Study on the Use of Vision and Laser Range Sensors with Graphical Models for the SLAM Problem

Ellon Paiva Mendes

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *12/07/2017* **par :**

Ellon PAIVA MENDES

STUDY ON THE USE OF VISION AND LASER RANGE SENSORS WITH GRAPHICAL MODELS FOR THE SLAM PROBLEM

JURY

Roland CHAPUIS	Professeur des Universités	Rapporteur
Luc JAULIN	Professeur des Universités	Rapporteur
Philippe BONNIFAIT	Professeur des Universités	Membre du jury
Rachid ALAMI	Directeur de recherche	Membre du Jury
Joan SOLÀ	Docteur	Membre du Jury
Simon LACROIX	Directeur de Recherche	Directeur de thèse

École doctorale et spécialité :
EDSYS : Robotique 4200046

Unité de Recherche :
Laboratoire d'analyse et d'architecture des systèmes

Directeur de Thèse :
Simon LACROIX

Rapporteurs :
Roland CHAPUIS et *Luc JAULIN*

Abstract

A strong requirement to deploy autonomous mobile robots is their capacity to localize themselves with a certain precision in relation to their environment. Localization exploits data gathered by sensors that either observe the inner states of the robot, like acceleration and speed, or the environment, like cameras and *Light Detection And Ranging* (LIDAR) sensors. The use of environment sensors has triggered the development of localization solutions that jointly estimate the robot position and the position of elements in the environment, referred to as Simultaneous Localization and Mapping (SLAM) approaches. To handle the noise inherent of the data coming from the sensors, SLAM solutions are implemented in a probabilistic framework. First developments were based on Extended Kalman Filters, while a more recent developments use probabilistic graphical models to model the estimation problem and solve it through optimization. This thesis exploits the latter approach to develop two distinct techniques for autonomous ground vehicles: one using monocular vision, the other one using LIDAR.

The lack of depth information in camera images has fostered the use of specific landmark parametrizations that isolate the unknown depth in one variable, concentrating its large uncertainty into a single parameter. One of these parametrizations, named Parallax Angle Parametrization, was originally introduced in the context of the Bundle Adjustment problem, that processes all the gathered data in a single global optimization step. We present how to exploit this parametrization in an incremental graph-based SLAM approach in which robot motion measures are also incorporated.

LIDAR sensors can be used to build odometry-like solutions for localization by sequentially registering the point clouds acquired along a robot trajectory. We define a graphical model layer on top of a LIDAR odometry layer, that uses the *Iterative Closest Points* (ICP) algorithm as registration technique. Reference frames are defined along the robot trajectory, and ICP results are used to build a pose graph, used to solve an optimization problem that enables the correction of the robot trajectory and the environment map upon loop closures.

After an introduction to the theory of graphical models applied to SLAM problem, the manuscript depicts these two approaches. Simulated and experimental results illustrate the developments throughout the manuscript, using classic and in-house datasets.

Résumé

La capacité des robots mobiles à se localiser précisément par rapport à leur environnement est indispensable à leur autonomie. Pour ce faire, les robots exploitent les données acquises par des capteurs qui observent leur état interne, tels que centrales inertielles ou l'odométrie, et les données acquises par des capteurs qui observent l'environnement, telles que les caméras et les Lidars. L'exploitation de ces derniers capteurs a suscité le développement de solutions qui estiment conjointement la position du robot et la position des éléments dans l'environnement, appelées SLAM (*Simultaneous Localization and Mapping*). Pour gérer le bruit des données provenant des capteurs, les solutions pour le SLAM sont mises en œuvre dans un contexte probabiliste. Les premiers développements étaient basés sur le filtre de Kalman étendu, mais des développements plus récents utilisent des modèles graphiques probabilistes pour modéliser le problème d'estimation et de le résoudre grâce à techniques d'optimisation. Cette thèse exploite cette dernière approche et propose deux techniques distinctes pour les véhicules terrestres autonomes: une utilisant la vision monoculaire, l'autre un Lidar.

L'absence d'information de profondeur dans les images obtenues par une caméra a mené à l'utilisation de paramétrisations spécifiques pour les points de repères qui isolent la profondeur inconnue dans une variable, concentrant la grande incertitude sur la profondeur dans un seul paramètre. Une de ces paramétrisations, nommé paramétrisation pour l'angle de parallaxe (ou PAP, *Parallax Angle Parametrization*), a été introduite dans le contexte du problème d'ajustement de faisceaux, qui traite l'ensemble des données en une seule étape d'optimisation globale. Nous présentons comment exploiter cette paramétrisation dans une approche incrémentale de SLAM à base de modèles graphiques, qui intègre également les mesures de mouvement du robot.

Les Lidars peuvent être utilisés pour construire des solutions d'odométrie grâce à un recalage séquentiel des nuages de points acquis le long de la trajectoire. Nous définissons une couche basée sur les modèles graphiques au dessus d'une telle couche d'odométrie, qui utilise l'algorithme ICP (*Iterative Closest Points*). Des repères clefs (*keyframes*) sont définis le long de la trajectoire du robot, et les résultats de l'algorithme ICP sont utilisés pour construire un graphe de poses, exploité pour résoudre un problème d'optimisation qui permet la correction de l'ensemble de la trajectoire du robot et de la carte de l'environnement à suite des fermetures de boucle.

Après une introduction à la théorie des modèles graphiques appliquée au problème de SLAM, le manuscrit présente ces deux approches. Des résultats simulés et expérimentaux illustrent les développements tout au long du manuscrit, en utilisant des jeux des données classiques et obtenus au laboratoire.

Acknowledgments

I would first like to thank my mother, Raimunda Paiva, for all the love and support she gave me not only when I decided to pursue my PhD degree in far away lands, but for all my life. *Muito obrigado por tudo, mãe.*

I would like to thank Simon Lacroix, my thesis supervisor, who always tried to help me with his knowledge and his optimism, especially in the difficult times.

I would also like to acknowledge the Brazilian National Council for Scientific and Technological Development (CNPq) for the financial support for most of my thesis.

I cannot forget to mention my *Guardian* friends that I left back in Brazil when I started my thesis. Even with the distance you were always in my mind, and your words of motivation kept me going.

I would also like to thank Petra Müllerová for her support and friendship in the last years. As well as her parents and family, who welcomed me many times and made me feel one of them, even if we did not speak the same language.

Finally, I would like to thank everyone at LAAS who shared some time with me during the years of my thesis. Many came and went, and I must express my very profound gratitude to Artur Maligo, Mamoun Gharbi, Elena Stumm, Alex Ravet, Mathieu Geisert, Christophe Reymann, and Amélie Barozet, with whom I could spend some time also out of the laboratory. I am very glad to have met you. The laughs we shared helped me keep my mental sanity along the way.

Many thanks to you all!

Contents

Chapter 1

Introduction

Mobile robot autonomous navigation requires the integration of a variety of processes that can be casted as an instantiation of the sense / plan / act paradigm for autonomous systems. Sensing encompasses the acquisition of information from the environment and their structuring into environment models, which are the basis upon which the planning processes decide the motions to execute, and execution being part of the acting process. Depending on the type of robots (*e.g.* ground, aerial or submarine), on the type of sensors that provide information on the environment (*e.g.* cameras, LIDARS, radars), the type of environments (*e.g.* indoors, urban), and on the context in which autonomous navigation is achieved (*e.g.* reach a far away goal, follow a road), these processes are declined into a large variety – not to mention that numerous different approaches have been proposed in the literature to implement them.

Besides the three sense / plan / act processes, there is an other essential process that is required for autonomous navigation, actually often classified as a sensing process: robot localization, which is the estimate of the robot position in a given reference frame. This process is actually one of the most important one in mobile robotics, as it conditions the success of all the others: environments models are not spatially consistent in the absence of localization, which precludes any consistent motion planning, and trajectory execution, whatever the servoing scheme is, strongly relies on localization. Localize oneself has motivated numerous work in the history of technology, and when it comes to mobile robotics, there is again a large variety of sensors, contexts and definition for the localization function, which generated a considerable amount of work in the community.

This thesis is a contribution to the localization problem in robotics, within a formalism that has recently been introduced and applied to localization: the graphical models. This section introduces the manuscript, by presenting first the context of the work, briefly reviewing the importance of the localization problem and the main types of solutions, and then the thesis contributions and outline.

1.1 Context

1.1.1 Outdoor robotics

The primary considered context of this work is large scale outdoor environments, either structured or not (fig. 1.1). The deployment of robots in such environments is typically considered for information gathering missions (to sample, map or monitor

some information in scientific contexts), transportation, logistics or surveillance applications.

Figure 1.1: Illustrations of some of the outdoor environments in which the two LAAS robots Mana and Minnie have been deployed (fields, rural and urban environments).

These contexts define a series of characteristics that relate to the environment and the robots: the environment is large, varied, may not be known beforehand, and missions are long lasting. Of course these characteristics are more or less pregnant, depending on the actual application. Yet we consider the generic case where the robot has to achieve long range navigation missions in such environments.

1.1.2 On the importance of localization

On-board an autonomous mobile robot, localization *per se* is not a functionality that is the goal of the robot mission. Yet, it is a key component necessary for three main functionalities that are at the core of the robot autonomy:

- It is required to ensure the proper *execution of trajectories*, that are either specified to the robot or autonomously planned. For this purpose, localization explicits the deviations with respect to the trajectories, which are the entry point of the lowest level locomotion and motion controls activities.
- It is required to ensure the *spatial consistency of the environment models* built by the robot. Such models (*e.g.* obstacles maps) are defined by the structuring and integration of exteroceptive sensor data: the positions from which these data are acquired have to be known, otherwise their integration would yield environment models that do not properly represent the robot environment.
- It is required to ensure the *achievement of the missions* the robot is tasked. This is trivial if the mission is a mere navigation one ("goto [goal]"), or when the mission is to ensure coverage of a given area. The proper handling of the missions is ensured by higher levels planners, that must reason of the progress of the mission: the robot localization information is of course necessary for this purpose.

These three functionalities are processes of very different nature, and are solved by various approaches. Hence they call for different instances of localization solutions: for instance executing a trajectory may require to localize the robot at a high

frequency relative to local features to servo the motions, such as a road line, or with respect to a local frame, such as the beginning of the trajectory. Building a local environment model may also only require precise localization over short ranges, but at the frequency at which the exteroceptive data are gathered. And to monitor the achievement of the overall mission, an estimate of the robot position is not needed at high frequencies, nor with a very good precision.

There is actually a wide range of localisation solutions, that can be characterized along the following criteria:

- the precision of the localisation, and its evolution over space and time,
- the frequency at which the robot position is estimated,
- the reference frame in which the position estimate is expressed,
- the availability of the localisation solution, which can depend on the area where the robot is, or even on the time of the day,
- and the integrity of the localisation solution, that is the level of guarantee one has on the position and precision estimates.

To our knowledge, no localisation solution is able to maximize all these criteria, and it is very likely that the state of the art will never reach this: a series of localisation solutions with varying performance for each criteria must be developed and integrated on-board an autonomous robot.

1.1.3 Localization solutions in robotics

A large variety of sensors can provide information on the motion and/or the localization of a mobile, and robotics has inherited all these technologies: accelerometers, gyrometers, magnetometer, speedometers, etc. But robotics introduced in the realm of localization the use of *exteroceptive sensors*, that is sensors that acquire information on the environment, and not on the robot inner state. These sensors are of course the entry point for the autonomy of robots that must *adapt to their environment*, and they have been used since the very beginning of researches on autonomous mobile robots in the beginning of the 70's. The most used are sonar telemeters, cameras, Lidars and radars, and they have been rapidly exploited in localization processes.

One can classify localization solutions in robotics into the following three large categories:

- Dead-reckoning. This consists in integrating a sequence of elementary motion measures to compute the current pose of the robot. The most straightforward dead-reckoning approach for wheeled mobile robot is odometry. Inertial navigation also falls in these approaches, as well as visual motion estimation

techniques from monocular or stereovision sequences – often coined "visual odometry". The main characteristic of dead-reckoning approaches is that the error on the position they estimate grows with time and/or distance traveled, and this growth can not be bounded. Hence dead-reckoning approaches are not applicable over the long term.

- Map-based localization. The only solution to guarantee bounded errors on the position estimates is to rely on stable environment features. If a spatially consistent map of the environment is available, map-based localization approaches can be applied: they consist in matching data acquired by the robot with a given initial map of the environment that is spatially consistent. Various approaches exist, depending on the kind of information that is extracted to be matched (*e.g.* mapped features that are detected in the environment by the robot, surface models like digital elevation map, etc), and on the estimation technique that derive position estimate in the map frame from the matched data. Map-based localization provide position estimates whose errors are bounded and depend on the resolution of the data encoded in the initial map. Yet, they require the existence of a prior map of the environment, and assume that the environment has not drastically changed between the time it has been mapped and the time the robot evolves within.

- Simultaneous localization and mapping. In the absence of an a priori map of the environment, the robot is facing a kind of "chicken and egg problem": it makes noisy observations of features on the environment, from positions which estimates are also corrupted with noise. These errors in the robot's pose have an influence on the estimate of the observed environment feature locations, and similarly, the use of the observations of previously perceived environment features to locate the robot provide position estimates that inherits from both errors: the errors of the robot's pose and the map features estimates are correlated. It has early been understood in the robotic community that the mapping and the localization problems are intimately tied together, and that they must therefore be concurrently solved in a unified manner, turning the chicken and egg problem into a virtuous circle. The approaches that solve this problem are commonly denoted as "Simultaneous Localization and Mapping" (SLAM), and have now been thoroughly studied.

The SLAM problem has received a lot of attention since the early stages of researches on mobile robotics. It is indeed a solution to both the important localization and mapping problems that is autonomous (the robot does not need to resort to any initial data or dedicated localisation infrastructure such as Global Navigation Satellite System), and is thus applicable in any context, provided the environment exhibit some structures that are perceivable by some sensors.

1.2 Thesis objective and contributions

This thesis builds upon the latest researches on the SLAM problem, that use probabilistic graphical model theory to solve SLAM problems through optimization [Grisetti 2010]. By exploring the sparsity of the SLAM problem, recent work have shown the possibility to considerably reduce the computation time needed to converge to a solution. Also, the graphical models used to model the optimization problem provide a powerful layer of abstraction that helps to better understand the problem and to design powerful solutions, like the iSAM2 algorithm [Kaess 2012].

The thesis tackles two variations of the problem for ground mobile robots evolving outdoors :

- SLAM with monocular vision. The well known issue with monocular vision is that the landmark depth is not directly perceivable: we consider the problem in the least favorable context, that is with a camera pointing towards the direction of the robot motion. In this contribution we tailor an optimization-based SLAM approach to a specific landmark parametrization that has proven its performance in the context of a batch bundle adjustment approach, the *parallax angle parametrization* [Zhao 2011].

- Long range SLAM with a Lidar. Here we adopt a pose-graph SLAM approach, in which no landmarks are extracted from the perceived data: the SLAM solutions builds upon the estimate of motions computed thanks to a scan alignment technique applied on the point clouds data provided by the Lidar, namely, the *Iterative Closest Points* (ICP) algorithm. A particular care is taken on augmenting the quality of the motion estimations computed by ICP, by properly filtering the input data on the one hand, and by properly managing the built "map" on the other hand, that is the selection of the point clouds to memorize along the robot route. This latter point also enables to detect loop closures.

1.3 Thesis outline

Chapter 2 is a synthetic introduction to the SLAM problem, focusing on its graphical modeling and on solutions based on non-linear optimization techniques. Its goal is to make the reader familiar with the concepts and tools that are latter used in the manuscript.

Chapter 3 is our first contribution. It proposes the use of a specific parametrization of point landmarks in the context of an incremental Graphical SLAM approach. All the details of the adaptation of a Graphical SLAM approach to this parametrization are given, and results in simulation are analysed.

Chapter 4 presents our second contribution, that is the development of a pose-graph SLAM approach based on the ICP scan matching algorithm. All the details

of the approach are provided, and results obtained in two different datasets are analysed.

Finally a conclusion ends the manuscript, and an appendix depicts some tools we have been developing to support our researches.

Chapter 2

Probabilistic Graphical Models for SLAM

Contents

The main subject of this thesis is the Simultaneous Localization and Mapping (SLAM) problem. This chapter presents this problem, with a focus on the current trend to model it using graphical models and to solve it using non-linear optimization techniques.

The SLAM problem has been studied in robotics for about thirty years now [Chatila 1985, Smith 1987, Crowley 1989], and many works that explains very well the problem were produced since [Bailey 2006, Durrant-Whyte 2006, Cadena 2016]. We do not have the ambition here to complement these texts: this chapter is a synthetic presentation of the SLAM problem, so as to present the basis into where the two contributions presented in chapters 3 and 4 of this thesis are built.

2.1 Modeling the SLAM problem

In a nutshell, the SLAM problem consists in estimating the pose of the robot in relation to a map, while computing an estimate for this map at the same time. This

is a fusion of the localization problem, where a fixed map is available and one estimates only the robot pose, with the mapping problem, where the robot localization is known and one seeks to find a map that best describes the environment.

This situation is very common in robotics. Often robots are requested to explore an unknown environment and perform tasks there, and for these purposes they need to know their location and their surroundings. Sometimes a map is available but it may be unprecise, incomplete or it may not correspond to the current state of the environment anymore: it is necessary to perform map updates as the robots perceive the current state of the environment. Thus, being able to perform SLAM is a key requirement to allow autonomous tasks and missions for robots, which has been reckoned early in the roboticists community.

2.1.1 Probabilistic Formulation of SLAM

The estimation is computed based on data acquired from the robot sensors. This data contains noise, which any SLAM solution should handle properly. The common way to do is to use a stochastic framework, where the observations and values to be estimated are random variables. Mathematically we can write the following: if $\mathbf{x}_{0:T} = \{\mathbf{x}_0, \dots, \mathbf{x}_T\}$ is a sequence of robot states (including the initial state $\mathbf{x}_0$), $\mathbf{o}_{1:T} = \{\mathbf{o}_1, \dots, \mathbf{o}_T\}$ is a sequence of observations made from the robot sensors, and $\mathbf{m}$ is set of random variables that represents the map of the environment, then to solve the SLAM problem is the same as to compute the following posterior probability:

$$p(\mathbf{x}_{0:T}, \mathbf{m} \,|\, \mathbf{o}_{1:T}) \tag{2.1}$$

with T representing the elapsed time. We can further divide the observations $\mathbf{o}_{1:T}$ depending on the type of sensor used to acquire the data. Proprioceptive sensors provide measurements of how the robot state changes, denoted by $\mathbf{u}_{1:T}$. On the other hand, exteroceptive sensors provide measurements of the environment relative to the robot pose and are denoted by $\mathbf{z}_{1:T}$. Using these two variables, and considering that the robot origin $\mathbf{x}_0$ is normally known or arbitrary, we can rewrite eq. (2.1) as:

$$p(\mathbf{x}_{1:T}, \mathbf{m} \,|\, \mathbf{x}_0, \mathbf{u}_{1:T}, \mathbf{z}_{1:T}) \tag{2.2}$$

Computing an estimate for the posterior in eq. (2.2) requires to operate in high dimensional states and would not be possible if the SLAM problem did not have a well defined structure [Grisetti 2010]. This structure is derived from the *Markov assumption*, that postulates that past and future are independent if the current state is known. This leads to two other assumptions that if not considered would violate the Markov one: the *static world* and the *independent noise* assumptions.

With the Markov assumption it is possible to derivate simplified probabilistic models that govern the SLAM problem. The first one is the *transition model*:

$$p(\mathbf{x}_t \,|\, \mathbf{x}_{0:t-1}, \mathbf{m}, \mathbf{u}_{1:t}, \mathbf{z}_{1:t}) = p(\mathbf{x}_t \,|\, \mathbf{x}_{0:t-1}, \mathbf{u}_{1:t}) = p(\mathbf{x}_t \,|\, \mathbf{x}_{t-1}, \mathbf{u}_t) \tag{2.3}$$

that expresses mathematically that *the current state depends only on the last state and the last control (or proprioceptive measurement)*. The second model describes the exteroceptive sensors. It is named the *observation model*:

$$p(\mathbf{z}_t \mid \mathbf{x}_{0:t}, \mathbf{m}, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}) = p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m}) \tag{2.4}$$

which means in other words that *the expected current world observations depends only on the current state and on the map*. These two models form the basis for the stochastic SLAM solutions.

2.1.2 Structure of a SLAM solution

An essential concern when defining a SLAM solution is the *map structure*. So far we considered the exteroceptive observations $\mathbf{z}_t$ in a generic way, where they are just "observations of the world", and the map has been represented only by the variable $\mathbf{m}$. While this may be convenient to write probabilities in a compact form, this hides an important aspect of the SLAM problem, due to the fact that sensor data are most of the times only observations of a *part* of the environment.

Mathematically, $\mathbf{z}_t = \{\mathbf{z}_{t1}, \ldots, \mathbf{z}_{tj}\}$ is a set of observations of *parts of the world*. In the same way, the variable $\mathbf{m}$ represents the set $\mathbf{m} = \{\mathbf{l}_1, \ldots, \mathbf{l}_n\}$, where each $\mathbf{l}_i$ represents one of these parts: a distinguishable world element that can be observed from the sensor data, or simply: a landmark.

Given the subdivision of both $\mathbf{z}_t$ and $\mathbf{m}$, we can talk about two different kind of processes that are required to define a SLAM solution:

- **Data association**. When we consider that $\mathbf{z}_t$ is composed by several observations of parts of the world, one question that arises is which part $\mathbf{l}_i$ of the world is observed by which observation $\mathbf{z}_{tj}$. Thus, before using the observations it is necessary to associate the observation with the part of the map it relates to. This problem is called *data association*, and it is crucial for the SLAM problem. The part responsible to perform data association in a SLAM system is commonly known as *front-end*. The front-end deals with data from the sensors, and thus is heavily sensor dependent. It is a critical part of SLAM: wrong data association may induce wrong estimations for the map, which results in a deformed map that does not represent well the environment and cannot be used for localization.

  It is not in the scope of this thesis to deal with data association: it is considered solved for the contribution presented in chapter 3, and it is implicit in the contribution presented in chapter 4.

- **Estimation**. The part responsible for the estimation is called *back-end*, and is normally sensor agnostic. Both front-end and back-end of a system may be independent from each other, but they may also be coupled. For instance, the front-end may use the current estimation from the back-end to perform guided data association (associate observations with the most probable parts of the

map); or the back-end may detect inconsistencies in the data-association and, if possible, remove or not use the inconsistent observations to compute the estimation.

Using these new variables and the process of data association, the observation model in eq. (2.4) then becomes:

$$p(\mathbf{z}_{tj} \,|\, \mathbf{x}_t, \mathbf{l}_i) \tag{2.5}$$

that is the probability of the measurement of one specific landmark given the robot pose, given the data association $\mathcal{D}$ and $(i, j) \in \mathcal{D}$.

The format of the variables that compose the map $\mathbf{m}$ depend on the sensors used by the robot and on the information extracted from the sensor data. For instance: robot systems that detect landmarks-features in images will contain the representation of these features in the map (points, lines, etc.); a robot moving in a plane that uses a 2D laser range finder may exploit an occupancy grid as the map, or the positions of features detected on the scans (corners, walls, etc).

Using several variables to describe the map has the benefit to exposing in details how the variables (composing the map and the trajectory) interact with each other. This will become clearer when we discuss graphical models below. A toy-example showing a robot moving and observing different landmarks is shown in fig. 2.1. This example will be used as a helper in the rest of this chapter.

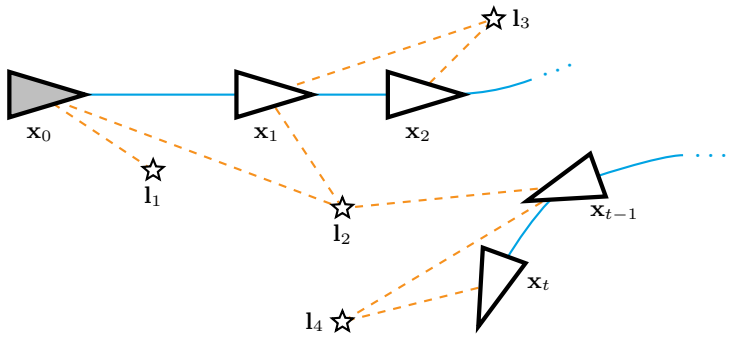


Figure 2.1: Toy-example used in the rest of this chapter. The robot moves in the world while observing some landmarks on its way. The colors in the figure show the different information available: ■ for odometry measurements, ■ for landmark measurements, and ■ for the prior information about the robot origin. The same colors will be used in following figures to show elements related with these information.

‘

2.1.3 Graphical Models

Graphical models are tools to model and visualize probabilistic relations between random variables. In general, graphical models provide a skeleton for representing the joint distribution compactly and in a factorized way [Koller 2007]. What

changes between each representation is how the joint probability is factorized, what each factor represent, and how the factors are represented in the graph.

2.1.3.1 Bayesian Networks

One of the most common types of graphical model is the *Bayesian Network* (BN), where the representation is a directed acyclic graph (DAG) $\mathcal{G}$. If $\mathcal{G}$ is a BN over the variables $X_1, \ldots, X_n$, then each node X_i represents a conditional probability distribution (CPD) $p(X_i \,|\, \mathbf{Pa}_{X_i})$ of X_i conditioned by its parents $\mathbf{Pa}_{X_i}$, or a local probabilistic distribution $p(X_i)$ (i.e. a CPD with an empty set of condition variables). Mathematically, we say that a distribution $p_{\mathcal{B}}$ over the same variable space as $\mathcal{G}$ factorizes according to $\mathcal{G}$ if $p_{\mathcal{B}}$ can be expressed as:

$$p_{\mathcal{B}}(X_1, \ldots, X_n) = \prod_{i=1}^{n} p(X_i \,|\, \mathbf{Pa}_{X_i}) \tag{2.6}$$

A BN where its variables relate to each other over adjacent time steps is called *Dynamic Bayesian Network* (DBN). For example, the joint probability of the SLAM problem can be represented by the DBN shown in fig. 2.2, modeled using eqs. (2.3) and (2.4). The factorization represented in this DBN is:

$$p(\mathbf{x}_{0:T}, \mathbf{m}, \mathbf{u}_{1:T}, \mathbf{z}_{1:T}) = p(\mathbf{x}_0) \prod_{i=1}^{T} p(\mathbf{x}_i \,|\, \mathbf{x}_{i-1}, \mathbf{u}_i) \prod_{(j,k) \,\in\, \mathcal{D}} p(\mathbf{z}_{ik} \,|\, \mathbf{x}_i, \mathbf{l}_j) \tag{2.7}$$

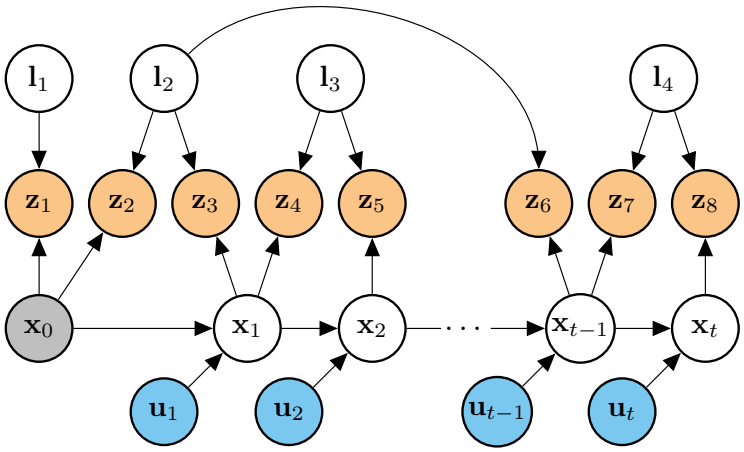


Figure 2.2: Dynamic Bayesian Network representing the toy-example of Fig. 2.1. Observed variables (data from the sensors and prior about initial state) are shown as colored nodes, and hidden variables (the ones we want to estimate) in white. This graph represents the factorization in eq. (2.7), using the models in eqs. (2.3) and (2.5).

2.1.3.2 Markov Random Fields

The second common class of probabilistic graphical models is called *Markov Random Field* (MRF). MRFs are represented by a undirected graph $\mathcal{H}$ where nodes represent variables and edges correspond to direct probabilistic interactions between neighboring variables. The graph shown in fig. 2.3 is a MRF for the DBN shown in fig. 2.2 after elimination of the observations and the initial pose.

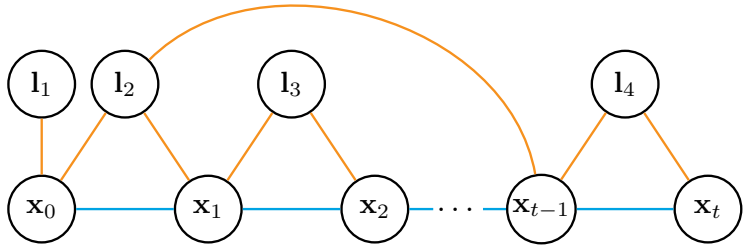


Figure 2.3: Possible Markov Random Field of the SLAM toy-example. The graph was generated from the elimination of control and measurement variables, resulting only in the robot trajectory and map. Different elimination order would generate a different MRF. Edges are loosely related with the measurements acquired by the robot, but this may be misleading. For example, the prior about the origin is represented implicitly by one of the edge potentials connecting to $\mathbf{x}_0$.

MRFs are parametrized into *factors*, which are functions of a set of random variables. This is a contrast with BNs that are parametrized by CPDs. We say that a distribution $p_{\mathcal{H}}$ factorizes over $\mathcal{H}$ if it is associated with: a set of subsets $\mathbf{D}_1, \ldots, \mathbf{D}_m$, where each $\mathbf{D}_i$ is a complete subgraph of $\mathcal{H}$; and with factors $\pi_1[\mathbf{D}_1], \ldots, \pi_m[\mathbf{D}_m]$, such that:

$$p_{\mathcal{H}}(X_1, \ldots, X_n) = \frac{1}{Z} p'_{\mathcal{H}}(X_1, \ldots, X_n) \tag{2.8}$$

where

$$p'_{\mathcal{H}}(X_1, \ldots, X_n) = \pi_1[\mathbf{D}_1] \times \cdots \times \pi_m[\mathbf{D}_m] \tag{2.9}$$

is a un-normalized measure, and where

$$Z = \sum_{X_1, \ldots, X_m} p'_{\mathcal{H}}(X_1, \ldots, X_m) \tag{2.10}$$

is a normalizing constant. Note that this definition is very close to the definition of BNs, but here the factors do not necessarily represents a probability, thus the need to normalize it using the constant Z.

2.1.3.3 Factor Graphs

Another useful graphical model for the SLAM problem, and the one most important for this thesis, is the *factor graph* representation, originally proposed by

[Kschischang 2001]. As it happens with both BN and MRF representations, the factor graphs allow a global function of several variables to be expressed as a product of factors over subsets of those variables, but factor graphs make this decomposition explicit by introducing additional nodes for the factors themselves, in addition to the nodes representing the variables [Bishop 2006, chapter 8].

If we write the joint probability of a set of random variables $\mathcal{X}$ in the form of a product of factors:

$$p(\mathcal{X}) = \prod_s f_s(\mathcal{X}_s) \tag{2.11}$$

where $\mathcal{X}_s$ is a subset of the variables in $\mathcal{X}$, the factor graph where this probability factorizes into would have each factor f_s connected to the variables $\mathcal{X}_s$, like the factor graph for the toy-example shown in fig. 2.4.

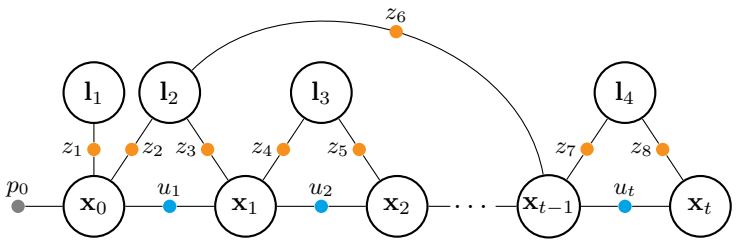
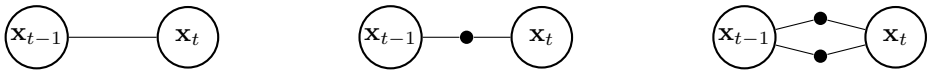


Figure 2.4: Factor graph representing the toy-example. The main difference between this graph and the MRF shown in fig. 2.3 is that all the factors are now are represented explicitly by factor nodes in the graph. For instance, now it is possible to see the prior about the origin.

The factor graph representation is more interesting for SLAM problems because it allows the visualization of the details of the problem, that are hidden in other representations. Take for example the situation of a robot that has two different odometry sensors, like for example wheel encoders and inertial odometry, and we want to model the joint probability of the robot poses along its trajectory. This could be modeled by the graphs in fig. 2.5, but it is clear that the graph shown in fig. 2.5c is more expressive than the others because it shows the contribution of each measurement separately. As we will see in section 2.2, normally when modeling the SLAM problem with factor graphs, each measurement can be related with a factor that imposes a constraint in the neighboring variables.

2.1.4 Types of SLAM problems

The SLAM problems can be distinguished according to various criteria. A taxonomy that shows several of these dimensions can be seen in [Thrun 2007]. And an other relevant taxonomy presented in [Forster 2016] casts existing contributions on visual-inertial odometry systems along three main criteria: number of camera poses, representation of uncertainty, and number of times the measurement model is linearized. Here, we concentrate on criteria relevant to this thesis.



(a) MRF (b) Single motion factor (c) Distinct motion factors

Figure 2.5: Different graph representations for a robot with multiple motion measurements. With MRFs we have only the representation in (a), while with factor graphs we could model it as a single factor (b) or isolate each measurement in a separated factor (c). The graph in (c) allows better visualization of the problem as each data is seen as one separated constraint in the graph.

2.1.4.1 *Full* versus *Online* versus *Keyframe*

One dimension to consider is related to the number of robot poses we want to estimate. The Full SLAM problem is when we are interested in estimating the maps and *all* the poses along the robot trajectory, represented by $p(\mathbf{x}_{1:t}, \mathbf{m} \mid \mathbf{o}_{1:t})$. Its counter-part is known as Online SLAM, that is when we want to estimate only the current (or most recent) pose of the robot together with the map: $p(\mathbf{x}_t, \mathbf{m} \mid \mathbf{o}_{1:t})$.

For each of the mentioned problems there is a kind of solution that is more adapted to solve it: solutions for the Full SLAM are called smoothing techniques, like graph-based optimization technique that rely on least-square minimization; while Online SLAM is better handled by filtering techniques, like Kalman filters, information filters, and particle filters. Solutions that estimate poses inside a sliding window of last poses are called fixed-lag smoothers. They may be implemented by means of a filter or optimization techniques, although optimization techniques tend to yield more precise results.

Sometimes it is also interesting to estimate only a subset of the robot poses. In the Keyframe-based SLAM one selects some poses along the robot trajectory to estimate the ones that are of some interest, named keyframes, while discarding or not estimating the others. The keyframe selection increases the distance between subsequent estimated keyframes, normally to allow better observation baseline or to avoid estimation of redundant poses. Graphs representing these SLAM specific problems and solutions are shown in fig. 2.6.

2.1.4.2 *Pose* versus *Landmark* SLAM

Another dimension of SLAM is related to the variables that are estimated. What is known as Pose SLAM is when only the poses of the robot are estimated (see graphical model in fig. 2.7). In this case the map is not explicitly part of the system.

When the map is estimated and composed of environment landmarks the problem is known as Landmark SLAM (already seen in fig. 2.4). In this case, the landmarks are features extracted from the sensor stream. A counter-part to this strategy is to use all the sensor data and compare with a volumetric map, but then the problem has a high dimension and requires much more computation. By us-

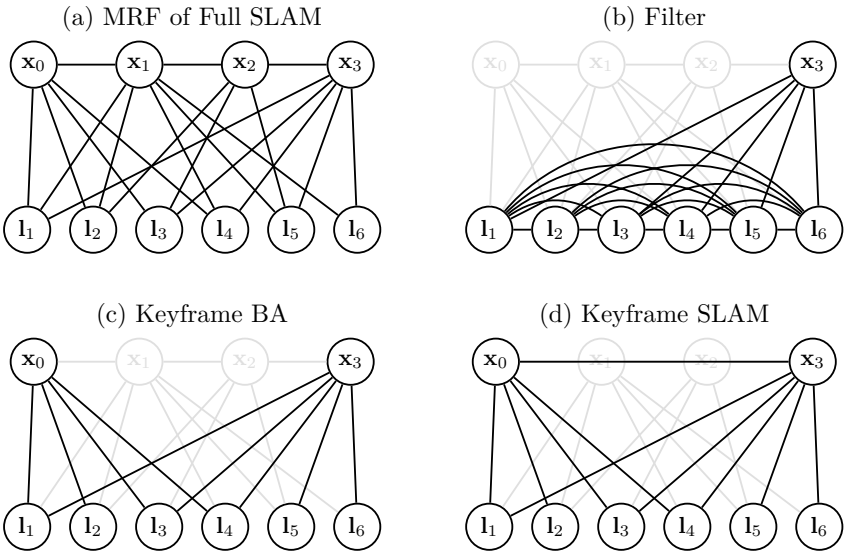


Figure 2.6: (a) Graphical model of the full SLAM problem as a Markov Random Field, after eliminating measurement variables. (b-c) shows how the inference progresses in a filter, keyframe-based optimization for Bundle Adjustment. (d) shows the inference progress in a keyframe-based optimization for Keyframe-SLAM, that is close to the BA case but with an extra edge linking the retained poses. Based on [Strasdat 2012, fig. 1].

ing only feature-landmarks as the map it is possible to be more efficient, but with the risk of having inferior results due to the discarded information in the feature extraction process.

These two SLAM types are important for this thesis because each of our main contributions deal with one of them: Landmark SLAM is addressed in chapter 3, and Pose SLAM chapter 4.

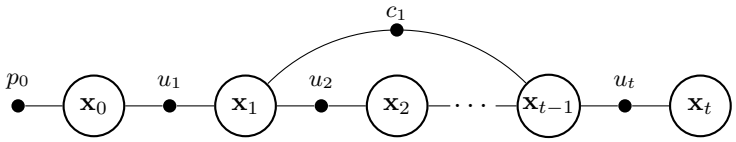


Figure 2.7: Possible factor graph of a Pose SLAM for the toy-problem. Factor $\mathbf{c}_1$ is a loop closing constraint that encodes a "virtual transformation" between the connected pose nodes. This transformation is derived from observations of the same part of the environment, that is only $\mathbf{l}_2$ in this example (see fig. 2.1), assuming that the measurements are rich enough to be used to compute such transformation (*e.g.*: enough overlap of laser scans or visual landmarks). Note that since $\mathbf{l}_2$ was also observed from $\mathbf{x}_0$ we also have the possibility to close the loop between $\mathbf{x}_{t-1}$ and $\mathbf{x}_0$ (not represented here).

2.1.4.3 Bundle Adjustment

There is a problem called *Structure from Motion* (SfM), originated from the fields of computer vision and photogrammetry, where one is interested in recovering the 3D structure of the scene along with the camera parameters (pose and intrinsic parameters), given only the images captured by the camera.

SfM is normally solved by a technique called Bundle Adjustment (BA), where the values of the points describing the scene and the camera parameters are refined simultaneously by minimizing the reprojection error between the image locations of observed and predicted image points. If all the data is available beforehand this computation can be done offline in a batch optimization, but real-time SfM, also known as visual SLAM, is also possible when the images come from a live video stream.

As it happens with SLAM solution, BA may be classified as Full, Fixed-lag or Keyframe, depending on the number of estimated poses. The key structural difference between BA and the SLAM solutions mentioned so far is that in BA there is no movement constrains between the camera poses, only the external measurements are considered. [Strasdat 2012] presents a comparison between filtering and keyframe BA in the context of visual SLAM, and shows that increasing the number of landmarks detected in the images increases accuracy, while increasing the number of intermediate keyframes has only a minor effect.

2.2 Solving the SLAM with graph-based optimization

The solutions for SLAM can be divided into two main categories: filtering and smoothing. The filtering category itself may be subdivided into the ones that use Extended Kalman Filters (EKF), information filter, or particle filters.

The online slam problem is usually solved by filtering techniques, while Full SLAM is solved by smoothing, but this may be misleading. Mathematically the state space of a filter may be increased to hold more than the last robot pose, allowing the implementation of fixed-lag, keyframe or even full SLAM solutions. In the same way, a graph optimization technique may be used to estimate only the current pose of a robot. The main reason for not doing so that is because there is no efficient way to compute the estimates of the given problem, either due to the extra computational cost (full SLAM with filters) or due to neglected benefits like the possibility to linearize more than once (online SLAM with graph-based optimization).

Recently there has been a lot of research on solving SLAM through smoothing, where the solution is computed using sparse optimization techniques. For long time the filtering techniques were seen as the way to solve SLAM, despite its drawbacks with the non-scalability regarding the map size. Since the seminal work presented in [Lu 1997], the community started to explore the sparsity of the SLAM more efficiently, with the aid of graphical models, making graph-based optimization techniques the current state-of-the-art solution for SLAM problems.

This thesis follows this trend and relies on graph-based optimization to solve SLAM. Next we will describe how we can solve SLAM using optimization based on a factor graph, mostly based on [Grisetti 2010].

2.2.1 Intuitive presentation

Before going into the mathematics of how to perform smoothing from a factor graph, I want to introduce the intuition behind it. Imagine a network of springs as the one in fig. 2.8, each spring having possible different length and stiffness. The springs are connected to by its endpoints, forming a graph. At the beginning we select the positions of the endpoints and keep then fixed. Then we release the endpoints, letting then be moved by the springs, until they are in a configuration of equilibrium.

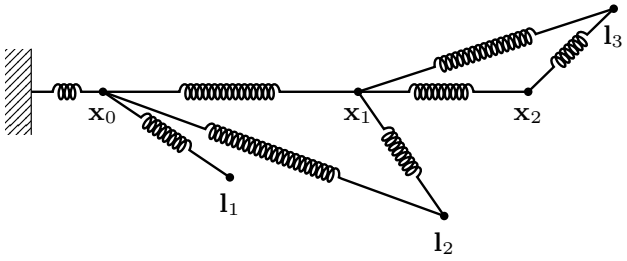


Figure 2.8: Section of the toy example abstracted as a spring network (ignore masses and gravity!). Coils represent factors that influence the variables in the graph, represented by joints and coils endpoints. Prior factors attach the variables with relation to a fixed object. Graph-based optimization may be seen as fixing the endpoints to a certain initial configuration and releasing them, letting the coils move the endpoints to a configuration of equilibrium.

This analogy is close to what happens in the graph based optimization. Each spring is a factor in the graph, and each endpoint is a variable. The different lengths and stiffness relates respectively to the measurements and covariances associated to each factor: factors with larger covariances carry less information, thus they constrain less their neighboring variables (smaller spring stiffness), while smaller covariances means more information, thus the neighboring variables will tend to respect more the constraint of this factor (higher stiffness). Prior factors can be seen as springs with one of the endpoints attached to a fixed object.

The main missing point in this simple analogy is that in graph-based optimization the relation between the variables and the error functions encoded in the factors are not always linear, as the endpoints and spring parameters. Depending on the starting configuration the variables may converge to local configurations that are not the one of minimal error (a local minimum), so it is desirable to start from a configuration close enough to the global minimum.

2.2.2 Modeling the problem as MLE

The objective of the graph-based optimization is to find a configuration for the variables that best matches the measurements encoded in the factors. If the observations are affected by Gaussian noise, this is the same as to compute a configuration that maximizes the likelihood of the observations. Note that the process is a point estimator, since only the values for the variables are estimated. This value is the mean of a Gaussian approximation over the graph variables. If necessary, the information matrix of this Gaussian can be computed once the mean is known, but it requires extra computation than what is needed to find only the mean.

Generically, each factor f_i encodes an error function that computes the difference between the expected observation and the acquired observation. For measurements in Euclidean spaces, this function has the form:

$$\mathbf{e}_{f_i} = \mathbf{o}_i - h(\mathcal{X}_i) \tag{2.12}$$

where we use $\mathbf{o}_i$ to denote the measurement encoded in f_i, and $h(\mathcal{X}_i)$ as the model that gives us the prediction of the measurement based on the set $\mathcal{X}_i$ of variables connected to the factor f_i (the same notation as in eq. (2.11)). If the measurement comes from a sensor, then $h(.)$ is the observation model of the sensor, but it may be for example a motion model if the measurements are the control inputs, or just the relative pose if the measurement is the robot displacement. This is the reason why we use $\mathbf{o}_i$ as a generic measurement, instead of specifying it as proprioceptive measurement $\mathbf{u}_i$ or exteroceptive measurement $\mathbf{z}_i$. The error function for non-Euclidean spaces is explained later in this chapter.

Let $\mathbf{\Sigma}_i$ the covariance of the measurement $\mathbf{o}_i$ and $\mathbf{\Omega}_i = \mathbf{\Sigma}_i^{-1}$ its information matrix. The likelihood of $\mathcal{X}_i$ given $\mathbf{o}_i$ encoded by f_i is:

$$\mathcal{L}_{f_i} = \mathcal{L}(\mathcal{X}_i\,;\,\mathbf{o}_i) \propto e^{-[\mathbf{o}_i - h(\mathcal{X}_i)]^\top \mathbf{\Omega}_i [\mathbf{o}_i - h(\mathcal{X}_i)]} \tag{2.13}$$

$$\propto e^{-\mathbf{e}_{f_i}^\top \mathbf{\Omega}_i \mathbf{e}_{f_i}} \tag{2.14}$$

To find the configuration $\mathcal{X}^*$ that maximizes the likelihood given all observations we need to consider all factors in the graph:

$$\mathcal{L}(\mathcal{X}) = \prod_i \mathcal{L}_{f_i} \tag{2.15}$$

and is the same to solve:

$$\mathcal{X}^* = \arg\max_{\mathcal{X}} \mathcal{L}(\mathcal{X}) \tag{2.16}$$

By using the logarithm of the likelihood function, that is monotone and has its maximum at the same point of the likelihood function, we can substitute the products of likelihoods by the sum of log likelihoods. And by negating it, we can transform

the maximization problem in a minimization problem, resulting in:

$$\mathcal{X}^* = \arg\min_{\mathcal{X}}(-\log \mathcal{L}(\mathcal{X})) \tag{2.17}$$

$$= \arg\min_{\mathcal{X}} \sum_i -\log \mathcal{L}_{f_i} \tag{2.18}$$

$$= \arg\min_{\mathcal{X}} \sum_i \mathbf{e}_{f_i}^\top \mathbf{\Omega}_i \mathbf{e}_{f_i} \tag{2.19}$$

2.2.3 Solving the minimization problem

If a good initial guess $\breve{\mathcal{X}}$ is available, eq. (2.19) can be solved by Gauss-Newton (GN) or Levenberg-Marquardt (LM) algorithms. For that, we start by approximating the error function of each factor by the first order Taylor expansion around the current initial guess:

$$\mathbf{e}_{f_i}(\breve{\mathcal{X}}_i + \Delta\mathcal{X}_i) = \mathbf{e}_{f_i}(\breve{\mathcal{X}} + \Delta\mathcal{X}) \tag{2.20}$$

$$\approx \mathbf{e}_{f_i} + \mathbf{J}_i \Delta\mathcal{X} \tag{2.21}$$

where $\mathbf{e}_{f_i} \triangleq \mathbf{e}_{f_i}(\breve{\mathcal{X}})$ is the error of the factor given the initial guess (the simplification of the notation will be handy in the equations below), $\mathbf{J}_i$ is the Jacobian of $\mathbf{e}_{f_i}(\mathcal{X})$ computed at $\breve{\mathcal{X}}$, and $\Delta\mathcal{X}$ is a small increment on the variables. Note that in eq. (2.20) we changed the domain of the error function of a factor f_i from its neighboring variables $\mathcal{X}_i$ to all variables $\mathcal{X}$, even if effectively only the neighboring variables are used to compute the error. This was done so that the Jacobian $\mathbf{J}_i$ can be computed with relation to all variables: the reason for this will become clear when we discuss the structure of the matrices used in the optimization problem.

By substituting eq. (2.21) in the terms of the sum in eq. (2.18) we get:

$$-\log \mathcal{L}_{f_i}(\breve{\mathcal{X}} + \Delta\mathcal{X}\,;\,\mathbf{o}_i) = \mathbf{e}_{f_i}(\breve{\mathcal{X}} + \Delta\mathcal{X})^\top \mathbf{\Omega}_i \mathbf{e}_{f_i}(\breve{\mathcal{X}} + \Delta\mathcal{X}) \tag{2.22}$$

$$\approx (\mathbf{e}_{f_i} + \mathbf{J}_i\Delta\mathcal{X})^\top \mathbf{\Omega}_i (\mathbf{e}_{f_i} + \mathbf{J}_i\Delta\mathcal{X}) \tag{2.23}$$

$$= \underbrace{\mathbf{e}_{f_i}^\top \mathbf{\Omega}_i \mathbf{e}_{f_i}}_{c_i} + 2\underbrace{\mathbf{e}_{f_i}^\top \mathbf{\Omega}_i \mathbf{J}_i}_{\mathbf{b}_i}\Delta\mathcal{X} + \Delta\mathcal{X}^\top \underbrace{\mathbf{J}_i^\top \mathbf{\Omega}_i \mathbf{J}_i}_{\mathbf{H}_i}\Delta\mathcal{X} \tag{2.24}$$

$$= c_i + 2\mathbf{b}_i\Delta\mathcal{X} + \Delta\mathcal{X}^\top \mathbf{H}_i \Delta\mathcal{X} \tag{2.25}$$

that is the local approximation of the likelihood for one observation. To approximate for all observations, we substitute eq. (2.25) in the sum being minimized in eq. (2.19)

to get:

$$\sum_i \mathbf{e}_{f_i}^\top \mathbf{\Omega}_i \mathbf{e}_{f_i} = \sum_i \left(c_i + 2\mathbf{b}_i \Delta\mathcal{X} + \Delta\mathcal{X}^\top \mathbf{H}_i \Delta\mathcal{X} \right) \tag{2.26}$$

$$= \underbrace{\sum_i c_i}_{c} + 2 \underbrace{\left(\sum_i \mathbf{b}_i \right)}_{\mathbf{b}^\top} \Delta\mathcal{X} + \Delta\mathcal{X}^\top \underbrace{\left(\sum_i \mathbf{H}_i \right)}_{\mathbf{H}} \Delta\mathcal{X} \tag{2.27}$$

$$= c + 2\mathbf{b}^\top \Delta\mathcal{X} + \Delta\mathcal{X}^\top \mathbf{H} \Delta\mathcal{X} \tag{2.28}$$

and finally, by differentiating the quadratic form in eq. (2.28) and making it equal to zero, we get the following linear system:

$$\frac{d}{d\Delta\mathcal{X}} \left(c + 2\mathbf{b}^\top \Delta\mathcal{X} + \Delta\mathcal{X}^\top \mathbf{H} \Delta\mathcal{X} \right) = \mathbf{0} \tag{2.29}$$

$$2\mathbf{b}^\top + 2\Delta\mathcal{X}^\top \mathbf{H} = \mathbf{0} \tag{2.30}$$

$$\mathbf{H}\Delta\mathcal{X} = -\mathbf{b} \tag{2.31}$$

that can be solved to find $\Delta\mathcal{X}^*$ that minimizes eq. (2.19) in $\Delta\mathcal{X}$. The linearized solution can be obtained by adding the computed increments to the initial guess:

$$\mathcal{X}^* = \breve{\mathcal{X}} + \Delta\mathcal{X}^* \tag{2.32}$$

Algorithms like GN and LM normally iterate between linearizing the system at the current solution with eq. (2.28), solving the linearized system for the increment with eq. (2.31), updating the solution with increment through eq. (2.32), and repeating the process using the new solution as the linearization point for the next iteration. This is done until convergence.

2.2.4 Matrix structures

The matrices $\mathbf{b}$ and $\mathbf{H}$ in the linear system shown in eq. (2.31) are sparse, thus the system may be efficiently solved by sparse techniques, like Cholesky sparse factorization. The sparsity of these matrices originates from the use of the Jacobians $\mathbf{J}_i$ in their construction (see eq. (2.24)). In fact, the error from each factor depends only on the variables connected to the factor, thus the Jacobian of the error *with relation to all variables* in the graph will have an structure where only the blocks related to the factor variables will have information, and it will be zero everywhere else. Through the sums performed in eq. (2.27), each factor produces contributions in the matrices $\mathbf{b}$ and $\mathbf{H}$ that are localized as well. In the end, one only need to iterate through the factors, computing these contribution for each factor and adding them in the right places in the both system matrices. This can be visualized in fig. 2.9.

The matrix $\mathbf{b}$ represents the differences between the expected measurements and the real measurements for each factor, weighted by the measurement errors in the respective factors (in the information form), and projected in the variable

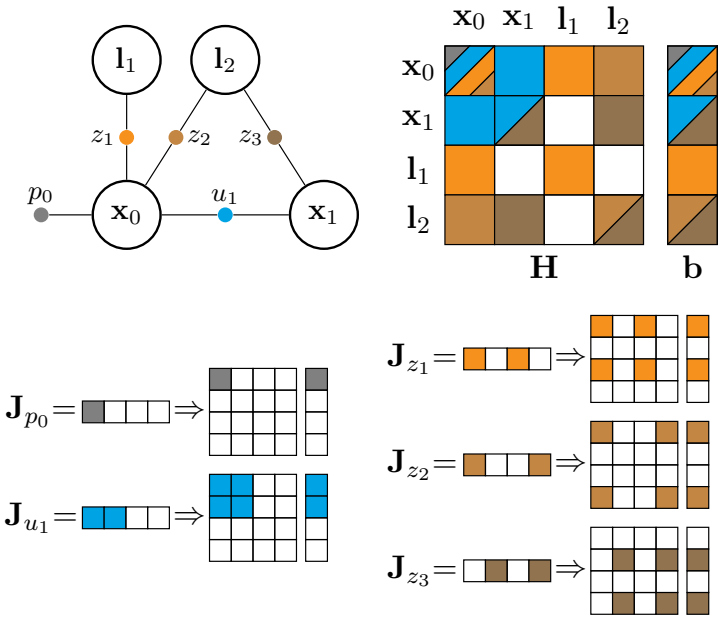


Figure 2.9: Example of the construction of the linearized system matrices from the factor graph. Different tones of the same color were used to differ each landmark observation factor. **Top-left**: small section of the factor graph for the toy example. **Bottom**: structure of the Jacobian matrices computed for each factor, and the resulting contributions in the matrices $\mathbf{H}$ and $\mathbf{b}$ of the linearized system. **Top-right**: resulting system matrices colored by the contributions from each factor.

space. The matrix $\mathbf{H}$ is the information matrix of the system, that projects all the measurement errors to the space of the variables. Since it is a symmetric matrix, an additional optimization is to compute only its upper (or lower) triangular part. For graphs where the measurements can be represented by Euclidean spaces, the final covariance on the variables can be obtained by inverting this matrix.

2.2.5 On the need of priors

A certain care should be taken to avoid building graphs with relative constraints *only*. In such cases, the function being minimized in eq. (2.19) may present some invariance (*e.g.*: a pose graph with only relative constraints is invariant under a rigid transformation of all poses). Prior factors can be added to avoid such problems. It is common to add a prior factor at the origin with a very small covariance, although setting the error at the origin to zero is not advisable since this is "too much informative" (zero covariance maps to infinity in the information form) and leads to numerical instability.

2.2.6 Dealing with manifolds

The approach explained above does not apply directly to the SLAM problem, in which the space for some variables may be non-Euclidean. For example, the orientation of the robot poses span over non-Euclidean spaces, like the group $SO(2)$ or $SO(3)$ for 2D and 3D rotations. If we apply eq. (2.32) for these variables, we may break the rules of the over-parametrization. This can be avoided by using a minimal representation for these variables, but then it would be subjected to singularities avoided by the over-parametrization.

One solution is to consider the variable space as a manifold and define an operator $\boxplus$ that maps the variation to a locally Euclidean tangent space to the manifold: $\Delta\mathcal{X} \mapsto \mathcal{X} \boxplus \Delta\mathcal{X}$. A new error function can be defined and linearized as:

$$\mathbf{e}_{f_i}(\breve{\mathcal{X}}_i \boxplus \Delta\tilde{\mathcal{X}}_i) = \mathbf{e}_{f_i}(\breve{\mathcal{X}} \boxplus \Delta\tilde{\mathcal{X}}) \tag{2.33}$$

$$\approx \mathbf{e}_{f_i} + \tilde{\mathbf{J}}_i \Delta\tilde{\mathcal{X}} \tag{2.34}$$

where we use $\tilde{(\cdot)}$ to distinguish from the previous variables in the non-manifold equations. The Jacobian $\tilde{\mathbf{J}}_i$ can be expressed by:

$$\tilde{\mathbf{J}}_i = \left.\frac{\partial \mathbf{e}_{f_i}(\breve{\mathcal{X}} \boxplus \Delta\tilde{\mathcal{X}})}{\partial \Delta\tilde{\mathcal{X}}}\right|_{\Delta\tilde{\mathcal{X}}=\mathbf{0}} \tag{2.35}$$

Since $\mathbf{e}_{f_i}$ depends only on $\mathcal{X}_i \subset \mathcal{X}$, and using the chain rule, we can write:

$$\frac{\partial \mathbf{e}_{f_i}(\breve{\mathcal{X}} \boxplus \Delta\tilde{\mathcal{X}})}{\partial \Delta\tilde{\mathcal{X}}} = \underbrace{\frac{\partial \mathbf{e}_{f_i}(\breve{\mathcal{X}})}{\partial \breve{\mathcal{X}}}}_{\mathbf{J}_i} \cdot \underbrace{\left.\frac{\breve{\mathcal{X}} \boxplus \Delta\tilde{\mathcal{X}}}{\partial \Delta\tilde{\mathcal{X}}}\right|_{\Delta\tilde{\mathcal{X}}=\mathbf{0}}}_{\mathbf{M}} \tag{2.36}$$

The term $\mathbf{J}_i$ in eq. (2.36) is the same as previously discussed for the Euclidean spaces. Since it has only a few block structures with non-zero values, the Jacobian $\tilde{\mathbf{J}}_i$ may be computed by multiplying only the non-zero blocks of $\mathbf{J}_i$ by the appropriate block of $\mathbf{M}$.

In a similar way, eq. (2.34) can be used in eq. (2.23) to produce the linear system:

$$\tilde{\mathbf{H}} \Delta\tilde{\mathcal{X}} = -\tilde{\mathbf{b}} \tag{2.37}$$

which is again sparse by construction, and may be solved using efficient sparse techniques. Once the system is solved for the increment in the local tangent space, the solution in the manifold can be updated using:

$$\mathcal{X} = \breve{\mathcal{X}}^* \boxplus \Delta\tilde{\mathcal{X}}^* \tag{2.38}$$

The final solution is then found by iterating the approximating with eq. (2.34), solving the system in eq. (2.37) and accumulating the increments in the manifold space using eq. (2.38). Also, note that $\mathbf{M}$ can be computed once for the current linearization point and have its blocks reused to compute each $\tilde{\mathbf{J}}_i$.

Note that the $\tilde{\mathbf{H}}$ now represents the information matrix in the tangent space $\Delta\tilde{\mathcal{X}}$, and not over the graph variables. If necessary, the covariance matrix over the graph variables may be computed once the configuration for the maximum likelihood is found through the projection of measurement errors through the Jacobians $\mathbf{J}_i$.

2.3 Insights on SLAM from graphical models

As we have seen, modeling SLAM as a graph-based optimization problem provides us with efficient ways to solve it. The graphical model analyze may also bring other insights, that may lead to innovative solutions to SLAM. Here I would like to discuss one of such interesting solution.

ISAM2

The ISAM2 algorithm was presented in [Kaess 2012]. It exploits graphical models to propose an incremental alternative to the linearize-solve-update strategy commonly used to find the solution.

Solving the linear system in eq. (2.37) usually involves factorization of the system matrix. It was shown in [Kaess 2012] that the factorization is the same as converting the factor graph into a Bayes network using the variable elimination order. Furthermore, the Bayes network can be converted into a novel structure, called Bayes Tree, that is a directed tree whose nodes represent cliques on the underlying chordal Bayes net. Nodes in the Bayes Tree can be mapped to blocks of the matrix resulting from the elimination. Figure 2.10 show an example of graphical models and associated sparse matrices.

In ISAM2 the Bayes Tree is used to store the square root information matrix used to find the update increments. The building of the tree is associated with a forward substitution where the information goes from the leaves to the root (conditioning of the node variables on the parent node variables), the back substitution that retrieves the solution is associated with passing the information from the root to the leaves (computing the value of the variable given the parents values).

When new factors are added to the graph, it is shown that it only causes local changes in the tree. More specifically, only the section from the nodes containing variables affected by new factors up to the root of the tree changes, and all the remaining sub-trees remains the same. Thus, when incrementally adding factors to the graph can be handled efficiently by re-building only a small portion of the tree, unless it is a loop-closing factor, where in this case a larger section of the tree needs to be re-built. The sub-trees that are not affected are removed from the Bayes Tree during the rebuilding and re-attached once the rebuilding is done.

ISAM2 also provides an strategy to control the linearization of the system, named *fluid relinearization*. It consists into track the validity of the linearization point for each variable separately, and only re-linearize the factors that connect to a variable whose linearization point moved more than a certain threshold (named β in the original paper). The linearization point of a variable that is not anymore valid

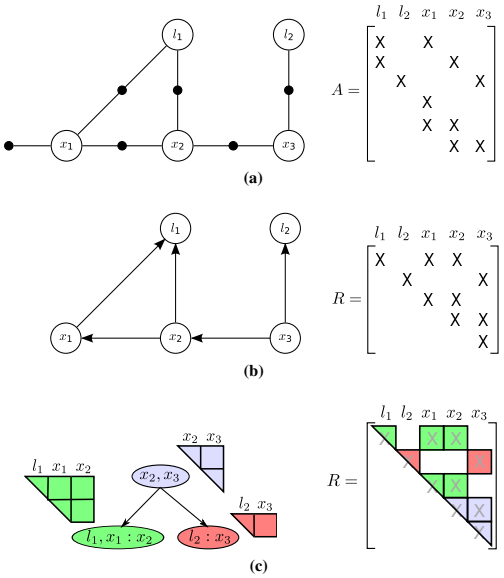


Figure 2.10: Graphical models and associated matrices, extracted from [Kaess 2012]. (a) Factor graph and Jacobian matrix A (the concatenation of Jacobians of all factors). (b) Bayes network and associated square root information matrix R resulting from the elimination of variables in the order l_1, l_2, x_1, x_2, x_3. (c) The Bayes Tree and the associated square root information matrix R. Matrix blocks encoded by the cliques are color-coded.

gets updated, and the nodes containing this variable and the ones up to the root will need to be relinearized (this is because the information about the linearization point goes up to the root of the tree). To be more efficient, the tree is rebuild only once per update, for both variables with non-valid linearization points and for variables affected by new factors.

The resulting Bayes Tree depends on the variable elimination order, and finding an elimination order that minimizes the fill-in is essential for a good performance. In ISAM2 an incremental variable ordering is used, where only the variables in the section of the tree being rebuild may be re-ordered, in a way to minimize the fill-in locally. While it causes more fill-in than if all variables were considered, authors claim that the incremental ordering still provide good solutions. For the ordering, the *constrained column approximate minimum degree* (CCOLAMD) algorithm is used: it is a COLAMD with the extra constraint to keep the variables affected by the new factors at the end of the elimination order, and thus closer to the root of the tree. The addition of this extra constraint is justified by the fact that the next factors being added are more likely to be connected to the last accessed variables (last pose or last seen landmarks). By putting the next accessed variables close to the root, the next incremental updated have a higher chance to affect a smaller region of the tree, speeding up its incremental reconstruction.

Finally, ISAM2 performs *partial updates*. The idea is that updates in variables that are local to the current robot position does not need to be propagated to

variables that are very far back in the robot trajectory. Since the last accessed variables are positioned close to the root, a threshold is used to control if updates should be propagated further variables, that are localized below in the tree: the update proceeds if changes in the variables of the current cluster are larger than the threshold, named α in the original paper.

The ISAM2 algorithm is a interesting example of how modeling the problem as a graphical model can bring insights that leads to the derivation of innovative and efficient algorithms. This algorithm was used as the optimization technique for the work presented in chapter 3.

2.4 Summary

This chapter presented SLAM as a probabilistic inference problem. The two problem main elements, namely the front-end and the back-end, were discussed, together with their respective relation with the data-association problem and the estimation of a solution. The main graphical models, Bayes networks, Markov random fields and factor graphs, were presented, using the same typical SLAM example as a guideline.

The typical way to solve the SLAM as a graph-based optimization was also presented, showing how the graph structure influence the matrices used to find the solution. Nevertheless, this is not the unique solution, and an alternative named ISAM2 was briefly presented, showing how important the insights brought by the graphical modeling are for solving the SLAM problem efficiently .

The focus in this chapter on the graph-based solution for SLAM is justified by the contributions in the next two chapters being based on such type of solution. Chapter 3 deals with the parametrization of landmarks for an incremental monocular visual-SLAM using ISAM2, and chapter 4 deals with Pose-SLAM using data from Lidar sensors, with constraints derived from a point cloud alignment algorithm.

Chapter 3

Incremental Parallax Angle Parametrization

Contents

3.1 Introduction

In this chapter we are dealing with the monocular SLAM problem, that is when the sole exteroceptive sensor used is a single camera. This problem is important because cameras are the smallest, lightest and the most power efficient exteroceptive sensors that can be used on-board robots. The possibility to detect and track environment features detected in images during motion make them a solution of choice for the SLAM problem.

Image data carry plenty of information about the world, for both close and far objects, but with the drawback that the depth information is not perceived because

of the perspective projection. This challenges the estimation, all the more when the camera is facing towards the main direction of motion. For instance, the cameras exploited for localization on board micro-UAVs are always oriented downwards (see *e.g.* [Weiss 2011]), so as to yield a good observability of depth, the UAVs evolving mostly in horizontal directions. On the contrary, autonomous ground robots are usually equipped with forward looking cameras, or endowing an UAV with a visual sensing and avoidance capacity requires the use of forwards facing cameras. In open areas, landmarks that lie far ahead are good reference point to correct orientation errors, but hardly provide information about translations. On the other hand, close landmarks can be used to correct translation errors *if* their depth from the image is known. But this information is not available from a single image and may not be easily recoverable when the robot is moving along the camera focal axis.

To overcome the problem of partial landmark observations in images, alternative ways to represent a point in space were proposed, other than the Cartesian coordinates in a 3D Euclidean space. Several different parametrizations for monocular SLAM are compared in [Solà 2012]. Among them, the *inverse depth parametrization* (IDP) [Civera 2008], is one of the most well known parametrizations for point landmarks, and has been applied successfully together with filtering techniques. In a nutshell, a landmark in IDP is represented by the direction and the distance (or depth) to a position in space from where it was seen. This position is called its *anchor*. Moreover, what is stored is the inverse of the distance (or inverse of the depth), which makes the handling of very distant landmarks numerically stable. This way, the unobserved dimension is isolated in one parameter, the inverse depth, instead of being spread into several parameters. The large uncertainty in this parameter can be well handled in a SLAM solution based on filtering.

In the context of the Bundle Adjustment problem, [Zhao 2011] propose the *parallax angle parametrization* (PAP), which constitutes an improvement over IDP, especially for features lying in the direction of the camera motion. PAP basically extends the concept of IDP by using two anchors instead of one, and by encoding the missing dimension in the *parallax angle* formed from vectors going from the anchors towards the 3D point, and in the distance between the anchors. The authors show that this parametrization avoids ill-conditioned equations, yields fast convergence due to the objective function having no small gradients with respect to the introduced parameters to describe the features location, and handles well the forward looking camera scenario.

The PAP presents a good potential for bundle adjustment, where the only measurements available are the camera images and all the processing is done offline and in batch. This potential motivates the study of how to use the PAP in incremental SLAM, in the same challenging scenario of a forward-looking camera. This chapter presents this study to the reader, showing the mechanics needed to use PAP in a incremental monocular SLAM solution, modeled using the theory of factor graphs that was introduced in chapter 2.

This chapter is structured as follows:

- First we present the basics of perspective camera models used in visual SLAM
- We then review the various visual point landmark parametrizations, focusing the specific PAP parametrization.
- Then we show how the PAP equations reflect in different factors that are used in the graph, and how to build and update the graph when new measurements arrive. This is the main contribution of the chapter, that ends with the definition of an algorithm for incremental SLAM using the PAP parametrization.
- Finally, we analyze results obtained from experiments with simulated datasets.

3.2 Basic notions of computer vision

The subject in this chapter is visual SLAM, so it is interesting to start presenting some fundamentals about reference frames and the perspective projection model that are used throughout the chapter.

3.2.1 Reference frames

The reference frames used in this work are defined to be *right-handed*. The position and orientation of a frame $\mathcal{F}$ relative to another frame $\mathcal{W}$ is specified by a translation vector $\mathbf{t} \in \mathbb{R}^3$ and a rotation matrix $\mathbf{R} \in SO(3)$. When appropriate, the orientation of a frame is represented by the quaternion $\mathbf{q} \in \mathbb{R}^4$ that is equivalent to the rotation matrix. The rotation matrix representation of a quaternion is written $\mathbf{R}(\mathbf{q})$.

We use the same notation as [Solà 2007] to indicate the alignment of the frames: a 3D frame $\mathcal{F}\{FLU\}$ has its x-axis pointing *forward*, its y-axis pointing *to the left*, and its z-axis pointing *upwards*. Similarly a frame $\mathcal{G}\{RDF\}$ has a *right-down-forward* alignment. The same notation is also used for 2D frames. For instance, $\mathcal{I}\{RD\}$ defines a 2D *right-down* frame. Frames representing the robot or the robot trajectory have the FLU alignment. The alignment notation is important to distinguish the camera frame from the sensor frame and from the image frame.

There are two operations that may be performed using frames. We can transform a point $\mathbf{p}^{\mathcal{F}}$, expressed in the frame $\mathcal{F}$, to a frame $\mathcal{W}$ with:

$$\mathbf{p}^{\mathcal{W}} = \mathbf{R}\mathbf{p}^{\mathcal{F}} + \mathbf{t} \tag{3.1}$$

and the opposite operation that transform a point from the frame $\mathcal{W}$ to the frame $\mathcal{F}$ is:

$$\mathbf{p}^{\mathcal{F}} = \mathbf{R}^{\top}\mathbf{p}^{\mathcal{W}} - \mathbf{R}^{\top}\mathbf{t} \tag{3.2}$$

The frame transformations operations can be written in homogeneous form. Let the homogeneous representation of a point $\mathbf{p}$ be:

$$\underline{\mathbf{p}} \triangleq \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \in \mathbb{R}^{n+1} \tag{3.3}$$

then we can rewrite the operations in eqs. (3.1) and (3.2) respectively as:

$$\begin{bmatrix} \mathbf{p}^{\mathcal{W}} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}^{\mathcal{F}} \\ 1 \end{bmatrix} \tag{3.4}$$

$$\begin{bmatrix} \mathbf{p}^{\mathcal{F}} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}^{\top} & -\mathbf{R}^{\top}\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}^{\mathcal{W}} \\ 1 \end{bmatrix} \tag{3.5}$$

This representation allow us to write the operations in compact form:

$$\underline{\mathbf{p}}^{\mathcal{W}} = \mathbf{H}^{\mathcal{W}\mathcal{F}}\underline{\mathbf{p}}^{\mathcal{F}} \tag{3.6}$$

$$\underline{\mathbf{p}}^{\mathcal{F}} = \mathbf{H}^{\mathcal{F}\mathcal{W}}\underline{\mathbf{p}}^{\mathcal{W}} \tag{3.7}$$

with

$$\mathbf{H}^{\mathcal{W}\mathcal{F}} \triangleq \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}; \qquad \mathbf{H}^{\mathcal{F}\mathcal{W}} \triangleq \begin{bmatrix} \mathbf{R}^{\top} & -\mathbf{R}^{\top}\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.8}$$

being known as *homogeneous matrices*. These matrices satisfy $\mathbf{H}^{\mathcal{W}\mathcal{F}} = (\mathbf{H}^{\mathcal{F}\mathcal{W}})^{-1}$. The advantage of writing the frame operations with homogeneous matrices is to perform *frame compositions*. For example, if we have the frame $\mathcal{F}^{\mathcal{W}}$ ($\mathcal{F}$ expressed in $\mathcal{W}$) and $\mathcal{C}^{\mathcal{F}}$ ($\mathcal{C}$ expressed in $\mathcal{F}$), we can convert a point $\mathbf{p}^{\mathcal{C}}$ to $\mathbf{p}^{\mathcal{W}}$ with:

$$\underline{\mathbf{p}}^{\mathcal{W}} = \mathbf{H}^{\mathcal{W}\mathcal{C}}\underline{\mathbf{p}}^{\mathcal{C}} = \mathbf{H}^{\mathcal{W}\mathcal{F}}\mathbf{H}^{\mathcal{F}\mathcal{C}}\underline{\mathbf{p}}^{\mathcal{C}} \tag{3.9}$$

3.2.2 Perspective projection model

The pin-hole camera model is shown in fig. 3.1. By applying triangle similarities, the point $\mathbf{p}^{\mathcal{S}} = \begin{bmatrix} X & Y & Z \end{bmatrix}^{\top}$ can be projected to the point $\mathbf{p}^{\mathcal{I}}$ with:

$$\mathbf{p}^{\mathcal{I}} = \begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{3.10}$$

with f being the focal distance. The same relation may be written using matrices and homogeneous coordinates as:

$$\underline{\mathbf{p}}^{\mathcal{I}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \propto s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.11}$$

where the relation $\propto$ means *proportional to*, and s is a scalar.

The point in the image frame $\mathcal{I}$ may further be converted to image pixel coor-

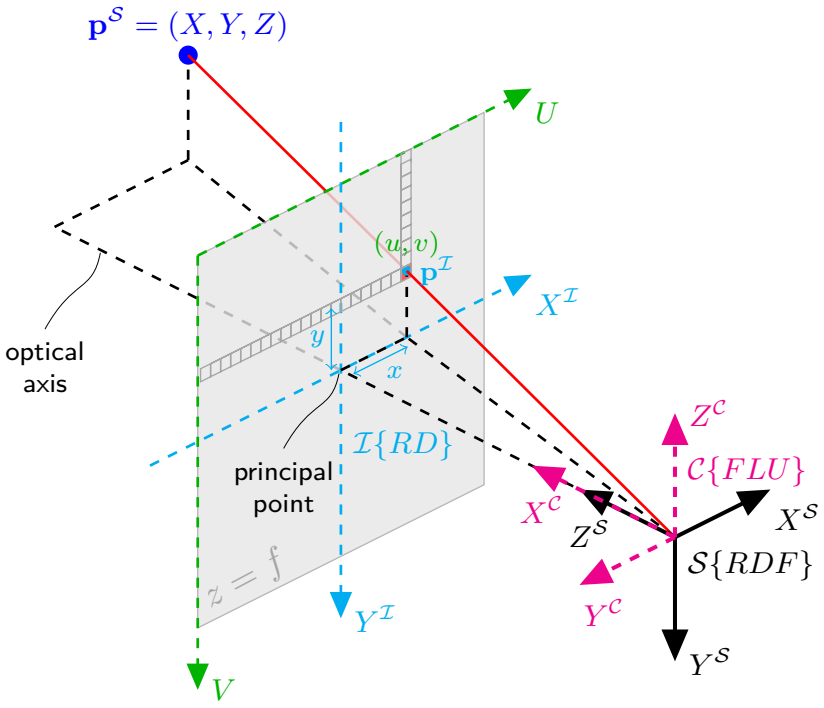


Figure 3.1: The pin-hole camera model. Three frames are defined: the camera body frame $\mathcal{C}$ with its x-axis pointing forward; the sensor frame $\mathcal{S}$ with its z-axis pointing forward; and the 2D image frame $\mathcal{I}$ that follows the *right-down* alignment. The image is represented by the gray rectangle, positioned at the focal distance f. Pixel coordinates (u, v) are expressed relative to the upper-left corner of the image.

dinates with:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} s_u & s_\theta & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.12}$$

Given all these conversions, we can write the conversion from a point $\mathbf{p}^{\mathcal{S}}$ to the image coordinates as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \propto \begin{bmatrix} s_u & s_\theta & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.13}$$

This can be further decomposed as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \propto \begin{bmatrix} s_u & s_\theta & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.14}$$

and its members being defined as:

$$\mathbf{K}_s \triangleq \begin{bmatrix} s_u & s_\theta & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}; \qquad \mathbf{K}_f \triangleq \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}; \qquad \mathbf{P}_0 \triangleq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{3.15}$$

The matrix $\mathbf{P}_0$ is usually referred as the *normalized projection matrix*. The matrices $\mathbf{K}_s$ and $\mathbf{K}_f$ define the intrinsic camera parameters. They normally appear together in a matrix $\mathbf{K}$, named *intrinsic matrix*, and defined as:

$$\mathbf{K} \triangleq \mathbf{K}_s\mathbf{K}_f = \begin{bmatrix} fs_u & fs_\theta & u_0 \\ 0 & fs_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & \alpha_\theta & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.16}$$

For most cameras the images can be considered squared, thus the *shear component* s_θ is usually zero, and consequently α_θ becomes zero as well. With this decomposition we can write the same conversion in the following compact form:

$$\underline{u} \propto \mathbf{K}\mathbf{P}_0\underline{\mathbf{p}}^{\mathcal{S}} \tag{3.17}$$

3.2.3 Inverse perspective projection model

The point $\mathbf{p}^{\mathcal{S}}$ can be recovered from the image pixel coordinates *up to a scale*. In other words, the depth of the point should be defined in advance because the depth information is lost in the projection of the point to the image frame. The depth is the free parameter s in the following equation:

$$\mathbf{p}^{\mathcal{S}} = s\mathbf{K}^{-1}\underline{\mathbf{u}} \tag{3.18}$$

The point may then be converted to other frames if desired, using the reference frame equations introduced in section 3.2.1.

3.2.4 Camera body frame and sensor frame

One important remark to be done about the perspective projection model is related to the different frames used in the model. As can be seen in fig. 3.1, the model has three frames: the body camera frame $\mathcal{C}\{FLU\}$, the sensor frame $\mathcal{S}\{RDF\}$, and the image frame $\mathcal{I}\{RD\}$, being this last one internal to the camera model. $\mathcal{C}$ is used to facilitate the specification of camera position with relation to the robot frame $\mathcal{R}\{FLU\}$. The homogeneous matrix that transforms a point in $\mathcal{S}$ to $\mathcal{C}$ is:

$$\mathbf{H}^{\mathcal{CS}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

Often in the parametrizations that will be presented next, 3D points are repre-

sented in an arbitrary frame $\mathcal{F}$. Thus to project the point it is necessary to convert it to $\mathcal{S}$ frame first. Supposing that the transformation $\mathbf{H}^{\mathcal{FC}}$ is known, this can be done by:

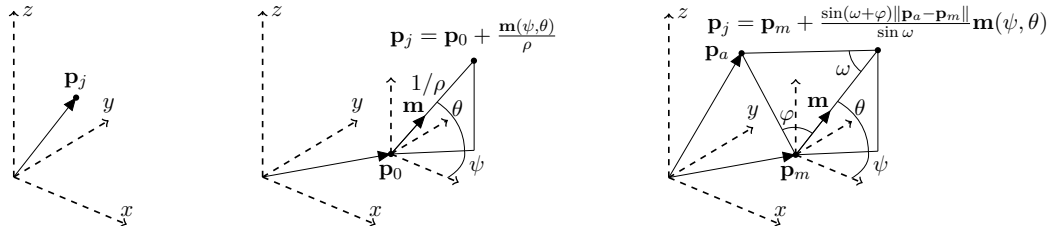
$$\mathbf{H}^{\mathcal{SF}} = \mathbf{H}^{\mathcal{SC}}\mathbf{H}^{\mathcal{CF}} = \left(\mathbf{H}^{\mathcal{CS}}\right)^{-1}\left(\mathbf{H}^{\mathcal{FC}}\right)^{-1} \tag{3.20}$$

3.2.5 About lens distortion

The reader may notice that we did not mention the possible effects of distortion caused by the lenses in the camera, even though this has to be taken into account into most vision systems. If considered, the lens distortion effect should be dealt in the projection model after projection of the point to the image plane and before conversion to image coordinates. Respectively, it should be dealt in the inverse projection model after the conversion from image coordinates to the image frame and before converting to the sensor frame.

In this thesis we worked direct with simulated data and rectified images from datasets, thus we are not developing further into the distortion subject – which does not mean it can be neglected: interested readers may refer to [Solà 2007] for the a in-depth discussion on this matter.

3.3 Landmark Parametrizations



(a) Euclidean (b) Inverse Depth (c) Parallax Angle

Figure 3.2: Three parametrizations for a point $\mathbf{p}_j$: (a) An Euclidean point is parametrized by its coordinates in a Cartesian frame; (b) An inverse depth point is parametrized by one anchor $\mathbf{p}_0$, two direction angles ψ and θ, and the inverse of the distance between p_0 and the point, named ρ. Note that the direction vector $\mathbf{m}$ is a function of the angles ϕ and θ; (c) A parallax angle point is parametrized by two anchors $\mathbf{p}_m$ and $\mathbf{p}_a$, the same two direction angles from inverse depth point, plus the parallax angle ω. The angle φ is an implicit parameter, being computed from the anchors and direction angles using eq. (3.34).

The simplest way to represent a 3D-point is by its Cartesian coordinates relative to some reference frame:

$$\mathbf{l}_{\text{EP}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.21}$$

where we use $\mathbf{l}$ with the meaning of *landmark parametrization*, and the subscript

as the name of the parametrization. In this case, $\mathbf{l}_{\text{EP}}$ means that the landmarks is expressed in *Euclidean* parametrization. It can be visualized in fig. 3.2a. The observation of this landmark in a camera $\mathcal{C}$ can be directly computed with the equations from section 3.2.2. If the landmark is expressed in the world frame $\mathcal{W}$, then we have:

$$\underline{\mathbf{u}} \propto \mathbf{K}\mathbf{P}_0\mathbf{H}^{\mathcal{SW}}\underline{\mathbf{l}}_{\text{EP}} \tag{3.22}$$

with:

$$\mathbf{H}^{\mathcal{SW}} = \mathbf{H}^{\mathcal{SC}}\mathbf{H}^{\mathcal{CW}}$$

As seen in section 3.2.3, the landmark can be initialized from one measurement *up to a scale* by eq. (3.20). In this equation we notice that the scale appears as a multiplicative factor in all parameters of the landmark representation. Thus, if we chose different scales, all parameters of the landmark will change. In other words, *the scale is encoded into all parameters of the Euclidean representation of a landmark*.

The real depth can be recovered by triangulation from two image measurements of the same point landmark. Due to noise in these measurements, a good baseline between the cameras used to acquire the measurements is necessary (either two different cameras, like in a stereo rig, or by displacing a single camera and acquiring the images at different times). In any case, some error in the depth estimation is considered normal, and this error affects all parameters of $\mathbf{l}_{\text{EP}}$.

In incremental systems, new measurements for a landmark may trigger updates in the depth estimation, especially when the baseline between the measurements is still small with relation to the true depth of the point. This results in strong nonlinear updates in all landmark parameters, and makes it difficult to produce proper estimates using linearization techniques.

3.3.1 Inverse Depth

Another way to parametrize point landmarks is through the inverse depth representation that can be seen in fig. 3.2b. The point $\mathbf{p}$ is represented in inverse depth by a point $\mathbf{p}_0$, the direction angles ψ and ω, and by the inverse depth ρ:

$$\mathbf{l}_{\text{ID}} = \begin{bmatrix} \mathbf{p}_0 \\ \psi \\ \theta \\ \rho \end{bmatrix} \tag{3.23}$$

The point $\mathbf{p}_0$ is the landmark's *anchor*, and is chosen to be one of the camera positions from where the point was seen (normally from the first time it was seen, but any of the camera positions can be used). The angles ψ and ω encode a direction vector $\mathbf{m}$ relative to a frame attached to the anchor $\mathbf{p}_0$ and oriented as the world frame (see fig. 3.2b). In this parametrization, the depth is encoded only in the parameter ρ, which is the inverse of the distance of the point to the anchor $\mathbf{p}_0$ through the vector $\mathbf{m}$.

The points detected in the images can be at a considerable distance from the sensor. For these points, the inverse representation of the depth is more appropriate because it behaves more linearly for large distances. The parameter ρ presents a non-linear behavior for small distances, which are actually way less numerous in practical cases – and must even be avoided when using a forward facing camera for safe robotics navigation. Moreover, this representation gives us the possibility of representing very far away points by making $\rho \to 0$.

The isolation of the depth information in one parameter makes the inverse depth parametrization more interesting than the Euclidean one when using camera measurements: with the main source of non-linearities isolated in one parameter, systems using inverse depth are less non-linear then ones using Euclidean landmarks. In solutions based on Gaussian filtering like EKF, it is common to initialize the inverse depth with a Gaussian prior whose mean is defined based on the size of the environment being explored, and with a standard deviation that covers the range of possible depths (*i.e.* the 3σ range should encompass $\rho = 0$ and $\rho = 1/d_{\min}$).

A drawback of using this representation is that six parameters are required to define a landmark, in contrast to three needed in the Euclidean representation. For filtering solutions this results in larger states, which in turn increases the computation needed per filter updates. A common workaround for this is to re-parametrize the landmarks from inverse depth to Euclidean once the covariance of the depth goes below a given threshold.

A point in this parametrization can be converted to Euclidean by:

$$\mathbf{l}_{\text{EP}} = \mathbf{p}_0 + \frac{\mathbf{m}(\psi, \theta)}{\rho} \tag{3.24}$$

where the direction vector $\mathbf{m}(.)$ is a function of the angles ψ and θ, given by:

$$\mathbf{m}(\psi, \theta) = \begin{bmatrix} \cos\psi \cos\theta \\ \sin\psi \cos\theta \\ \sin\theta \end{bmatrix} . \tag{3.25}$$

The pin-hole projection of a point represented in inverse depth could be computed by firstly converting to Euclidean with eq. (3.24) and then projecting with eq. (3.22), but we can do better and avoid the potentially dangerous division by ρ in eq. (3.24). For this we rewrite eq. (3.22) in its non-homogeneous form:

$$\underline{\mathbf{u}} \propto \mathbf{K}\mathbf{R}(\mathbf{q}^{\mathcal{S}})(\mathbf{l}_{\text{EP}} - \mathbf{t}^{\mathcal{S}}) \tag{3.26}$$

where $\mathbf{t}^{\mathcal{S}}$ and $R(\mathbf{q}^{\mathcal{S}})$ are respectively the translation and orientation in rotation matrix form of the frame $\mathcal{S}$ with relation to the frame the euclidean landmark is represented (normally the world frame $\mathcal{W}$). Substituting eq. (3.24) in eq. (3.26) we

have:

$$\begin{aligned}\underline{\mathbf{u}} &\propto \mathbf{K}\mathbf{R}(\mathbf{q}^{\mathcal{S}})^{\top}\left(\mathbf{p}_0 + \frac{\mathbf{m}(\psi,\theta)}{\rho} - \mathbf{t}^{\mathcal{S}}\right) \\ &= \mathbf{K}\underbrace{\mathbf{R}(\mathbf{q}^{\mathcal{S}})^{\top}\left(\frac{\mathbf{m}(\psi,\theta)}{\rho} - (\mathbf{t}^{\mathcal{S}} - \mathbf{p}_0)\right)}_{\text{point in sensor frame}}\end{aligned} \tag{3.27}$$

Due to the proportionality property of projection, we can multiply the point in sensor frame by a constant and have it project to the same place in the image. Thus, if we multiply it by ρ we get:

$$\begin{aligned}\underline{\mathbf{u}} &\propto \mathbf{K}\left(\mathbf{R}(\mathbf{q}^{\mathcal{S}})^{\top}\left(\frac{\mathbf{m}(\psi,\theta)}{\rho} - (\mathbf{t}^{\mathcal{S}} - \mathbf{p}_0)\right)\rho\right) \\ &= \mathbf{K}\mathbf{R}(\mathbf{q}^{\mathcal{S}})^{\top}\left(\mathbf{m}(\psi,\theta) - \rho(\mathbf{t}^{\mathcal{S}} - \mathbf{p}_0)\right)\end{aligned} \tag{3.28}$$

Now we have the parameter ρ as a multiplicative factor in the camera position $\mathbf{t}^{\mathcal{S}}$ and the anchor $\mathbf{p}_0$. In the cases the projected point is too far ($\rho \to 0$) or when the base line is to little ($\mathbf{t}^{\mathcal{S}} - \mathbf{p}_0 \to 0$), only the direction vector $\mathbf{m}(\psi,\theta)$ will be rotated and projected, as we would expect to happen in these cases.

3.3.2 Parallax Angle

The Parallax Angle Parametrization (PAP) was first presented in [Zhao 2011] and further analyzed in [Zhao 2015]. Its main characteristic is that it uses *two* anchors instead of one for the IDP, and the depth is encoded in a *parallax angle* between vectors from these anchors to the point. Since PAP is the main parametrization used in this chapter, it will be presented below in more detail than the other parametrizations.

3.3.2.1 Definition

An image feature $\mathbf{l}_{\text{PAP}}$ of a 3D point $\mathbf{p}$ represented in Parallax Angle Parametrization as:

$$\mathbf{l}_{\text{PAP}} = \begin{bmatrix}\mathbf{p}_m \\ \mathbf{p}_a \\ \psi \\ \theta \\ \omega\end{bmatrix} \tag{3.29}$$

where $\mathbf{p}_m$ and $\mathbf{p}_a$ are points in space known as *main anchor* and *associated anchor*, and ω is the *parallax angle* after which the representation is named. Similar to IDP, the anchors are camera positions from where the landmark was seen. The extra anchor in PAP increases the number of values to represent a landmark to nine, three more than what is needed for IDP, and therefore even less suitable to be used in a filtering technique. However, if used in a smoothing solution, the anchors can

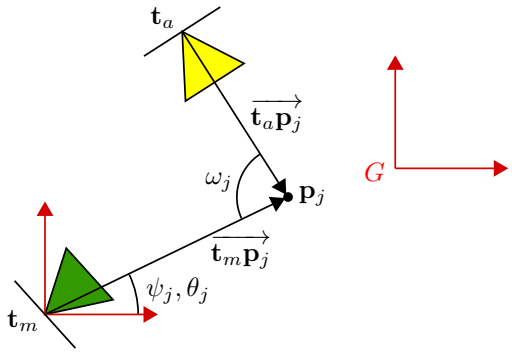


Figure 3.3: Geometrical relations between PAP parameters. For simplicity the relations are drawn in 2D, from a top view, and hence the angles ψ_j and θ_j are represented as the same arc. The triangles represent the field of view of the cameras. The main anchor camera is in green, the associated anchor camera is in yellow, and the global frame is in red.

be chosen to be camera positions that are already being estimated by the system, and thus the landmark can be parametrized only by the angle values, that will give its position in relation to its anchors.

Let's then redefine a landmark in PAP. From now on we only deal with PAP landmarks, so we drop the parametrization name from the subscripts to improve readability. An image feature $\mathbf{l}_j$ of a 3D point $\mathbf{p}_j$ observed from two cameras represented by frames C_m, C_a is represented in PAP as:

$$\mathbf{l}_j = \begin{bmatrix} \psi_j \\ \theta_j \\ \omega_j \end{bmatrix} \tag{3.30}$$

where ψ_j and θ_j are the azimuth and elevation angles in a frame centered at $\mathbf{t}_m$ and oriented as the global frame, (the same as IDP) and ω_j is the parallax angle, which is the angle between the vectors $\overrightarrow{\mathbf{t}_m\mathbf{p}_j}$ and $\overrightarrow{\mathbf{t}_a\mathbf{p}_j}$ (see fig. 3.2c for the 3D representation, or fig. 3.3 for a simpler 2D view). The feature $\mathbf{l}_j$ is said to be *anchored* to the frames C_m and C_a: it has $\mathbf{t}_m$ as the *main anchor* and $\mathbf{t}_a$ as the *associated anchor*.

Throughout the rest of this chapter, main anchor related variables are written as $(.)_m$, associated anchor related variables as $(.)_a$, variables not related with any of the anchors as $(.)_o$ (which stands for "other than anchors"), and variables related with landmark j as $(.)_j$. Camera frames written as C_i represents any camera frame, possibly the main frame C_m or the associated frame C_a.

3.3.2.2 Observation Function

Given a general camera frame $C_i = \begin{bmatrix} \mathbf{t}_i & \mathbf{q}_i \end{bmatrix}^\top$ and a PAP representation $\mathbf{l}_j$ of a 3D point $\mathbf{p}_j$, let $\mathbf{v}_i$ be the vector in the global frame that goes from the camera center

at $\mathbf{t}_i$ towards $\mathbf{p}_j$ (fig. 3.4). $\mathbf{v}_i$ can be computed in two distinct ways:

$$\mathbf{v}_i = \begin{cases} \mathbf{v}_m & \text{if } C_i = C_m \\ \tilde{\mathbf{v}}_i & \text{if } C_i \neq C_m \end{cases} \tag{3.31}$$

When $C_i = C_m$, $\mathbf{v}_i = \mathbf{v}_m$ is the direction vector from the main anchor to the point, given by[1]:

$$\mathbf{v}_m = \mathbf{m}(\psi_j, \theta_j) = \begin{bmatrix} \cos\psi_j \cos\theta_j \\ \sin\psi_j \cos\theta_j \\ \sin\theta_j \end{bmatrix} . \tag{3.32}$$

The reader should notice that $\mathbf{v}_m$ is the same director vector $\mathbf{m}(.)$ for IDP in eq. (3.25) and fig. 3.2. When $C_i \neq C_m$, $\mathbf{v}_i = \tilde{\mathbf{v}}_i$ is the vector proportional to $\overrightarrow{\mathbf{t}_i\mathbf{p}_j}$, given by [Zhao 2011, Zhao 2015]:

$$\begin{aligned} \tilde{\mathbf{v}}_i &\triangleq \sin\omega_j \overrightarrow{\mathbf{t}_i\mathbf{p}_j} \\ &= \sin(\omega_j + \varphi)\|\overrightarrow{\mathbf{t}_m\mathbf{t}_a}\|\mathbf{v}_m - \sin\omega_j \overrightarrow{\mathbf{t}_m\mathbf{t}_i} \end{aligned} \tag{3.33}$$

where φ is the angle between $\overrightarrow{\mathbf{t}_m\mathbf{t}_a}$ and $\mathbf{v}_m$, and can be computed as[2]

$$\varphi = \text{atan2}\left(\|\mathbf{v}_m \times \overrightarrow{\mathbf{t}_m\mathbf{t}_a}\|, \mathbf{v}_m \cdot \overrightarrow{\mathbf{t}_m\mathbf{t}_a}\right) \tag{3.34}$$

The image projection $\mathbf{z}_i = \begin{bmatrix} u_i & v_i \end{bmatrix}^\top$ of a point $\mathbf{p}_j$ to a camera C_i can then be computed as

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \propto \mathbf{K}\mathbf{R}(\mathbf{q}_i)^\top \mathbf{v}_i \tag{3.35}$$

with $\mathbf{K}$ being the camera calibration matrix, and $\mathbf{R}(\mathbf{q})$ being the rotation matrix representation of the quaternion $\mathbf{q}$. A 2D representation of the variables that define this observation function can be seen in fig. 3.4.

3.3.2.3 Initialization From Measurements

Given a feature $\mathbf{l}_j$ first observed from a camera at C_m, its azimuth and elevation angles in PAP are computed as:

$$\begin{aligned} \psi_j &= \text{atan2}\left(\hat{y}_m, \hat{x}_m\right) \\ \theta_j &= \text{atan2}\left(\hat{z}_m, \sqrt{(\hat{x}_m)^2 + (\hat{y}_m)^2}\right) \end{aligned} \tag{3.36}$$

[1] Here we use the *Front- Left-Up* (FLU) frame convention [Solà 2007, Section 1.2.1], for the XYZ axes, instead of *Right-Down-Forward* (RDF) convention of the original PAP papers.

[2] In the original PAP papers this angle was computed using arccos, we opt for a more stable version using atan2

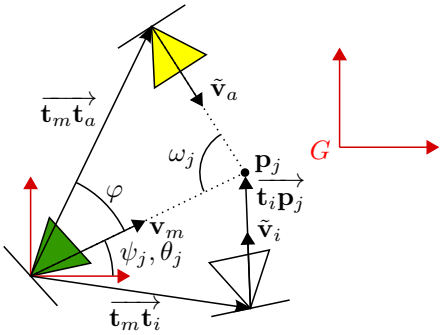


Figure 3.4: Variables involved in the observation function in PAP (again, the relations are drawn in 2D, from a top view). The white triangle represents the field of view of a camera other than the main and associated cameras. The vector $\mathbf{v}_m$, the vector $\tilde{\mathbf{v}}_i$ and the angle φ are respectively computed using eqs. (3.32) to (3.34). The vector $\tilde{\mathbf{v}}_a$ can be also computed from eq. (3.33) when $i = a$.

where $\hat{\mathbf{v}}_m$ is the vector computed by back-projecting the image measurement $\mathbf{z}_m = \begin{bmatrix} u_m & v_m \end{bmatrix}^\top$ taken from C_m:

$$\hat{\mathbf{v}}_m = \begin{bmatrix} \hat{x}_m \\ \hat{y}_m \\ \hat{z}_m \end{bmatrix} = \mathbf{R}(\mathbf{q}_m)\mathbf{K}^{-1} \begin{bmatrix} u_m \\ v_m \\ 1 \end{bmatrix} . \tag{3.37}$$

If the feature is re-observed from a camera at C_a, then the parallax angle ω_j can be initialized as

$$\omega_j = \operatorname{atan2}\left(\|\hat{\mathbf{v}}_m \times \hat{\mathbf{v}}_a\|, \hat{\mathbf{v}}_m \cdot \hat{\mathbf{v}}_a\right) \tag{3.38}$$

where $\hat{\mathbf{v}}_a$ is the vector computed by back-projecting the new measurement $\mathbf{z}_a$, with an expression akin to $\hat{\mathbf{v}}_m$ in eq. (3.37).

3.4 PAP in Incremental Graphical SLAM

As already discussed in chapter 2, the first step to solve graphical SLAM is to build the graph whose structure will be used to fill efficiently the matrices that are used in the minimization. Doing it incrementally means that we update the graph (adding new variables and/or factors) as new measurements are available, to then find an updated solution using the new graph structure. The updated solution may be computed every time the graph changes, or after the graph has accumulated enough changes.

Normally data acquisition from robot sensors is performed in a rate that is faster than the robot dynamics. Updating the graph at sensor rate would create many redundant variables, *e.g.*: several poses variables representing robot poses too close to each other. The increased number of variables would reduce the system performance without further benefit for the trajectory estimation. Besides that, for visual SLAM, graph updates as such would result in landmark variables connected

to many projection factors that can not provide new information about the landmark parameters because the poses variables they are connected to are not distant enough from each other to provide a good baseline to observe the landmark.

To avoid these problems we made some assumptions. We assume that there exists:

- A lower-level odometry system that is able to provide short term measurement of the robot movement without accumulating too much drift;
- A lower-level visual tracking system to detect and track landmarks in the image; and
- A criteria that decides when to add a new pose to the graph, according to some heuristics. For visual SLAM, the usual heuristics are:
  - Minimum number of common features between the current image and the image taken from the pose that was last added to the graph;
  - Maximal linear and angular distance between the current pose estimated by the lower-level odometry and the pose that was last added to the graph;
  - Both previous heuristics.

When there is a decision to add a new pose to the graph, the current estimation of the lower-level odometry system (mean and covariance) is used as the odometry measurement that is then encoded in a odometry factor. Also, measurements to landmarks in the image taken from this pose may be encoded in projection factors. This process is described in section 3.4.3 further below.

3.4.1 Projection Factors for PAP

When using PAP in a graphical SLAM, the observation function shown in eq. (3.35) may involve values from up to four different variables (main anchor, associated anchor, camera pose, and landmark). Due to different ways to compute the vector $\mathbf{v}_i$ in eq. (3.31), we have three types of projection factors for PAP landmarks. They are shown in fig. 3.5.

The *ordinary factor* is shown in fig. 3.5(c). In these cases the camera to where we wish to project the point is not an anchor camera, and thus it uses eq. (3.33) to compute $\mathbf{v}_i$ (which is in fact $\tilde{\mathbf{v}}_i$), and it depends on both landmark, anchors and the camera to where the landmark is being projected.

The other two factors are special cases when the measurements are taken from one of the anchor frames: the factor shown in fig. 3.5(a) is called *main factor* and uses the equation eq. (3.32) to compute $\mathbf{v}_i$ (which in this case is $\mathbf{v}_m$). The projection function depends only on the landmark parameters ψ and θ, and on the orientation stored on x_m, thus we have a factor with only two connections. The factor shown on fig. 3.5(b) is the special case of equation eq. (3.33) when $i = a$, and it is called *associated factor*, thus it depends on the main and associated anchor

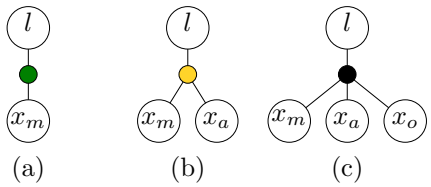


Figure 3.5: Three possible projection factors for a PAP feature. (a) main factor; (b) associated factor; and (c) ordinary factor. The subscripts for the camera variables represents the anchors: m for the main anchor frame, a for the associated anchor frame, and o for all the other cameras from where the landmark was observed.

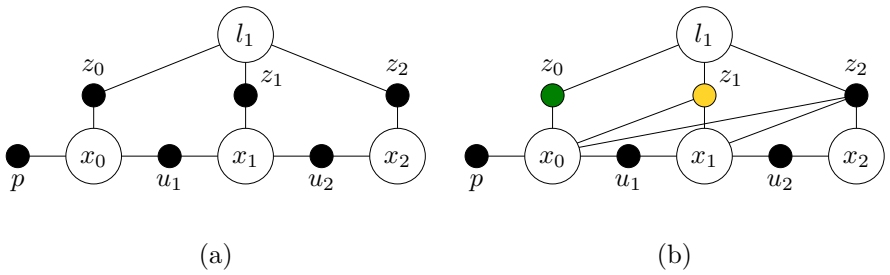


Figure 3.6: Formulation of the SLAM problem using projection factors. Variable nodes are shown in large circles and factor nodes as small solid circles ($\mathcal{V}=\{x_0, x_1, x_2, l_1\}$, $\mathcal{F}=\{p, u_1, u_2, z_0, z_1, z_2\}$). The factors represent a prior information p about the origin, odometry measurements u, and landmark measurements z. (a) Typical factor graph of the monocular SLAM when landmarks are parametrized as Euclidean coordinates. Note that every factor z here are of the same type, connecting only to the landmark and the camera pose from which the measurement was taken. (b) Factor graph when using the specific PAP factors. The color code is the same as in the former figures: the measurements from the main and associated anchors are respectively in green and yellow. Here the landmark l_1 is anchored in the camera frames represented by x_0 and x_1.

values, resulting in three connections. Note that for each fully initialized landmark there is always one main and one associated factor, all other factors for the same landmark being of the ordinary type.

Figure 3.6 shows examples of monocular SLAM factor graphs using the aforementioned PAP factors (fig. 3.6b), together with the graph with Euclidean projection factors (fig. 3.6a) for comparison. The main structural difference between them is the increased number of edges due to the use of projection factors that connect to more than one camera frame in the PAP factor graph.

3.4.2 Incremental Anchor Selection

PAP was originally conceived to be used in *Bundle Adjustment*. In this context, initial estimates of the camera frames are computed from two-view geometry techniques, like the eight point algorithm. The parallax angle is then computed using these initial camera frame estimates for each pair of measurements to a landmark, and the camera frames that provides the best parallax angle, *i.e.* maximum par-

allax found, or a parallax angle larger than a threshold, are selected as landmark anchors. This threshold is used to stop the anchor search when the parallax is good enough, avoiding unnecessary computation. The anchors are then kept fixed during the optimization phase.

Indeed, for PAP to perform well it is important to choose anchors that provide a good observability of the landmark depth, that is encoded in the parallax angle *and* the anchors. In an incremental SLAM solution, it is not possible to know in advance which camera frames will be the best anchors for a landmark: it is necessary to allow the anchors to be actively selected when new observations are made. This is done in two steps: first the landmark should be *anchored*, and then possibly be *re-anchored* if better anchor frames are found.

3.4.2.1 Anchoring

To *anchor* a landmark it is necessary to decide which frames are adequate anchors. Observing fig. 3.4 we can see that the anchors should not define a vector $\overrightarrow{\mathbf{t}_m\mathbf{t}_a}$ that is collinear with vector $\mathbf{v}_m$. Such configuration produces an angle φ and potentially an initial parallax angle ω_j that, when combined, make $\tilde{\mathbf{v}}_i \to \mathbf{0}$ due to the coefficients $\sin(\omega_j + \varphi)$ and $\sin \omega_j$ in eq. (3.33), that we repeat here for convenience:

$$\tilde{\mathbf{v}}_i = \sin(\omega_j + \varphi)\|\overrightarrow{\mathbf{t}_m\mathbf{t}_a}\|\mathbf{v}_m - \sin \omega_j \overrightarrow{\mathbf{t}_m\mathbf{t}_i}$$

As a consequence of $\tilde{\mathbf{v}}_i \to \mathbf{0}$, eq. (3.35) becomes ill-posed, and the associated and ordinary factors of this landmark can not compute the expected image measurement properly.

Figure 3.7(a)-(b) shows the cases to avoid when anchoring a landmark. This problem is more likely to happen when the camera is moving along its focal axis and observing a landmark situated on the focal point. Note that cases when the point is very far away (zero or near zero initial parallax) but the anchors are not collinear should not cause problems in PAP, as shown in fig. 3.7(d).

3.4.2.2 Re-anchoring

To *re-anchor* a landmark, a re-anchoring test is performed *after the optimization step*, whenever an anchored landmark is observed from a new camera frame and the current parallax is below a threshold. Equation (3.38) is used to compute the parallax angle as if the new frame was one of its anchors. This angle is computed twice: one time considering the main-new frame pair as anchors; and another time for the associated-new frame pair. In both cases the current frame estimates and the respective measurements from these frames to the landmark are used.

If any of the pairs provides a better parallax, the frames that compose this pair are selected as the new anchors. Then, we have two cases to deal with:

- If main-new is a better pair of anchors, we need to change the current associated anchor. For this, the current associated factor is replaced by a ordinary

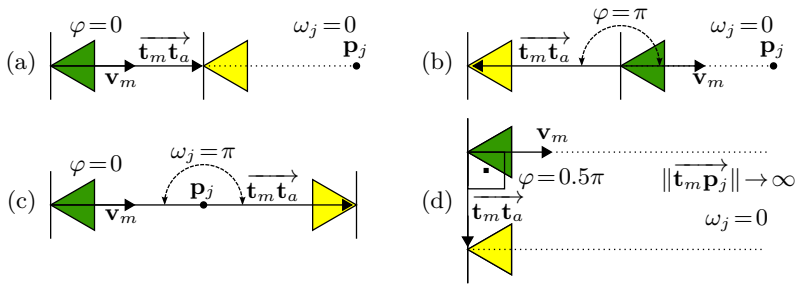


Figure 3.7: Particular geometrical configurations of landmark anchors and the angles φ and ω_j (again presented in 2D, compare with fig. 3.4): (a) The associated anchor is between the main anchor and point; (b) The main anchor is between the associated anchor and the point; (c) The point is in the middle of anchors; (d) The point is at infinity and the anchors are not aligned. Configurations (a) and (b) should be avoided because they produce a vector $\tilde{\mathbf{v}}_i = \mathbf{0}$ in eq. (3.33) that cannot be properly projected by eq. (3.35). (c) will hardly occur in practice because the re-anchoring process stops once the anchors provide good parallax angle for the landmark, not to mention that visual features can hardly be matched in such conditions. (d) shows that PAP can handle very far away points (theoretically at infinity) as long as the anchors are not aligned with the point.

factor that has the new frame as the associated anchor, and the factor with the new measurement is replaced by an associated factor. Also, all other ordinary factors need to be adapted to link to the new associated anchor, and the landmark's ω has to be updated to reflect this new anchor as well. The main factor is the only that remains unchanged;

- If associated-new is a better pair of anchors, both current anchors should change. The current main factor is replaced by a ordinary factor, and the current associated factor is replaced by a main factor. The factor with the new measurement is replaced by an associated factor, and all other ordinary factors are adapted to link to the new anchors. The landmark variable has to be fully updated as well: ψ and θ should reflect the orientation from the new main anchor, and ω should be set according to the new anchor pair.

In the case none of the tested pairs provide a better parallax, the new landmark measurement is kept as an ordinary projection factor. This re-anchoring procedure is shown in fig. 3.8. Note that replacing some projection factors by other types implies the use of different functions to compute the error between the expected and acquired measurements.

3.4.3 Incremental PAP Algorithm

The algorithm for incremental SLAM with PAP is described in algorithm 1. The initialization step goes from lines 1 to 5. Here the origin and the prior factor are normally set to the identity frame (line 1). The prior covariance is set to be very small, but not zero (line 2). Setting it to zero would introduce a hard constraint to the system that can cause problems during optimization. Then we

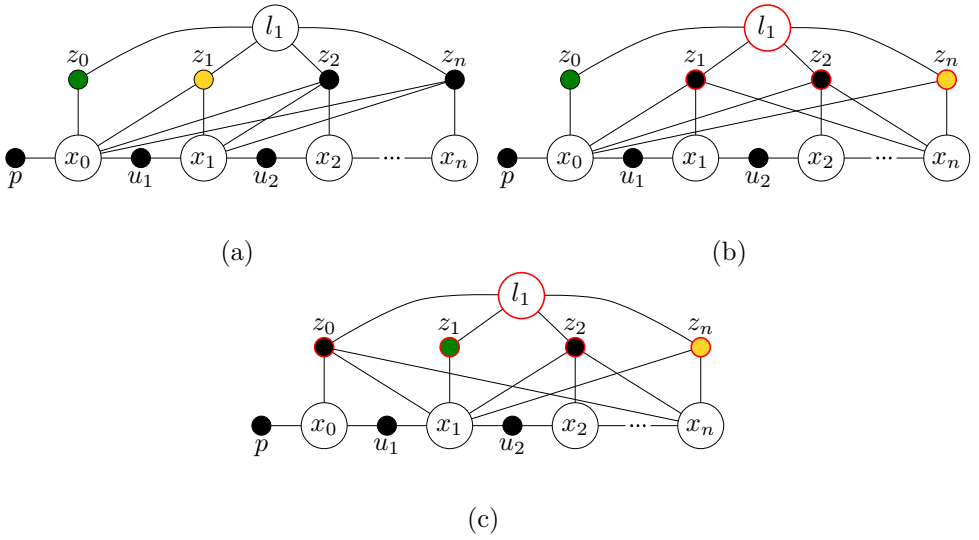


Figure 3.8: Landmark re-anchoring after addition of a factor z_n. All affected factors and variables are highlighted in red. (a) New measurement does not provide better parallax *or* current landmark's parallax angle ω is above a threshold, so the measurement is kept as an ordinary factor; (b) A new measurement provides better parallax with respect to the main anchor: x_n becomes the associated anchor, all projection factors *but the main one* are updated, and landmark's ω is updated using eq. (3.38); (c) A new measurement provides better parallax with respect to the associated anchor: the current associated anchor x_1 becomes the new main anchor, x_n becomes the new associated anchor, all projection factors are updated, and landmark's ψ, θ and ω are updated using eq. (3.36) and eq. (3.38) according to the new anchors.

store a frame-measurement pair for each landmark observed from the origin as the main anchor information for that landmark (line 4), and compute its direction angles (line 5). The stored anchor information is used later to initialize the landmark variable and to create the projection factors to the anchors.

The main loop of the algorithm starts at line 6 and goes until line 29. For each new camera frame and new odometry measurement we initialize a new frame variable by composing the previous frame with the odometry measurement (lines 7 and 8), and we add a odometry factor between subsequent camera frames to the graph (line 9).

Then, from lines 10 to 22 we handle the landmark measurements from this new frame. For new landmarks we store their main anchor information (line 12) and compute their azimuth and elevation angles (line 13). Observed landmarks that had been previously seen but are not yet anchored are tested to avoid the collinearity problem (line 15). If the test succeeds they have their associated anchor information stored (line 16), their parallax angle computed (line 17), their anchor factors created and added to the graph (line 18), their variable node initialized (line 19), and finally they are set as anchored (line 20). Observed landmarks that were already anchored receive a new ordinary projection factor (line 22).

Next, the current factor graph is solved (line 23) and all variables are updated

with the optimization solution (line 24).

Finally we deal with re-anchoring in lines 25 to 29, the final part of the main loop. The re-anchoring test is performed for every landmark observed from the new frame to which the measurement was added as an ordinary factor (lines 26 and 27), and updates in the factor graph and the landmark variables are performed if needed, as described in section 3.4.2.

Note that the re-anchoring step is performed only *after* the optimization step. This is needed to guarantee that the tested parallax angles use frame values compatible with the anchor frame values, *i.e.* that all frame values used in the test come from the solution of the optimization.

Algorithm 1 Incremental PAP SLAM.
Camera frames x_i and landmark variables l_i are stored in the variable set $\mathcal{V}$. Odometry measurements u_i and landmark measurements z_{ij} are stored as factors in the factor graph $\mathcal{F}$. Anchor information for a landmark j is stored as a pair (C_i, z_{ij})

1: Add camera frame C_0 as origin x_0 to $\mathcal{V}$
2: Add prior about the origin as a factor p to $\mathcal{F}$
3: **for** each measurement z_{0j} of a landmark j visible from C_0 **do**
4: Store (C_0, z_{0j}) as main anchor of landmark j
5: Compute ψ_j and θ_j of l_j ▷ eq. (3.36)

6: **for** each new odometry measurement u_i from C_{i-1} to C_i **do**
7: Compose x_{i-1} in $\mathcal{V}$ with u_i to initialize variable x_i
8: Add variable x_i to $\mathcal{V}$
9: Add u_i as odometry factor between x_i and x_{i-1} to $\mathcal{F}$
10: **for** each measurement z_{ij} of a landmark j visible from C_i **do**
11: **if** z_{ij} is first measurement of landmark j **then**
12: Store (C_i, z_{ij}) as main anchor of landmark j
13: Compute ψ_j and θ_j of l_j ▷ eq. (3.36)
14: **else if** landmark j is **not** anchored **then**
15: **if** $\overrightarrow{\mathbf{t}_{mj}\mathbf{t}_{aj}}$ is **not** collinear with $\mathbf{v}_{mj}$ **then** ▷ section 3.4.2.1
16: Store (C_i, z_{ij}) as associated anchor of landmark j
17: Compute ω_j of l_j ▷ eq. (3.38)
18: Add both anchor factors of l_j to $\mathcal{F}$ ▷ figs. 3.5 and 3.6b
19: Add variable l_j to $\mathcal{V}$
20: Set landmark j as anchored
21: **else if** landmark j is anchored **then**
22: Add it as an ordinary factor of l_j to $\mathcal{F}$ ▷ figs. 3.5 and 3.6b

23: Solve optimization problem using $\mathcal{F}$ and $\mathcal{V}$ ▷ chapter 2
24: Update $\mathcal{V}$ with current solution

25: **for** each measurement z_{ij} of a landmark j visible from C_i **do**
26: **if** measurement z_{ij} is encoded as a ordinary factor **then**
27: Perform re-anchoring test for landmark j ▷ section 3.4.2.2
28: **if** landmark j should be re-anchored **then**
29: Update $\mathcal{F}$ and $\mathcal{V}$ to match new anchors ▷ fig. 3.8

3.5 Analysis

The experiments were realized in a simulated environment using SLAMTB[3], a SLAM Toolbox for Matlab. They consisted in a robot represented as a 3D pose moving in a plane, in an environment filled with 3D landmarks. The robot has an odometry sensor and one camera, which is positioned with the focal axis looking forward. Key-frames were selected after a certain number of odometry measurements. These measurements were integrated and the result is used as odometry measurements in algorithm 1, together with image measurements of landmarks visible from the key-frame.

The front-end used is similar to the one used in RT-SLAM [Roussillon 2011]: we limit the number of measurements in each key-frame to the first N_{up} measurements to landmarks whose projections provide the best innovations. We always try to initialize N_{init} new landmarks every key-frame to deal with possible landmarks going out of the field-of-view.

The optimization back-end used ISAM2 [Kaess 2012] as it is implemented on *GTSAM 3.1.0*. The PAP variables with initialization methods and the different projection factors described in this work are implemented in GTSAM and are available at `github.com/Ellon/gtsam`. As a wrapper is used to access GTSAM libraries from Matlab, it is impossible to have a computational analysis of time performance of the algorithm, although some theoretical analysis are discussed below.

The proposed algorithm is placed in-between the SLAMTB front-end and the ISAM2 back-end. It decides which landmarks can be anchored and initialized, and when to do so, dealing also with projection factor creation and possible removals, modifications and re-insertions of these factors in the factor graph inside ISAM2 due to incremental re-anchoring. Contrary to the original PAP papers, here we estimate the robot frames instead of camera frames. While this increases the non-linearity of the system, it is the easiest way to fuse data from multiple sensors (odometry and camera in our case). The landmark anchors are taken from the composition of estimated robot frames with the camera frame with respect to the body frame, which is considered known and constant.

The trajectory estimated for a long run along a squared path can be seen in fig. 3.9.

3.5.1 Performance on ISAM2

The decision to use ISAM2 as the incremental optimization back-end was motivated by its popularity and reported performance when used in SLAM solutions. Here we present the results of using PAP with ISAM2.

The main features of ISAM2 are:

1. restructure the factor graph in a tree of cluster-nodes called Bayes Tree [Kaess 2011] in a way incremental updates tends to modify only a small part of the tree;

[3] `github.com/joansola/slamtb`

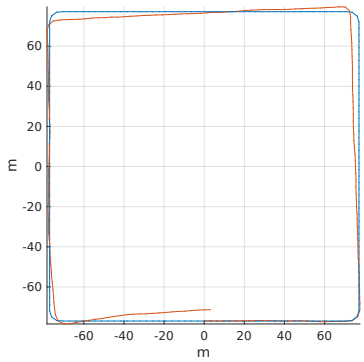


Figure 3.9: Robot trajectory for a long run: estimated in red, and simulated in blue

Table 3.1: Average size of frontal variables and separators of Bayes trees

	Front Variables	Separator set
Euclidean	1.33	9.19
PAP	2.09	11.31

2. the ability to perform partial updates of the variables to reduce the computational cost while recovering a nearly exact solution; and

3. track the validity of the linearization point for each variable *and* only re-linearize when needed.

3.5.1.1 Incremental Updates

We investigated the impact of the extra edges needed by the associated and ordinary projection factors presented in section 3.4 on the incremental update. We compared the Bayes trees built for the monocular SLAM using Euclidean XYZ and using PAP, run over the scenario shown in fig. 3.10. The differences between the average number of frontal variables and the sizes of the separator is shown in table 3.1. They suggest a denser fill-in of the system matrices when using PAP, which yields slower incremental updates. The Hessian matrices in fig. 3.11 confirm the increase of the fill-in caused by the additional edges needed to implement PAP with factor graphs.

3.5.1.2 Sensibility to ISAM2 parameters

The ISAM2 features mentioned in items 2 and 3 are controlled respectively by two parameters α and β. They were found to have great impact on the results, especially β. Setting β to a large number trigger re-linearizations less often, providing faster updates. We suppose that by estimating the landmarks values in the system we increase the non-linearity of the system, being necessary to keep the system

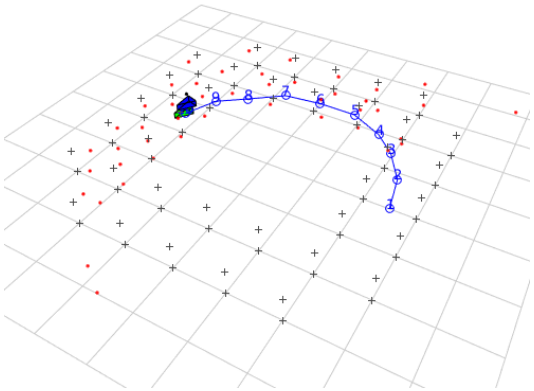


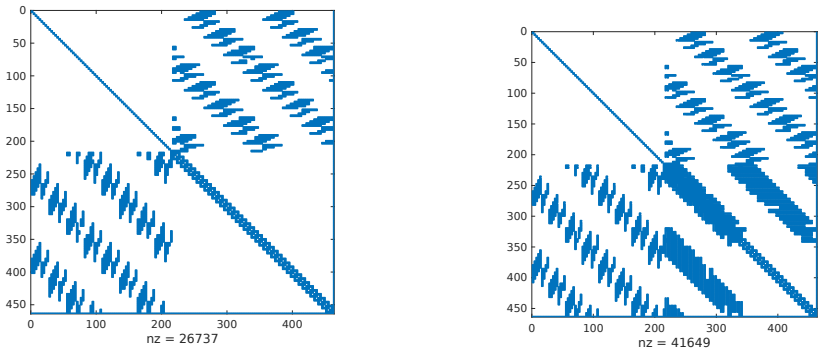
Figure 3.10: Scenario used for testing of incremental updates. The polygons are the simulated robot: simulated in blue, and estimated (little below the simulated) in green. The crosses are the simulated landmarks. Red marks are estimated landmarks and the blue circles and line are the key-frames and trajectory.

Jacobians up to date to achieve a good solution. In the experiments we normally set this parameter to a very low value.

3.5.2 Projection Factors and Cheirality

The measurements associated to a projection factor are the image coordinates of the landmark (i.e. their projection on the camera). To evaluate the error contribution of a projection factor (and to derive the Jacobians of the observation function) we use the equations from section 3.3.2.2. For a projection to be valid, the vector $\mathbf{R}(\mathbf{q}_i)^\top \mathbf{v}_i$ in eq. (3.35) needs to be in the *front side* of the image plane of the camera at C_i, also named *cheirality* [Hartley 1993]. If the cheirality is negative, the point lays behind the camera, and then the projection factor should not be considered (the Jacobians are set to zero). This can be dangerous for monocular vision systems because it may reduce the rank of the matrix used in the optimization step that is built from the factor graph, possibly leading to *ill-conditioned local systems* that cannot be solved.

When using PAP points, we noticed that sometimes the estimated points are indeed behind the camera. This mostly occurs with points lying on the ground and during the few first observations. Since the robot is always moving forward in our test case, a landmark estimated by a PAP that is close to the robot and with a reduced observability (for example, with only measurements from its anchors) may have big changes on its depth with respect to the main anchor on each iteration of the algorithm, and possibly move behind another camera that has a measurement to it. We noticed that the point became stable as more measurements are incorporated and the re-anchoring is applied.



(a) Euclidean. (b) PAP.

Figure 3.11: Hessian matrices for the SLAM problem using Euclidean and PAP. The PAP Hessian has a higher number of non-zeros (41649) with respect to the Euclidean solution Hessian (26733).

3.6 Conclusions

This chapter presented how to use Parallax Angle Parametrization (PAP) for 3D points in an incremental graphical SLAM setting. The main difference from the original PAP approach is the incremental selection of anchors, and the estimation of robot frames instead of camera frames. We presented the factors needed to project a PAP points into the cameras, and how to re-anchor then when needed to obtain an observation of the landmark parallax angle. Some analysis of the performance in the incremental estimation were presented, based on simulations. We used ISAM2 as the incremental optimization back-end and the projection factors for PAP implemented on GTSAM framework were made publicly available.

In the original PAP for BA, the search for the best anchors is done only once at the beginning. The camera poses are initialized by composing the transformations between subsequent keyframes. These transformation are obtained with the 8-point geometry technique with fixed scale, using the common set of measurements in the images taken from the keyframes. The search stops once there is a pair of keyframes that provide a minimal initial parallax for the landmark, or after testing all keyframes, in which case the best pair is selected as anchors.

For the incremental version proposed here, the decision to re-anchor a landmark is done by comparing the current estimated landmark's parallax with those computed from the new anchor candidates, and we emphasized that this comparison is done *after the optimization*. The reason behind this is that since the new frames are initialized by composing the current odometry estimate with the estimated pose of the last keyframe, by performing the test after the optimization we avoid using a noisy value for the new frame to compute the parallaxes that is compared in the test, giving a chance for the new frame to be optimized at least once. Regarding the values being compared, an alternative would be to recompute the parallax using the current estimate of the current anchors, instead of

using the current landmark's parallax estimate, and compare it with the one computed from both candidate pairs. The interest of this alternative has to be assessed.

The proposed system was only tested using simulated data, and experiments on real datasets were tried, but not concluded. Running the approach with real datasets requires the presence of a front-end to detect and keep track of landmarks between the keyframes. The front-end used to process the images in our trials with real datasets was based on a naive feature-based matching on subsequent images, and turned out to be not able to keep track of features over long sequence of images.

Also, neither the algorithm 1 nor the used optimization back-end is able to handle outliers in the data association. Thus the front-end should be able to perform data-association with few outliers, otherwise errors from bad matches would be propagated in the system. The use of robust error functions for the image measurements can reduce the effect of such outliers, but may also disturb the convergence, and increase risk of being trapped in a local minimum.

A complete visual odometry system would be the most appropriated choice for a front-end. As showed by [Strasdat 2010, Strasdat 2012], increasing the number of observations in visual SLAM increases the accuracy of the system, while increasing the number of intermediate keyframes has only minor effect. In this sense, an active-search filter front-end is not the best choice, since it would only detect a few landmarks (the ones that provide the best innovation) on each image. The needed front-end must be able to track a high number of landmarks to provide enough observations to the back-end to be accurate on its estimation. Loop closings in the system are simply the re-observation of the landmarks when revisiting a known place: this ability could be provided by a place-recognition system, that would provide matches between the landmarks.

Possible future extensions to this work could be to find a better elimination order that increases the sparsity when using PAP, and performing the re-anchoring inside ISAM2 by only reconnecting edges, without the need to explicitly remove and reinsert the modified factors in the factor graph.

This chapter presented the first adaptation of PAP to incremental SLAM setting, presenting the mechanics that should be used in such adaptation. The main caveat of the proposed method is the need to perform re-linearization more often than it would be necessary in a pose-graph SLAM, which goes against one of the main ideas of ISAM2, and surely hinders its performance. We believe this happens because the landmarks are also estimated as part of the state, resulting in a system with more sources of non-linearity than it would be if the only non-linear factor would be the orientation of robot poses. The same mechanisms presented here can be applied to build and adapt a graph of the SLAM problem incrementally, and whose solution would be found by classical non-linear optimization techniques, other than ISAM2.

If only the trajectory of the robot is of interest, one possible alternative would be to use *structureless landmarks*, as described in [Forster 2015], that uses *smart projection factors* [Carlone 2014] to avoid estimating landmark parameters directly

while still using landmark measurements to constrain the robot trajectory.

Chapter 4

ICP-Based Pose-Graph SLAM

Contents

As we have seen in the previous chapter, maintaining a landmark map is a burden for the estimation process, due to the increased size of the state to be estimated and the non-linear relations between the parameters composing the map and the poses composing the robot trajectory. One way to avoid this is to use the external observations to derive constraints directly on the poses, and not explicitly represent the map in the system state being estimated. This is more complicated for image measurements due to the lack of depth information in the landmark observations, but may be not the case when using different sensors. For instance, point clouds provide data that is rich in depth information, and such constraints can be computed by means of registration algorithms. This is the idea that motivates the contribution presented in this chapter: it describes a way to exploit scan registration algorithms and graphical model optimization to build a localization system for

autonomous mobile ground robots equipped with LIDAR sensors, for missions in semi-structured environments. Besides the integration of these two techniques, the configuration of the registration algorithm to work with point clouds acquired from the Velodyne HDL LIDAR is precisely described.

4.1 LIDARs and robot localization

LIDAR sensors, like the Velodyne HDL sensors, have primarily been used for obstacle detection, but they have also rapidly been exploited for localization using scan registration algorithms. These algorithms find the transformation that best aligns the points of one point cloud with respect to a reference point cloud. They have been used for several LIDAR odometry solutions, that compute the overall robot position by integrating elementary displacement estimates. Some solutions perform registration on the basis of matches established between features extracted from the scans [Zhang 2014], others directly exploit the scan points [Pomerleau 2011].

The main scan registration technique is the *Iterative Closest Points* (ICP) algorithm, originally introduced in [Besl 1992]. ICP is a rather simple algorithm with a clear sequence of steps, but numerous variants have been developed along the years, so that selecting the proper configuration and its parameters requires empirical tests and experience. The designer needs to consider the environment in which the robot evolves, and especially the sensor being used, the amount of data to process, how the data is organized, the measurement errors in the data, and so on.

When properly configured, the scan registration is precise. But as it is inherent to any dead-reckoning system, the accumulation of errors made at every registration leads to a drift of the overall position estimate. In the absence of any prior map of the environment, resorting to a *Simultaneous Localization and Mapping* (SLAM) approach is the only way to reduce this drift.

4.2 System Overview

This section provides an overview of the system, and the outline for the remainder of the chapter. The diagram shown in fig. 4.1 describes an hypothetical state of the overall system after processing some scans. It exhibits the two main layers of the proposed system, plus the scan stream, that is simply the history of acquired scans. The ICP layer is composed by *keyframes* K_i, and their associated scans (gray cloud shapes in fig. 4.1), also called *keyframe scans*. Each keyframe K_i is associated to a node x_i in the graph layer. Some of these keyframe scans are selected to compose *local maps* (blue clouds, local map of size 3), that are used as reference input for the ICP process. The keyframe scans not used as local map are stored, and may be later used to compose a local map if the robot revisits this part of the environment. The robot frame R is always expressed with respect to the local map frame L, which is coincident with closest keyframe (K_{18} in fig. 4.1). The system does not require a complete map of the environment, yet such a map can be computed on demand:

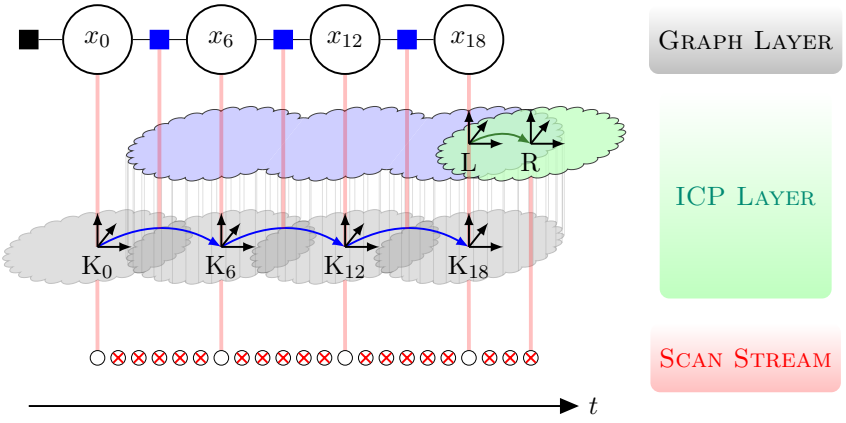


Figure 4.1: Overview of the system, composed by the ICP layer and the graph layer. The scan stream is also shown to explicit the acquisition of data with the elements of each layer. Red vertical lines exhibit correspondences in-between these elements, and between elements and the scans being acquired.

it consists of the concatenation of all keyframe scans positioned according to the respective keyframe values.

As the robot starts, a first keyframe K_0 is associated to the first acquired scan. The associated node x_0 is also created in the graph, and a factor representing the initial robot position with respect to the world origin is added to the graph (black square).

Every time a new scan (green cloud) is available at the current robot position R, the ICP finds the transformation (green arrow) that aligns it with the current local map, correcting the robot position locally. If the overlap between the current scan (still the green cloud) and the current local map is lower than a threshold, a new keyframe is created – otherwise the scan is just discarded (crossed circles in the scan stream level). Section 4.3 depicts the ICP algorithm used to align the scans and how it is configured for the Velodyne HDL data.

Newly created keyframes are initialized to be coincident with the robot frame after the local correction from ICP. A new frame variable that is associated to the new keyframe is added as a node to the graph (named white circles on top of fig. 4.1), as well as a factor containing the transformation between the new and former keyframes (blue squares for factors, blue arrows for transformations). Finally the local map is rebuilt by incorporating the newest keyframe scan and removing the furthest keyframe scan (i.e. in fig. 4.1, K_0 was removed from the local map upon creation of K_{18} and its addition to the local map). Section 4.4 depicts the selection of keyframes and management of the local maps. The graph layer and details of the factors being used are presented in section 4.6.

When the system detects a potential loop closing between two keyframes (not shown in fig. 4.1 for simplicity), a local map is built around the oldest between these loop closing keyframes, and an ICP call tries to align the scan of the newest loop

closing keyframes with this local map. If the ICP is successful, a new factor is added to the graph between the variables associated with these loop closing keyframes, creating a loop in the graph level. An optimization using the graph data is then triggered. The loop closing process ends with the repositioning of keyframes using the optimization result, and local maps are reconstructed according to these new keyframe values. Since the robot frame R is always expressed with respect to the closest keyframe, its global pose is implicitly updated by the loop closing process. Section 4.5 presents the details of this loop closing procedure.

4.3 Point Cloud Registration

The first building block of the system is the well known *Iterative Closest Points* (ICP) registration algorithm. This section presents an overview of the ICP algorithm, followed by the details of the ICP solution designed for this work.

4.3.1 Overview of ICP

We present the ICP algorithm according to the in-depth review in [Pomerleau 2013b]. The ICP is responsible to find the transformation that best aligns a geometric shape called *reading*, to another shape called *reference*. This operation is known as *registration*. In our case, a shape $\mathcal{S}$ is a set of 3D points extracted from a LIDAR scan. As it is usual for ICP implementations, scans are preprocessed by filters before running the optimization that finds the best alignment. They may, for example, add more information to the scan (e.g. add normal vectors to points based on the point neighborhood), or to remove points that do not bring valuable information for the registration.

Let $\mathcal{P}^A$ the reading set in a coordinate frame $\mathcal{A}$, $\mathcal{Q}^B$ the reference set in a coordinate frame $\mathcal{B}$, and $\overline{\mathcal{P}}^{\mathcal{A}}$,$\overline{\mathcal{Q}}^{\mathcal{B}}$ the filtered version of the respective sets (the need to apply filters on the input sets is discussed in section 4.3.2). The registration algorithm estimates the transformation $\hat{\mathcal{T}}^{\mathcal{BA}}$ that minimizes an error function $e(\mathcal{P}, \mathcal{Q})$:

$$\hat{\mathcal{T}}^{\mathcal{BA}} = \arg\min_{\mathcal{T}}(e(\mathcal{T}(\overline{\mathcal{P}}^{\mathcal{A}}), \overline{\mathcal{Q}}^{\mathcal{B}})) \tag{4.1}$$

where $\mathcal{T}(\mathcal{S})$ is the transformation applied to a set $\mathcal{S}$.

Note that the error function in eq. (4.1) has the filtered sets as inputs. In fact, the error function is computed on pairs of points that have been associated between the filtered input sets. The association is produced by a *matching function*, and is usually solved by associating to each point of the reading the closest point in the reference. For the sake of robustness, weights can be provided by an *outlier rejection function* to change the influence of the matches in the error function. Thus, if $\mathcal{M} = \text{match}(\mathcal{P}, \mathcal{Q}) = \{(p, q) : p \in \mathcal{P}, q \in \mathcal{Q}\}$ and $\mathcal{W} = \text{outlier}(\mathcal{M}) =$

$\{w(p,q) : \forall (p,q) \in \mathcal{M}\}$, the error function is given by:

$$e(\mathcal{P}, \mathcal{Q}) = \sum_{(p,q)\in\mathcal{M}} w(p,q)d(p,q) \tag{4.2}$$

where $d(p,q)$ is a *distance function* between two points.

In practice the associations from the matching function are not perfect, and the best estimate for $\mathcal{T}^{\mathcal{BA}}$ cannot be found perfectly by eq. (4.1). Nevertheless, the idea behind ICP is that even with imperfect matches, minimizing the error can provide an estimation that, in turn, provides better matches, and so on. This leads to an iterative algorithm where each iteration provides an intermediary transformation $\mathcal{T}^{i+1,i}$ from:

$$\mathcal{T}^{i+1,i} = \arg\min_{\mathcal{T}}(e(\mathcal{T}(\overline{\mathcal{P}}^i), \overline{\mathcal{Q}}^{\mathcal{B}})) \tag{4.3}$$

where $\overline{\mathcal{P}}^i$ is the filtered cloud that is iteratively transformed by the intermediary transformations. The final transformation estimate is given by:

$$\hat{\mathcal{T}}^{\mathcal{BA}} = \left(\bigcirc_i \mathcal{T}^{i+1,i}\right) \circ \mathcal{T}_{\text{init}} \tag{4.4}$$

with $\bigcirc_i \mathcal{T}^{i+1,i}$ being the iterative composition of all intermediary transformations and $\mathcal{T}_{\text{init}}$ being the initial transformation between the shapes that is supplied to ICP at the beginning of this iterative process. Note that the initial transformation is important, since initial transformations that do not provide enough good matches may cause the ICP minimization procedure to be trapped in a local minimum.

4.3.2 Configuration of ICP

There is a wide spectrum of ICP solutions that can be implemented by different combinations the ICP elements, namely the filters, the match, the outlier detection, and the distance functions. The good choices for an ICP solution are strongly dependent on the environment, the computing resources, and the sensing capabilities of the robot: assessing them requires a significant amount of expertise. This section presents the details of the ICP solution used in this work for mobile robots with the Velodyne HDL 64 sensor in mostly urban areas, along with the rationale behind them. The presentation of the solution is divided into the main elements already discussed in section 4.3.1. Figure 4.2 show examples of intermediary products of the ICP pipeline.

4.3.2.1 Filters

The filters handle the transformation of input scans $\mathcal{S}$ into filtered scans $\mathcal{S}'$. The scans acquired from the Velodyne sensor contain a very large number of points, of which a large part is redundant data. To reduce the computational time of ICP, a first filter is applied to sub-sample the point cloud by keeping one of every N points

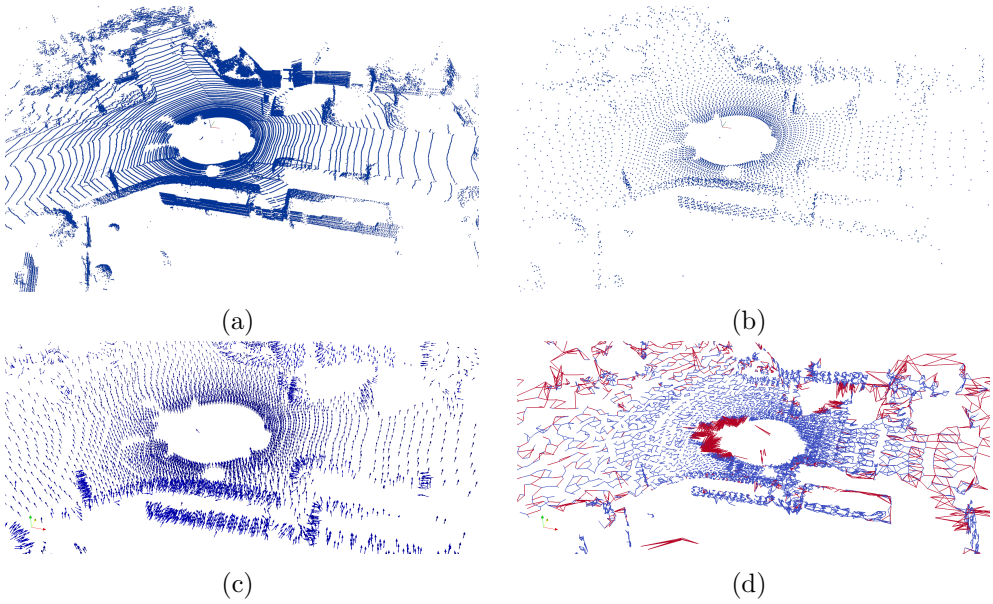


Figure 4.2: Typical scan acquired by a LIDAR and scans produced by the ICP pipeline. (a) Scan from a Velodyne HDL-64 sensor, from the KITTI dataset. Note how the scan lines gets more sparse with the distance from the sensor. (b) The same scan sub-sampled by the ICP filters (section 4.3.2.1). (c) Computed normals for the same scan (also section 4.3.2.1). (d) Good matches (blue) and rejected matches (red) between the same scan and a later one, after alignment. Good matches are given weight one and rejected matches are given weight zero, as explained in section 4.3.2.3.

in the scan. This systematic sub-sampling can be used because the scan data is somehow organized (points are stored in rows – if not, some randomness can be used to sub-sample the points). With Velodyne HDL 64 data, up to 80% to 90% of points can be removed without loosing the environment structure.

A second filter computes the normals for each point using *Principal Component Analysis* (PCA) with the K nearest-neighbors of each point. The good number of nearest neighbors depends on the resultant structure of the cloud after applying the first filter. Normals are important scan features for our ICP solution: they are used in the outlier rejection and distance function, as explained in sections 4.3.2.2 and 4.3.2.3.

The use of PCA alone provides only the plane that fits the point and its neighbors, thus there is an ambiguity on the direction of the normal vector that should be solved. The last two filters adjust the direction of the computed normals: one computes the observation vectors that points to the sensor origin from each point in the scan, and the second one orients the normal vector of each point so that the angle between the observation vector and the normals are minimal. This way all normals are oriented to the side of the plane that faces the sensor.

4.3.2.2 Match Function and Distance Function

The matching function $\mathcal{M} = \text{match}(\mathcal{P}, \mathcal{Q})$ pairs each point of the reading shape $\mathcal{P}$ with the 3 nearest neighbor points in the reference cloud $\mathcal{Q}$.

For the distance function $d(p, q)$ it is used the distance from a point p in the reading cloud to the plane defined by the normal of the matched point q in the reference cloud. This is known as *point-to-plane* distance, and is best suited for the environments that partly exhibit planar structures.

The decision to match the same point with several neighbors should accord with the distance function being used. We do multiple matching because matches of the same reading point to reference points that belong to the same plane (and thus have similar computed normals) still produces an accumulated error that is minimal. Also, in case outliers does not get eliminated by the outlier rejection function, we expect that multiple matching still provides matches to points that belongs to the correct plane, and can contribute to find the best transformation for the current iteration of the algorithm.

Note that by matching the point with its 3 nearest neighbors, there will be 3 times more matches to compute errors from. The decision to use 3 matches has shown to be a good compromise between the increased computational cost and added robustness.

4.3.2.3 Outlier Rejection

The rejection function $\mathcal{W} = \text{outlier}(\mathcal{M})$ uses a *hard outliers rejection*, meaning that the weights $w(p, q)$ for a match are either zero or one. We consider that even if noisy, the initial transformation is enough to provide several correct matches. Thus matches whose euclidean distance are greater than a threshold are rejected (i.e. given zero weight). Matches whose point normals present angles greater than a threshold are also rejected. This makes ICP to consider only matches between points that belong to planes oriented in a similar way, and avoid matches made in cluttered areas (e.g. vegetation), that impede a good convergence of the algorithm.

4.3.2.4 Convergence Tests

As an iterative algorithm, ICP needs some criteria to assess the minimization convergence. The main convergence test is the differential test, where we consider the ICP converged if the translational and rotational differences between transformation increments drops below given thresholds. This computed difference is normally smoothed through a few iterations to avoid oscillations around the cost function minimum. Also, we consider that the convergence failed if we reach a maximum number of allowed iterations or if the relative transformation between the initial and last computed transformation exceeds a maximum rotational and translational threshold.

Table 4.1: ICP Configuration for Velodyne HDL-64

ICP elements	Details
Filters	Keep one point in every 20 points Compute normals using 10 nearest neighbors Add observation vector pointing to sensor origin Orient normals using the observation vectors
Matching	Match to 3-nearest neighbors
Rejection	Remove pairs if distance is greater than 1 m Remove pairs if normal angle is greater than 60°
Distance	Use distance from the point to plane
Convergence	Success if relative motions goes below 0.01 m and 0.001 rad. Failure if reaches 80 iterations Failure if transform goes beyond 0.8 rad and 15 m

A summary of the various parameters and associated threshold values is shown in table 4.1.

4.4 Keyframes and Local Maps

An issue with Velodyne LIDARs (as with most LIDARs actually) is that the produced point clouds have a non uniform spatial resolution that reduces drastically with the distance from the sensor, as can be seen in fig. 4.2a. Thus, the overlap between different clouds acquired from distant poses may have very different spatial resolutions. As a result, points that are far from the sensor will have their normals computed using nearest neighbors that are farther from the considered points, potentially belonging to another plane, and thus reducing the precision of these normals. Another undesirable result is that matching may occur between points that does not correspond to the same scene element, increasing the difficulty to find a good transformation that aligns the clouds. An example of scan region that produce low quality normals can be seen in fig. 4.3.

4.4.1 Local maps

An essential prerequisite for the registration is to have enough overlap of the observed environment in both reading and reference scans. Otherwise the matching process can not produce enough matches with quality (*i.e.* correct matches, or matches to points near the correct one), compromising the quality of the resulting transformation.

In our approach, reference clouds used by ICP are *local maps*, that are *the concatenation of n selected keyframe scans whose keyframes compose a connected*

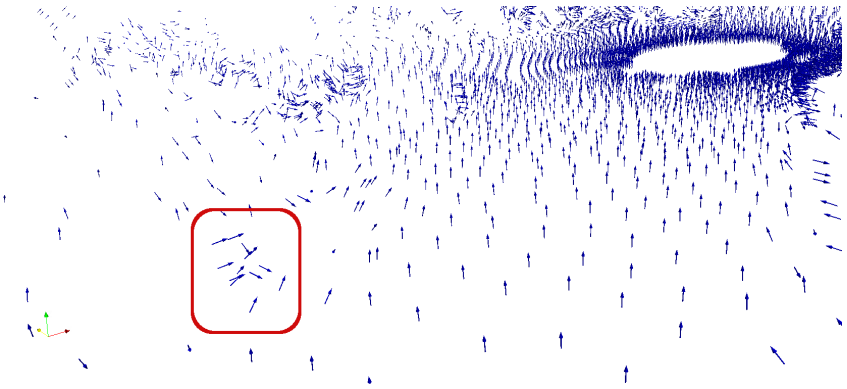


Figure 4.3: Example of region distant from the sensor origin where normals cannot be properly computed, *i.e.* nearest neighbors does not belong to the same plane.

sub-graph in the graph layer. The points of these scans are expressed in the keyframe of the local map that is the closest to the reading scan frame. The reading frame is usually the robot frame, but it can also be another keyframe when attempting to close a loop. Local maps are buffers ordered by the distance to the robot, in a way that if we push a keyframe to a full local map, the most distant keyframe will be removed from it.

The overlap between the new scans and the current local map decreases as the robot moves towards the borders of the local map. In order to always have enough overlap, the overlap is estimated after every ICP call, and a new local map is composed if this estimate drops below a threshold. The overlap estimation used is simply the ratio between the number of matched points in the filtered reading scan after outlier rejection and the total number of points in the filtered reading scan.

4.4.2 Controlling the building of local maps

To control the local map building we use the concept of *state*: the robot may be *exploring* if it is moving towards a unknown part of the environment; or it may be *revisiting* if it is moving in a region that was already mapped. The selection of keyframes used to build the local maps behave differently depending on the current state. The idea is that when exploring we prioritize the creation of new keyframes, adding the new keyframes to the system and to the current local map. When revisiting, we compose the local map starting from the closest keyframe and expanding with the nearest keyframes while following the graph topology. The use of states forbids the system to re-enter a mapped region without a loop being closed in the graph level, as explained in section 4.5.

When the first scan is acquired an initial local map is composed from that scan only and the system initial state is set to exploring. The system keeps acquiring scans and re-localizing the robot by means of ICP between the scans and local map,

until the overlap drops below a threshold.

Then, if exploring, it verifies if the last local map does not provide a better overlap, meaning that the robot is coming back along the explored trajectory. In this case the last local map is re-composed and the system is set to revisiting. If not, a new keyframe is created and a loop closing test is performed using this new keyframe (more about this on section 4.5). In a successful loop closing, the state also changes to revisiting. Otherwise the new keyframe is added to the local map (removing the more distant keyframe from it), meaning that the robot is moving towards an unknown region. The state is kept as exploring.

If the robot is revisiting, we search for another set of keyframes that compose a local map with enough overlap to re-localize the robot. This set should contain the closest keyframe and the ones in the vicinity of the closest, following the graph topology. If that is the case, the new local map is composed from this set and the robot keeps revisiting. Otherwise we create a new keyframe and test for loop closing, and then proceed as explained before according to the loop closing test result. Figure 4.5 shows examples of local maps of size $n = 3$ for both system states.

By following this procedure for creating local maps, we see that at the very beginning the local map is composed of only one scan, and it is concatenated with a new scan every time a new keyframe is created. These first runs of ICP with a reduced local map are critical for the system, since they are performed with reference clouds that exhibits the spatial resolution problem described before.

Note that no new keyframes will be added when the robot moves in already mapped regions. This allows the number of keyframes to scale with the size of the environment being explored, instead of the length of the robot path, as done in FrameSLAM [Konolige 2008]. Note also that when the robot starts again to explore the environment from a region it was revisiting, new keyframes being created during this exploration will gradually replace the keyframes that are more distant to the robot in the local map of the last revisited region. This is shown in fig. 4.4.

The keyframes are important in the system because it is through them that the information flows between the ICP localization level and the pose graph level. Every time a new keyframe is created, a pose variable is added to the pose graph, and the last transformation from ICP along with the estimation of its covariance is added as a constraint (factor) in the pose graph. The keyframes are always positioned at the poses represented by their associated pose variables in the pose graph. Updates on these variables happens when loop closings are performed, as explained in the following section.

4.5 Loop Closing

Every time a new keyframe is created the system verifies if a loop can be closed, i.e. if ICP is able to align the newest keyframe scan to a local map from region the robot is potentially revisiting. We assume that the robot is a ground vehicle

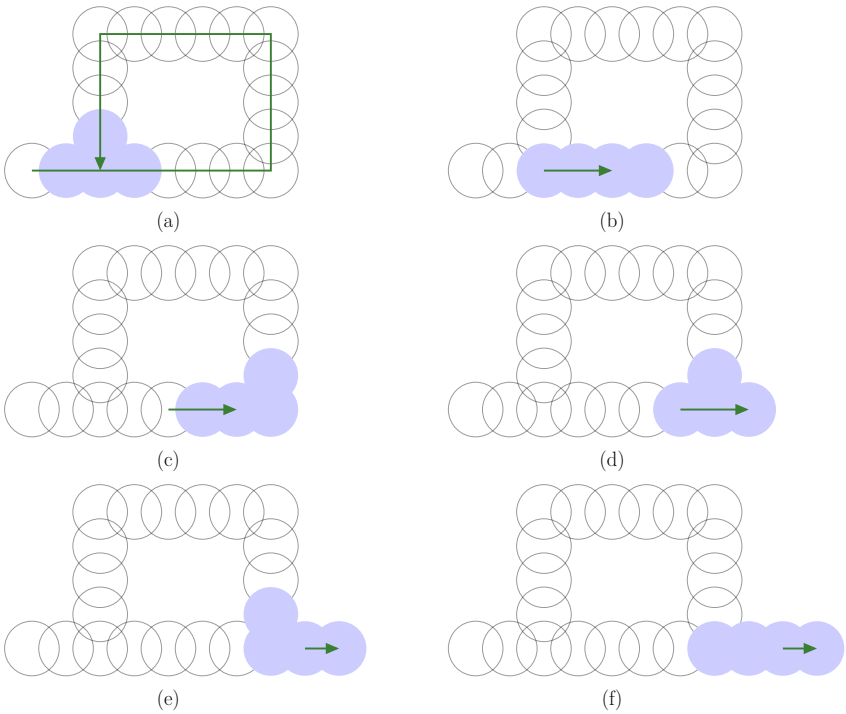


Figure 4.4: Example of selected local maps of size 4: (a) Local map after closing the loop; (b-c) Robot moves while revisiting, changing local maps when the overlap gets low; (d-f) Robot starts explore environment again, replacing distant keyframes from the the local map with newly created ones.

and that the environment being explored does not contain tunnels or bridges: if the robot returns to a previously visited place it should be at the same elevation as before.

To perform the test, first the system selects a loop closing candidate keyframe. The candidate is the closest keyframe inside the certain distance. Due to the mentioned assumption, the distances are computed only in the 2D xy-plane of the origin keyframe. Also, we do not consider the m most recent keyframes as possible candidates. This favors the creation of longer loops that provides better corrections of odometry drift. This set of m keyframes is called *no-loop window*, and is set to be greater than the local map size. Once a candidate is selected, the loop closing local map is built around it, using close keyframes and following the graph topology. This procedure is illustrated in fig. 4.6.

The loop closing ICP is then performed using the candidate local map and the new keyframe scan. The initial transformation for this ICP is the relative pose between the candidate frame and new keyframe projected into the 2D xy-plane of the origin. Due to accumulated errors along the loop, the filters for the loop closing ICP should be less restrictive than the one used to re-localize the robot during exploration. If not, there is the risk that there will not be enough matches to allow the transformation to be found. This is normally done by relaxing (increasing the

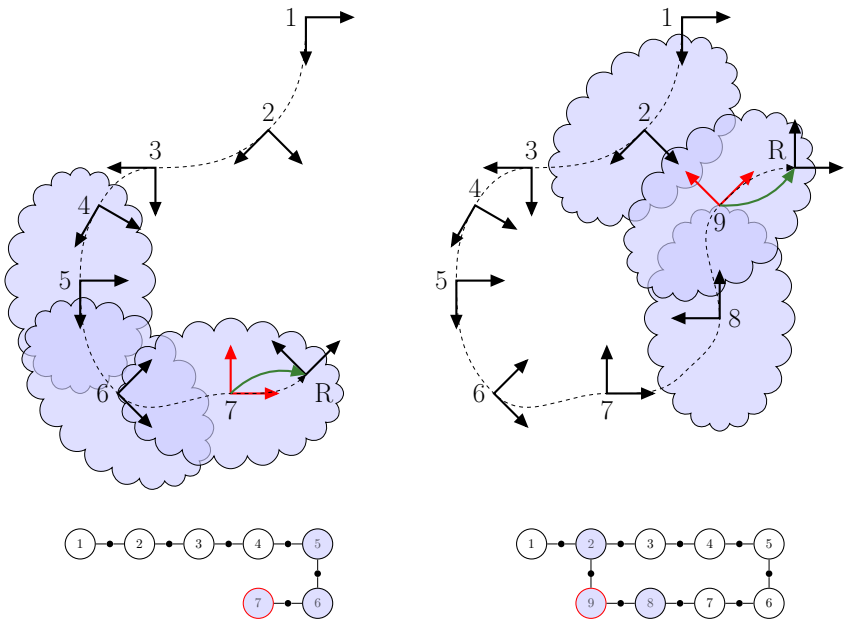


Figure 4.5: Local map building according to state, with respective graphs. The robot trajectory is represented by the dashed line. Clouds represent keyframe scans. Keyframes are identified by numbers, and R represents the robot frame. Local maps are composed of 3 scans, and are expressed in the frames shown in red. Green arrows represents the transformation from the local map to robot frame. For clarity reading scans are not shown. **Left**: Robot explores new part of the environment, using most recent keyframe scans as local map (5, 6 and 7); **Right**: local map used when the robot is revisiting a mapped environment, right after a loop closing (keyframes 2, 8 and 9). Note how the local map on the right uses the keyframe 2, that is far back on the robot path, but connected in the graph through the loop closing link.

value of) the parameters of the outlier rejection. The different values used in our loop closing ICP is shown in table 4.2.

Besides the normal ICP convergence tests described on section 4.3.2.4, the final value for the error function is also analyzed, and the loop closing is considered successful if it also lies below a threshold. This is important because sometimes the initial transform does not provide enough overlap between the scans of the loop closing ICP, or has a large angular errors. In this situation ICP needs many more iterations to converge. It may then reach the maximum number of allowed iterations before converging, or it may be trapped into a local minimum. This additional test helps to avoid closing the loop with an incorrect transformation.

It should be noted that since this error is the sum of weighted distances of the matches between the two clouds, it depends on the filtered clouds, the matches, and the weights computed by the outlier rejection. The system designer should compute the average error for clouds that are verified to be aligned (probably using a software to visualize and assess the alignment) to select an appropriate value for this threshold.

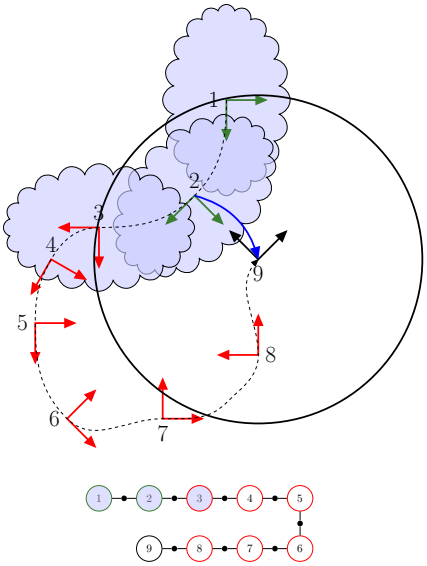


Figure 4.6: Loop closing candidate selection and candidate local map. The circle shows the maximum distance from keyframe 9 to allow the loop closing. Keyframes 3 and 8 are not possible candidates because they are inside the no loop window (the keyframes shown in red). Possible candidates are keyframes 1 and 2, in green. The loop closing candidate is keyframe 2 since it is the closest between the possible candidates. A local map is build around it using the current connectivity of the graph layer, selecting the scans from keyframes 1, 2 and 3.

A successful loop closing ICP then adds a loop closing constraint to the factor graph between the variables associated to the newest keyframe and the candidate keyframe. The optimization using the graph data can be performed, which updates the variables in a way that the overall robot drift along the loop is reduced. The optimization is reflected back to the ICP level by repositioning the keyframes with the poses in the associated graph values.

The use of states to build local maps for re-localization also plays a role in the loop closing process. If the local map could be always built from any set of keyframes, there could be a situation where the current scan has overlap to both the current local map and a potential loop closing local map, and the system would chose to use the second to re-localize the robot into, without creating the loop in

Table 4.2: Modified ICP Configuration for Velodyne HDL-64 used in Loop Closing

Modified ICP elements	**Details**
Rejection	Remove pairs if distance is greater than 5 m Remove pairs if normal angle is greater than 90°

(Elements not mentioned here are the same of table 4.1)

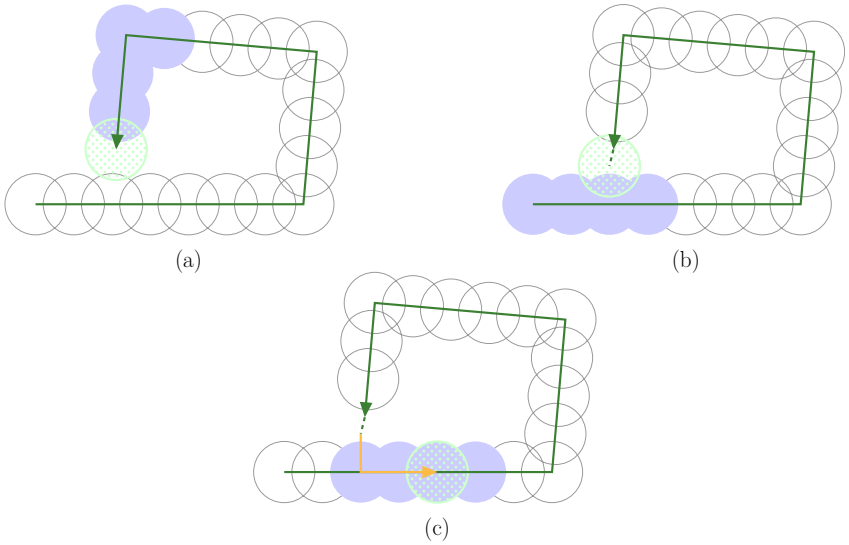


Figure 4.7: Example of missed loop closing that is avoided by using states. (a) The robot moves in the environment while accumulating drift in this pose estimation, and its current scan overlaps a previous visited place; (b) If the system did not prioritize the creation of new keyframes when exploring, a local map in the revising region could be used to re-localize the robot, creating a gap in the robot trajectory; (c) The robot continues revising, missing the loop closing and creating an inconsistent map.

the graph level. This is shown in fig. 4.7. By using states we favor the creation of new keyframes when exploring, and possibly closing the loop in the graph.

4.6 Pose-Graph Building

The second main feature of the proposed system is the use of a pose graph to reduce the accumulated errors from ICP odometry when a loop closing is detected. A pose graph is a special case of a factor graph when all the variables being estimated are robot poses along the robot trajectory, and the factors are relative pose measurements between these poses.

The main theory behind the graph based optimization was already explained in chapter 2. In the proposed approach of this contribution, each variable in the pose graph represents a 3D pose and is associated with a keyframe of the ICP layer, and each factor is a constraint between two variables. Each factor encodes the following error function:

$$e(x_a, x_b, z_{ab}) = (\ominus z_{ab}) \oplus ((\ominus x_a) \oplus x_b) \tag{4.5}$$

where $\ominus$ and $\oplus$ are respectively the inverse and composition operators, a and b the indexes of keyframes before and after the movement, and z_{ab} is the transformation obtained from the ICP, when exploring or when closing the loop, between the scan from keyframe b and the local map expressed in keyframe a. Since the keyframes are 3D poses, the error is in fact computed in a local tangent space as explained in

chapter 2.

Each measurement used in a factor should also be associated to a covariance matrix Σ representing the measurement error. Since the ICP only produces transformations as outputs, the covariance matrix Σ_{ab} for z_{ab} estimated using the technique proposed on [Censi 2007].

4.7 Experiments

4.7.1 Setup

The overall flow chart of the implemented system is shown in fig. 4.8. It exploits *libpointmatcher* [Pomerleau 2013a] for the scan registration in the ICP layer, and *GTSAM* [Dellaert 2012] for the optimizations in the graph layer. Experiments were performed using a dataset obtained on a parking lot using one of the ground robot of LAAS, and later with the sequence `2011_10_03_drive_0027` of the KITTI dataset [Geiger 2013]. The details of each dataset is shown in table 4.3.

Table 4.3: Description of datasets used in this work

Dataset Details	**Datasets**	
	LAAS Parking	**KITTI**
LIDAR sensor	Velodyne HDL 64	Velodyne HDL 64
Environment Type	Parking	Urban area
Trajectory length	306 m	3.72 km
Ground truth from	Centimeter-level GPS (Novatel RTK)	Fusion of LIDAR and Centimeter-level GPS (XTS RT 3003)
Average speed	1 m/s	28.46 km/h
No. of scans	306	4541

Noisy odometry measurements were drawn from the ground truth of both datasets in order to provide a realistic initial transformation to ICP when the robot explores the environment. A Gaussian noise of $\mathcal{N}(\mu, \sigma) = \mathcal{N}(0, 0.1\,\text{m})$ was set for each translation component and $\mathcal{N}(0, 1°)$ for each orientation component of both datasets. The scans are converted from the sensor frame to the robot frame after filtering but before the calls to ICP, in a way that the resulting transformations represent robot frame displacements, instead of sensor frame displacements.

4.7.2 Parameters

The system thresholds used for each dataset are shown in table 4.4. They required some tuning, that is discussed below.

The condition to create a new keyframe is to have the overlap between the current scan going below a threshold. This threshold was set to a relative high

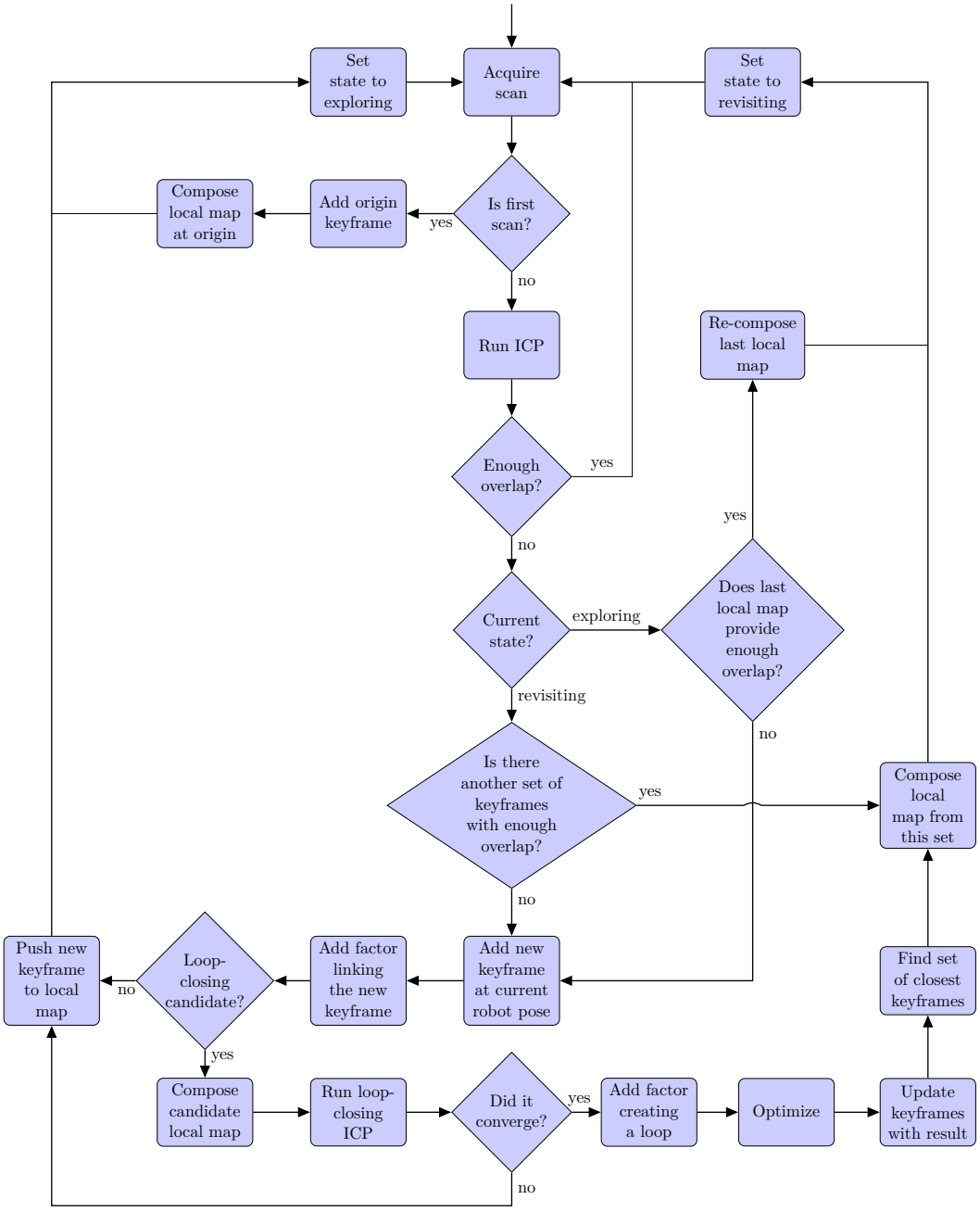


Figure 4.8: Simplified flow chart for the algorithm proposed in this chapter.

Table 4.4: Thresholds Used In The System

Threshold	Datasets	
	LAAS Parking	KITTI
Create keyframe if overlap below	75 %	75 %
Local map size	3	3
No-loop window size	6	10
Max distance to loop closing candidate	5 m	15 m
Max residual error for loop closing	2000 m	2000 m

value to guarantee that when going below this threshold, the overlap is still high enough for the resulting transformation to be trusted and used to build the factors in the graph. The overlap being simply the ratio between the number of matched points over the number of points in the current cloud, it is highly influenced by the matcher and outliers rejection in the ICP pipeline, and consequently by the environment. The overlap computed from scans acquired from near poses may present small values if most of the points lay near the sensor (*i.e.* narrow corridor-like environment) or if the environment has elements without enough structure to allow normal computation. This second example is shown in fig. 4.9. The increase in frequency of keyframe creation can be seen also in the plots shown in fig. 4.10. With the thresholds shown in table 4.4, the final number of created keyframes was 30 for the LAAS dataset, and 443 for the KITTI dataset.

The size for the local map was set to three after visual verification that it was enough to reduce the non-uniform density problem of the acquired scans, although in the KITTI dataset, one could benefit of larger local maps in the regions with vegetation as shown in fig. 4.9. It is also worth mentioning that larger local maps would increase the computation time of each ICP (larger reference clouds results into more computation for the matchings) and to re-build the local maps when needed.

The no-loop window size and the max distance for the loop closing candidate were set to a higher value for the KITTI dataset because there the Velodyne sensor is positioned higher with respect to the environment (on top of a car) than for the LAAS dataset (on top of a small robot, figure 1.1 page 2). The higher sensor position produces scans that cover larger areas, so we can increase the distance threshold while increasing also the no-loop window threshold to reduce the creation of small loops.

It is important to notice that while the no-loop window reduces the creation of smaller loops, it does not avoid it completely. When tuning the system some small loops along the robot path were observed in runs with with the KITTI dataset. The main drawback of closing small loops is triggering an optimization that cost some extra computational resources and that does not provide significant correction of the drift error.

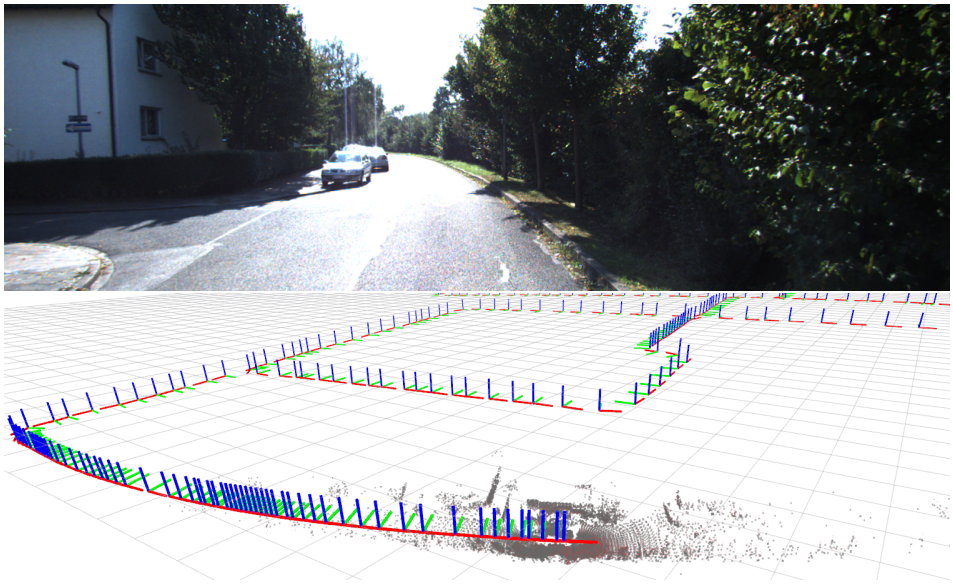


Figure 4.9: Region of the KITTI dataset showing the influence of vegetation in the keyframe creation. **Top**: path in the dataset where vegetation is present on the right side of the road. **Bottom**: frequency of creation of keyframes in the region with vegetation is higher than other regions (compare with the regions far back in the robot trajectory). This happens because normals for points laying on vegetation cannot be properly computed, increasing the rejection of the matches. The increase in match rejection causes a reduction in the estimated overlap, triggering keyframe creation more often.

Finally, the max residual error for loop closing is set to be a little larger than the maximal residual from the local ICP. The rationale behind this decision is that since the loop closing ICP is less restrictive than the local ICP, its residual error is expected to be larger than the residuals for the local ICP. So we select a threshold that is larger but not too large to be all permissive. Plots showing the residual error for the local ICP for both datasets can be seen in fig. 4.11. We remind the reader that this value is the sum of the error of all the retained matches between the scans processes by ICP. Its large value is due to the number of points in the filtered scans (around 6000 point are kept in each scan) and the fact that we match each point to its 3 nearest neighbors (see table 4.1).

4.7.3 Results

The plots of the final estimated path for the LAAS dataset can be seen in fig. 4.12, and the plots of each variable in fig. 4.13. The only angle available in this dataset is the one around the z-axis, thus this is the only ground truth angle shown in fig. 4.13.

In the plots it can be seen that the path estimated using ICP only presents a significant translation drift along the z-axis. This drift is accentuated when the robot moves away from the origin, getting reduced as the robot moves back near

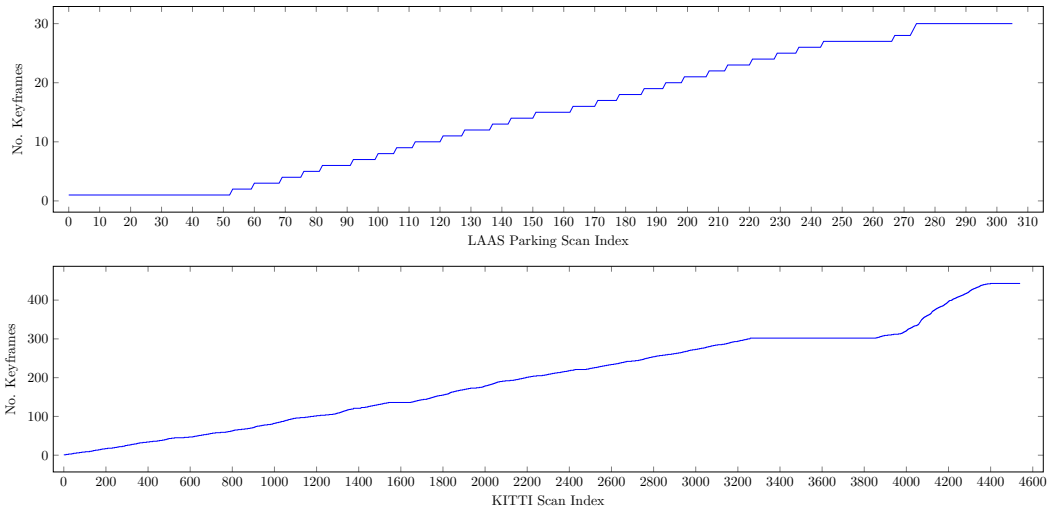


Figure 4.10: Plots of the number of keyframes per scan indexes. Long flat regions indicate that the robot was revisiting. The steep slope near the end of KITTI dataset shows the influence of the vegetation on the frequency at which keyframes are created.

the origin. This shows that it is being probably caused by accumulation of small errors in the orientation from the ICP. Since both paths estimated with and without the graph level follows the provided ground truth, as it can be seen from the yaw plot in fig. 4.13, it is mainly influenced by the errors in the roll and pitch angles, although the lack of ground truth on this variables make it impossible to confirm this hypothesis in this dataset. It is noticeable that through the use of the graph layer this drift is considerably reduced, although it is not completely eliminated.

To better assess the error in the proposed system, an analysis of the translation and rotation errors for the estimated path were performed. We used the same evaluation used in the *KITTI Vision Benchmark Suite* [Geiger 2012]. It is performed as follows:

- Search in the ground truth for the pose indexes that define the start and end points of all possible sub-paths of predefined lengths;
- For each sub-path, compute the last pose with relation to the first pose in both ground truth and estimated path, called *pose deltas*;
- For pair of pose deltas, compute the composition of the inverse of the estimated pose delta with the ground truth pose delta. The result is the *pose error*, and should be equal to the identity transformation for a perfect match in the sub-paths;
- For each pose error, compute the *translation error* as the module of the pose error, and the *rotation error* as the rotation angle of the pose error.
- Sum all the translation and rotation errors for the sub-paths with the same length to get the final error with relation to this specific length.

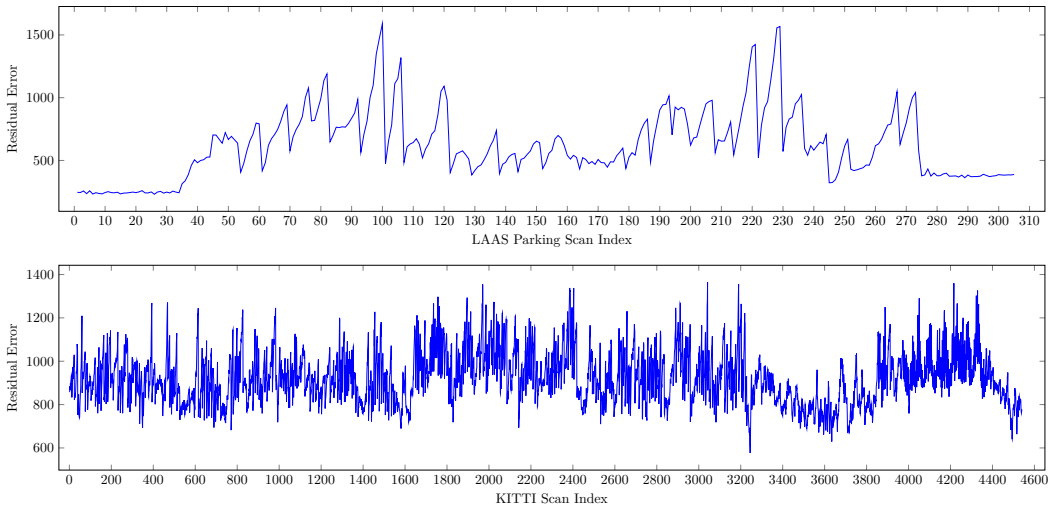


Figure 4.11: Plot of residual errors of local ICP for both LAAS and KITTI datasets. From these plots we can estimate a value for the max residual error allowed for the loop closing ICP. The saw-like pattern especially noticeable in the LAAS plot is a result of the robot moving away from the current local map, increasing the error, and then creating a new keyframe and re-building the local map, that drastically reduces the error.

The code for the evaluation explained above is available in the *KITTI odometry development kit*. It was adapted to evaluate the results for a dataset acquired from a slower robot traversing a shorter path, as it was the case in the LAAS dataset. The error plots can be seen in fig. 4.14 with relation to sub-path lengths, and in fig. 4.15 for the robot speed. Note that the curves represent the *error relative to the segment length*, *i.e.* the translation errors are given as percentage of the segment length, and rotation errors are given as degrees per segment length. This explains the tendency of the curves to decrease as longer segments are considered.

We can see that in general the errors are smaller after the loop closings, an exception being the rotational errors for small lengths, which were higher than before the loop closing. A similar behavior can be seen in the curves for the errors in relation to the robot speed.

The trajectory and variable plots for the KITTI dataset can be seen in fig. 4.12, and fig. 4.13. A video showing the system running on the KITTI dataset is available in `http://youtu.be/KYvOqUB_odg`.

Note the difference of scale between the plots for the LAAS dataset and the ones for the KITTI dataset, as the robot traversed a much longer path in the second case. It can be seen that the curves in these plots show similar features as the ones for the LAAS dataset, namely that they follow the ground truth curves for the x, y and *yaw* angles, and present a drift in z. The longer trajectory in this dataset causes a noticeable drift in the estimation of x and y for the ICP only case. By comparing the estimated roll and pitch angles with the ground truth, one can see that they are estimated with some errors, which is the main source of drift that

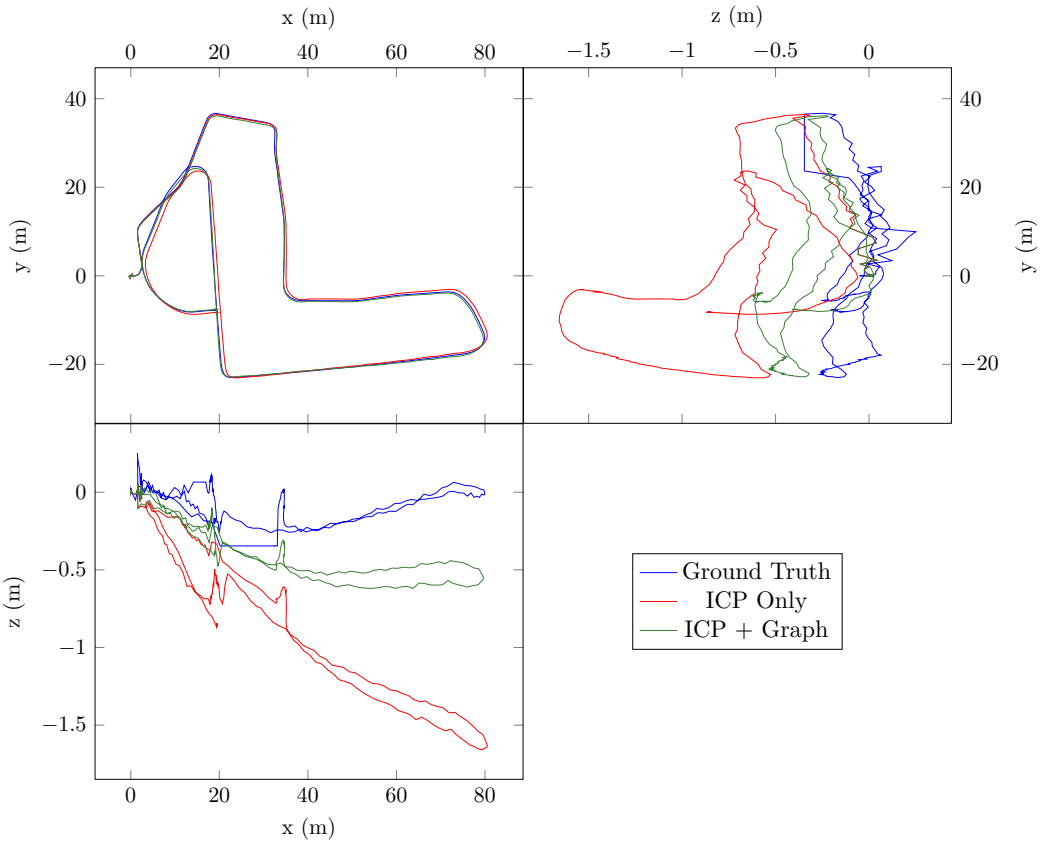


Figure 4.12: Plot of final trajectories for the LAAS parking dataset. Note the drift in the z-axis, which is smaller with the combination of ICP and Pose-graph optimization than when using only ICP.

accumulates along the z-axis.

One can see that this drift is larger when away from the origin, as it was seen in the results with the LAAS dataset. Nevertheless, the robot was able to close a loop with the origin near the end of its trajectory (loop closing 5). The visualization of this loop closing can be seen in fig. 4.18. Again, the overall drift is smaller when the estimation is performed with the graph layer.

The error plots for the KITTI dataset can be seen in fig. 4.19 and fig. 4.20. Five loop closings were performed along the robot trajectory. One can see that the error is reduced, excepted after the forth loop closing. The error as a function of the robot speed also show that the error was reduced after the loop closings.

4.8 Discussion

This chapter presented a system that uses transformations computed from ICP to feed a pose graph structure, that in turn is used on loop closings to build an

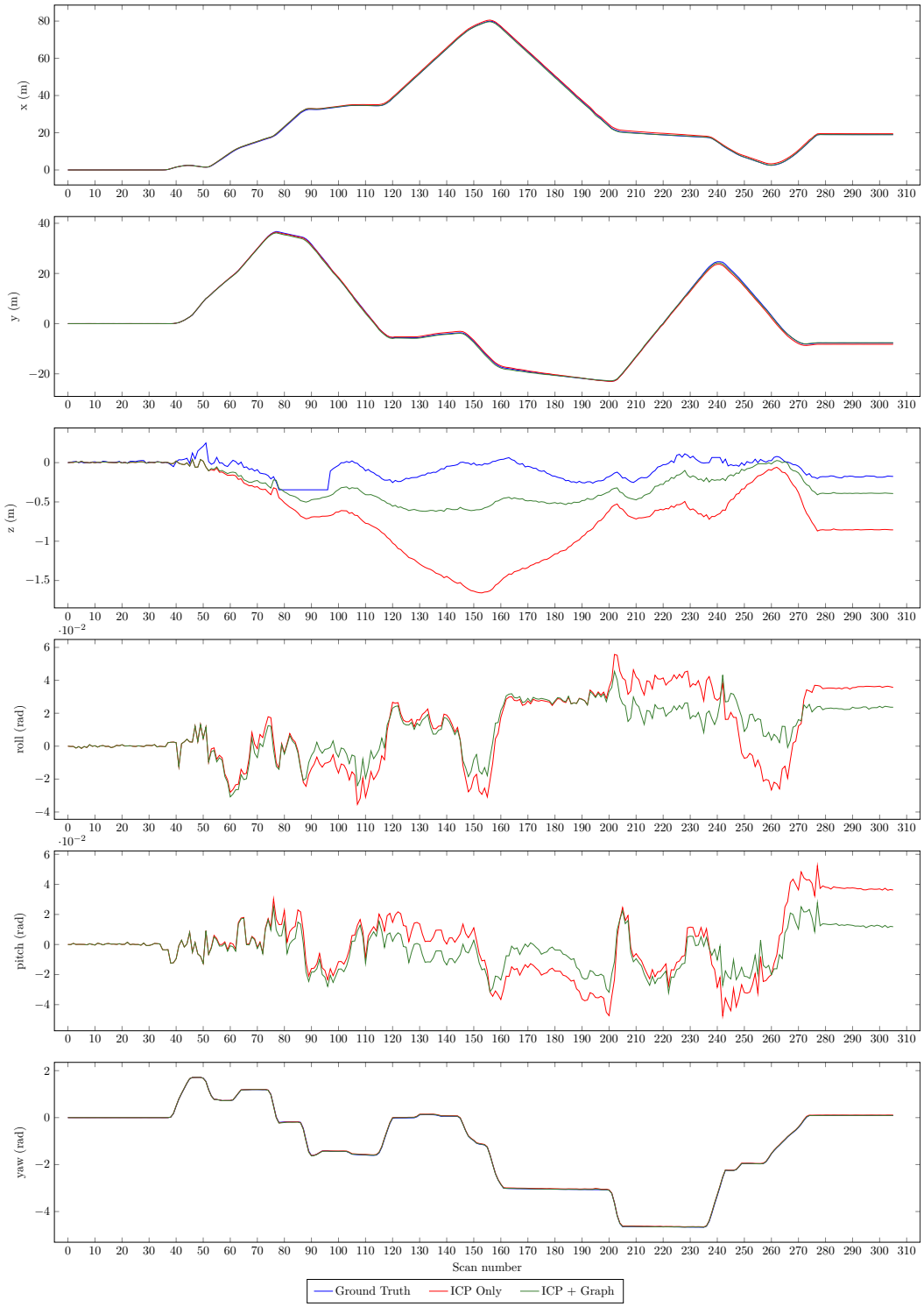


Figure 4.13: Plot of estimated pose parameters for the LAAS parking dataset. The ground truth for roll and pitch angles is not available. Note a GPS acquisition problem from scans 79 to 97, where the value for z-axis is kept fixed at -0.345777.

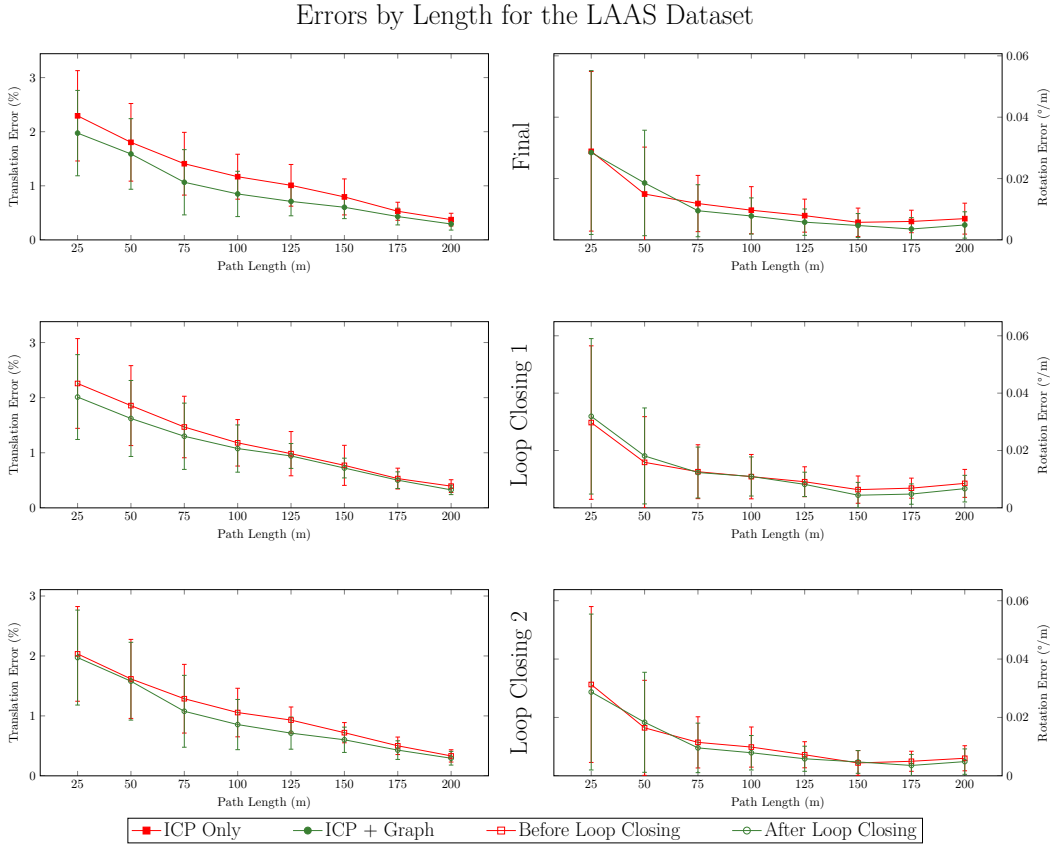


Figure 4.14: Plot of relative translational and rotational errors by the path length for the LAAS dataset. Only the orientation around the z-axis was considered in the rotational error due to the lack of ground truth for the other two angles. See text for details.

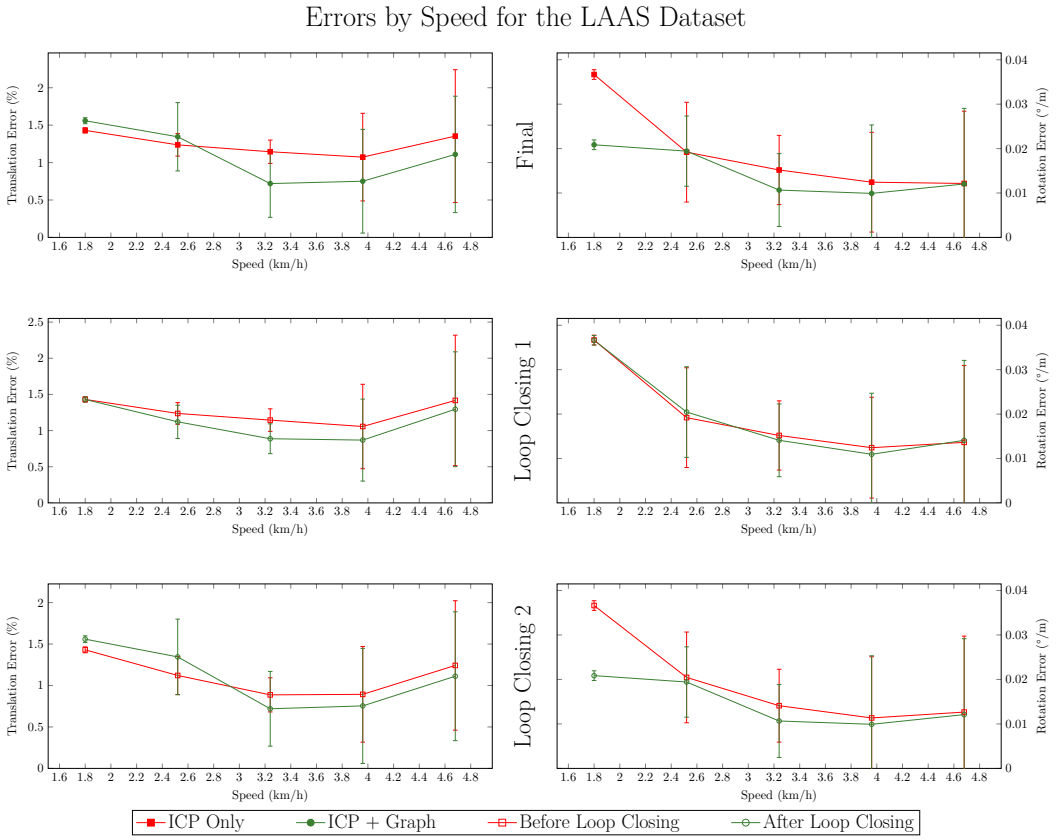


Figure 4.15: Plot of relative translational and rotational errors by the robot speed for the LAAS dataset. Only the orientation around z-axis was considered in the rotational error due to lack of ground truth for the other two angles. See text for details.

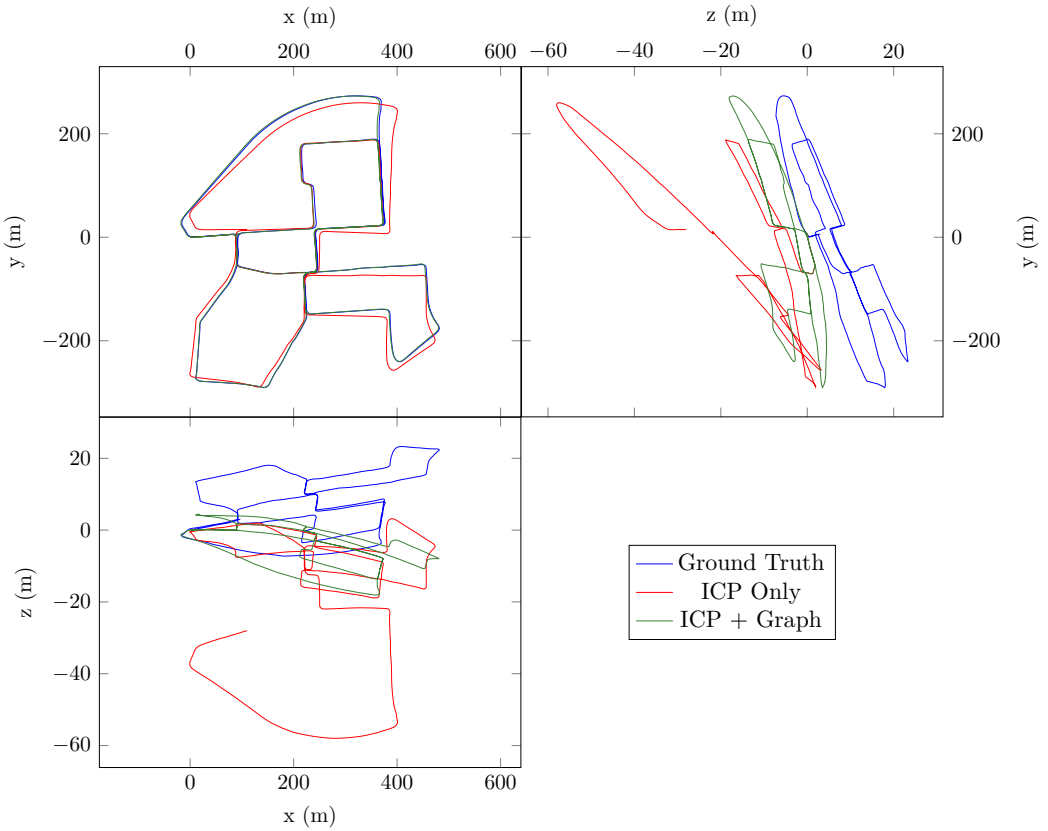


Figure 4.16: Plot of final trajectories for the KITTI dataset. The trajectory traversed by the robot is considerably longer than the LAAS dataset.The visible deformations when using ICP only is caused mostly by accumulated angular errors, that makes the estimation drift along the z-axis. A smaller drift occur when using the ICP together with the pose graph optimization.

optimization problem that provide updates of *keyframes* selected along the robot trajectory. These updates correct the built map of the environment and reduce the accumulated errors from the ICP odometry. The chapter also detailed the ICP configuration for Velodyne HDL-64 sensor.

We now discuss various aspects of this work, sketching possible improvements.

4.8.1 Comparison wih the state of the art

The contribution presented here has similarities with the work in [Borrmann 2008] (a 3D extension of the work presented in [Lu 1997] for the 2D case) that also uses ICP to localize the robot and achieves a global consistent map on loop closures using graph based optimization. The main difference between this work and [Borrmann 2008] is the way the optimization problem is constructed: in [Borrmann 2008] the optimization finds the robot poses that minimizes the positional error of points matched between two neighbor scans in the graph, resulting

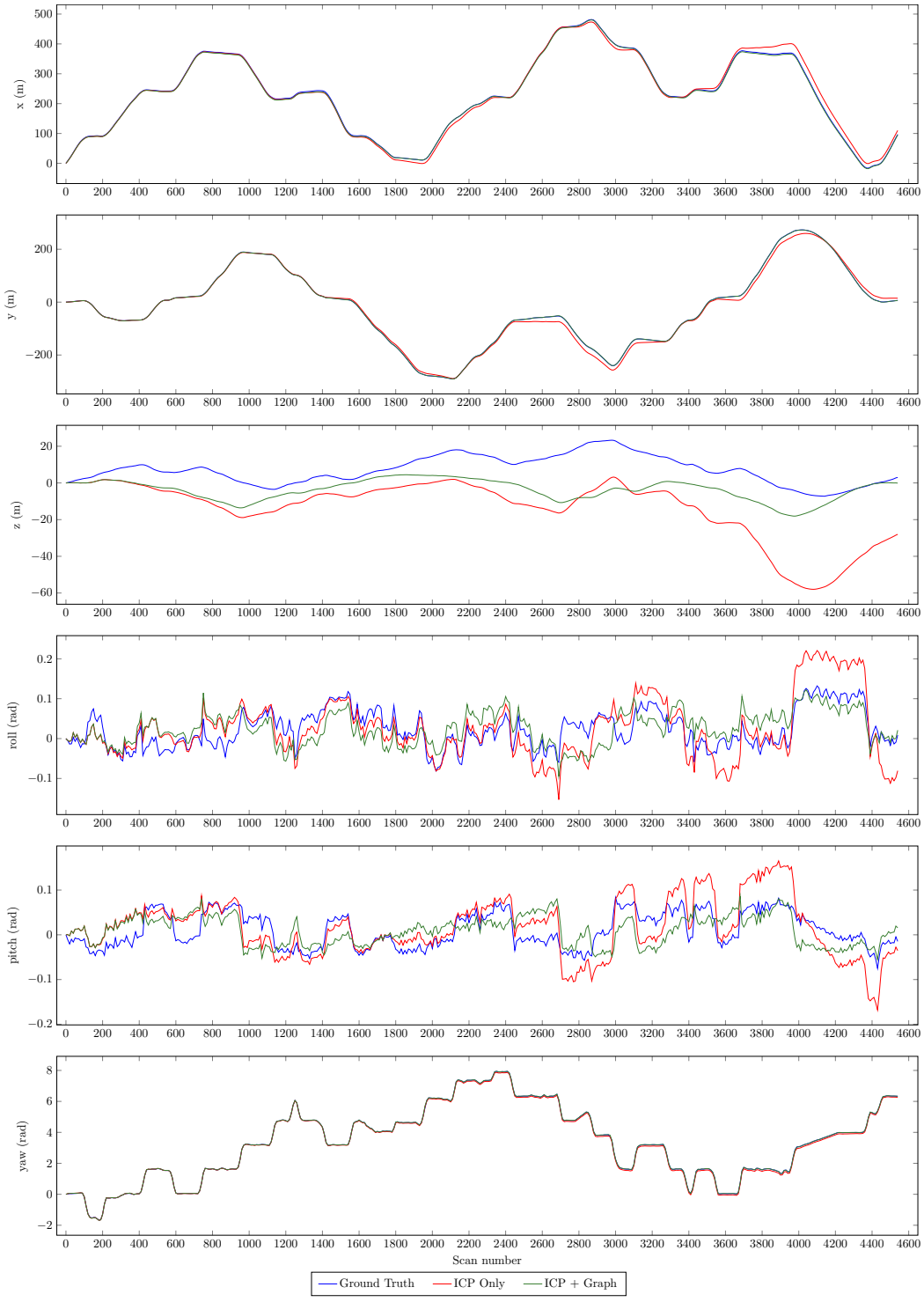
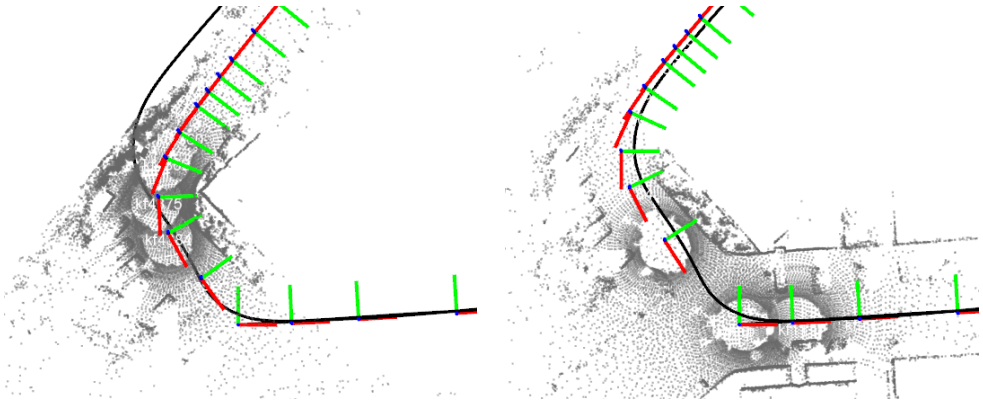


Figure 4.17: Plot of estimated pose parameters for the KITTI dataset.



(a) before (b) after

Figure 4.18: Top-view before and after closing the loop at the origin of the KITTI dataset. The black line shows the ground truth, and the current local map is shown in gray. Compare the correction in the keyframe positions, and the local maps used before and after the loop closure.

in very large linearized system to be solved at every iteration, and requiring a large amount of time to converge to the solution. In our work the transformations between keyframes outputed from ICP are used as measurements between keyframes in the graph, and the covariances are estimated as proposed by [Censi 2007]. This leads to a system that is faster to solve, and may benefit from the new techniques based on factor graphs, like ISAM2.

One interesting feature of [Borrmann 2008] is that edges may be added to or removed from the graph in between the optimization iterations, while in this work the graph is static during optimization. Other differences are: we use an overlap criteria to create new keyframes, in contrast to a fixed distance criteria; and the restriction to create new keyframes only when exploring new parts of the environment, that was not addressed in [Borrmann 2008].

The overlap threshold used to define new keyframes is a critical parameter of the system. This is due to the way the overlap is estimated and the influence that the filters have in the estimation: hard rejection based on normal angles helps to avoid unexpected drifts caused by vegetation, but also makes the overlap estimation go fast below the threshold. This results in keyframes that are too close each other, and consequently local maps using these keyframes cover smaller areas of the environment. A more robust way to compute the overlap between shapes would benefit the proposed system.

4.8.2 Avoiding ICP local minima

Even if the proposed system improves the estimation of the robot trajectory through loop closings, it remains prone to the ICP being trapped in a local minimum: the resulting transformations in such cases are propagated to the graph level and cause discrepancies in the final map. Local minima may occur when the environ-

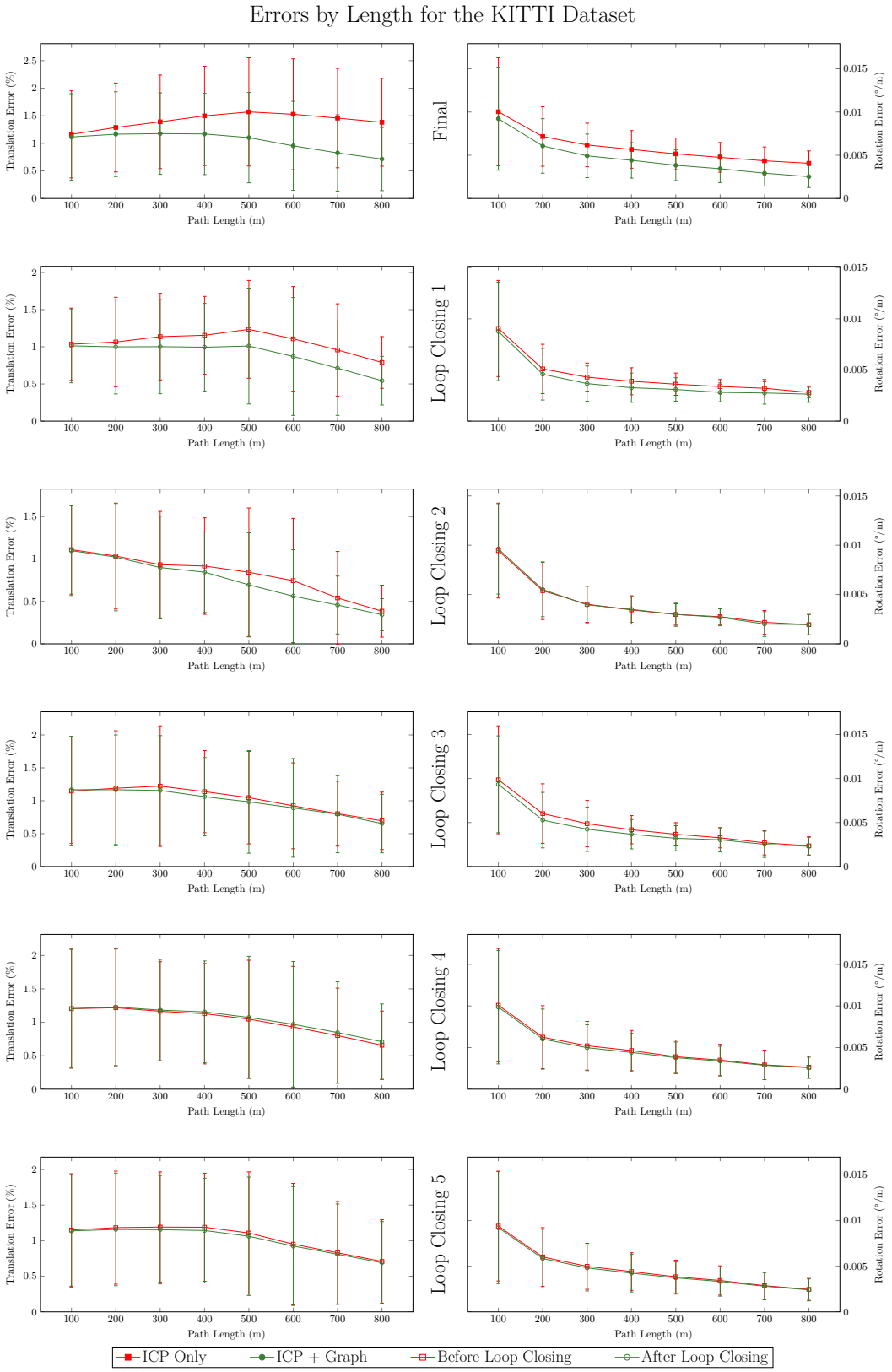


Figure 4.19: Plot of translational and rotational errors by the path length for the KITTI dataset. See text for details.

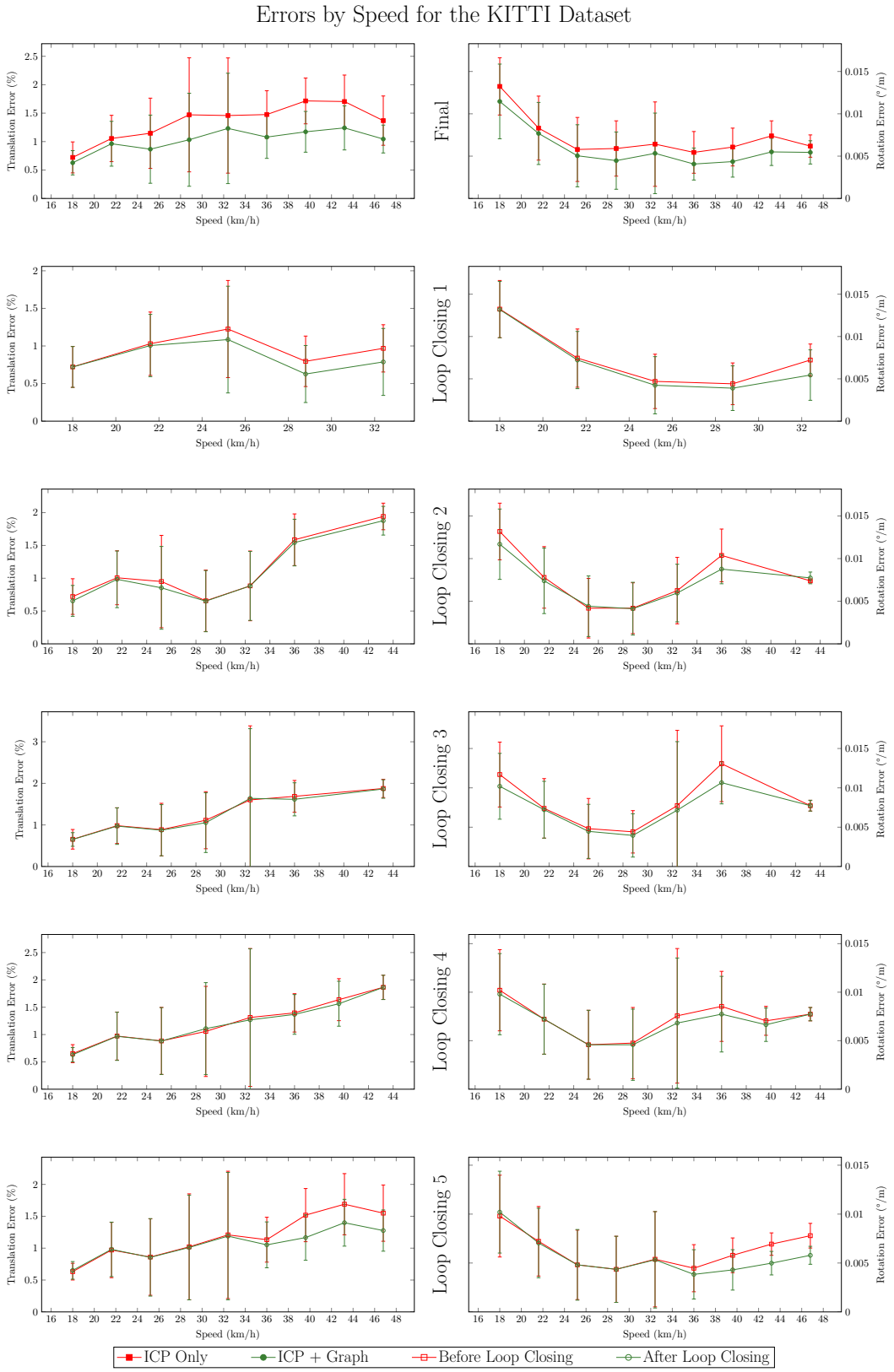


Figure 4.20: Plot of translational and rotational errors by the robot speed for the KITTI dataset. See text for details.

ment is mostly symmetric, or if it has repetitive patterns. They can be avoided with good initial estimations for the registration, for example exploiting IMU-based initial estimations. IMU data can also be used with pre-integrated IMU factors [Forster 2015, Lupton 2012] that can be added to the graph as constraints between keyframe variables instead of ICP-based ones. In this case, more variables would need to be added per keyframe (namely the robot velocity, and the accelerometer and gyrometer biases) and the graph would not be a pose graph anymore. Still the ICP transforms could be used to create small loop closings between keyframe poses along the robot path, helping to constrain these extra variables and allowing successful optimizations in the graph level. This idea is shown in fig. 4.21.

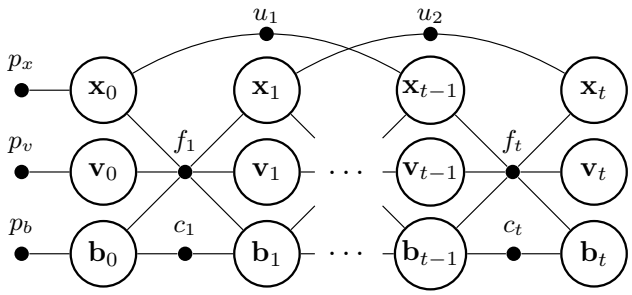


Figure 4.21: Example of factor graph that uses the pre-integrated IMU factors [Forster 2015, Lupton 2012]. Estimated variables are the robot poses $\mathbf{x}_i$, robot velocity $\mathbf{v}_i$ and IMU linear and angular biases $\mathbf{b}_i$. $p_{x,v,b}$ represents priors over the initial state variables. Factors f_i are constraints over sequential robot states based on pre-integration of IMU data, and factors c_i models the biases dynamics as random walks (*i.e.* allow them to change a little over time, according to some error encoded in the factors). Finally, the factors u_i are loop closing factors between the connected poses. They may be computed from ICP between overlapping scans along the robot trajectory, and are important to constrain the state variables composing the loop.

The initial 3D transformation used in the loop closing ICP is computed from the projections of the candidate and newly created keyframe on the xy-plane of the origin keyframe, *i.e.* it is a 3D transformation but confined to this plane. This transformation must provide enough overlap between both reference and reading scans passed to the loop closing ICP, as it is always the case with ICP algorithm. This strategy may fail if the robot accumulates too much angular errors along the loop, and if this angular error makes the robot position drift on z-axis, like climbing up a hill or going down a valley. In this case, if the robot approaches a loop closing point, the projected 2D distance would be larger than it should, and thus no candidate would be found, or the initial transformation for the loop closing ICP would not provide enough overlap. In such a case, the system would not be able to close the loop. This can be avoided with an ICP configuration that provides low error on pitch and roll angles.

For ICP it is less difficult to correct the translation that aligns the clouds than the orientation, which is the second main source of local minima. Thus, another

problem may happen when trying to close the loop is that the initial yaw angle is far from the one that aligns the clouds. If computational resources are available, one strategy to overcome this problem is to perform several loop closing ICPs with different initial yaw angles, and compare the residual error from the ones that converged, choosing the one with least error. Since this multiple ICP calls takes longer, it may be performed in different threads on a multi-core computer, to avoid blocking the ICP used to re-localize the robot locally. Also, since the residual error depends on the outlier weights, the outlier rejection filter should be much less restrictive, notably not rejecting by angular difference between the normals. If not, the residual error for a transformation that provides a bad alignment may be lower than a better transformation, because it has less matches contributing to compute this error (*i.e.* most of match distances were multiplied by zero in eq. (4.2)).

4.8.3 Estimating ICP relative motion errors

During the experiments, the covariances estimated with the method in [Censi 2007] and that are used to create factors in the graph were found to be remarkably small.

To assess the quality of the covariance estimation, transformations were computed with ICP using 33 scans captured with the robot standing still, so the expected transformation is the identity transformation. The result can be seen in table 4.5. The standard deviations computed using the method proposed by [Censi 2007] are at least one order of magnitude smaller than the ones computed by comparing the ICP results with the identity transformation. Although we acknowledge that the quality of the transformations obtained in this experiment depends on many factors, notably the ICP pipeline and the environment described by the scans, it still shows that covariance estimation using [Censi 2007] may lead to overconfident systems.

Table 4.5: Comparison of standard deviation for transformations computed from ICP using data acquired with a stationary robot.

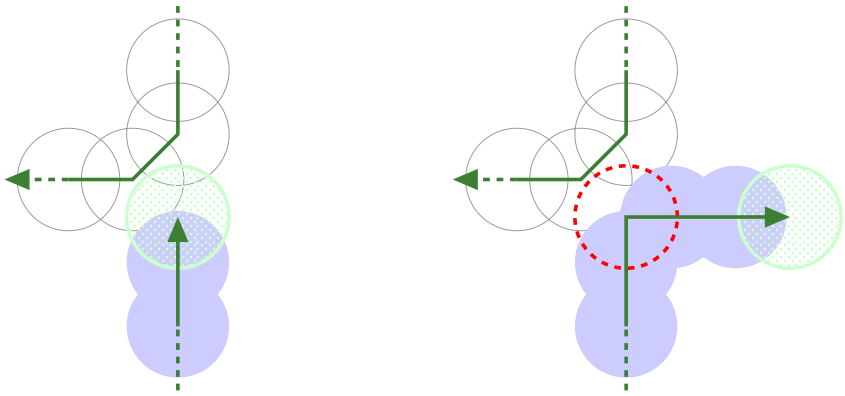
Pose component	**Standard deviation**	
	From ICP result	**Using [Censi 2007]**
x	3.231×10^{-3}	4.703×10^{-4}
y	4.210×10^{-3}	5.949×10^{-4}
z	4.962×10^{-3}	7.316×10^{-5}
roll	5.276×10^{-4}	1.498×10^{-5}
pitch	4.624×10^{-4}	1.778×10^{-5}
yaw	5.494×10^{-4}	3.334×10^{-5}

Our system show good results using the errors as computed in [Censi 2007], but there is a risk that the use of such small covariances may lead to pose graphs in which the optimization matrix is ill-posed and cannot be solved. A better error model should be exploited: building an empirical model that relates the errors to

some characteristics of the ICP process, or even to some signatures computed on the scans that relates the type of environment in which the robot is to the precision on the estimates of each estimated degree of freedom is an interesting work direction to explore.

4.8.4 Detecting loop-closures

The test for loop closing is only performed upon creation of a new keyframe in the graph level. Thus, a loop closing is dependent on the overlap threshold (to create the keyframes) and the candidate selection test, other than the ICP convergence itself. An alternative way to perform loop closings would be as follows: always verify the distances between the robot pose and keyframes, and/or check if there is an overlap between the last scan and keyframe scans outside the *no-loop window*. In a positive case, trigger the loop closing ICP and, upon convergence, create the new keyframe, close the loop in the graph, and optimize. In summary, this alternative strategy would force the creation of a keyframe that would be used to close a loop. This alternative is more costly, due to the computation of distances and overlaps for each scan, but reduces the risk of missing a loop closing in situations like the one described in fig. 4.22.



(a) (b)

Figure 4.22: Situation where the proposed system may miss a loop closing. (a) The robot passes through a loop closing region, but does not create the keyframe because it still has enough overlap with the current local map. (b) When the keyframe is created later, the associated keyframe scan can not be used to close the loop, and the robot misses the loop closing point (red dashed circle).

The use of local maps helps to overcome difficulties with the alignment of velodyne scans with ICP, notably due to the non-uniform density of points in the scans. Using local maps also reduces the chance of having reference clouds whose points are too close to the robot, a situation that may occur if the robot passes through a narrow space between large obstacles. In this case, using a scan-to-scan ICP would

be very likely to produce a bad transformation, or to make the ICP to fail during its the minimization step.

Since the scans for a local map are positioned at the estimated keyframe poses, the local maps may contain little discontinuities that influences the ICP result. Upon a loop closing and update of keyframe poses, one may consider to recompute all the transformations of the loop, using the updated poses to position the local map scans, and then optimize the loop in the graph again, and repeat upon convergence. This strategy was not implemented due to the trouble of storing the extra information that would be needed (*i.e.* the keyframes used to build the local maps for each factor), and the computation overhead that it would produce when closing the loop. Thus, the proposed system considers that the transformations computed from ICP are good enough to position the scans of the next local maps.

Chapter 5

Conclusion

In the last years, the state of the art of SLAM solutions experienced a significant shift from filtering approaches to optimization-based approaches. Modeling the SLAM problem with probabilistic graphical models has brought a new level of understanding of the problem, and the graphical optimization frameworks now available to the community, like GTSAM and g2o, makes it easier for the engineers to design SLAM solutions, reasoning in terms of variables to be estimated and constraints applied to these variables.

After having synthesized these recent methods, this thesis presented two contributions that extend and exploit them.

Chapter 3 presented the application to an incremental SLAM approach of a landmark parametrization previously developed for the Bundle Adjustment problem. It developed the specific factors needed by the approach, and proposed a way to incrementally update the graph with new measurements. The developments were performed in a simulated environment, and more work is still necessary to make it applicable on-board a real robot. In the presented work an EKF based front-end was used, that limits the number of observed landmarks at each image to reduce computational time, whereas numerous landmarks would be required to obtain good results [Strasdat 2010].

Regarding the optimization back-end, the relation between landmark re-anchoring and the changes in the Bayes tree should be further studied. ISAM2 was originally not developed with the replacement of factors in mind, and our re-anchoring of a landmark required the removal and re-insertion of a large number of factors. One should investigate a way to minimize the changes or computations needed to rebuild the Bayes tree in the update step, possibly achieving better performance on incremental updates.

During the course of our work on the PAP landmarks, we noticed a tendency in the robotics community towards using structureless projection factors in graphical-based SLAM, which avoids the problem of explicitly estimating the landmark parameters. This approach is currently being applied in numerous visual-SLAM systems with success, and its use tends to increase: further developments and comparisons are required to compare our proposal with these approaches.

A system to perform SLAM based on point-cloud alignments was proposed in chapter 4. It uses the ICP algorithm to estimate local motions, which defines constraints that are fed to a pose-graph, which is used to reduce the drift along the

robot trajectory upon loop closings, by means of graph-based optimization. In the ICP, we perform the alignment of one scan to a local map built of several scans to overcome problems due to the non-uniform density in the Velodyne scans. The configuration of ICP for Velodyne scans was precisely described .

Tests were performed using two datasets, showing that the graph layer helps to reduce the overall drift after loop closings, especially along the z-axis, although some error is still present.

The first main weakness of the system is that it is completely based on the convergence of the ICP process. But an ICP process may fail, e.g. in cases where the robot passes through clogged environments, which makes most of the scan points to lay into objects very close to the sensor: such scans has not enough structure to allow a good alignment. This may be critical if the overlap gets too low, making the robot unable to localize itself with respect to the previously seen environment. The use of local maps reduces the risk of such failures, but does not guarantee they are excluded.

Further work to monitor the ICP process may help to detect faulty motion estimates. But the graph layer can also provide robustness with respect to such failures. If the incremental ICP fails, a new keyframe can be created in the graph level and connected to a prior factor built from the lower level odometry and with a small covariance, in contrast of being connected to the previous keyframe. This makes the graph disconnected, and the prior on the newly created keyframe fixes the origin of the new disconnected part of the graph. With some chance the robot will re-enter an area that would allow the system to perform a loop closure: if the system manages to relocalize the robot using the loop closing ICP, the prior can be removed and the disconnected sub-graph can be reconnected to the rest of the graph through the constraint produced by the loop closing ICP, although it does not create a loop in the graph. The effectiveness of this solution needs to be assessed.

A second critical point arises when closing loops. Since the accumulated drift along a loop may be large, the initial transformation may be far from the correct one, and the loop closing ICP may be trapped in a local minima, yielding a spatially inconsistent map. We reduced the chances of this happening by considering that the environment is mostly planar, and constraining the initial transformation to a plane. In any case, the correction of the orientation is critical, due to the non-linearities introduced by the angular terms. One possible solution to cope with such faulty ICP loop closures is to add a discrete switch variable to the loop closing constraints, a proposed in [Sünderhauf 2012]. The idea is to model a switch that may turn off the contribution of the loop closing constraint if it is detected that this constraint is causing the optimization error to increase. While this introduces more variables to the system, this makes it more robust to loop closing ICP failures.

The same strategy mentioned before to recover from possible incremental ICP failure could be applied to a multi-robot setting. When using multiple robots, a graph could be built for each robot, each graph fixed to its respective robot origin. Then, if two different robots visit the same part of the environment, the loop closing

ICP can be used to find a constraint that connects the robot graphs, forming a single graph between these two robots. If the priors that encode the robot origins are not certain, one of the two priors may be removed following a predefined priority over the robots composing the team, resulting in single graph for the two robots that has one fixed pose. Otherwise, if the priors are certain, both can be kept in the graphs, and the constraint connecting both robot graphs works like a loop closing constraint that would lead to a correction in both robot trajectories and maps. The merging of graphs could be repeated and potentially result in a full connected graph.

Note that one should be careful with the second option, as it may lead to ill-posed optimization systems if the priors are not consistent with the reality, *i.e.* if the distance between the origins coded in the priors differs considerably from the real distance between the real robot origins. Regarding the merging process, centralized solutions are simpler to handle than distributed ones. If the merging is performed in a distributed manner, the system designer should be careful to avoid the duplication of the same constrains across the robots graphs. This may happen with a team of three robots or more. For example, if the graphs of robot A and C are merged with a robot B, both A and C will contain constraints originated from B. Then if robots A and C attempt to merge their graphs later, the information of B contained in both robot graphs should be handled carefully to avoid duplication.

Appendix A

Developed tools

Some tools were necessary to develop during the thesis. Most of them rely on existing open source projects that *almost* provided the functionality I was looking for, but for which I ended up implementing the missing needed functionalities. This appendix briefly present these tools, which have been made accessible to the community as open source code.

A.1 Python Wrapper for GT-SAM

The GT-SAM library comes with a wrapper for MATLAB that let call its C++ functions from MATLAB, allowing fast prototyping with the library. At some point I needed to use GT-SAM with some python scripts, and at the time there only existed a draft of python wrapper that was not yet functional, mostly implemented by Andrew Melim. So I set myself to wrap the GT-SAM functionalities I needed to python.

The MATLAB wrapper is generated by parsing a interface file with a grammar implemented using Boost Spirit library to generate a MATLAB executable (a mex file), that is in turn processed by MATLAB to provide access to the library from the MATLAB interpreter. Initially the wrapper auto-generation was adapted to produce Boost Python code that would work as the bridge between the python interpreter and GT-SAM, but soon it was found that some grammar changes were needed that would must probably disturb the MATLAB wrapper. We then decided to directly write the code to wrap the needed functionalities, following some handwritten files in their repository.

Most of the functionalities needed to solve SLAM using ISAM2 in python were coded this way (poses and landmark variables, factors between poses, projection factors, nonlinear graph structures, etc), and other ones could be easily wrapped using these ones as examples. Codes such as the one in listing A.1 were made possible with the implemented wrapper.

The developments were shared back for the main maintainers of GT-SAM and incorporated in the source code, available at `https://bitbucket.org/gtborg/gtsam`. At the time of writing the python wrapper is a milestone for the 4.0.0 version of GT-SAM, and there are some efforts from Duy Nguyen Ta to wrap it automatically using Cython.

Listing A.1: Example of code using the GT-SAM python module, slightly edited to better fit this document. The original file can be found at `python/gtsam_examples/VisualISAM2Example.py` in the GT-SAM source code directory.

```
from __future__ import print_function

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import time  # for sleep()

import gtsam
from gtsam_examples import SFMdata
import gtsam_utils

# shorthand symbols:
X = lambda i: int(gtsam.Symbol('x', i))
L = lambda j: int(gtsam.Symbol('l', j))

def visual_ISAM2_plot(poses, points, result):
    # VisualISAMPlot plots current state of ISAM2 object
    # Author: Ellon Paiva
    # Based on MATLAB version by: Duy Nguyen Ta and Frank Dellaert

    # Declare an id for the figure
    fignum = 0

    fig = plt.figure(fignum)
    ax = fig.gca(projection='3d')
    plt.cla()

    # Plot points
    gtsam_utils.plot3DPoints(fignum, result, 'rx')

    # Plot cameras
    i = 0
    while result.exists(X(i)):
        pose_i = result.atPose3(X(i))
        gtsam_utils.plotPose3(fignum, pose_i, 10)
        i += 1

    # draw
    ax.set_xlim3d(-40, 40)
```

```
    ax.set_ylim3d(-40, 40)
    ax.set_zlim3d(-40, 40)
    plt.pause(1)

def visual_ISAM2_example():
    plt.ion()

    # Define the camera calibration parameters
    K = gtsam.Cal3_S2(50.0, 50.0, 0.0, 50.0, 50.0)

    # Define the camera observation noise model
    # (one pixel in u and v)
    measurementNoise = gtsam.noiseModel.Isotropic.Sigma(2, 1.0)

    # Create the set of ground-truth landmarks
    points = SFMdata.createPoints()

    # Create the set of ground-truth poses
    poses = SFMdata.createPoses()

    # Create an iSAM2 object. Unlike iSAM1, which performs periodic
    # batch steps to maintain proper linearization and efficient
    # variable ordering, iSAM2 performs partial
    # relinearization/reordering at each step. A parameter
    # structure is available that allows the user to set various
    # properties, such as the relinearization threshold and type of
    # linear solver. For this example, we we set the
    # relinearization threshold small so the iSAM2 result will
    # approach the batch result.
    parameters = gtsam.ISAM2Params()
    parameters.relinearize_threshold = 0.01
    parameters.relinearize_skip = 1
    isam = gtsam.ISAM2(parameters)

    # Create a Factor Graph and Values to hold the new data
    graph = gtsam.NonlinearFactorGraph()
    initialEstimate = gtsam.Values()

    #  Loop over the different poses, adding the observations to
    # iSAM incrementally
    for i, pose in enumerate(poses):

        # Add factors for each landmark observation
```

```
        for j, point in enumerate(points):
            camera = gtsam.PinholeCameraCal3_S2(pose, K)
            measurement = camera.project(point)
            graph.push_back( gtsam.GenericProjectionFactorCal3_S2(
                measurement, measurementNoise, X(i), L(j), K))

        # Add an initial guess for the current pose
        # Intentionally initialize the variables off from the
        # ground truth
        initialEstimate.insert( X(i), pose.compose( gtsam.Pose3(
            gtsam.Rot3.Rodrigues(-0.1, 0.2, 0.25),
            gtsam.Point3(0.05, -0.10, 0.20))))

        # If this is the first iteration, add a prior on the first
        # pose to set the coordinate frame and a prior on the first
        # landmark to set the scale.
        # Also, as iSAM solves incrementally, we must wait until
        # each is observed at least twice before adding it to iSAM.
        if(i == 0):
            # Add a prior on pose x0
            # (30cm std on x,y,z 0.1 rad on roll,pitch,yaw)
            poseNoise = gtsam.noiseModel.Diagonal.Sigmas(
                np.array([0.3, 0.3, 0.3, 0.1, 0.1, 0.1]))
            graph.push_back( tsam.PriorFactorPose3(X(0),
                poses[0], poseNoise))

            # Add a prior on landmark l0
            pointNoise = gtsam.noiseModel.Isotropic.Sigma(3, 0.1)
            graph.push_back( gtsam.PriorFactorPoint3(
                L(0), points[0], pointNoise))

            # Add initial guesses to all observed landmarks
            # Intentionally initialize the variables off from the
            # ground truth
            for j, point in enumerate(points):
                initialEstimate.insert(L(j), point
                    + gtsam.Point3(-0.25, 0.20, 0.15))
        else:
            # Update iSAM with the new factors
            isam.update(graph, initialEstimate)
            # Each call to iSAM2 update(*) performs one iteration
            # of the iterative nonlinear solver. If accuracy is
            # desired at the expense of time, update(*) can be
```

```
            # called additional times to perform multiple optimizer
            # iterations every step.
            isam.update()
            currentEstimate = isam.calculate_estimate()

            print("****************************************************")
            print("Frame", i, ":")
            for j in range(i + 1):
                print(X(j), ":", currentEstimate.atPose3(X(j)))

            for j in range(len(points)):
                print(L(j), ":", currentEstimate.atPoint3(L(j)))

            visual_ISAM2_plot(poses, points, currentEstimate)

            # Clear the factor graph and values for the next
            # iteration
            graph.resize(0)
            initialEstimate.clear()

    plt.ioff()
    plt.show()

if __name__ == '__main__':
    visual_ISAM2_example()
```

A.2 KITTI Dataset ROS Player

The `kitti_player` is a software that publishes data from KITTI datasets into ROS topics. It is original from the IRALab, from *Università degli Studi di Milano - Bicocca*, and available at `https://github.com/iralabdisco/kitti_player`. It was mainly developed by Francesco Sacchi and Augusto Luis Ballardini.

The IRALab version of `kitti_player` was used in my first experiments with KITTI dataset and ROS, and soon I had to perform some improvements to fit specific needs. My main developments consisted of some fixes in the original code; the addition of options to publish sensor transformations through ROS `tf` mechanism; options to publish odometry from GPS/IMU data as Odometry messages and through `tf`; and an option to convert the dataset directly to a rosbag. The code was also re-factored into classes to improve maintenance.

At the time of writing, the software received some contributions from the community allowing it to read KITTI datasets for visual odometry and to work

with newer versions of ROS. It is available in `https://github.com/Ellon/kitti_player`.

A.3 RViz Covariance Plugin

Rviz is a 3D visualizer for ROS. It allows the visualization of several entities that are common to the ROS ecosystem, like messages passed between nodes through topics and the robot frames in the transformation tree. Visualization for the most common types of data in robotics are already available for the system designer, *e.g.* 2D and 3D poses, points, images, laser scans, point clouds, among others.

Often mathematical entities like poses, accelerations and twists are associated to errors stored in covariance matrices. ROS has standard messages to carry such information if necessary, but there is no built-in visualization for the covariance in rviz. To provide this missing functionality, Thomas Moulard, a former LAAS PhD student, developed a rviz plug-in called `rviz_plugin_covariance` that allowed the visualization of covariances for `PoseWithCovariance` and `Odometry` messages, available in `https://github.com/laas/rviz_plugin_covariance`. As I started to use his plug-in code to visualize the covariances from ICP transformations, I felt some functionalities were still missing, so I took the role of maintainer of the code and added new functionality to it.

The code was re-factored to incorporate features already present for the equivalent messages without covariances, like better control of colors, transparencies, scales of the drawings, and possibility to hide the covariance display or its specific elements. The possibility to visualize poses as arrows or frames was also added, as well as a better drawing of covariances for 2D poses, automatically set when the variances on z, and on the roll and pitch angles are zero. The possibility to visualize the covariance of several last `Odometry` messages was made possible through the use of a buffer whose size is configurable through the rviz interface, and the pose and covariance values can also be visualized by the user though the rviz interface. Figure A.1 show a the visualization of `PoseWithCovariance` messages, while `Odometry` messages with covariances can be seen in fig. A.2.

The main addition to the plug-in was a new way to visualize the angular errors. For the 2D case, the error in yaw is shown as a 2D triangle on the xy-plane. For the 3D case, 2D ellipses are associated to each axis, and they are drawn in a plane perpendicular to the respective axis. Each ellipse represents the influence of the errors of the other two axis on its axis. The sizes and orientation of the ellipses are computed from the Principal Component Analysis of a subsection of the covariance matrix. The details of which section is used to draw each element can be seen in fig. A.3. While this does not allow the visualization of the covariance between all variables (notably between the position variables and the angles variables), the strategy of decomposing the covariance matrix into smaller ones manages to be enough informative about the characteristics of the pose error.

At the time of writing, the maximum angular error that can be visualized is set

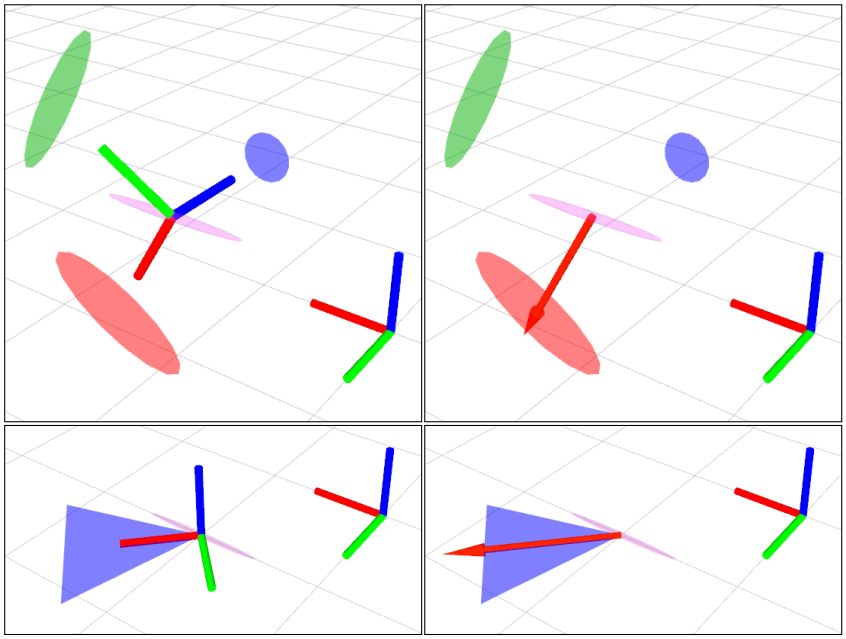


Figure A.1: Visualization of poses with covariance. 3D poses are shown in on top and 2D poses on the bottom. Left images show the poses as 3D frames, while right images show them as arrows along pose's x-axis. The extra frame in each image represents the origin.

to 89°. This is a limitation caused by the way the angular covariances are drawn: by projecting the error into 2D ellipses fixed in a plane perpendicular to the pose axes, angular errors of 90° would produce ellipses of infinite dimensions (fig. A.4). This should not cause problem to most of the applications since most angular errors are usually very small (to allow the representation of a wider range of angular errors, one could project the covariances into a circular sphere around the pose).

A.4 Libpointmatcher Paraview Plugin

Finding a good configuration for the ICP may be cumbersome due to the number of different pipeline elements and their parameters. A visualization software is essential to inspect the convergence of the algorithm. Libpointmatcher provides inspection modules to help with this task. In particular, the `VTKInspector` dumps the intermediary point clouds of an ICP run into VTK files, that may be loaded in *Paraview*, an application for visualization and analysis of data. This is very handy to inspect the convergence, and see if the ICP provided a good result or got trapped in a local minima. If needed, one may write small programs to test a certain pipeline, dump the filtered clouds into VTK files, and visualize in Paraview, but such a work-flow is a burden.

To improve this work-flow I started developing a Paraview plug-in for the libpointmatcher library. The idea was to allow the user to build a libpointmatcher

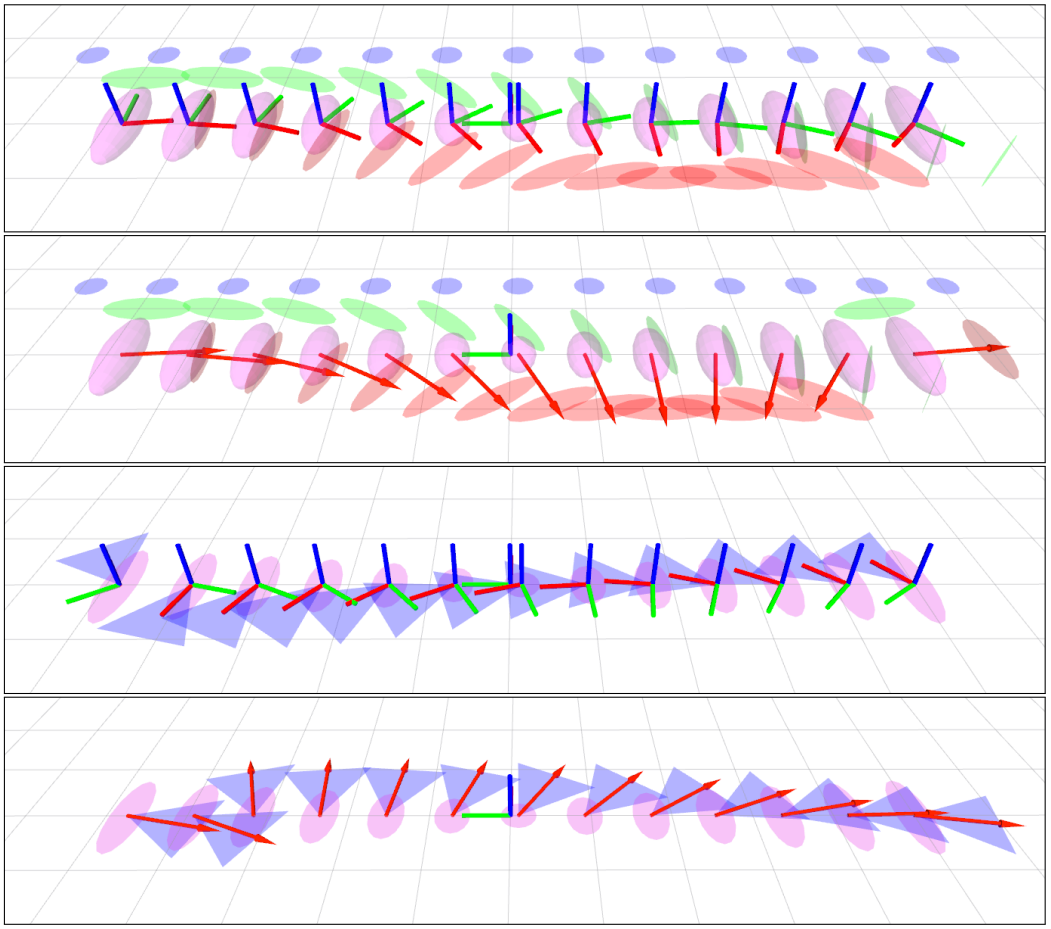


Figure A.2: Visualization of odometry messages with covariance, using a buffer of length 13. From top to bottom: odometry with 3D poses shown as 3D frames and as arrows; 2D poses shown as 3D frames and arrows. The origin is shown as a 3D frame in the middle of the images.

pipeline inside Paraview, specifying the data filters for the reading and reference clouds, as well as the matcher, outlier filters and minimizer, together with their parameters. The user would then be able to inspect the clouds in the outputs of each data filter, as well as the alignment process from the ICP minimization, without the need to dump the files using the `VTKInspector` and reload in Paraview every time. Ultimately, the user would be able to save the configuration file for the current pipeline in a YAML file, that could then be used directly to load the ICP configuration in applications using libpointmatcher.

Figure A.5 shows the main elements of PointMatcherPlugin in Paraview (pipeline, available filters and properties of the main ICP filter). Since the library is in continuous development, some helper scripts were developed to auto-generate the code wrapping the filters from the output of `pmicp -l`, a command that lists all available modules in libpointmatcher. This way, new filters may be added with

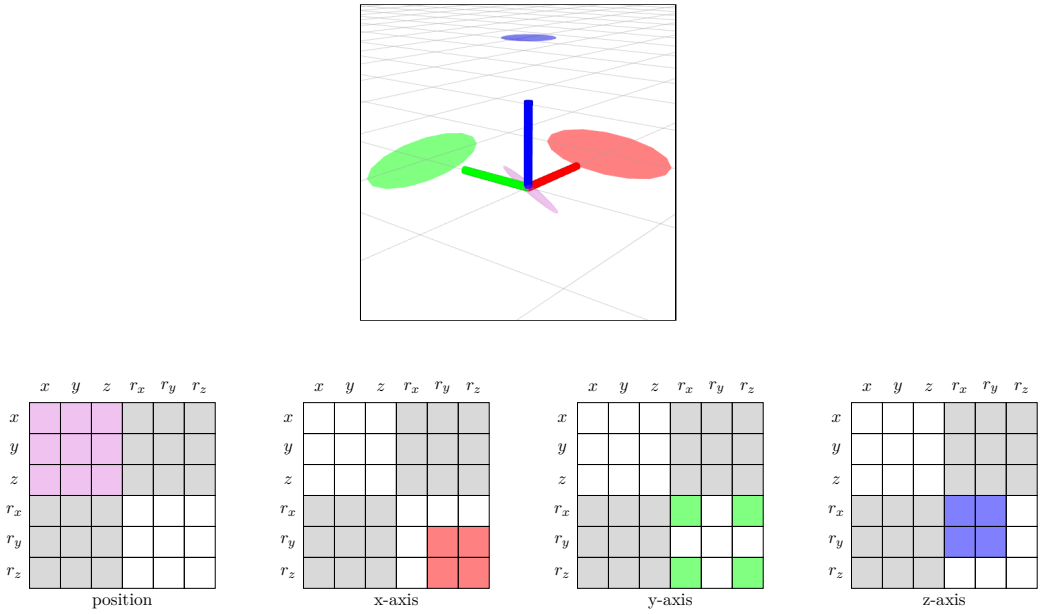


Figure A.3: Covariances of a 3D pose and covariance sub-matrices used to draw each element. Colors in the matrices match the color of the element drawn in the top figure. The position error is draw as a 3D ellipse with lengths computed from the top-left 3×3 sub-matrix. Different 2×2 sub-matrices are used to compute the 2D ellipses representing the angular error on each axis. Each of these sub-matrices represent the covariance on the other two axis. Gray regions of the covariance matrix are never used.

almost no effort, and the auto-generated code may be used as a base for filter wrappers that use more advanced features of Paraview, like the use of some widgets to set filter parameters. The matcher, outlier filters, minimizer and transformation checkers are set from the properties of a single pipeline element, the `PMIcpFilter`.

At the time of writing the input and output of configuration files is also not yet implemented in the plugin. Also, the visualization of the alignment process is not yet available due to limitations on Paraview's pipeline management, to which a workaround was still not found. Nevertheless, the filter modules are already functional and the plugin may be used to inspect the intermediary reading and reference clouds modified by the filters in the pipeline. The plugin is available in `https://github.com/Ellon/PointMatcherPlugin`.

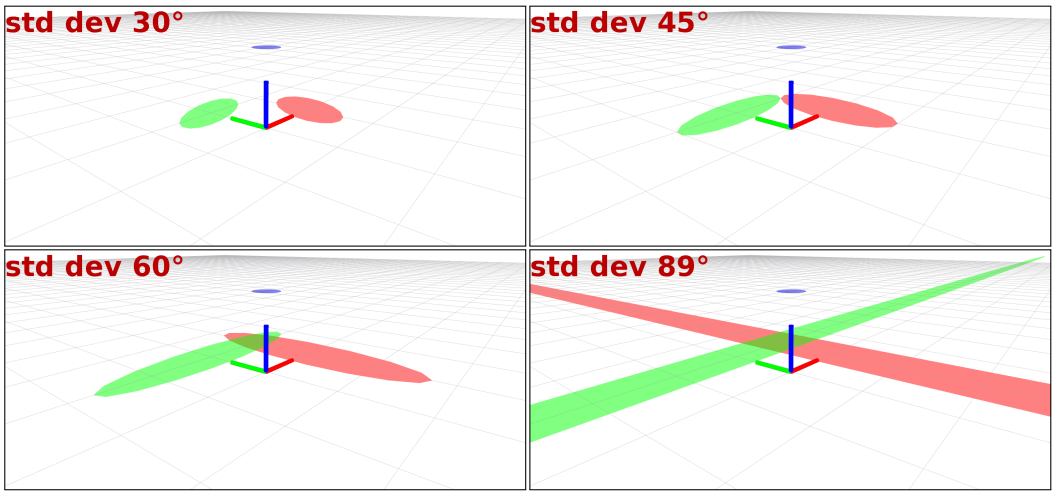


Figure A.4: Visualization of different covariances in the angle around z-axis. Errors in one axis deforms the ellipses drawn on the other two axis. As the angle approaches a standard deviation of 90° the ellipses get stretched to infinity (bottom-right figure), thus the maximal error that can be visualized is set to 89°. Such large errors should not happen in practice.

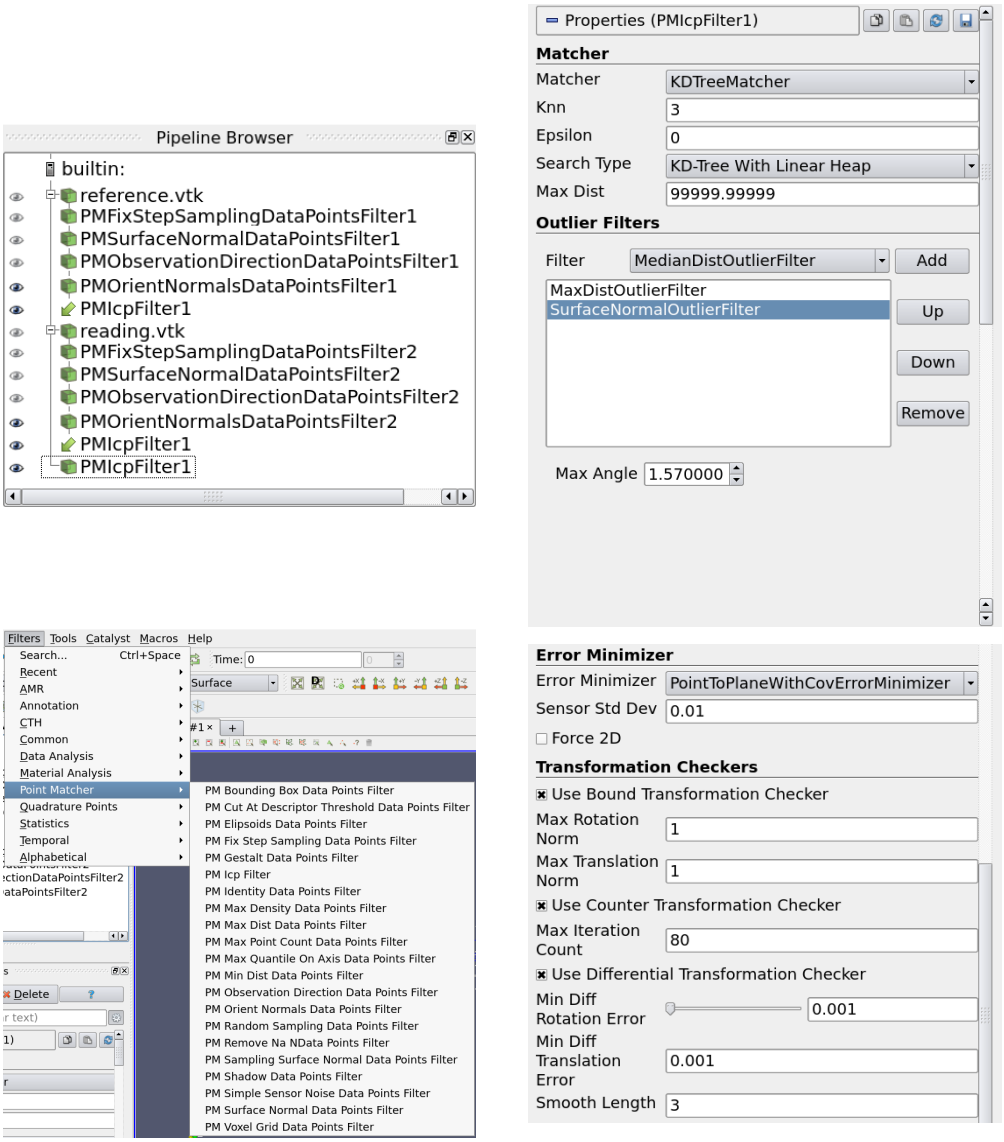


Figure A.5: `PointMatcherPlugin` in Paraview. **Top-left**: an example of ICP configuration as modules in Paraview's pipeline. Note how the reading and reference filter pipelines connect to the inputs of a `PMIcpFilter`. **Bottom-left**: the *Point Matcher* filter menu with the available filters. **Right**: parameters of libpointmatcher's matchers, outlier filters, error minimizers and transformation checkers, accessible through properties of `PMIcpFilter`.

Bibliography

[Bailey 2006] T. Bailey and H. Durrant-Whyte. *Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms.* IEEE Robotics and Automation Magazine, vol. 13, no. 2, pages 99–110, June 2006. (Cited in page 7.)

[Besl 1992] P. J. Besl and H. D. McKay. *A method for registration of 3-D shapes.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pages 239–256, February 1992. (Cited in page 54.)

[Bishop 2006] Christopher M. Bishop. Pattern recognition and machine learning (information science and statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. (Cited in page 13.)

[Borrmann 2008] Dorit Borrmann, Jan Elseberg, Kai Lingemann, Andreas Nüchter and Joachim Hertzberg. *Globally Consistent 3D Mapping with Scan Matching.* Robot. Auton. Syst., vol. 56, no. 2, pages 130–142, February 2008. (Cited in pages 77 and 79.)

[Cadena 2016] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid and J. J. Leonard. *Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age.* IEEE Transactions on Robotics, vol. 32, no. 6, pages 1309–1332, December 2016. (Cited in page 7.)

[Carlone 2014] Luca Carlone, Zsolt Kira, Chris Beall, Vadim Indelman and Frank Dellaert. *Eliminating conditionally independent sets in factor graphs: a unifying perspective based on smart factors.* In Int. Conf. on Robotics and Automation (ICRA), Hong Kong, 2014. (Cited in page 50.)

[Censi 2007] A. Censi. *An accurate closed-form estimate of ICP's covariance.* In Robotics and Automation, 2007 IEEE International Conference on, pages 3167–3172, April 2007. (Cited in pages 67, 79, and 83.)

[Chatila 1985] R. Chatila and J-P. Laumond. *Position Referencing and Consistent World Modeling for Mobile Robots.* In IEEE International Conference on Robotics and Automation, St Louis (USA), pages 138–145, 1985. (Cited in page 7.)

[Civera 2008] J. Civera, A.J. Davison and J. Montiel. *Inverse Depth Parametrization for Monocular SLAM.* Robotics, IEEE Transactions on, vol. 24, no. 5, pages 932–945, October 2008. (Cited in page 28.)

[Crowley 1989] J. L. Crowley. *World modelling and position estimation for a mobile robot using ultrasonic ranging.* In IEEE International Conference on Robotics and Automation, pages 674 – 680, 1989. (Cited in page 7.)

[Dellaert 2012] Frank Dellaert. *Factor Graphs and GTSAM: A Hands-on Introduction*. Technical Report GT-RIM-CP&R-2012-002, GT RIM, September 2012. (Cited in page 67.)

[Durrant-Whyte 2006] H. Durrant-Whyte and T. Bailey. *Simultaneous Localisation and Mapping (SLAM): Part II State of the Art*. IEEE Robotics and Automation Magazine, vol. 13, no. 3, pages 108–117, June 2006. (Cited in page 7.)

[Forster 2015] Christian Forster, Luca Carlone, Frank Dellaert and Davide Scaramuzza. *IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation*. In Robotics: Science and Systems XI, Sapienza University of Rome, Rome, Italy, July 2015. (Cited in pages 50 and 82.)

[Forster 2016] Christian Forster, Luca Carlone, Frank Dellaert and Davide Scaramuzza. *On-Manifold Preintegration for Real-Time Visual-Inertial Odometry*. IEEE Transactions on Robotics (TRO), 2016. (Cited in page 13.)

[Geiger 2012] Andreas Geiger, Philip Lenz and Raquel Urtasun. *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite*. In Conference on Computer Vision and Pattern Recognition (CVPR), 2012. (Cited in page 71.)

[Geiger 2013] Andreas Geiger, Philip Lenz, Christoph Stiller and Raquel Urtasun. *Vision meets Robotics: The KITTI Dataset*. International Journal of Robotics Research (IJRR), 2013. (Cited in page 67.)

[Grisetti 2010] G. Grisetti, R. Kümmerle, C. Stachniss and W. Burgard. *A Tutorial on Graph-Based SLAM*. Intelligent Transportation Systems Magazine, IEEE, vol. 2, no. 4, pages 31–43, winter 2010. (Cited in pages 5, 8, and 17.)

[Hartley 1993] Richard I. Hartley. *Cheirality Invariants*. In In proc. DARPA Image Understanding Workshop, pages 745–753, 1993. (Cited in page 48.)

[Kaess 2011] Michael Kaess, Viorela Ila, Richard Roberts and Frank Dellaert. *The Bayes Tree: An Algorithmic Foundation for Probabilistic Robot Mapping*. In David Hsu, Volkan Isler, Jean-Claude Latombe and MingC. Lin, editors, Algorithmic Foundations of Robotics IX, volume 68 of *Springer Tracts in Advanced Robotics*, pages 157–173. Springer Berlin Heidelberg, 2011. (Cited in page 46.)

[Kaess 2012] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard and Frank Dellaert. *iSAM2: Incremental smoothing and mapping using the Bayes tree*. The International Journal of Robotics Research, vol. 31, no. 2, pages 216–235, 2012. (Cited in pages 5, 23, 24, and 46.)

[Koller 2007] D. Koller, N. Friedman, L. Getoor and B. Taskar. *Graphical Models in a Nutshell*. In L. Getoor and B. Taskar, editors, Introduction to Statistical Relational Learning. MIT Press, 2007. (Cited in page 10.)

[Konolige 2008] K. Konolige and M. Agrawal. *FrameSLAM: From Bundle Adjustment to Real-Time Visual Mapping*. Trans. Rob., vol. 24, no. 5, pages 1066–1077, October 2008. (Cited in page 62.)

[Kschischang 2001] F. R. Kschischang, B. J. Frey and H. A. Loeliger. *Factor graphs and the sum-product algorithm*. IEEE Transactions on Information Theory, vol. 47, no. 2, pages 498–519, February 2001. (Cited in page 13.)

[Lu 1997] F. Lu and E. Milios. *Globally Consistent Range Scan Alignment for Environment Mapping*. Autonomous Robots, vol. 4, no. 4, pages 333–349, 1997. (Cited in pages 16 and 77.)

[Lupton 2012] T. Lupton and S. Sukkarieh. *Visual-Inertial-Aided Navigation for High-Dynamic Motion in Built Environments Without Initial Conditions*. Robotics, IEEE Transactions on, vol. 28, no. 1, pages 61–76, February 2012. (Cited in page 82.)

[Pomerleau 2011] F. Pomerleau, S. Magnenat, F. Colas, M. Liu and R. Siegwart. *Tracking a depth camera: Parameter exploration for fast ICP*. In 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3824–3829, September 2011. (Cited in page 54.)

[Pomerleau 2013a] François Pomerleau, Francis Colas, Roland Siegwart and Stéphane Magnenat. *Comparing ICP Variants on Real-World Data Sets*. Autonomous Robots, vol. 34, no. 3, pages 133–148, February 2013. (Cited in page 67.)

[Pomerleau 2013b] François Pomerleau, Francis Colas and Roland Siegwart. *A Review of Point Cloud Registration Algorithms for Mobile Robotics*. Foundations and Trends® in Robotics, vol. 4, no. 1, pages 1–104, 2013. (Cited in page 56.)

[Roussillon 2011] C. Roussillon, A. Gonzalez, J. Solà, , J-M Codol, N. Mansard, S. Lacroix and M. Devy. *RT-SLAM: a generic and real-time SLAM architecture*. In International Conference on Vision Systems, Sophia Antopolis (France), September 2011. (Cited in page 46.)

[Smith 1987] R. Smith, M. Self and P. Cheeseman. *A Stochastic Map for Uncertain Spatial Relationships*. In Robotics Research: The Fourth International Symposium, Santa Cruz (USA), pages 468–474, 1987. (Cited in page 7.)

[Solà 2007] Joan Solà. *Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot: a Geometric and Probabilistic Approach.* PhD thesis, École Doctorale Systèmes, 2007. (Cited in pages 29, 33, and 38.)

[Solà 2012] Joan Solà, Teresa Vidal-Calleja, Javier Civera and JoséMaríaMartínez Montiel. *Impact of Landmark Parametrization on Monocular EKF-SLAM with Points and Lines.* International Journal of Computer Vision, vol. 97, no. 3, pages 339–368, 2012. (Cited in page 28.)

[Strasdat 2010] H. Strasdat, J. M. M. Montiel and A. J. Davison. *Real-time monocular SLAM: Why filter?* In 2010 IEEE International Conference on Robotics and Automation, pages 2657–2664, May 2010. (Cited in pages 50 and 87.)

[Strasdat 2012] Hauke Strasdat, J. M. M. Montiel and Andrew J. Davison. *Editors Choice Article: Visual SLAM: Why Filter?* Image Vision Comput., vol. 30, no. 2, pages 65–77, February 2012. (Cited in pages 15, 16, and 50.)

[Sünderhauf 2012] Niko Sünderhauf and Peter Protzel. *Switchable constraints for robust pose graph SLAM.* In IROS, pages 1879–1884. IEEE, 2012. (Cited in page 88.)

[Thrun 2007] Sebastian Thrun and John J. Leonard. *Simultaneous Localization and Mapping.* In Bruno Siciliano and Oussama Khatib, editors, Springer Handbook of Robotics, chapter 37. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. (Cited in page 13.)

[Weiss 2011] Stephan Weiss, Davide Scaramuzza and Roland Siegwart. *Monocular-SLAM-Based Navigation for Autonomous Micro Helicopters in GPS-denied Environments.* Journal of Field Robotics, vol. 28, no. 6, pages 854–874, November 2011. (Cited in page 28.)

[Zhang 2014] Ji Zhang and Sanjiv Singh. *LOAM: Lidar Odometry and Mapping in Real-time.* In Robotics: Science and Systems Conference, July 2014. (Cited in page 54.)

[Zhao 2011] Liang Zhao, Shoudong Huang, Lei Yan and G. Dissanayake. *Parallax angle parametrization for monocular SLAM.* In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 3117–3124, May 2011. (Cited in pages 5, 28, 36, and 38.)

[Zhao 2015] Liang Zhao, Shoudong Huang, Yanbiao Sun, Lei Yan and Gamini Dissanayake. *ParallaxBA: bundle adjustment using parallax angle feature parametrization.* The International Journal of Robotics Research, vol. 34, no. 4-5, pages 493–516, 2015. (Cited in pages 36 and 38.)