldsymbol{e}_i(\check{\boldsymbol{X}} + \boldsymbol{\Delta X})\right] \\ &\simeq \left(\boldsymbol{e}_i + \boldsymbol{J}_i \, \boldsymbol{\Delta X}\right)^T \, \boldsymbol{\Omega}_i \, \left(\boldsymbol{e}_i + \boldsymbol{J}_i \, \boldsymbol{\Delta X}\right) \\ &= \left(\boldsymbol{e}_i^T + \boldsymbol{\Delta X}^T \boldsymbol{J}_i^T\right) \, \boldsymbol{\Omega}_i \, \left(\boldsymbol{e}_i + \boldsymbol{J}_i \, \boldsymbol{\Delta X}\right) \\ &= \boldsymbol{e}_i^T \boldsymbol{\Omega}_i \boldsymbol{e}_i + \boldsymbol{e}_i^T \boldsymbol{\Omega}_i \boldsymbol{J}_i \boldsymbol{\Delta X} + \boldsymbol{\Delta X}^T \boldsymbol{J}_i^T \boldsymbol{\Omega}_i \boldsymbol{e}_i + \boldsymbol{\Delta X}^T \boldsymbol{J}_i^T \boldsymbol{\Omega}_i \boldsymbol{J} \boldsymbol{\Delta X} \end{aligned} \tag{3.5}$$

in which dependency on $\check{\boldsymbol{X}}$ of $\boldsymbol{e}_i$ and $\boldsymbol{J}_i$ has been omitted for simplicity. Since $e_i^2\left(\check{\boldsymbol{X}} + \boldsymbol{\Delta X}\right)$ is a scalar, all summands on the right-hand side of (3.5) are scalars and therefore we can transpose any of them. Transposing the third term and considering that $\boldsymbol{\Omega}_i = \boldsymbol{\Omega}_i^T$ (it is an information matrix) we obtain:

$$\begin{aligned} e_i^2(\check{\boldsymbol{X}} + \boldsymbol{\Delta X}) \; &\simeq \; \underbrace{\boldsymbol{e}_i^T \boldsymbol{\Omega}_i \boldsymbol{e}_i}_{c_i} + 2 \, \underbrace{\boldsymbol{e}_i^T \boldsymbol{\Omega}_i \boldsymbol{J}_i}_{\boldsymbol{b}_i^T} \boldsymbol{\Delta X} + \boldsymbol{\Delta X}^T \underbrace{\boldsymbol{J}_i^T \boldsymbol{\Omega}_i \boldsymbol{J}_i}_{\boldsymbol{H}_i} \boldsymbol{\Delta X} \\ &= c_i + 2 \, \boldsymbol{b}_i^T \boldsymbol{\Delta X} + \boldsymbol{\Delta X}^T \boldsymbol{H}_i \, \boldsymbol{\Delta X}. \end{aligned}$$

The global error function is the sum of the squared errors of the individual measurements, therefore:

$$\begin{aligned} F(\check{\boldsymbol{X}} + \boldsymbol{\Delta X}) &= \sum_{i=1}^{n} e_i^2 \, (\check{\boldsymbol{X}} + \boldsymbol{\Delta X}) \\ &\simeq \sum_{i=1}^{n} (\, c_i + 2 \, \boldsymbol{b}_i^T \boldsymbol{\Delta X} + \boldsymbol{\Delta X}^T \boldsymbol{H}_i \, \boldsymbol{\Delta X} \,) \\ &= \underbrace{\sum_{i=1}^{n} c_i}_{c} + 2 \underbrace{\left(\sum_{i=1}^{n} \boldsymbol{b}_i^T \right)}_{\boldsymbol{b}^T} \boldsymbol{\Delta X} + \boldsymbol{\Delta X}^T \underbrace{\left(\sum_{i=1}^{n} \boldsymbol{H}_i \right)}_{\boldsymbol{H}} \boldsymbol{\Delta X} \end{aligned}$$

in which c, $\boldsymbol{b}^T$ and $\boldsymbol{H}$ do not depend on $\boldsymbol{\Delta X}$: the global error function in the neighbourhood of the current solution $\check{\boldsymbol{X}}$ is thus approximated with a quadratic form in $\boldsymbol{\Delta X}$,

$$F(\check{\boldsymbol{X}} + \boldsymbol{\Delta X}) \;\simeq\; c + 2\, \boldsymbol{b}^T \boldsymbol{\Delta X} + \boldsymbol{\Delta X}^T \boldsymbol{H} \boldsymbol{\Delta X} \tag{3.6}$$

with

$$\boldsymbol{b}^T = \sum_{i=1}^{n} \boldsymbol{e}_i^T(\check{\boldsymbol{X}}) \, \boldsymbol{\Omega}_i \, \boldsymbol{J}_i(\check{\boldsymbol{X}}) \tag{3.7}$$

$$\boldsymbol{H} = \sum_{i=1}^{n} \boldsymbol{J}_i^T(\check{\boldsymbol{X}}) \, \boldsymbol{\Omega}_i \, \boldsymbol{J}_i(\check{\boldsymbol{X}}) \tag{3.8}$$

The first derivative of a quadratic form $f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{H} \boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x}$ is [13]

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = (\boldsymbol{H} + \boldsymbol{H}^T) \, \boldsymbol{x} + \boldsymbol{b}$$

Hence, taking the derivative of (3.6) with respect to $\boldsymbol{\Delta X}$ yields

$$\frac{\partial F(\check{\boldsymbol{X}} + \boldsymbol{\Delta X})}{\partial \boldsymbol{\Delta X}} \simeq \frac{\partial}{\partial \boldsymbol{\Delta X}} (c + 2\boldsymbol{b}^T \boldsymbol{\Delta X} + \boldsymbol{\Delta X}^T \boldsymbol{H} \boldsymbol{\Delta X})$$

$$= 2\boldsymbol{b} + (\boldsymbol{H} + \boldsymbol{H}^T)\boldsymbol{\Delta X}$$

Setting it to zero and considering that from (3.8) it follows $\boldsymbol{H} = \boldsymbol{H}^T$ (since $\boldsymbol{\Omega}_i = \boldsymbol{\Omega}_i^T$) we obtain

$$0 = 2\boldsymbol{b} + 2\boldsymbol{H}\boldsymbol{\Delta X}$$

Which in turn leads to the linear system

$$\boldsymbol{H}\boldsymbol{\Delta X} = -\boldsymbol{b} \tag{3.9}$$

And the solution for the increment $\boldsymbol{\Delta X}$ that minimizes the approximated global error function is therefore

$$\boldsymbol{\Delta X} = -\boldsymbol{H}^{-1}\boldsymbol{b}$$

which can be used to update the current estimate of the state $\check{\boldsymbol{X}}$. Thus, the iterative scheme can be summarized this way:

1. Given the initial guess/current state estimate $\check{\boldsymbol{X}}$, calculate for each measurement $\boldsymbol{e}_i(\check{\boldsymbol{X}})$ from (3.1) and $\boldsymbol{J}_i(\check{\boldsymbol{X}})$ from (3.4)
2. Calculate $\boldsymbol{b}$ from (3.7) and $\boldsymbol{H}$ from (3.8)
3. Solve the linear system (3.9) for the increment $\boldsymbol{\Delta X}$
4. Update the state $\check{\boldsymbol{X}}_{new} = \check{\boldsymbol{X}} + \boldsymbol{\Delta X}$ and iterate until convergence.

The following section will investigate the application of this result to the optimization of the SLAM graph.

3.3 Graph Optimization

Let us give a formulation of the SLAM problem appropriate to apply the method of least squares introduced in section 3.2. The notation is as follows:

$(\hat{\boldsymbol{I}}, \hat{\boldsymbol{J}}, \hat{\boldsymbol{K}})$	Global reference frame
(X, Y, Z)	Global coordinates
$(\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}, \hat{\boldsymbol{k}})$	Robot reference frame
(x, y, z)	Robot coordinates

In graph-based SLAM, we are interested in determining the position of a number of nodes, each representing a robot pose or a landmark. Once a global reference frame ($\hat{\boldsymbol{I}}$, $\hat{\boldsymbol{J}}$, $\hat{\boldsymbol{K}}$) has been defined (generally fixing the first pose as the origin), the parameters necessary to define a robot pose i are its location (X_i, Y_i, Z_i) and the three attitude angles (α_i, β_i, γ_i). This is equivalent to specifying the origin and orientation of the robot reference frame with respect to the global reference frame. On the other hand, for a point landmark j the only geometric parameters necessary to characterize it are its global coordinates in the map (X_j, Y_j, Z_j). A given configuration of the nodes in the graph can then be described by a single state vector $\boldsymbol{X}$ storing all these variables (poses and landmarks):

$$\boldsymbol{X}^T = \begin{pmatrix} \boldsymbol{X}_1^T & \boldsymbol{X}_2^T & \cdots & \boldsymbol{X}_n^T \end{pmatrix}$$

Where each $\boldsymbol{X}_i$ is a column vector representing a pose (6 variables) or landmark (3 variables).

Since it was assumed that no odometry information is available, the measurements are just the observations of the landmarks from the various poses. In the following the subscript i will be used to refer to a pose, while the subscript j will always denote a landmark. Using a stereo camera we are able to triangulate points, and therefore each observation will be a measurement of the position of landmark j relative to pose i (i.e. the coordinates of landmark j in the robot reference frame of pose i):

$$\boldsymbol{z}_{ij} = \begin{pmatrix} x_{ij} \\ y_{ij} \\ z_{ij} \end{pmatrix},$$

with an associated information matrix $\boldsymbol{\Omega}_{ij}$ that encodes triangulation uncertainty (see section 4.2.3).

The functions $f_{ij}(\boldsymbol{X})$ mapping the state vector to the expected observations are simply the geometric transformations that map the global coordinates of a pair pose-landmark to a predicted relative measurement (see section 3.4). Clearly, this geometric transformation has the same form for all pairs pose-landmark and the expected observation of landmark j from pose i only depends on the relative geometry between the two, thus:

$$\hat{\boldsymbol{z}}_{ij} = f_{ij}(\boldsymbol{X}) = f(\boldsymbol{X}_i, \boldsymbol{X}_j). \tag{3.10}$$

We now have all instruments necessary to apply the Gauss-Newton algorithm to the SLAM problem. Recalling the procedure discussed in section 3.2:

1. Given the current state estimate $\boldsymbol{X}$, calculate for each measurement

$$\boldsymbol{e}_{ij}(\boldsymbol{X}) = \boldsymbol{z}_{ij} - \hat{\boldsymbol{z}}_{ij} \qquad \text{and} \qquad \boldsymbol{J}_{ij}(\boldsymbol{X}) = \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{X})}{\partial \boldsymbol{X}}$$

2. Build the linear system from (3.7) and (3.8):

$$\boldsymbol{b}^T = \sum_{ij} \boldsymbol{e}_{ij}^T \, \boldsymbol{\Omega}_{ij} \, \boldsymbol{J}_{ij} \qquad \text{and} \qquad \boldsymbol{H} = \sum_{ij} \boldsymbol{J}_{ij}^T \, \boldsymbol{\Omega}_{ij} \, \boldsymbol{J}_{ij}$$

3. Solve the linear system $\boldsymbol{H}\boldsymbol{\Delta}\boldsymbol{X} = -\boldsymbol{b}$ for the increment $\boldsymbol{\Delta}\boldsymbol{X}$, update the state and iterate.

Since the expected observation of a landmark j from a pose i only depends on the relative geometry between the two, it is clear that also the error term $\boldsymbol{e}_{ij}(\boldsymbol{X})$ will depend on $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ alone. This is apparent from (3.10): since $\boldsymbol{e}_{ij}(\boldsymbol{X}) = \boldsymbol{z}_{ij} - \hat{\boldsymbol{z}}_{ij}$, it follows that $\boldsymbol{e}_{ij}(\boldsymbol{X}) = \boldsymbol{e}_{ij}(\boldsymbol{X}_i, \boldsymbol{X}_j)$. This implies that

$$\frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{X})}{\partial \boldsymbol{X}_k} = \boldsymbol{0} \qquad \text{if } k \neq i, j$$

and therefore the Jacobian $\boldsymbol{J}_{ij}$ will be zero everywhere except in the columns corresponding to $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$:

$$\boldsymbol{J}_{ij}(\boldsymbol{X}) = \begin{pmatrix} \boldsymbol{0} & \cdots & \boldsymbol{0} & \underbrace{\dfrac{\partial \boldsymbol{e}_{ij}(\boldsymbol{X}_i, \boldsymbol{X}_j)}{\partial \boldsymbol{X}_i}}_{\boldsymbol{A}_{ij}} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \underbrace{\dfrac{\partial \boldsymbol{e}_{ij}(\boldsymbol{X}_i, \boldsymbol{X}_j)}{\partial \boldsymbol{X}_j}}_{\boldsymbol{B}_{ij}} & \boldsymbol{0} & \cdots & \boldsymbol{0} \end{pmatrix}.$$

The sparse nature of the Jacobian influences the structure of the linear system (3.9). The right-hand side is

$$\boldsymbol{b}^T = \sum_{ij} \boldsymbol{b}_{ij}^T = \sum_{ij} \boldsymbol{e}_{ij}^T \, \boldsymbol{\Omega}_{ij} \, \boldsymbol{J}_{ij}$$

In which each $\boldsymbol{b}_{ij}^T$ can be written as

$$\begin{aligned}
\boldsymbol{b}_{ij}^T &= \boldsymbol{e}_{ij}^T \, \boldsymbol{\Omega}_{ij} \, \boldsymbol{J}_{ij} \\
&= \boldsymbol{e}_{ij}^T \, \boldsymbol{\Omega}_{ij} \, (\, \mathbf{0} \, \cdots \, \boldsymbol{A}_{ij} \, \cdots \, \boldsymbol{B}_{ij} \cdots \, \mathbf{0} \,) \\
&= \big(\, \mathbf{0} \, \cdots \, \boldsymbol{e}_{ij}^T \, \boldsymbol{\Omega}_{ij} \, \boldsymbol{A}_{ij} \, \cdots \, \boldsymbol{e}_{ij}^T \, \boldsymbol{\Omega}_{ij} \, \boldsymbol{B}_{ij} \, \cdots \, \mathbf{0} \, \big)
\end{aligned}$$

and hence it is non-zero only at the blocks corresponding to $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ (figure 3.2):

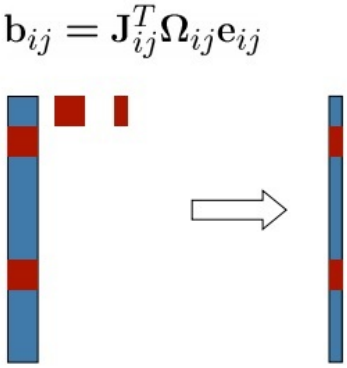

Figure 3.2: Structure of $\boldsymbol{b}_{ij}$

Similarly, for the matrix $\boldsymbol{H}$ we can write

$$\boldsymbol{H} = \sum_{ij} \boldsymbol{H}_{ij} = \sum_{ij} \boldsymbol{J}_{ij}^T \, \boldsymbol{\Omega}_{ij} \, \boldsymbol{J}_{ij}$$

with

$$
\begin{aligned}
\boldsymbol{H}_{ij} &= \boldsymbol{J}_{ij}^T \, \boldsymbol{\Omega}_{ij} \, \boldsymbol{J}_{ij} \\
&= \begin{pmatrix} \mathbf{0} \\ \vdots \\ \boldsymbol{A}_{ij}^T \\ \vdots \\ \boldsymbol{B}_{ij}^T \\ \vdots \\ \mathbf{0} \end{pmatrix} \boldsymbol{\Omega}_{ij} \begin{pmatrix} \mathbf{0} & \cdots & \boldsymbol{A}_{ij} & \cdots & \boldsymbol{B}_{ij} & \cdots & \mathbf{0} \end{pmatrix} \\
&= \begin{pmatrix} \ddots & & & & & & \\ & \boldsymbol{A}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{A}_{ij} & \cdots & \boldsymbol{A}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{B}_{ij} & & \\ & \vdots & \ddots & \vdots & & \\ & \boldsymbol{B}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{A}_{ij} & \cdots & \boldsymbol{B}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{B}_{ij} & & \\ & & & & \ddots & \end{pmatrix}
\end{aligned}
$$

which is non-zero only at the four blocks ii, ij, ji and jj (figure 3.3).

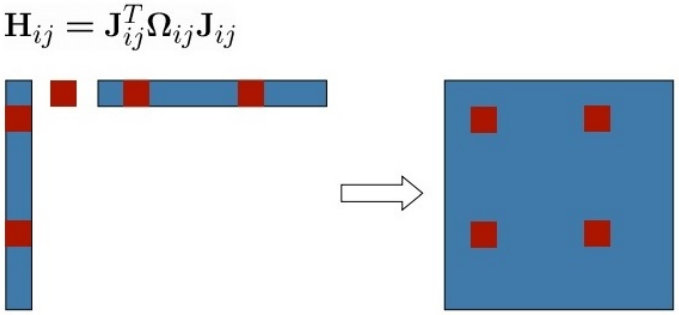

Figure 3.3: Structure of $\boldsymbol{H}_{ij}$

Taking the sum of all terms results in the structure shown in figure 3.4. Since every node is involved in at least one constraint, each block of $\boldsymbol{b}$ will have a contribution and therefore the resulting vector will be dense. On the other hand, each node is usually connected to a small number of other nodes, and therefore the matrix $\boldsymbol{H}$ typically retains a sparse structure. For the matrix to turn dense it would require every node in the graph to be connected to every other; this corresponds to the situation in which the entirety of the features in the environment is observed from each pose, which is quite unlikely to occur.

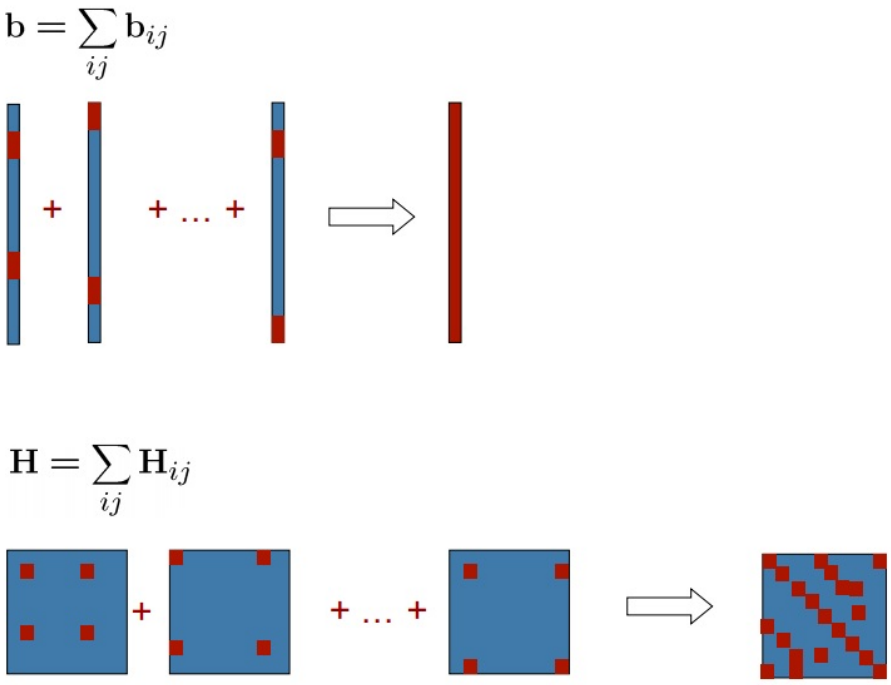

Figure 3.4: Resulting structure of the system

In addition to being sparse, $\boldsymbol{H}$ is positive definite by construction [9]. This allows the system to be solved very efficiently using sparse Cholesky decomposition. As shown in chapter 5, this results in the time required for

optimization being linear in the size of the map; it is this remarkable propriety that gives graph-based algorithms the great computational advantage discussed in chapter 1.

From figure 3.4 it is apparent that each measurement $\boldsymbol{z}_{ij}$ contributes only to the i-th and j-th block of $\boldsymbol{b}$, and to blocks ii, ij, ji, jj of $\boldsymbol{H}$. Therefore the linear system (3.9) can be built from (3.7) and (3.8) in an efficient way. For each observation:

- compute the error $\boldsymbol{e}_{ij} = \boldsymbol{z}_{ij} - \hat{\boldsymbol{z}}_{ij}(\boldsymbol{X}_i, \boldsymbol{X}_j)$
- compute the blocks of the Jacobian

$$\boldsymbol{A}_{ij} = \frac{\partial \boldsymbol{e}\,(\boldsymbol{X}_i, \boldsymbol{X}_j)}{\partial \boldsymbol{X}_i} \qquad\qquad \boldsymbol{B}_{ij} = \frac{\partial \boldsymbol{e}\,(\boldsymbol{X}_i, \boldsymbol{X}_j)}{\partial \boldsymbol{X}_j}$$

- update the coefficient vector:

$$\bar{\boldsymbol{b}}_i^T \mathrel{+}= \boldsymbol{e}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{A}_{ij} \qquad\qquad \bar{\boldsymbol{b}}_j^T \mathrel{+}= \boldsymbol{e}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{B}_{ij}$$

- update the blocks ii, ij, ji and jj of the system matrix:

$$\bar{\boldsymbol{H}}^{ii} \mathrel{+}= \boldsymbol{A}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{A}_{ij} \qquad\qquad \bar{\boldsymbol{H}}^{ij} \mathrel{+}= \boldsymbol{A}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{B}_{ij}$$

$$\bar{\boldsymbol{H}}^{ji} \mathrel{+}= \boldsymbol{B}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{A}_{ij} \qquad\qquad \bar{\boldsymbol{H}}^{jj} \mathrel{+}= \boldsymbol{B}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{B}_{ij}.$$

Once the system has been built, one more step is necessary to be able to solve it. Since the error of a constraint $\boldsymbol{e}_{ij}$ depends only on the relative

position of the connected nodes $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$, it follows that the error $F(\boldsymbol{X})$ of a particular configuration is invariant under a rigid transformation of all the nodes. This results in the system (3.9) being under determined. To solve the problem it is therefore necessary to fix the position of one of the nodes; one way to do this is by constraining the increments to that node to be zero. If k is the index of the node to be fixed, this can be done by simply adding the identity matrix to $\boldsymbol{H}^{kk}$ (the k-th diagonal block of $\boldsymbol{H}$): from (3.9) it can easily be seen that this operation is equivalent to adding a constraint $\boldsymbol{\Delta X}_k = 0$. In this implementation, the first pose is initialized and fixed in the origin for the entire run.

3.3.1 Uncertainty Estimation

Once the optimum configuration of the graph has been computed, estimating the uncertainty of the various nodes is a straightforward operation. Given the linearization point, the system matrix $\boldsymbol{H}$ represents also the information matrix of the current system configuration (that is a Gaussian estimate of the true state) [9]. The diagonal blocks of its inverse represent the covariances of the various nodes of the graph, that can be used to draw the absolute uncertainty ellipses. Computing the uncertainties relative to a given node i is also straightforward: these covariances can be obtained as the diagonal blocks of the inverse of a reduced Hessian $\boldsymbol{H}_{red}$ obtained from $\boldsymbol{H}$ by removing the rows and the columns corresponding to node i. This was the method use to compute the uncertainties shown in chapter 5.

3.3.2 Robustness to Outliers and Bad Initialization

Most SLAM approaches assume that the constraints generated by the front-end are affected by noise but no outliers are present, i.e. there are no con-

straints that identify actually different places as being the same one. This corresponds to the assumption of having a perfect SLAM front-end. As will be seen in chapter 4 however, developing the perfect front-end that produces graphs which are free of outliers is hard to achieve due to perceptual aliasing, and a single data association error is often sufficient for traditional methods to produce inconsistent maps, compromising the entire process. Moreover, even in the absence of outliers, converging to the correct solution is challenging for non-linear error minimization algorithms if the initial guess is far from the correct solution. Therefore, optimization back-ends need to be resilient to outliers, as well as be robust to bad initialization.

Agarwal et al. [14] successfully demonstrated an effective method for optimizing constraint networks in presence of outliers and bad initial guesses. In their so-called dynamic covariance scaling (DCS) approach, the original least squares formulation (3.2)

$$\boldsymbol{X}^* = \underset{\boldsymbol{X}}{\operatorname{argmin}} \sum_{ij} \boldsymbol{e}_{ij}^T(\boldsymbol{X}) \, \boldsymbol{\Omega}_{ij} \, \boldsymbol{e}_{ij}(\boldsymbol{X})$$

is augmented introducing for each constraint a scaling factor s_{ij} :

$$\boldsymbol{X}^* = \underset{\boldsymbol{X}}{\operatorname{argmin}} \sum_{ij} \boldsymbol{e}_{ij}^T(\boldsymbol{X}) \, (s_{ij}^2 \, \boldsymbol{\Omega}_i) \, \boldsymbol{e}_{ij}(\boldsymbol{X}).$$

This weight is used to dynamically adjust the effect of constraints on the optimizer based on the original error term $\chi_{ij}^2 = \boldsymbol{e}_{ij}^T(\boldsymbol{X}) \, \boldsymbol{\Omega}_{ij} \, \boldsymbol{e}_{ij}(\boldsymbol{X})$ that they would introduce in the system. The intuition behind this approach is simple: if the error relative to a certain constraint is large, it means that the current configuration of the graph is far away from what the constraint is telling, so the degree of uncertainty of that constraint is increased by scaling

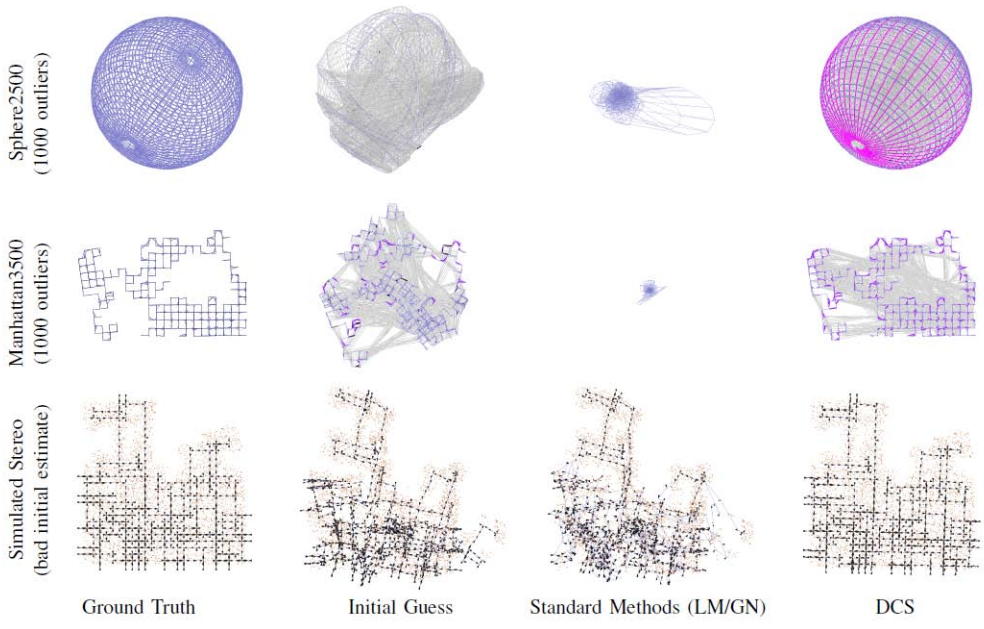

Figure 3.5: Performance of Dynamic Covariance Scaling (DCS) in the presence of outliers (top two rows) and bad initialization (bottom row) for publicly available datasets [14]. DCS is shown to converge to the correct result where standard methods fail to reach the optimum solution.

down its information matrix. Thus, constraints that are far away from the expectation will have smaller influence on the optimization. The scaling variable s_{ij} is computed as

$$s_{ij} = \min\left(1, \frac{2\Phi}{\Phi + \chi^2_{ij}}\right)$$

where Φ is a free parameter. A detailed derivation of this scaling function and an analysis of the impact of Φ can be found in [14]. The scaling function for a constraint remains flat when $\chi^2_{l_{ij}} \leq \Phi$. In this region, DCS behaves like a normal squared kernel without any scaling. As the error increases, DCS scales the information matrix gradually. This has the effect of the error still

being squared but with reduced weight. As $\chi^2_{l_{ij}} \longrightarrow \infty, s_{ij} \longrightarrow 0$.

Figure 3.5 shows the results obtained with DCS in [14]. As can be seen, the algorithm is remarkably robust compared to traditional methods (Levenberg-Marquardt and Gauss-Newton) and converges to the correct solution even in presence of a relevant number of outliers. Discussion of DCS for this implementation can be found in chapter 5.

3.4 Appendix - Derivation of Jacobian Blocks

In this section an explicit expression will be derived for computing the blocks of the Jacobian $\boldsymbol{A}_{ij}$ and $\boldsymbol{B}_{ij}$ in the case at hand. Given the geometry of the problem, the expected observation of a landmark j from pose i is the position of j relative to i, projected in the reference frame of i. With the notation introduced in section 3.3:

$$\hat{\boldsymbol{z}}_{ij}(\boldsymbol{X}_i, \boldsymbol{X}_j) = \begin{pmatrix} \hat{x}_{ij} \\ \hat{y}_{ij} \\ \hat{z}_{ij} \end{pmatrix} = \boldsymbol{R}_i \begin{pmatrix} X_j - X_i \\ Y_j - Y_i \\ Z_j - Z_i \end{pmatrix}$$

where $\boldsymbol{R}_i$ is the rotation matrix from the global reference frame to the robot reference frame. If $(\alpha_i, \beta_i, \gamma_i)$ are the classic yaw, pitch and roll angles of the robot in pose i, then (the subscript i is dropped for simplicity):

$$\boldsymbol{R}_i = \begin{pmatrix} \cos\alpha\cos\beta & \sin\alpha\cos\beta & -\sin\beta \\ \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \cos\beta\sin\gamma \\ \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \cos\beta\cos\gamma \end{pmatrix}$$

Therefore

$$\begin{aligned}
\boldsymbol{e}_{ij}(\boldsymbol{X}_i, \boldsymbol{X}_j) = \; & \boldsymbol{z}_{ij} - \hat{\boldsymbol{z}}_{ij}(\boldsymbol{X}_i, \boldsymbol{X}_j) \\
& = \begin{pmatrix} x_{ij} \\ y_{ij} \\ z_{ij} \end{pmatrix} - \boldsymbol{R}_i \begin{pmatrix} X_j - X_i \\ Y_j - Y_i \\ Z_j - Z_i \end{pmatrix} \\
& = \begin{pmatrix} x_{ij} - R_{11}(X_j - X_i) - R_{12}(Y_j - Y_i) - R_{13}(Z_j - Z_i) \\ y_{ij} - R_{21}(X_j - X_i) - R_{22}(Y_j - Y_i) - R_{23}(Z_j - Z_i) \\ z_{ij} - R_{31}(X_j - X_i) - R_{32}(Y_j - Y_i) - R_{33}(Z_j - Z_i) \end{pmatrix}
\end{aligned} \tag{3.11}$$

Assuming that the robot is equipped with an inertial measurement unit, the pitch and roll angles can be determined directly [15]. Therefore, the variables of the problem reduce to

$$\boldsymbol{X}_i = \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ \alpha_i \end{pmatrix} \qquad \boldsymbol{X}_j = \begin{pmatrix} X_j \\ Y_j \\ Z_j \end{pmatrix}$$

for a pose and a landmark respectively. Thus, the blocks of the Jacobian for an observation of landmark j from pose i are:

$$\boldsymbol{A}_{ij} = \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{X}_i, \boldsymbol{X}_j)}{\partial \boldsymbol{X}_i} = \begin{pmatrix} \frac{\partial e_{ij,1}}{\partial X_i} & \frac{\partial e_{ij,1}}{\partial Y_i} & \frac{\partial e_{ij,1}}{\partial Z_i} & \frac{\partial e_{ij,1}}{\partial \alpha_i} \\ \frac{\partial e_{ij,2}}{\partial X_i} & \frac{\partial e_{ij,2}}{\partial Y_i} & \frac{\partial e_{ij,2}}{\partial Z_i} & \frac{\partial e_{ij,2}}{\partial \alpha_i} \\ \frac{\partial e_{ij,3}}{\partial X_i} & \frac{\partial e_{ij,3}}{\partial Y_i} & \frac{\partial e_{ij,3}}{\partial Z_i} & \frac{\partial e_{ij,3}}{\partial \alpha_i} \end{pmatrix}$$

$$\boldsymbol{B}_{ij} = \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{X}_i, \boldsymbol{X}_j)}{\partial \boldsymbol{X}_j} = \begin{pmatrix} \frac{\partial e_{ij,1}}{\partial X_j} & \frac{\partial e_{ij,1}}{\partial Y_j} & \frac{\partial e_{ij,1}}{\partial Z_j} \\ \frac{\partial e_{ij,2}}{\partial X_j} & \frac{\partial e_{ij,2}}{\partial Y_j} & \frac{\partial e_{ij,2}}{\partial Z_j} \\ \frac{\partial e_{ij,3}}{\partial X_j} & \frac{\partial e_{ij,3}}{\partial Y_j} & \frac{\partial e_{ij,3}}{\partial Z_j} \end{pmatrix}$$

Confronting with equation 3.11, it is apparent that the left 3x3 block of matrix $\boldsymbol{A}_{ij}$ is the matrix $\boldsymbol{R}_i$. For the last column, we can write:

$$\frac{\partial e_{ij,1}}{\partial \alpha_i} = -\frac{\partial R_{11}}{\partial \alpha_i}(X_j - X_i) - \frac{\partial R_{12}}{\partial \alpha_i}(Y_j - Y_i) - \frac{\partial R_{13}}{\partial \alpha_i}(Z_j - Z_i)$$

$$\frac{\partial e_{ij,2}}{\partial \alpha_i} = -\frac{\partial R_{21}}{\partial \alpha_i}(X_j - X_i) - \frac{\partial R_{22}}{\partial \alpha_i}(Y_j - Y_i) - \frac{\partial R_{23}}{\partial \alpha_i}(Z_j - Z_i)$$

$$\frac{\partial e_{ij,3}}{\partial \alpha_i} = -\frac{\partial R_{31}}{\partial \alpha_i}(X_j - X_i) - \frac{\partial R_{32}}{\partial \alpha_i}(Y_j - Y_i) - \frac{\partial R_{33}}{\partial \alpha_i}(Z_j - Z_i)$$

with

$$\frac{\partial R_{11}}{\partial \alpha_i} = -\sin\alpha_i \cos\beta_i = -R_{12}$$

$$\frac{\partial R_{12}}{\partial \alpha_i} = \cos\alpha_i \cos\beta_i = R_{11}$$

$$\frac{\partial R_{13}}{\partial \alpha_i} = 0$$

and similarly for the other rows, yielding

$$\boldsymbol{A}_{ij} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & R_{12}(X_j - X_i) - R_{11}(Y_j - Y_i) \\ R_{21} & R_{22} & R_{23} & R_{22}(X_j - X_i) - R_{21}(Y_j - Y_i) \\ R_{31} & R_{32} & R_{33} & R_{32}(X_j - X_i) - R_{31}(Y_j - Y_i) \end{pmatrix}$$

while for the matrix $\boldsymbol{B}_{ij}$ it is apparent from 3.11 that

$$\boldsymbol{B}_{ij} = -\boldsymbol{R}_i \,.$$

Chapter 4

Front End

4.1 Introduction

The front end of the SLAM algorithm is the part that builds the graph based on sensor measurements. Clearly, the procedure needed to accomplish this task strongly depends on the type of sensor the robot is equipped with. In the case of this implementation, the front end must convert stereo images in spatial measurements, and use them to add landmarks and constraints to the graph. This implies two fundamental steps:

1. Images must be processed to extract quantitative information about the environment. This includes identifying correspondences in the two views and triangulating points to obtain spatial measurements, as will be discussed in section 4.2.

2. To build the graph, each measurement must then be associated to the correct landmark in the map (or used to create a new one). This so-called data association problem is one of the most challenging parts when designing SLAM algorithm and will be discussed in section 4.3.

4.2 Image processing

Sensor information in a given time step comes in the form of a pair of stereo views. To extract spatial information, these images are processed using the C++ library OpenCV. The procedure involves selecting interest points in an image, finding their counterparts in the other view, and then triangulating points to obtain quantitative measurements.

Interest points in the images are selected using the SURF algorithm implemented in OpenCV. SURF was chosen among other feature detectors for its good compromise between speed and repeatability [16]. For each detected keypoint, a SURF descriptor is computed. Descriptors are 64 or 128-entries vectors calculated from image parameters in the neighbourhood of the considered pixel, and are invariant to small scaling and rotation transformations (i.e. to small camera movements). In the case of stereo images, the change in perspective between the two views is limited, and therefore the descriptors of corresponding points will be similar. This enables the matching of features in the two images comparing their descriptors; the matching procedure assigns the correspondence to points that are closest to each other in the descriptors space, with the distance computed as a Euclidean norm. This procedure relies solely on image parameters, and does not consider the geometry of the problem; a typical result is shown in figure 4.3. It can be seen that although some features are matched correctly, a relevant number of outliers is present. Triangulating erroneous matches would result in meaningless environment measurements; the presence of such outliers can cause a substantial degradation in the final solution of the SLAM algorithm, and therefore these false positives must be carefully removed. To do so, the geometry of the problem must be taken into account; the next sections will provide more insight into the subject.

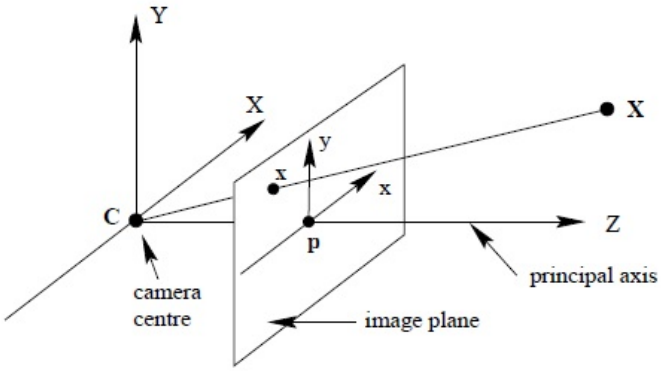

Figure 4.1: The pinhole camera model [17]. In real cameras, the image plane is actually behind the centre of projection, and produces an image that is rotated 180 degrees; here however the projection problem is simplified by placing a virtual image plane in front of the centre of projection to produce an unrotated image.

4.2.1 Epipolar geometry

The 2D image captured by a camera can be thought of as the projection of the 3D world on an image plane through a centre of projection $\boldsymbol{C}$ (pinhole camera model, figure 4.1). Within the framework of this simple model, consider now a system of two cameras portraying the same scene from different points of view (figure 4.2). Denote with $\boldsymbol{x}$ and $\boldsymbol{x}'$ the projections on the two image planes of the same 3D point $\boldsymbol{X}$. Supposing that we only know the projection in the left image $\boldsymbol{x}$, we may ask how its counterpart in the right view $\boldsymbol{x}'$ is constrained. From the information given by the left image alone, the 3D point $\boldsymbol{X}$ resulting in the projection $\boldsymbol{x}$ could lie anywhere on the ray emanating from the camera centre $\boldsymbol{C}$ through $\boldsymbol{x}$. This ray in space is imaged in the right view as a line $\boldsymbol{l}'$; hence, the projection $\boldsymbol{x}'$ cannot lie anywhere on the right image plane, but must belong to this line. This constraint can be used to narrow the search for matching pairs in stereo images and to reject outliers.

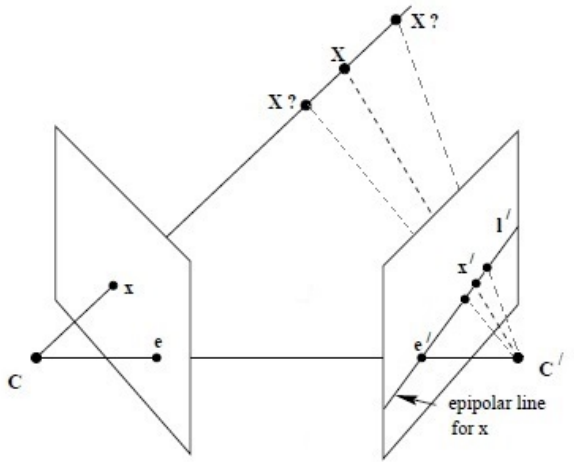

Figure 4.2: Point correspondence geometry [17]. The two cameras are indicated by their centres $\boldsymbol{C}$ and $\boldsymbol{C}'$ and image planes. An image point $\boldsymbol{x}$ projects to a ray in 3D space defined by the first camera centre, $\boldsymbol{C}$, and $\boldsymbol{x}$. This ray is imaged as a line $\boldsymbol{l}'$ in the second view. The 3D point $\boldsymbol{X}$ which projects to $\boldsymbol{x}$ must lie on this ray, so the image of $\boldsymbol{X}$ in the second view must lie on $\boldsymbol{l}'$.

Referring to figure 4.2, let $\boldsymbol{e}'$ denote the projection on the right image plane of the left camera centre $\boldsymbol{C}$. Then it can be seen that the line $\boldsymbol{l}'$ necessarily intersects this point, regardless of the position of $\boldsymbol{x}$. The point $\boldsymbol{e}'$ is called an epipole, and therefore the line $\boldsymbol{l}'$ is called an *epipolar line*. To each point $\boldsymbol{x}$ in the left image, there exists a corresponding epipolar line $\boldsymbol{l}'$ in the right image (and vice-versa). Therefore a map

$$\boldsymbol{x} \mapsto \boldsymbol{l}'$$

exists from a point in one view to its corresponding epipolar line in the other view [17]. If image points are expressed in homogeneous coordinates [18], this correspondence can be written in the form

$$\boldsymbol{l}' = \boldsymbol{F}\,\boldsymbol{x} \tag{4.1}$$

in which $\boldsymbol{F}$ is a 3x3 matrix representing a map from the 2-dimensional projective plane of the first image to the pencil of epipolar lines through the epipole of the second image $\boldsymbol{e}'$. $\boldsymbol{F}$ is called the *fundamental matrix* of the system and clearly depends on the relative geometry between the two cameras.

A key implication follows from the above relationship. If points $\boldsymbol{x}$ and $\boldsymbol{x}'$ are the projections of the same 3D point $\boldsymbol{X}$, then $\boldsymbol{x}'$ lies on the epipolar line $\boldsymbol{l}' = \boldsymbol{F}\boldsymbol{x}$ corresponding to $\boldsymbol{x}$. In homogeneous coordinates this is written as $\boldsymbol{x}'^T\boldsymbol{l}' = 0$. Confronting with (4.1) it immediately follows that if points $\boldsymbol{x}$ and $\boldsymbol{x}'$ correspond, then

$$\boldsymbol{x}'^T\boldsymbol{F}\,\boldsymbol{x} = 0. \tag{4.2}$$

This so-called *epipolar constraint* is of utmost importance, in that it gives a way of characterizing the fundamental matrix without any reference to the geometric configuration of the two cameras, i.e. only in terms of corresponding image points. It can be shown that $\boldsymbol{F}$ has seven degrees of freedom, and therefore at least seven point correspondences are needed to estimate it [17]. The ability to compute the fundamental matrix from image correspondences alone provides an effective tool to remove outliers in the matching process, as will be discussed in the next section.

4.2.2 Match Validation

Figure 4.3 showed an example of keypoints matched based on their SURF descriptors. As it was seen, descriptor matching relies on image parameters only and produces many outliers, making the result unusable for detecting

consistent landmarks. Fortunately, the epipolar constraint provides a useful tool to validate matches taking into account the geometry of the problem. This can be done (for undistorted images) by estimating the fundamental matrix F from (4.2) with a Random Sample Consensus (RANSAC) scheme. The procedure can be summarized as follows.

1. From the set of matched point pairs, many random subsets of seven pairs each are drawn.

2. For each subset, the corresponding fundamental matrix is estimated from eq. 4.1 with a least-squares algorithm.

3. The goodness of the computed fundamental matrix for each case is measured by counting how many of the other point pairs satisfy equation 4.1 (within a certain threshold).

4. The fundamental matrix with the highest number of inliers is selected as the best solution; matches that do not satisfy the epipolar constraint for this $\boldsymbol{F}$ are considered outliers.

This method effectively identifies point pairs that do not fit the most likely geometry of the problem. It can handle practically any ratio of outliers, although care must be taken to adjust the threshold according to image resolution and noise. The result of applying the procedure to the stereo pair of figure 4.3 is shown in figure 4.4: as it can be seen, erroneous matches that do not satisfy the epipolar constraint have been removed. Although this geometric validation is usually very effective, however, it is not guaranteed to always eliminate *all* outliers. The epipolar constraint states that if $\boldsymbol{x}$ and $\boldsymbol{x}'$ correspond, then ${\boldsymbol{x}'}^T \boldsymbol{F} \boldsymbol{x} = 0$, but the vice-versa is not necessarily true: if two image points satisfy the constraint ${\boldsymbol{x}'}^T \boldsymbol{F} \boldsymbol{x} = 0$, this does not imply that

$\boldsymbol{x}$ and $\boldsymbol{x}'$ correspond. Erroneous correspondences that happen to satisfy the epipolar constraint thus will not be discarded. An example of this case is shown in figure 4.5 (in which both the left and right keypoint sets are drawn in the same image). This situation suggests a somehow naive but effective approach to identify surviving outliers based on anomalous image coordinates of matching points. The mean distance between the image coordinates of points in the left and right view is then calculated along with the standard deviation, and the pairs for which the distance exceeds the 3σ boundary are removed; the process is repeated iteratively until no further change is produced.

Although a 100% effectiveness in providing only correct matches is not realistic, the above procedure has been seen to produce quite reliable results. Given the calibration parameters of the stereo camera, the surviving matches can then be triangulated to estimate their position in space.

4.2.3 Triangulation

For this step, the algorithm relies on a third-party mid-point triangulation method, that however does not provide a measure of the uncertainty. To estimate it, a Gaussian approximation is used [19, 20]. Each triangulated observation is obtained from a stereo measurement of the type

$$\boldsymbol{x} = \begin{bmatrix} x_L \\ y_L \\ x_R \\ y_R \end{bmatrix} + \mathcal{N}(0, N_t)$$

where (x_L, y_L) and (x_R, y_R) are the image coordinates in pixels of the observed feature in the left and in the right view, while $\mathcal{N}(0, N_t)$ represents a random sample drawn from the normal distribution with mean 0 and co-

variance N_t used to model sensor noise. Assuming that for each camera the image coordinate errors are decorrelated, N_t can be written as

$$N_t = \text{diag}\left(\sigma_{x_L}^2,\ \sigma_{y_L}^2,\ \sigma_{x_R}^2,\ \sigma_{y_R}^2\right)$$

with σ_{x_L}, σ_{y_L}, σ_{x_R} and σ_{y_R} the standard deviations in pixels of the match measurement. This observation results in a triangulated measurement

$$\boldsymbol{X} = \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \mathcal{N}(0, \Sigma_t)$$

with (X_w, Y_w, Z_w) the world coordinates of the landmark and Σ_t the observation covariance matrix. A transformation matrix from the known N_t (that depends on camera parameters) to the unknown Σ_t can be estimated with a first order Taylor approximation as

$$W_t = \begin{bmatrix} \frac{\partial X_w}{\partial x_L} & \frac{\partial X_w}{\partial y_L} & \frac{\partial X_w}{\partial x_R} & \frac{\partial X_w}{\partial y_R} \\ \frac{\partial Y_w}{\partial x_L} & \frac{\partial Y_w}{\partial y_L} & \frac{\partial Y_w}{\partial x_R} & \frac{\partial Y_w}{\partial y_R} \\ \frac{\partial Z_w}{\partial x_L} & \frac{\partial Z_w}{\partial y_L} & \frac{\partial Z_w}{\partial x_R} & \frac{\partial Z_w}{\partial y_R} \end{bmatrix}$$

Then, the observation covariance matrix can be expressed as

$$\Sigma_t = W_t\, N_t\, W_t^T.$$

This matrix can be used to draw the confidence ellipses for the triangulated points. Figure 4.7 shows an example of confidence ellipses obtained with this method; as expected, the depth measure is the most uncertain, with the error increasing with the distance from the camera (that is located in the origin).

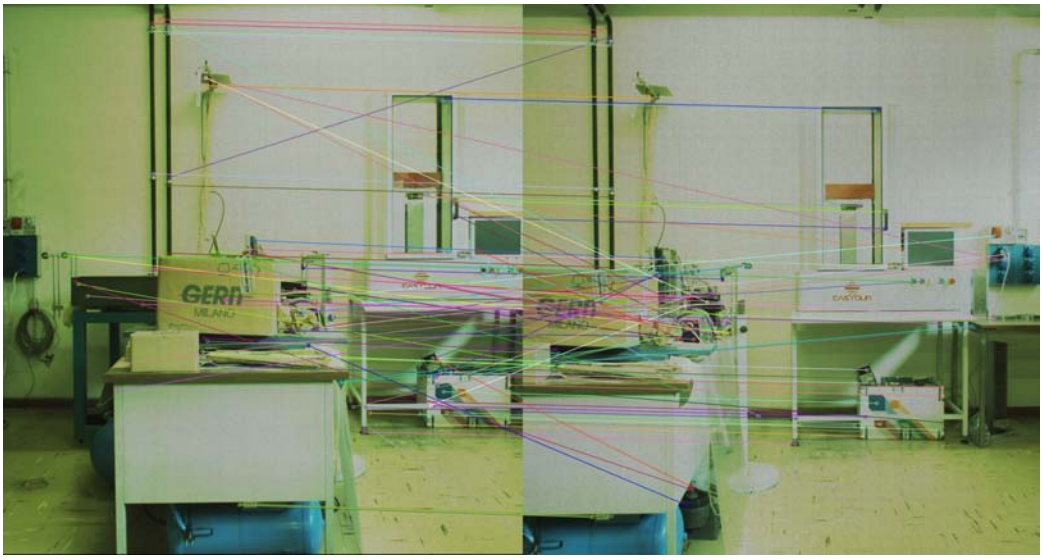

Figure 4.3: Results of SURF descriptor matching in stereo images (detail).

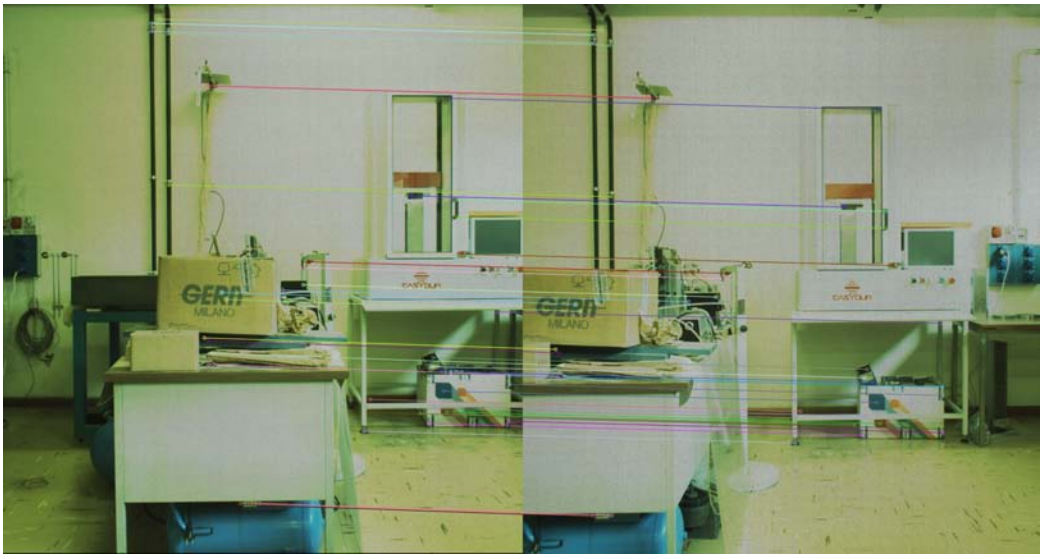

Figure 4.4: Surviving matches that satisfy the epipolar constraint after application of the RANSAC scheme (detail).

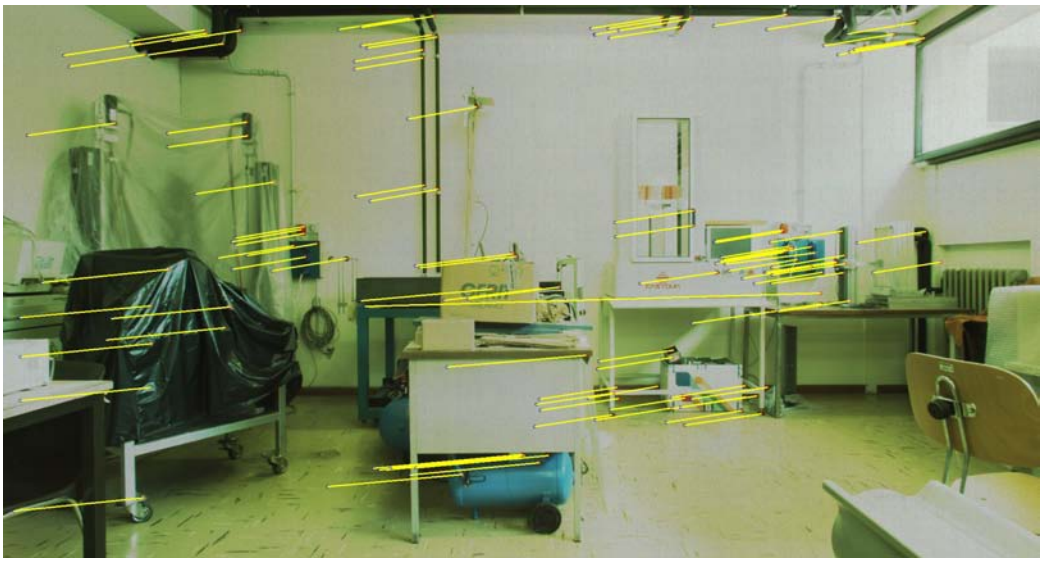

Figure 4.5: Stereo matches after the epipolar validation. Both left and right keypoints are shown in the left view, respectively with red and blue circles. In the centre of the image, an apparent outlier survived the geometric validation.

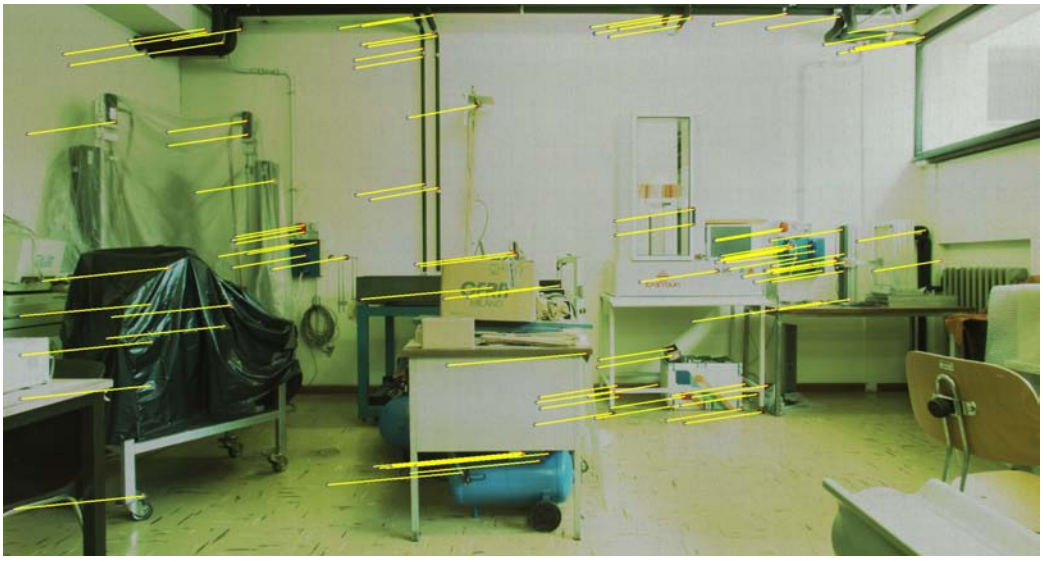

Figure 4.6: The outlier has been removed by the 3σ validation scheme.

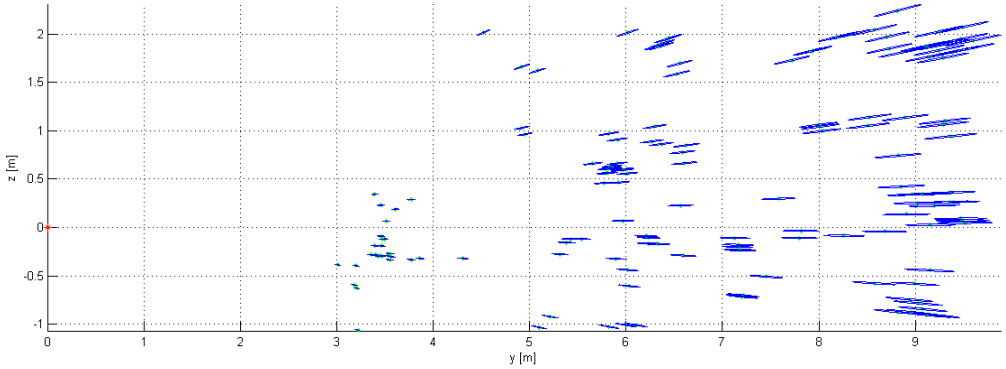

Figure 4.7: Confidence ellipses for triangulated points; the stereo camera sits in the origin.

4.3 Data Association

Triangulating stereo matches yields a set of spatial observations that are however useless if we do not know which landmark originated a given observation. Since the robot poses and the locations of the various landmarks are not known precisely, it is not straightforward to associate a certain measurement to the correct landmark. In vision-based SLAM, the task can be aided by visual information provided by the cameras.

In most existing approaches, visual data association to a landmark in the map is based on the squared euclidean distance between descriptors

$$E = (\boldsymbol{d}_i - \boldsymbol{d}_j)\,(\boldsymbol{d}_i - \boldsymbol{d}_j)^T,$$

where $\boldsymbol{d}_i$ and $\boldsymbol{d}_j$ denote the mean descriptors of an observed feature (obtained averaging the left and right view descriptors) and of a stored landmark. The landmark in the map that minimizes the distance is regarded as the

correct data association if E is below a certain threshold; otherwise, a new landmark is created. As explained in section 4.2, when the same point is seen from slightly different viewpoints, the values in its descriptor remain quite similar. However, if the point of view changes significantly, the difference in the descriptor will be remarkable and the check using the Euclidian distance is likely to produce a wrong data association.

Gil et al. [21] proposed an improvement of this method based on the Mahalanobis distance. For each landmark in the map, a mean descriptor $\bar{\boldsymbol{d}}_l$ is stored, while a descriptor covariance matrix $\boldsymbol{S}_l$ keeps track of the landmark appearance in various views. The covariance is calculated assuming the elements in the descriptor are independent of each other. When searching for a correspondence for a feature with descriptor $\boldsymbol{d}_f$, the Mahalanobis distance

$$M = (\bar{\boldsymbol{d}}_l - \boldsymbol{d}_f)\, \boldsymbol{S}_l^{-1}\, (\bar{\boldsymbol{d}}_l - \boldsymbol{d}_f)^T$$

is computed for all stored landmarks. The correspondence is assigned to the landmark that minimizes this distance if M is below a predefined threshold; otherwise, $\boldsymbol{d}_f$ is considered a new landmark. This method was shown in [21] to produce better results compared to euclidean distance data association, but still relies on image parameters only.

The procedure developed for this implementation stems from this approach, but also takes into account the geometric configuration of the system. The absence of odometry information however complicates the process significantly. As a new set of observations comes, in fact, the pose from which these are obtained is not even known approximately, making a geometric association impossible. Therefore, tracking of features across multiple frames is used to obtain information on their identity for immediate association (figure 4.8), while for features that are seen for the first time, association must be delayed to the subsequent time step, when more information on the actual

Figure 4.8: Tracking of features across consecutive frames.

pose will be available. The resulting procedure has been seen to produce good results. In the following section, description of the data association process is given in the context of the entire SLAM algorithm. For the rest of the chapter, the term *feature* will denote a 3D point observed in a given time step, while *landmark* will refer to a 3D point stored in the map.

4.3.1 SLAM Algorithm Overview

1. At the initial time step t_0, pose $\boldsymbol{p}_0$ (the origin) is added to the (still empty) graph. The stereo pair S_0 is triangulated, but the features observed from this pose are not added to the graph for the moment. Features are added to the map as landmarks only when they have been seen at least from two consecutive poses.

2. At time t_1, a new pose is added to the graph. Since no odometry information is available, the previous pose is always used to initialize the new one (i.e. $\boldsymbol{p}_n = \boldsymbol{p}_{n-1}$). Unless abnormal displacement has occurred between the two, this provides a reasonable initialization.

3. The stereo pair S_1 is triangulated. We now have two sets of observed features.

4. L_1 (the left view of S_1) is matched with L_0 (the left view of S_0). The resulting m correspondences are features that have been seen in two consecutive views. Therefore, they are added to the graph as landmarks. Their position is initialized with their coordinates relative to $\boldsymbol{p}_0$ (the origin) which are known from the observations in t_0. The observations $\{\boldsymbol{z}_{0j}, \boldsymbol{z}_{1j}\}_{j=0\ldots m}$ of these landmarks from $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$ are constraints between poses and landmarks that are also added to the graph. For each landmark that was added to the map, the mean descriptor and covariance are stored.

5. The back end optimizes the graph. This among other things moves $\boldsymbol{p}_1$ from the origin to its optimum location.

6. Time step t_2. Pose $\boldsymbol{p}_2$ is created and set equal to $\boldsymbol{p}_1$. S_2 is triangulated. L_2 is matched with L_1. Each match can now be of two types.

   (a) A feature in L_2 can be matched to one in L_1 that was already added to the map in t_1 (i.e. it is a landmark). In this case, the only thing that is added to the graph is a constraint between that landmark and $\boldsymbol{p}_2$. The landmark descriptor and covariance are updated with the new information from L_2.

   (b) A feature in L_2 can also be matched to one in L_1 that is not a landmark. In this case, a new landmark is added to the graph. It is initialized with the global coordinates given by its observation from $\boldsymbol{p}_1$ (which is now known) and constrained to $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ with the relative observations. In this case therefore one landmark

and two constraints are added to the graph. The mean landmark descriptor and covariance are also stored.

Once the procedure has been repeated for all matches between L_2 and L_1, the graph has acquired a number of new landmarks and constraints.

7. The graph is optimized.

8. Timestep t_3 (and t_n in general). $\boldsymbol{p}_n$ is initialized as $\boldsymbol{p}_{n-1}$. S_n is triangulated; L_n is matched with L_{n-1}. From now on, three cases can be individuated.

   (a) A feature in L_n can be matched to one in L_{n-1} for which correspondence to a landmark was already established in the previous time step. In this case, the correspondence is propagated to this feature, and the only thing that is added to the graph is a constraint between that landmark and $\boldsymbol{p}_n$. The landmark descriptor and covariance are updated.

   (b) A feature in L_n can be matched to one in L_{n-1} that was not identified as a landmark in the previous time step. In this case, before adding a new landmark, the algorithm tries to establish a correspondence between this pair of features and an existing landmark. Two strategies have been implemented to do this. Strategy A finds the n landmarks in the map that are nearest to the location where the new one would be initialized (given by its coordinates relative to $\boldsymbol{p}_{n-1}$). Then, among the n nearest landmarks, correspondence is assigned to the one that minimizes the Mahalanobis distance

$$M = (\bar{\boldsymbol{d}}_l - \bar{\boldsymbol{d}}_f)\, \boldsymbol{S}_l^{-1}\, (\bar{\boldsymbol{d}}_l - \bar{\boldsymbol{d}}_f)^T,$$

if M is below a certain threshold. Here $\bar{\boldsymbol{d}}_l$ is the mean descriptor of the landmark, $\bar{\boldsymbol{d}}_f$ the mean descriptor of the pair of features, and $\boldsymbol{S}_l$ the covariance matrix of the landmark descriptor. Strategy B on the other hand simply assigns the correspondence to the nearest landmark in the map if M does not exceed the threshold; the performances of the two strategies will be compared in section 5.1. If the correspondence is confirmed, two constraints to the landmark are added to the graph (one from $\boldsymbol{p}_{n-1}$ and one from $\boldsymbol{p}_n$). The mean landmark descriptor and covariance are updated.

(c) If the minimum Mahalanobis distance is above the defined threshold, the two features are considered to be a new landmark, that is therefore added to the graph in the same way as before. It is initialized using its position relative to $\boldsymbol{p}_{n-1}$ and constrained to $\boldsymbol{p}_{n-1}$ and $\boldsymbol{p}_n$.

Steps 7 and 8 are then repeated until the end of the run. They form the backbone of the algorithm, while steps 1-6 provide an initialization made necessary by the delayed data association scheme. The algorithm was tested with a stereo data set acquired in the laboratory of the department of mechanical engineering at the University of Padova; the results will be presented in the next chapter.

Chapter 5

Results and discussion

5.1 Experimental Setup

To test the algorithm, an indoor stereo dataset was acquired in the laboratory of the department of mechanical engineering at the University of Padova. The experimental setup is the same used in [22], with two Basler ace acA2000-340kc cameras fixed on a 54 cm arm and mounted both on a rotary stage and on a linear slide (figure 5.1). The rotary stage is driven by a high precision stepper motor while the linear slide is provided with a graduated scale, in order to compare the measurements obtained by the visual system with known rotations and displacements. The stereo camera has a height from the ground of 1.1 m and a downward tilt angle of 0°. A Kowa LM6HC 6mm wide-angle lens is mounted on the cameras; this nominal focal length combined with the camera sensor size results in a field of view of 86°x 53°. This lens was seen in [22] to be the most suitable to operate a visual odometry algorithm in the same indoor environment with the same stereo bench (the other lenses tested being f = 10 mm and f = 50 mm). Figure 5.9 shows three views of the laboratory obtained with this set-up; image resolution is 2040 x

Figure 5.1: The set-up used to acquire stereo images.

1086 pixels.

Before analyzing a simulated robot trajectory, the algorithm was tested with elementary translation and rotation datasets. In the translation tests, the stereo camera was displaced along the 1350 mm linear slide with a step of 10 mm; after each step the motion was stopped and two images were acquired, so that a stereo pair is available every 10 mm. The same procedure was repeated using the rotary stage to obtain -90°/+90° datasets; in this case the stereo images were acquired with a 1° step.

According to the results summarized in figures 5.2 to 5.5 and to extensive testing, data association strategy A (see section 4.3.1) appears to yield the most precise results when a low value is used for the Mahalanobis distance threshold, both in the case of translation and rotation (figures 5.2, 5.4). However, it also appears to be far less robust than strategy B to the outlier associations that result from an increase of said threshold. In fact, raising the Mahalanobis threshold means increasing the number of uncertain land-

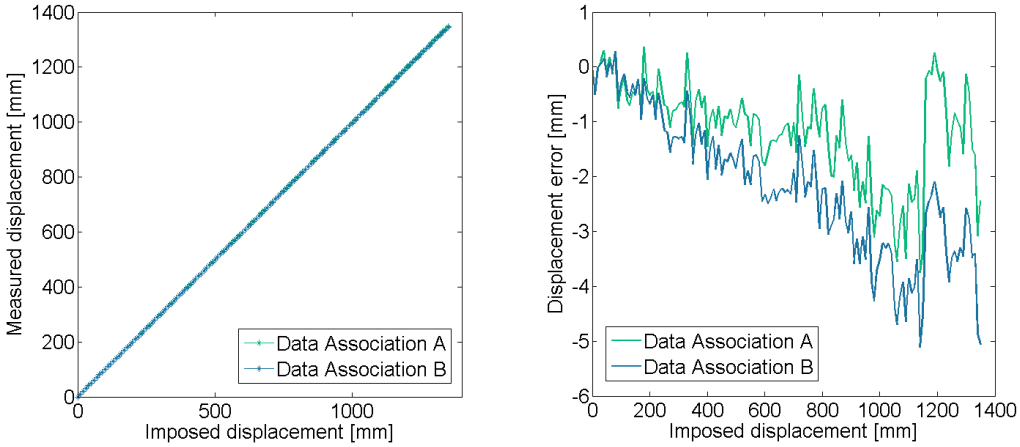

Figure 5.2: Data association comparison for a low Mahalanobis threshold translation test (M = 500). a: Measured vs imposed displacement. b: Ground-truth error. Other relevant parameters are: SURF detector threshold = 700, step = 10 mm.

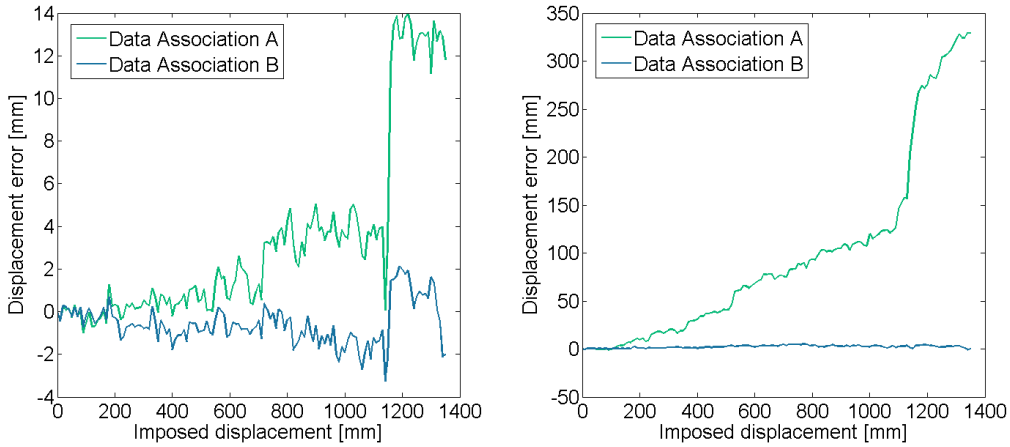

Figure 5.3: Ground truth errors for high Mahalanobis threshold translation tests. a: M = 3,000; b: M = 10,000. Other relevant parameters are: SURF detector threshold = 700, step = 10 mm.

mark associations that are confirmed. This can be desirable to decrease the size of the map (figure 5.6) and therefore the computational load (if the level

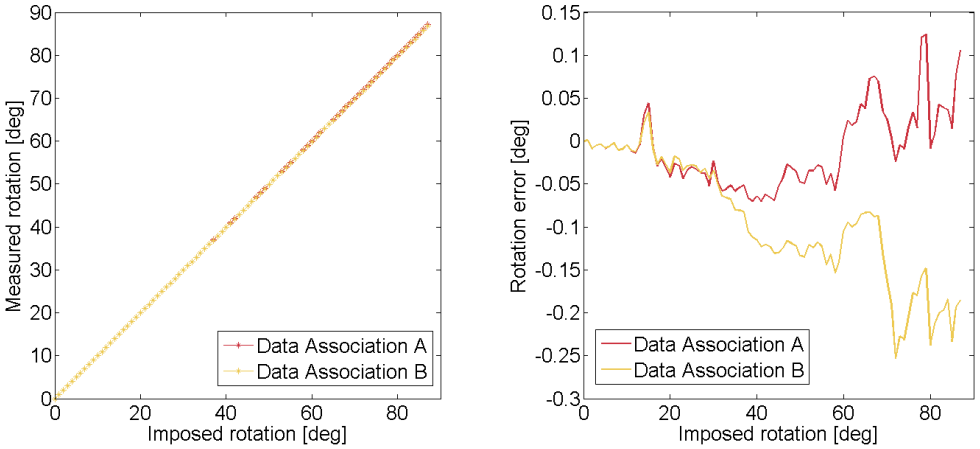

Figure 5.4: Data association comparison for a low Mahalanobis threshold rotation test (M = 500). a: Measured vs imposed rotation. b: Ground-truth error. Other relevant parameters are: SURF threshold = 700, step = 1°.

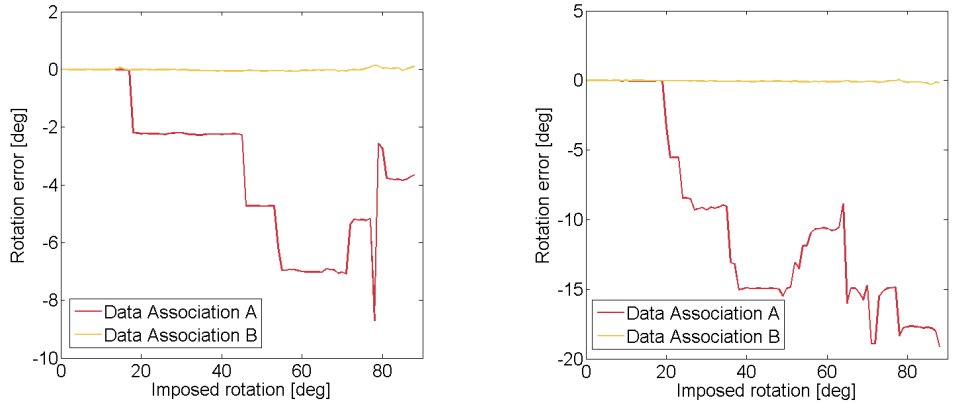

Figure 5.5: Ground truth errors for high Mahalanobis threshold rotation tests. a: M = 3000; b: M = 10000. Other relevant parameters are: SURF detector threshold = 700, step = 1°.

of confidence required to confirm a landmark correspondence is decreased, a part of the observed features that would be added to the map will be associated to already existing landmarks instead). However, this will necessarily also increase the ratio of erroneous constraints introduced in the graph. Fig-

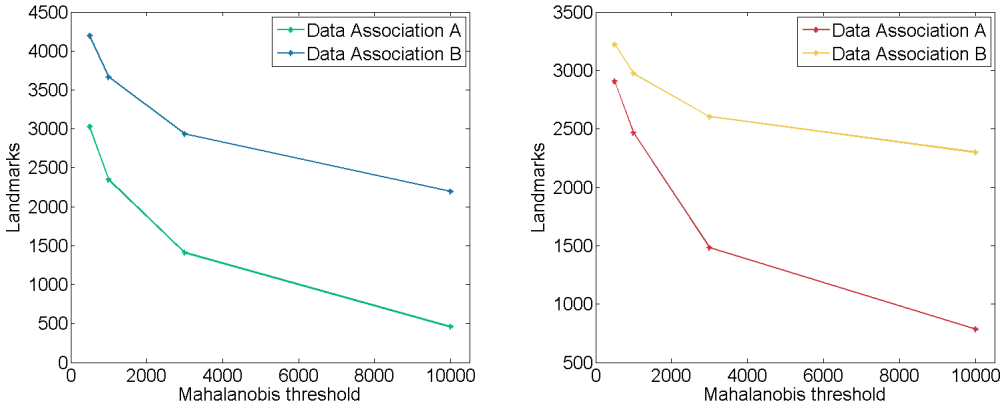

Figure 5.6: Final number of landmarks in the map vs Mahalanobis threshold for (a) 1350 mm translation, 10 mm step and (b) 90° rotation, 1° step. SURF detector threshold was set to 700 in both cases.

ures 5.3 and 5.5 compare the performances of the two strategies when the threshold is increased; as it can be seen, strategy B appears to be far more robust to outliers, retaining good precision even when strategy A fails to converge to the right solution (figures 5.3b, 5.5). This is due to the fact that strategy B takes as a candidate for association to a feature only the landmark that is nearest to it in the map; therefore, wrong associations identify as referring to the same landmark observations that are not far from each other anyway. Thus, the error introduced by a wrong strategy B association is limited; strategy A on the other hand can result in greater discrepancies.

From figure 5.6 it can be seen that for a given Mahalanobis threshold, strategy A results in a smaller number of identified landmarks compared to strategy B; this is due to the fact that strategy A takes n candidates for association (in this case $n = 20$), increasing the probability to find a match in the map for a feature that would be otherwise be added to it. This partially reduces the computational overhead resulting from the much lower Mahalanobis threshold needed by scheme A to retain precision.

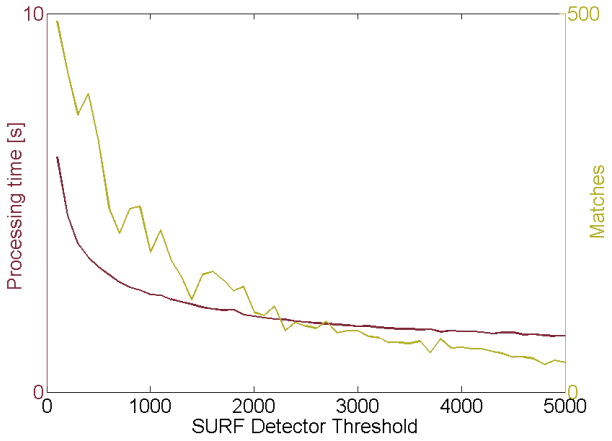

Figure 5.7: Number of stereo matches found and relative processing times on an Intel i5 480M CPU vs. SURF detector threshold.

Finally, in a fashion similar to the Mahalanobis threshold increase:

- Increasing the SURF threshold decreases computational times, at the cost of a decreased quality of the solution. A higher SURF threshold in fact diminishes the number of features that are extracted from the images, decreasing image processing times (figure 5.7) and also the size of the map, that will be less detailed but also more efficient to optimize. However, since less landmarks will be available to estimate the trajectory, this also decreases the quality of the solution. As in the case of the Mahalanobis threshold increase, data association scheme A is the most sensible to this dilution of precision, while scheme B can cope with greater SURF threshold increases without extreme losses. Reasonable values of the SURF threshold are in the order of 500 - 3000, depending on the expected size of the map and desired accuracy.

- Increasing the translation/rotation steps reduces not only the number of poses but also that of landmarks (fewer poses mean fewer observations and therefore fewer landmarks added to the map). This results

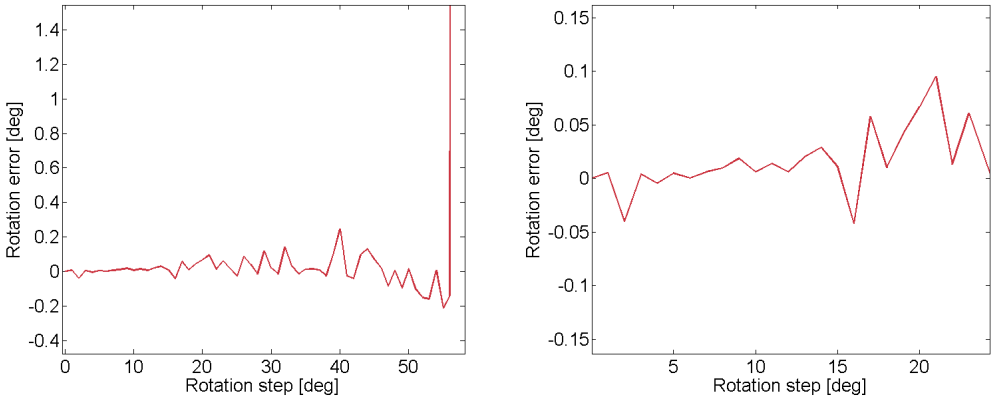

Figure 5.8: a: Error in the estimated rotation vs. rotation imposed in a single step. b: Detail of figure (a). Other relevant parameters are: SURF threshold = 500, data association scheme = A, Mahalanobis threshold = 500.

in a smaller graph (faster optimization) but also in a decrease of precision, that once again is greater for scheme A than for scheme B. For the datasets examined, translations were reconstructed correctly even with the maximum step (1350 mm). The case of rotation however is different, because while in the case of pure translation the features in the center of the image remain visible even with large steps, moderate rotations can change the scene seen from the camera completely, depending on the width of the field of view. This can make the tracking of features across subsequent frames impossible, and if no known landmarks are observed from the new pose either (e.g. because the robot is exploring a new area), the algorithm will fail to estimate the pose. Figure 5.8 shows the error in the estimated rotation versus the imposed rotation step for a favorable case. As it can be seen, precision is remarkable in the range $0° \div 10°$, where the fields of view of the two subsequent steps overlap greatly and many common features can be individuated. As the rotation step increases however, the number

of tracked features decreases, reducing precision until angle estimation ultimately fails. Although in this case the algorithm succeeded in estimating rotation steps up to 55°, this was achieved in optimal conditions (low SURF and Mahalanobis threshold, high distance from the objects in the environment (⇒ greater number of detected features, see below)). The maximum estimable step decreased to 10°/15° when using higher thresholds with objects near to the camera (2-3 m), suggesting that in real applications where image acquisition is regulated by time rather than displacement, the need to cover rotations with a higher number of frames could be the bottleneck of the SLAM system.

5.2 Results

After testing the algorithm with elementary translations and rotations, various datasets taken at different locations in the laboratory were combined to simulate robot motion through the environment and obtain a full map of the room. The ideal robot starts its trajectory from the location shown in figure 5.9a, then moves forward 130 cm, looks to its left (+90°) and to its right(-90°), then continues forward towards the wall that can be seen in figure 5.9a and 5.9b, and finally turns backwards to its right (figure 5.9c). A first simulation aimed at obtaining a precise map of the environment was run on this simulated path with data association scheme A, a SURF threshold of 2500, a Mahalanobis threshold of 500, a rotation step of 3° and a translation step of 11 cm. This resulted in the map shown in figure 5.10; the final graph is comprised of 284 robot poses and 4140 landmarks.

The calculated trajectory and map agree with the simulated robot path and the actual environment. Observing the map, shapes and objects of the

Figure 5.9: Sample views of the laboratory (left camera view).

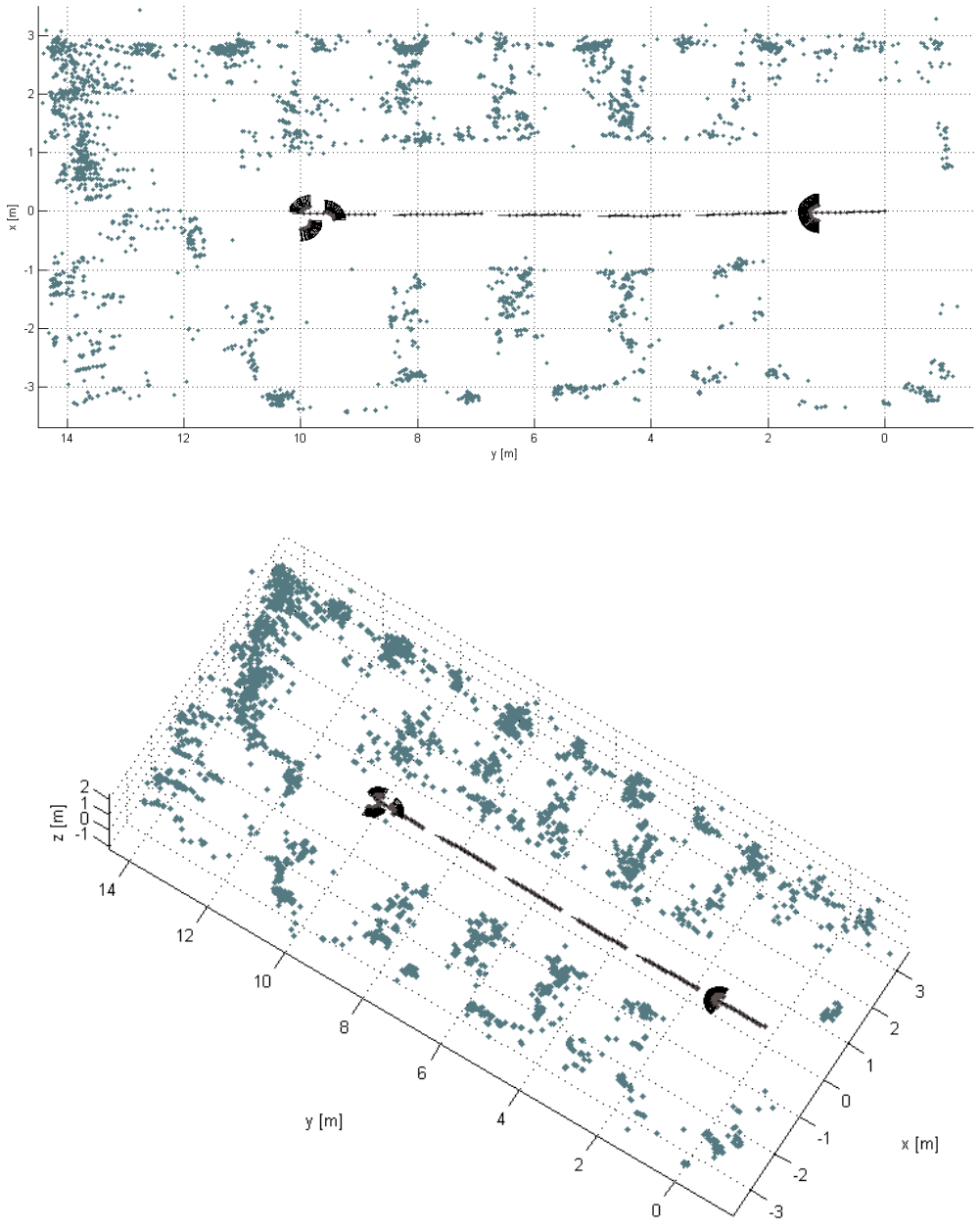

Figure 5.10: 2D and 3D views of the generated map. Grey dots represent landmarks, while brown dots (poses) represent the midpoint of the stereo camera baseline, with an arrow indicating the direction of the field of view.

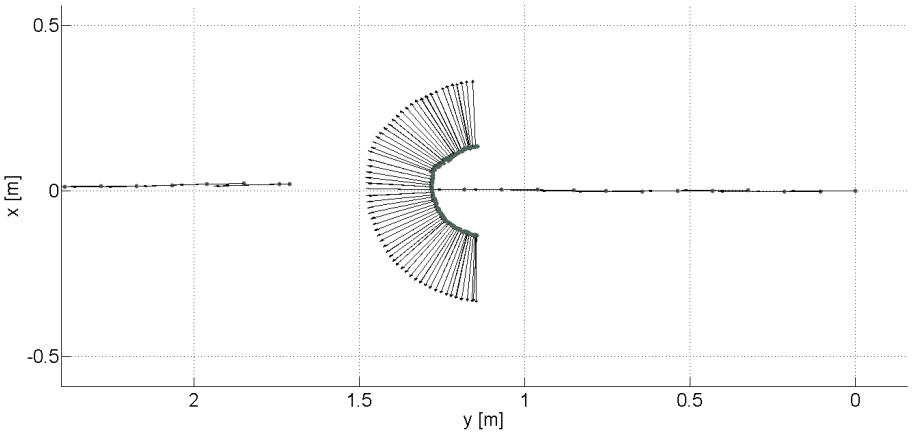

Figure 5.11: Detail of the robot's trajectory.

laboratory can be recognized. The upper side of view 5.9a is the side where the windows are located; five desks can be individuated on that part. On the opposite side, four desks are comprised between the two doors, and another lies after the second door. On the right-hand side of the map, where the robot's path begins, the empty area near the blackboard that is visible in figure 5.9c can be recognized, while on the opposite wall, towards the end of the robot's trajectory, the machinery of figure 5.9b can be identified. The measures of the calculated map match the true size of the laboratory (15 m x 6 m). As far as trajectory is concerned, figure 5.11 shows the reconstruction of the 11 cm translation and 3° rotation pattern. Unfortunately, ground-truth is not available in this case because the simulated path was obtained merging independent datasets; this is also the reason for the visible discontinuities in the estimated trajectory. Another feature that is apparent in figure 5.11 is the motion on a circular arc in correspondence of rotations. This is due to the fact that the axis around which rotation takes place does not intersect the stereo camera baseline; thus, imposing a rotation of the rotary stage also causes the midpoint of the baseline to move on a small circular arc.

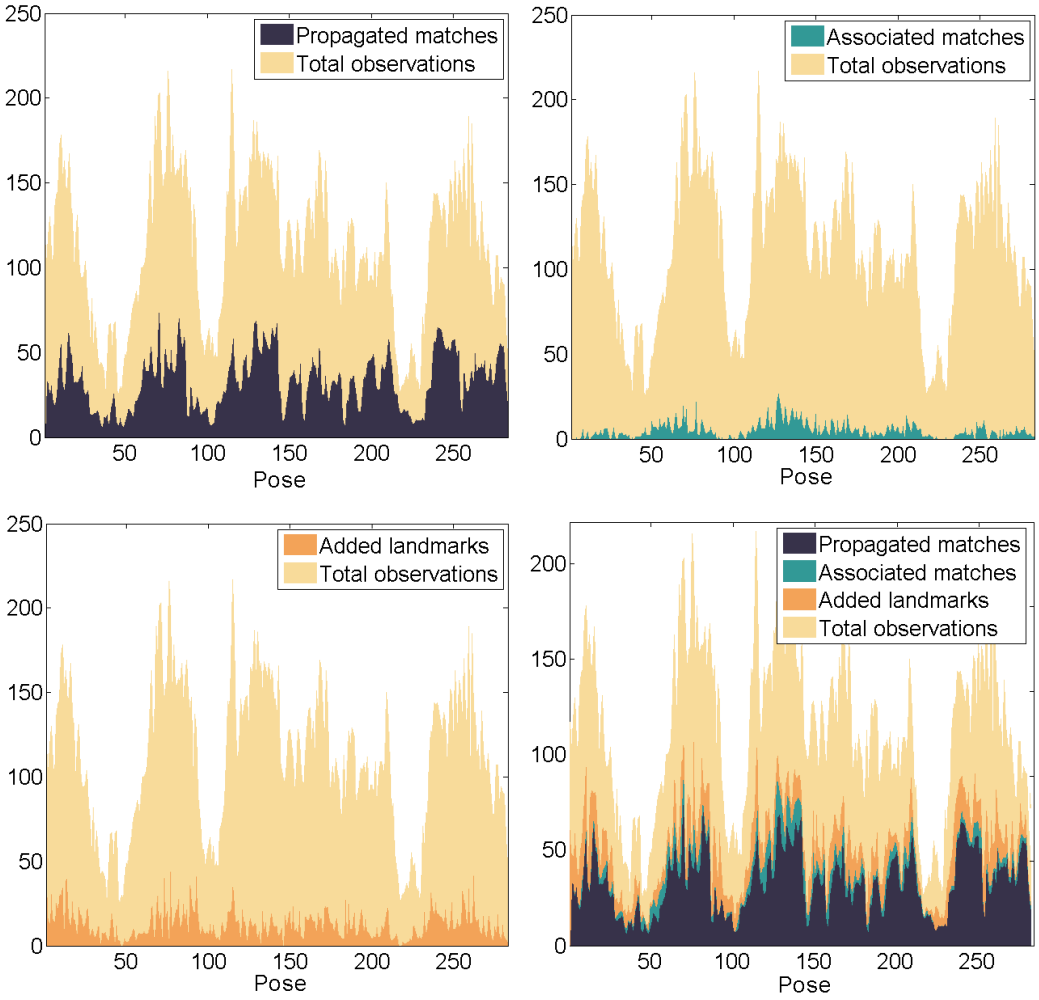

Figure 5.12: Data association.

Figure 5.12 provides details on the data association process. Fig. 5.12a shows for each step the number of landmarks identified by propagating the correspondence from previous steps through frame matching (see section 4.3.1), while fig. 5.12b shows associations by map search. As it can be seen, the majority of landmark correspondences are found by the tracking of features across subsequent frames rather than by map searches. Finally, fig. 5.12c shows that the number of landmarks added to the map at each step only accounts for a minor part of the total observations.

Three major decreases in the number of observed features can be individuated around steps 50, 100 and 220 (events 1, 2 and 3). The reason for this can be tracked down to particular orientations of the stereo camera that occur in correspondence of these three events. The tests were performed in a rectangular room, and when the camera rotation angle is $\pm 90°$, the optical axes are parallel to the short walls. In this configuration, the observed objects are closer to the cameras (up to a distance of about 2-3 m), resulting in a decrease in the number of observable features, as reported also in [22]. This affects the precision with which the robot pose is estimated. As it can be seen in figure 5.13, as the rotation angle approaches $+90°$ (event 1) or $-90°$ (events 2 and 3), both position and orientation uncertainty (estimated as explained in section 3.3.1) grow significantly. However, as the robot rotates back, known landmarks re-enter the field of view, providing information that helps localizing the robot so that the uncertainty decreases again. This does not happen however after the third event (step 220), because in that case the camera after rotating continues exploring new parts of the environment instead of re-observing known features. The situation in which not enough features are observed can also lead to erroneous pose estimates, as in the case near event 1 shown in fig 5.14a; fortunately, subsequent information propagates through the map, allowing the previous pose estimates to be corrected (fig 5.14b).

A possible solution to this loss of precision could be to adjust the SURF detector threshold when the count of observed features falls below a certain number, in order to obtain more matches. However in this case this was not considered necessary, since the map and trajectory are reconstructed correctly anyway, and decreasing the SURF threshold would increase the time required for both stereo matching and optimization.

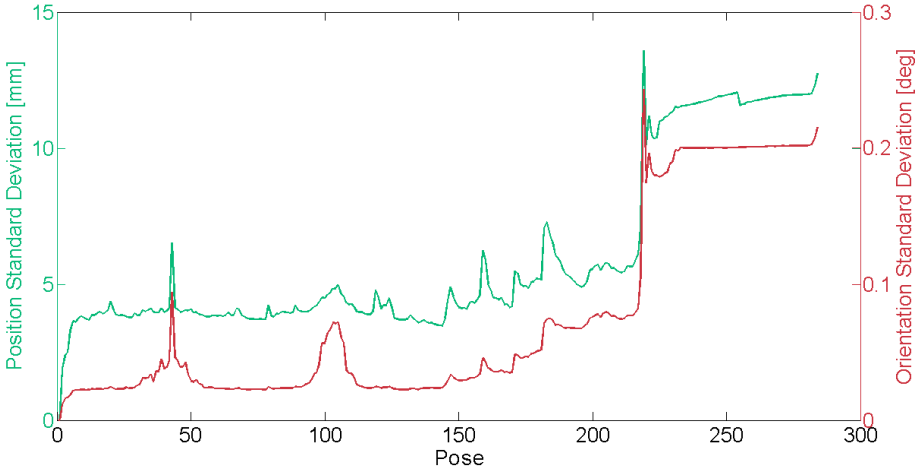

Figure 5.13: Estimated pose uncertainty at the end of the run.

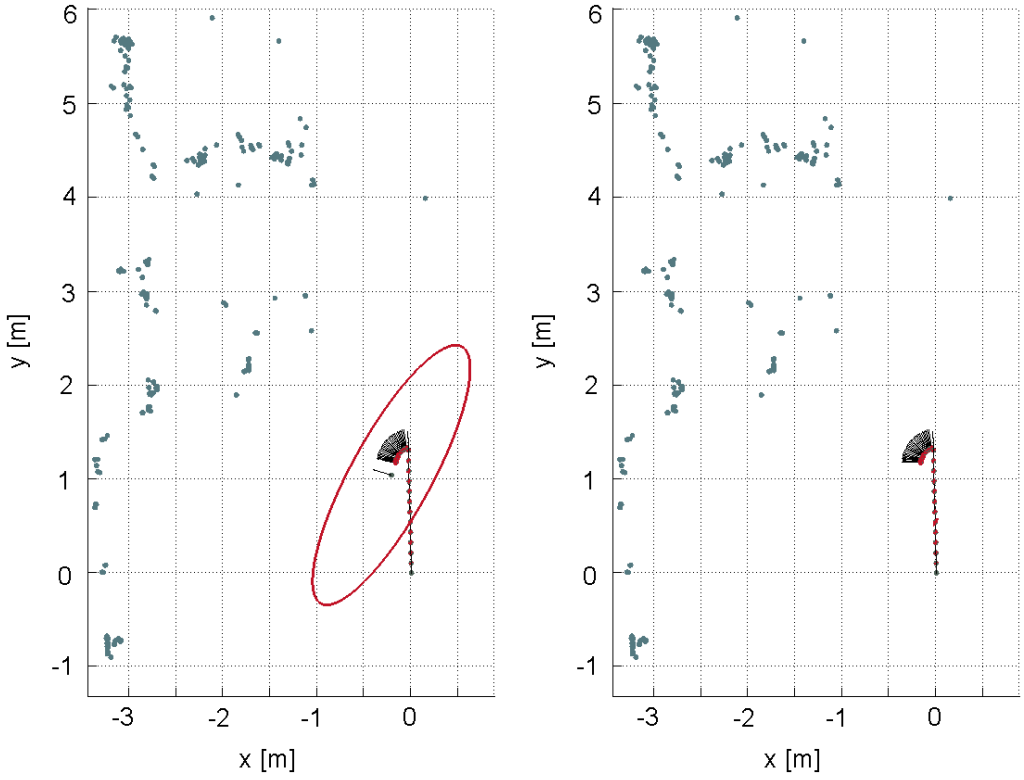

Figure 5.14: Pose estimate correction. a: Erroneous pose estimate with the relative uncertainty ellipse. b: The pose has been corrected and the uncertainty reduced in the light of subsequent information.

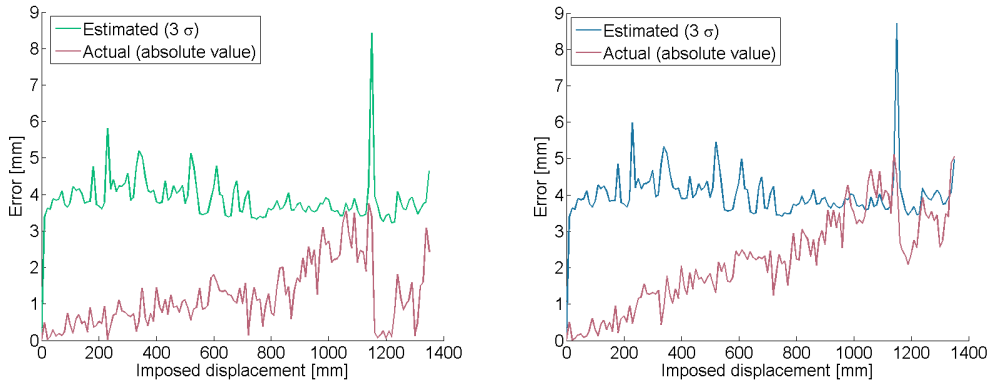

Figure 5.15: Estimated and actual errors for a low Mahalanobis threshold translation test (M = 500). a: Scheme A. b: Scheme B. Other relevant parameters are: SURF detector threshold = 700, step = 10 mm.

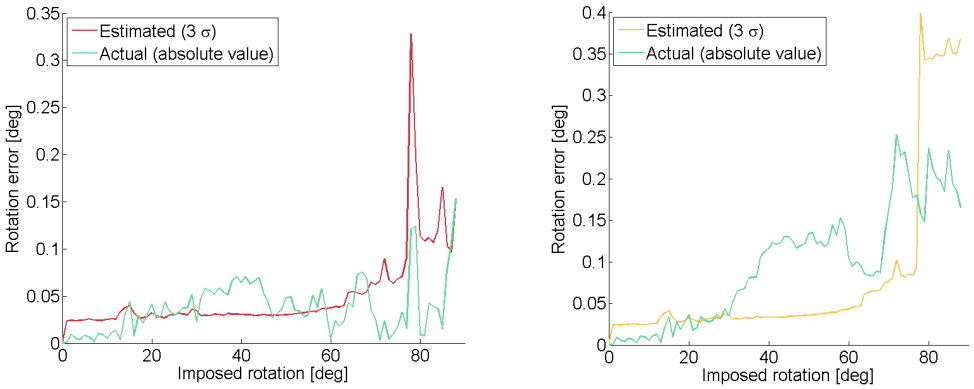

Figure 5.16: Estimated and actual errors for a low Mahalanobis threshold rotation test (M = 500). a: Scheme A. b: Scheme B. Other relevant parameters are: SURF detector threshold = 700, step = 1°.

As far as the precision of the estimated pose is concerned, although a proper ground-truth reference for the whole path is not available, an estimation of the uncertainty can be calculated as explained in section 3.3.1. Figures 5.15 to 5.18 show a comparison between the estimated uncertainty and the actual error for translation and rotation datasets for which ground truth is available. As it can be seen, when the Mahalanobis threshold is set

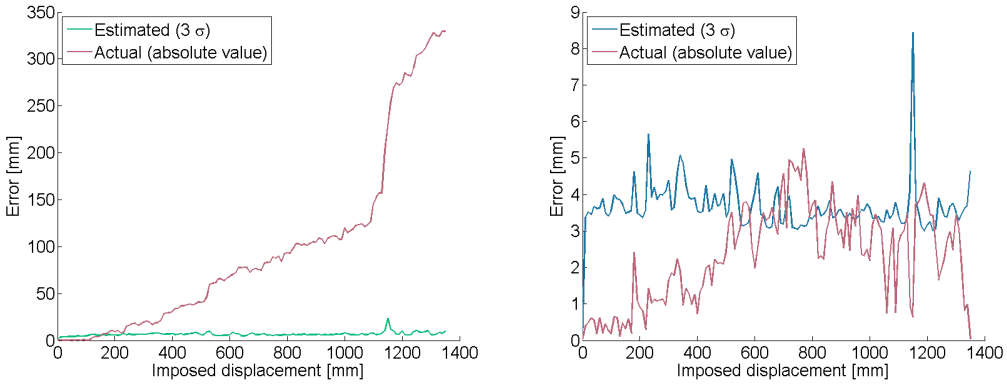

Figure 5.17: Estimated and actual errors for a high Mahalanobis threshold translation test (M = 500). a: Scheme A. b: Scheme B. Other relevant parameters are: SURF detector threshold = 700, step = 10 mm.

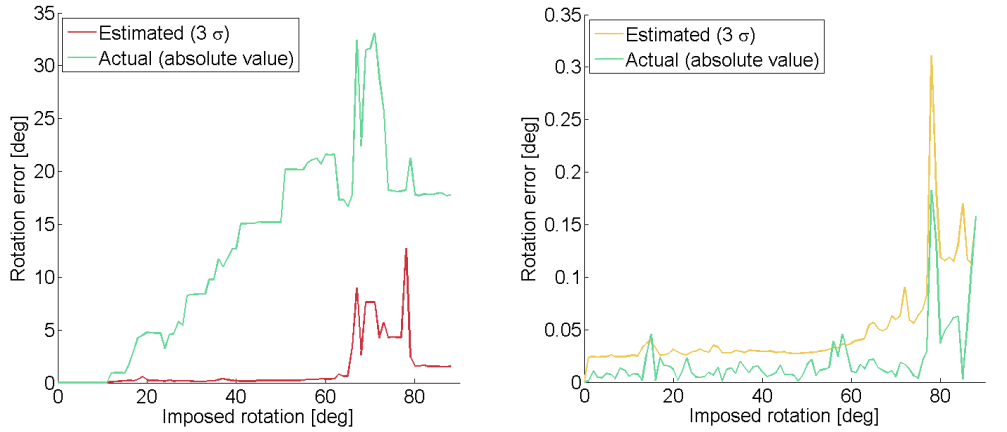

Figure 5.18: Estimated and actual errors for a high Mahalanobis threshold rotation test (M = 10000). a: Scheme A. b: Scheme B. Other relevant parameters are: SURF detector threshold = 700, step = 1°.

to a low value (figures 5.15, 5.16) the computed uncertainty provides a reasonable estimate of the actual error, especially in the case of scheme A; furthermore, rotations seem to cause a less precise estimation of the actual error compared to translations.

When the Mahalanobis threshold is increased, it can be seen from figures

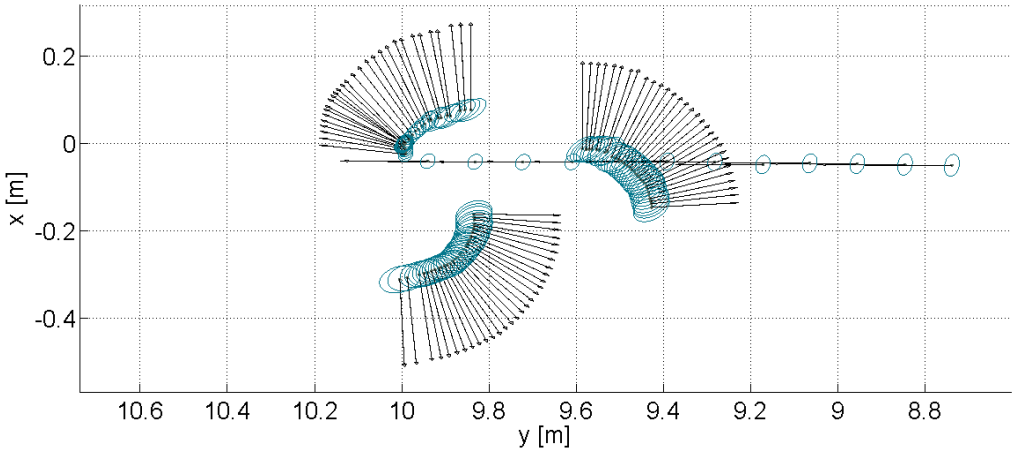

Figure 5.19: Uncertainty ellipses increasing in size in the neighbourhood of event 3.

5.17 and 5.18 that scheme A fails to estimate not only the correct displacement, but also the relative uncertainty. This is due to the fact that the uncertainty estimation method of section 3.3.1 assumes that the various constraints are uncertain due to sensor noise, but correct in the data association. As the Mahalanobis threshold is increased and data association outliers are introduced in the system, therefore, the estimation of uncertainty is not reliable any more (figures 5.17a, 5.18a). Scheme B however continues to provide a good estimation of both pose and uncertainty even with high Mahalanobis thresholds; this is probably due to the smaller errors that wrong scheme B associations introduce (as already discussed), to the smaller probability of establishing wrong constraints (since fewer potential candidates are examined for each association compared to scheme A), or both.

Figure 5.13 shows the estimated uncertainty over the entire robot's path; from the above considerations and figures, a conservative estimate of the error ($\sim$ 4 standard deviations to account for outliers in the data association) is 20 mm for the position and 0.15° for the orientation before event 3, and 50mm,

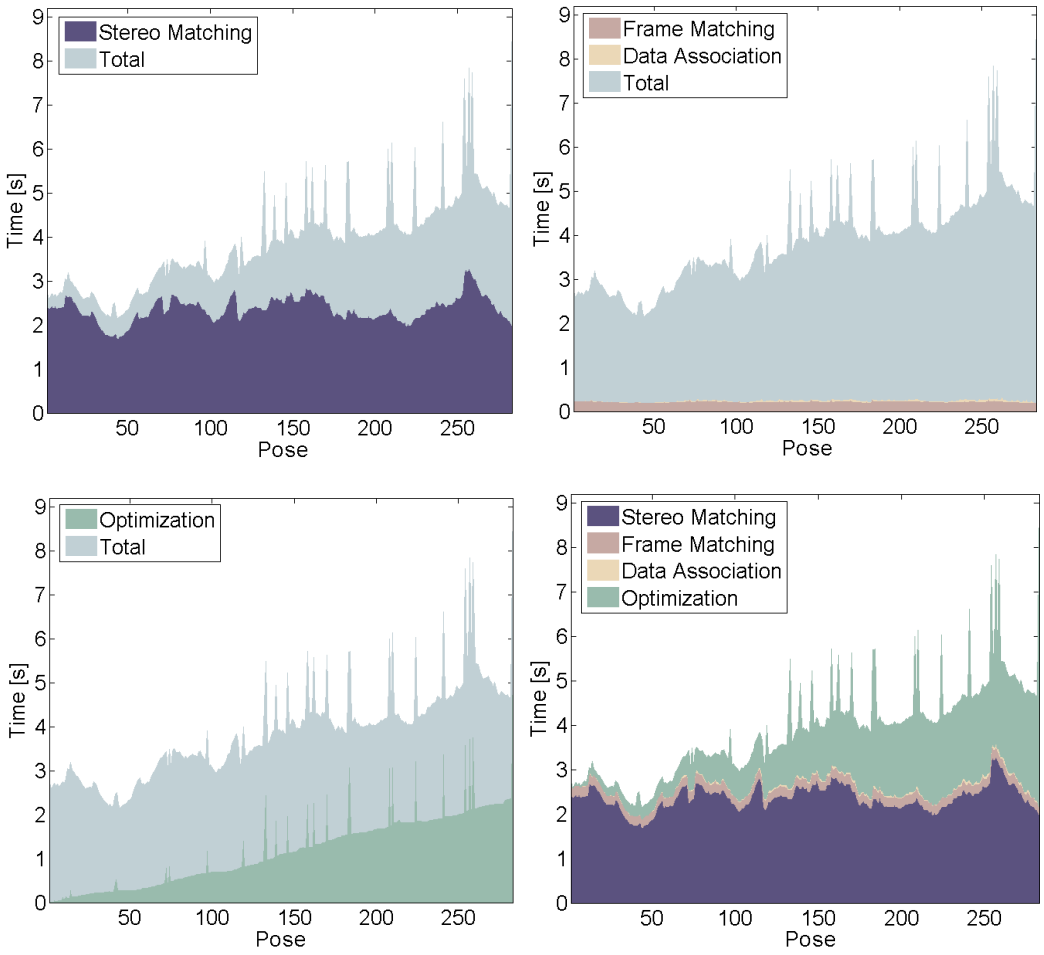

Figure 5.20: Processing times.

0.8° after event 3 (figure 5.19). At the end of the path, where the robot faces the door of the laboratory perpendicularly, its ground-truth displacement is unknown, but the nominal rotation of the frame in the relative dataset is -270°. The robot's estimate of its orientation, after a relevant displacement and several rotations, is -269.8° ,with an estimated standard deviation of 0.2°.

Figure 5.20 shows the processing times to run the simulation on a 2.67 GHz Intel Core i5-480M CPU. As it can be seen, feature detection, description and matching in stereo images account for a relevant part of the total

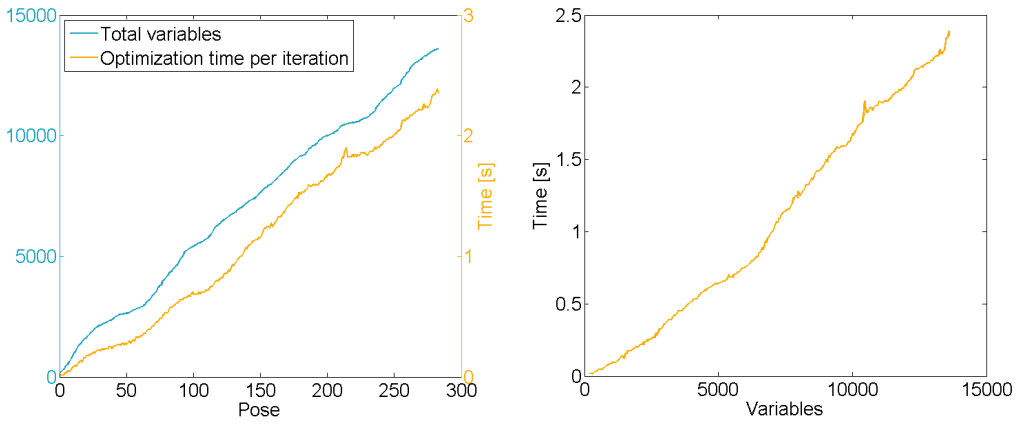

Figure 5.21: Variables and computational time growth.

time (5.20a). Frame matching on the other hand operates on the same keypoints already detected and described for stereo matching, and therefore requires a smaller amount of time; data association is a simple search in the map and is carried out very rapidly (5.20b). As expected, optimization time increases linearly with the size of the graph (figure 5.21b). Since the size of the graph is roughly linear in the number of poses considered, this results in a linear increase of the optimization time with the number of poses (figure 5.20c); peaks in the graph are relative to steps in which more than one Gauss-Newton iteration was carried out. The total time required to process a step at the beginning of the simulation is mainly due to stereo matching, while towards the end graph optimization demands about as much time.

Although a good precision was obtained with this simulation, processing times were too high for real-time application; therefore, a second simulation with increased steps and thresholds was run to investigate online operation capability. The steps were increased to 50 cm for translations and 10° for rotations, while the SURF threshold was increased to 3500, with the additional possibility to adjust it when the number of observed features is insufficient.

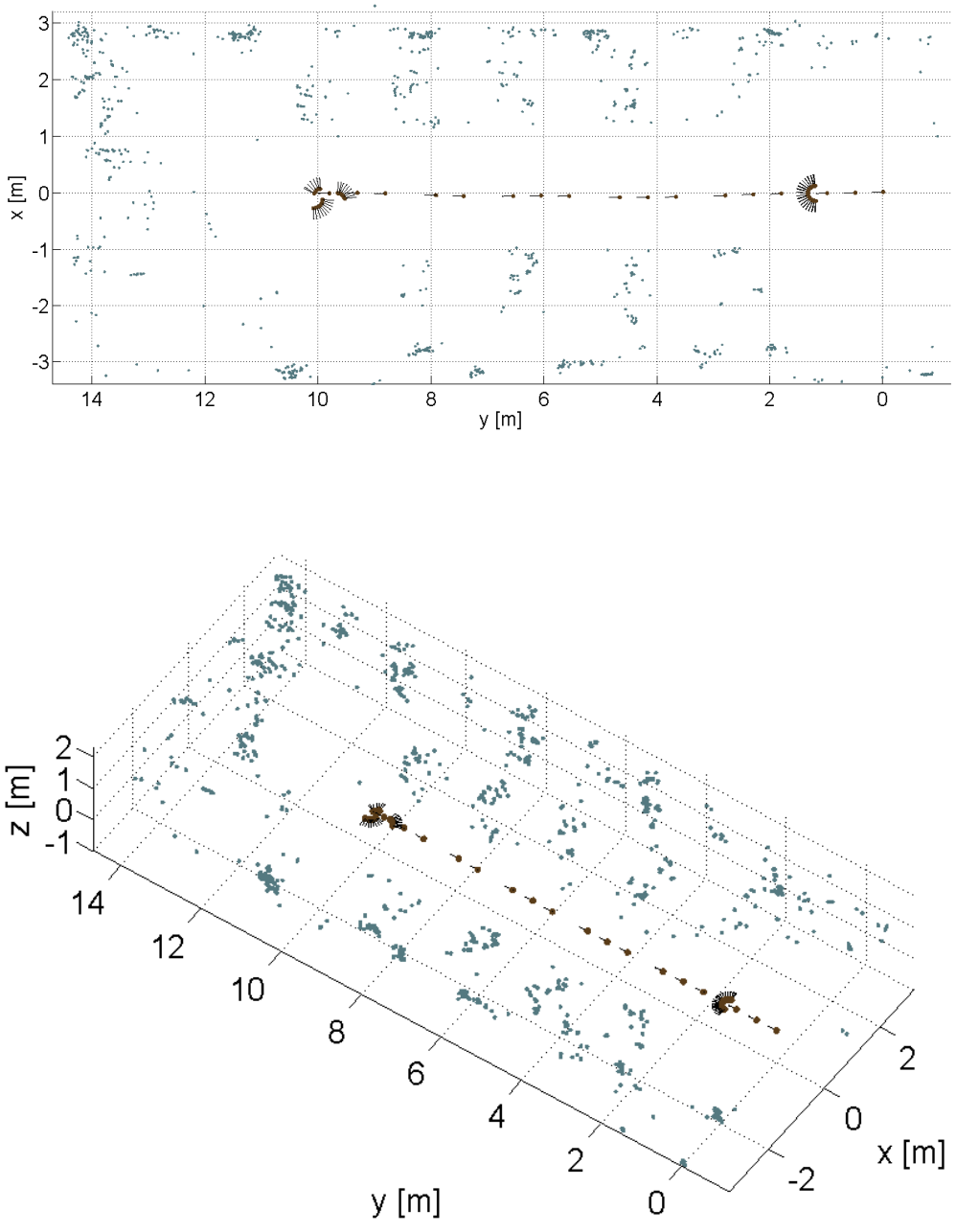

Figure 5.22: 2D and 3D views of the generated map.

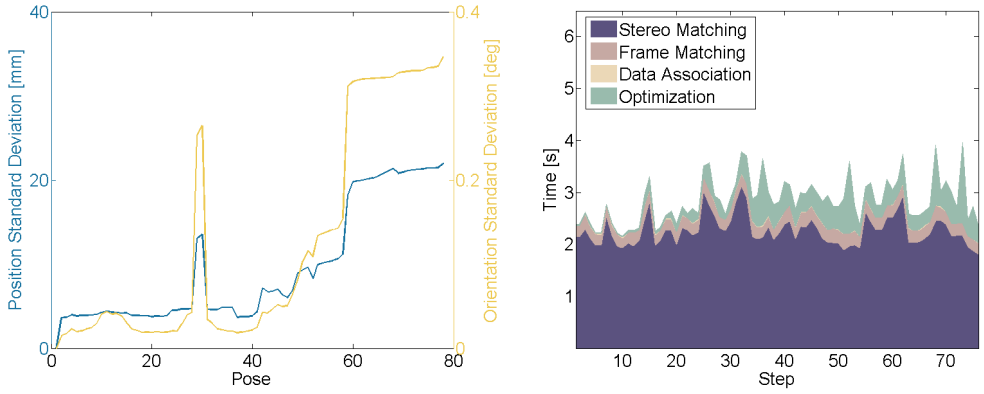

Figure 5.23: a: Estimated uncertainty and b: processing times for the fast simulation.

Since robustness is to be privileged over precision in online applications, scheme B was used for data association; to speed up the computation, the Mahalanobis threshold was increased to 5000. The resulting map shown in figure 5.22 is comprised of 78 robot poses and 979 landmarks. As expected, the map is now much sparser; recognizing objects of the environment is harder, but the shape and size of the obtained map and the estimated trajectory agree with the ones in figure 5.10.

Figure 5.23a shows the uncertainty estimation for this second run. Similarly to what happened in the previous simulation (figure 5.13), +90°/-90° orientations of the camera result in an increased uncertainty, the last being permanent since no loop closures take place afterwards. After the final event, an error estimate of 80 mm for the position and 1.3° for the orientation ($\sim 4\sigma$) can be inferred for this run. Thus, the computed map is sparser and the uncertainty greater compared to the previous simulation, but the overall loss of precision is limited.

Processing times are shown in figure 5.23a; as it can be seen, optimization is now much faster, although the number of iterations carried out per step is

greater (see table 5.1) due to the fact that the increased displacements result in a worse initialization of the new nodes added to the graph. The time required for image processing on the other hand does not decrease much, as expected from figure 5.7. These results suggests that potential for online application is present if image processing is sped up; a possible way to achieve this other than transferring image processing on a GPU is to decrease frame resolution. The stereo images used in this work have a resolution of 2040 x 1086 pixel each, and a parallel work on the same data showed that decreasing image size dramatically reduces the amount of time required for processing while still retaining a good precision of the resulting observations [23].

Finally, table 5.1 summarizes significant figures for the two simulations discussed in this chapter.

	Simulation 1 (Precise)	Simulation 2 (Fast)
Translation Step	11 cm	50 cm
Rotation Step	3°	10°
Data Association	Scheme A	Scheme B
Mahalanobis Threshold	500	5000
SURF Threshold	2500	3500
Total Poses	284	78
Total Landmarks	4140	979
Total Constraints	30958	3229
Image Processing (Avg.)	2.58 s	2.49 s
Optimization (Avg.)	1.212 s	0.341 s
Iterations Per Step (Avg.)	1.10	2.05
Final Position Uncertainty (4σ)	50 mm	80 mm
Final Orientation Uncertainty (4σ)	0.8°	1.3°

Table 5.1: Comparison of the two simulations.

Conclusions

A graph-based Simultaneous Localization And Mapping algorithm was implemented and tested. As a robot explores its surroundings, images coming from a stereo camera are processed to identify features in the environment and calculate their coordinates by triangulation. The estimated position and visual appearance of the observed features are then used to establish a correspondence with previously seen landmarks, and a non-linear optimization algorithm estimates the followed path and the structure of the environment that are most likely in the light of these informations.

The precision and performance of the algorithm was studied with both elementary transformations and a simulated complex trajectory in an indoor environment. Pose estimation precisions in the order of $\sim$3 mm over a 1350 mm translation and of $\sim 0.1^\circ$ over a 90° rotation were achieved for elementary transformations for which ground truth was available. At the end of the complex trajectory, a conservative estimate of pose uncertainty was (50 mm, 0.8°) for a detailed, computationally intensive simulation and (80 mm, 1.3°) for a sparser efficient run; the resulting map was in both cases coherent with the structure of the environment.

Two data association schemes were tested, one relying more on the visual appearance of features and one giving more weight to geometric proximity; the former was found to be more precise, while the latter proved to be more

robust in less favourable cases. For both schemes, the precision of the estimated solution was seen to increase as the image detector threshold, data association threshold and displacement step were decreased, but this also resulted in increased processing times.

As far as real applications are concerned, pose estimation was seen to fail when relevant rotations of the field of view occurred between consecutive images; therefore, a minimum frame rate requisite for the whole system would likely result from the need to cover rotations with a higher number of frames. Finally, while the optimization time is linear in the size of the map and shows potential for real-time operation, image processing was seen to be the bottleneck of the system that must be removed before considering an online application.

Bibliography

[1] P. Mountney, D. Stoyanov, A. Davison, G.Z. Yang, *Simultaneous stereoscope localization and soft-tissue mapping for minimal invasive surgery.* Medical Image Computing and Computer-Assisted InterventionMICCAI, 347-354, Springer Berlin Heidelberg (2006).

[2] S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics.* MIT Press, Cambridge (2005).

[3] B. Siciliano, O. Khatib, *Springer Handbook of Robotics.* Springer-Verlag Berlin Heidelberg, 1st edition (2008).

[4] R. Smith, M. Self, P. Cheeseman, *Estimating uncertain spatial relationships in robotics.* Uncertainty in Artificial Intelligence 2, 435-461, Elsevier Science Publishers B.V. (1988).

[5] P.S. Maybeck, *The Kalman filter: An introduction to concepts.* Autonomous Robot Vehicles, ed. by I.J. Cox, G.T. Wilfong. Springer, Berlin, Heidelberg (1990).

[6] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *FastSLAM: A factored solution to the simultaneous localization and mapping problem.* Proceedings of the Edmonton AAAI National Conference on Artificial Intelligence (2002).

[7] D. Hähnel, W. Burgard, D. Fox, S. Thrun, *An efficient FastSLAM Algorithm for Generating Maps of Large-Scale Cyclic Environments from Raw Laser Range Measurements*. Proceedings of the 2003 IEEE/RSJ Conference on Intelligent Robots and Systems (2003).

[8] F. Lu, E. Milios, *Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans*. Journal of Intelligent and Robotic Systems 18, 249-275 (1997).

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

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

[11] E. Olson, P. Agarwal, *Inference on networks of mixtures for robust robot mapping*. The International Journal of Robotics Research 32(7), 826-840 (2013).

[12] T. Lemaire, C.Berger, I.K. Jung, *Vision-Based SLAM: Stereo and Monocular Approaches*. International Journal of Computer Vision 74(3), 343-364 (2007).

[13] K.B. Petersen, M.S. Pedersen, *The Matrix Cookbook* (2012). http://matrixcookbook.com

[14] P. Agarwal, G.D. Tipaldi, L. Spinello, C. Stachniss, W. Burgard, *Robust map optimization using dynamic covariance scaling*. Proceedings of 2013

IEEE International Conference on Robotics and Automation (ICRA), 62-69 (2013).

[15] J. Vaganay, M.J. Aldon, A. Fournier, *Mobile Robot Attitude Estimation by Fusion of Inertial Data*. Proceedings of 1993 IEEE International Conference on Robotics and Automation, 277-282 (1993).

[16] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool, *Speeded-Up Robust Features (SURF)*. Computer Vision and Image Understanding 110, 346-359 (2008).

[17] R. Hartley, A. Zisserman, *Multiple View Geometry In Computer Vision*. Cambridge University Press, 2nd edition (2003).

[18] D. Marsh, *Applied Geometry for Computer Graphics and CAD*. Springer, 2nd edition (2005).

[19] L. Matthies, S. Shafer, *Error modeling in stereo navigation*. IEEE Journal of Robotics and Automation 3(3), 239-248 (1987).

[20] W. Brink, C. E. Van Daalen, W. Brink, *Probabilistic outlier removal for robust landmark identification in stereo vision based SLAM*. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2822-2827 (2012).

[21] A. Gil, O. Reinoso, O. M. Mozos, C. Stachniss, W. Burgard, *Improving Data Association in Vision-based SLAM*. Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems.

[22] M. Pertile, S. Chiodini, S. Debei, E. Lorenzini, *Laboratory calibration and comparison of three visual odometry systems*. IX Congresso MMT, 48-55 (2014).

[23] G. Soldà, *Implementation and experimental verification of a 3D-to-2D Visual Odometry algorithm for real-time measurements of a vehicle pose.* Master Thesis, University of Padova (2015).