Pontifícia Universidade Católica
do Rio de Janeiro

João Carlos Virgolino Soares

Graph Optimization and Probabilistic SLAM of Mobile Robots using an RGB-D Sensor

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Engenharia Mecânica of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Mecânica.

Advisor: Prof. Marco Antonio Meggiolaro

Rio de Janeiro
March 2018

Pontifícia Universidade Católica
do Rio de Janeiro

João Carlos Virgolino Soares

Graph Optimization and Probabilistic SLAM of Mobile Robots using an RGB-D Sensor

Dissertation presented to the Programa de Pós–graduação em Engenharia Mecânica of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Mecânica. Approved by the undersigned Examination Committee.

Prof. Marco Antonio Meggiolaro
Advisor
Departamento de Engenharia Mecânica – PUC-Rio

Prof. Wouter Caarls
Departamento de Engenharia Elétrica – PUC-Rio

Prof. Paulo Fernando Ferreira Rosa
Seção de Engenharia de Computação – IME

Prof. Márcio da Silveira Carvalho
Vice Dean of Graduate Studies
Centro Técnico Científico – PUC-Rio

Rio de Janeiro, March 26th, 2018

João Carlos Virgolino Soares

Graduated in Mechanical Engineering at Pontifícia Universidade Católica do Rio de Janeiro in 2015

Bibliographic data

Soares, João Carlos Virgolino

Graph optimization and probabilistic SLAM of mobile robots using an RGB-D sensor / João Carlos Virgolino Soares; advisor: Marco Antonio Meggiolaro. – 2018.
109 f. : il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro , Departamento de Engenharia Mecânica, 2018.
Inclui bibliografia

1. Engenharia Mecânica – Teses. 2. SLAM. 3. Robótica probabilística. 4. Otimização de grafos. 5. Sensor RGB-D. 6. ROS. I. Meggiolaro, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro . Departamento de Engenharia Mecânica. III. Título.

CDD: 621

In memory of my father, João Carlos Pinheiro Soares.

Acknowledgments

To my advisor, Professor Marco Antonio Meggiolaro, for the oportunity, orientation and incentive.

To Professors Paulo Fernando Ferreira Rosa and Wouter Caarls, for the interest in the work.

To Professor Carlos Alberto de Almeida, for the fundamental role in my formation.

To Professors Ivan Menezes and Mauro Speranza for all support.

To CAPES, for the financial support.

To my colleagues Rodrigo Bianchi, Marisa Bazzi, Eduardo Cota, Thiago Almeida and André Xavier.

To all my friends from team RioBotz, specially Fischer, Adriel, Homsi, Maria Vitória, Gustavo, Júnior, Ivan, Mariana, Montesanto, Zig, Luiz Santarelli, Ziliani, Daniel, David, Henrique, Lohan, Grativol, Robalo, Coutinho, Letícia, Malu, Tatiana, Rodrigo D'Amico, Rodrigo Nogueira, Rafael Schoenfelder, Rodrigo Duque, Yann and Victória.

To Guilherme Rodrigues, João Almeida Ramos and Eduardo von Ristow, for all advice.

To my friends from PUC-Rio, Igor Girsas, Juliana Leão, Bruno Calasães, Gabriel Barsi, Matheus Cosenza, Pedro Froner, José Benatti, Antonio Rodrigues, Felipe Salles and Thaís Joffe.

To my long time friends, Fernando and José Renato.

To Isaías, Rosemary, Lucas and Letícia, my family from the heart.

To my dear aunties Norma and Nadir.

To all my relatives.

To my parents João and Imar, for the unconditional love.

To my wife Ana, for always been by my side.

To God, who allowed me to get here.

Abstract

Soares, João Carlos Virgolino; Meggiolaro, Marco Antonio (Advisor). **Graph Optimization and Probabilistic SLAM of Mobile Robots using an RGB-D Sensor**. Rio de Janeiro, 2018. 109p. Dissertação de Mestrado – Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro .

Mobile robots have a wide range of applications, including autonomous vehicles, industrial robots and unmanned aerial vehicles. Autonomous mobile navigation is a challenging subject due to the high uncertainty and nonlinearity inherent to unstructured environments, robot motion and sensor measurements. To perform autonomous navigation, a robot need a map of the environment and an estimation of its own pose with respect to the global coordinate system. However, usually the robot has no prior knowledge about the environment, and has to create a map using sensor information and localize itself at the same time, a problem called Simultaneous Localization and Mapping (SLAM). The SLAM formulations use probabilistic algorithms to handle the uncertainties of the problem, and the graph-based approach is one of the state-of-the-art solutions for SLAM. For many years, the LRF (laser range finders) were the most popular sensor choice for SLAM. However, RGB-D sensors are an interesting alternative, due to their low cost. This work presents an RGB-D SLAM implementation with a graph-based probabilistic approach. The proposed methodology uses the Robot Operating System (ROS) as middleware. The implementation is tested in a low cost robot and with real-world datasets from literature. Also, it is presented the implementation of a pose-graph optimization tool for MATLAB.

Keywords

SLAM; Probabilistic robotics; Graph-Optimization; RGB-D Sensor; ROS.

Resumo

Soares, João Carlos Virgolino; Meggiolaro, Marco Antonio. **Otimização de Grafos e SLAM Probabilístico de Robôs Móveis usando um Sensor RGB-D**. Rio de Janeiro, 2018. 109p. Dissertação de Mestrado – Departamento de Engenharia Mecânica, Pontifícia Universidade Católica do Rio de Janeiro .

Robôs móveis têm uma grande gama de aplicações, incluindo veículos autônomos, robôs industriais e veículos aéreos não tripulados. Navegação móvel autônoma é um assunto desafiador devido à alta incerteza e não-linearidade inerente a ambientes não estruturados, locomoção e medições de sensores. Para executar navegação autônoma, um robô precisa de um mapa do ambiente e de uma estimativa de sua própria localização e orientação em relação ao sistema de referência global. No entando, geralmente o robô não possui informações prévias sobre o ambiente e deve criar o mapa usando informações de sensores e se localizar ao mesmo tempo, um problema chamado Mapeamento e Localização Simultâneos (SLAM). As formulações de SLAM usam algoritmos probabilísticos para lidar com as incertezas do problema, e a abordagem baseada em grafos é uma das soluções estado-da-arte para SLAM. Por muitos anos os sensores LRF (laser range finders) eram as escolhas mais populares de sensores para SLAM. No entanto, sensores RGB-D são uma alternativa interessante, devido ao baixo custo. Este trabalho apresenta uma implementação de RGB-D SLAM com uma abordagem baseada em grafos. A metodologia proposta usa o Sistema Operacional de Robôs (ROS) como middleware do sistema. A implementação é testada num robô de baixo custo e com um conjunto de dados reais obtidos na literatura. Também é apresentada a implementação de uma ferramenta de otimização de grafos para MATLAB.

Palavras-chave

SLAM; Robótica probabilística; Otimização de grafos; Sensor RGB-D; ROS.

Table of contents

List of figures

List of tables

List of Abreviations

AGV – Autonomous Guided Vehicle
ATE – Absolute Trajectory Error
DBN – Dynamic Bayesian Network
DOF – Degrees of Freedom
EKF – Extended Kalman Filter
ICP – Iterative Closest Point
IMU – Inertial Measurement System
LiPo – Lithium Polymer
LM – Levenberg-Marquardt
PCG – Preconditioned Conjugate Gradient
PCL – Point Cloud Library
PDF – Probability Density Function
RANSAC – Random Sample Consensus
RMSE – Root Mean Squared Error
ROS – Robot Operating System
RPE – Relative Pose Error
SIFT – Scale Invariant Feature Transform
SLAM – Simultaneous Localization and Mapping
UAV – Unmanned Aerial Vehicle
VO – Visual Odometry

They that sow in tears shall reap in joy.

Psalm 126:5.

1
Introduction

1.1
Motivation

Robotic systems are becoming more commom in society, improving life quality performing tasks that are tedious, dangerous or even impossible for humans. Robotic manipulators, for example, allowed a considerable improvement in manufacturing industry with a faster production and a more reliable quality. Despite their success, they suffer from a limited mobility [1]. In certain applications is neccessary a robot that is able to move freely through the environment. Mobile robots are robotic systems capable of moving and operating without a fixed location. The Mars Exploration Rover, shown in Fig. 1.1, for example, allowed cientific research in a planet to which is currently infeasible to send humans.

Figure 1.1: Mars Exploration Rover

Mobile robots can be teleoperated, automated guided vehicles or fully autonomous [1]. AGVs are robots that follow a pre-determined path and are often used in industrial applications such as product transportation. Teleoperated robots are usually used in a situation of high risk for humans. However, many applications need a fully autonomous robot. Self-driving cars,

for example, are a topic of extense discussion and research nowadays, and are being developed by several companies such as Google, Tesla and GM.

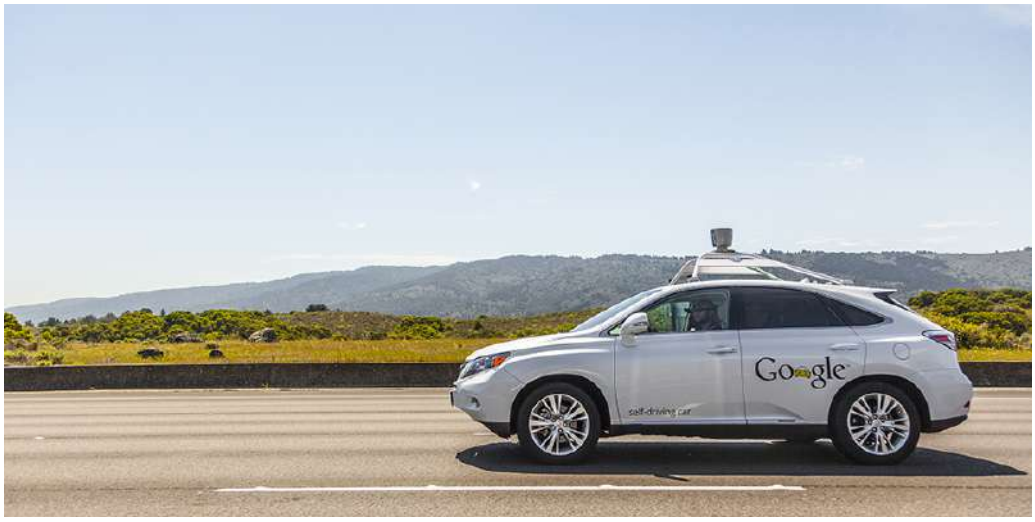


Figure 1.2: Self-driving car

Achieving fully autonomous navigation is increasingly becoming essential in mobile robotics. Autonomous mobile robots have several other applications, including mobile industrial robots that are more flexible to changes in environment or interaction with humans in a workspace than AGVs, tour-guide robots, aerial surveillance of UAVs and home-service robots, such as the house-cleaning robot roomba, shown in Fig. 1.3.

Figure 1.3: Roomba - robotic vacuum cleaner

Autonomous mobile navigation is currently one of the most challenging subjects in robotics, due to the high uncertainty and non-linearity inherent to unstructured and unpredictable environments, sensor measurements, and locomotion, in contrast to manufacturing robot arms that are usually fixed

and perform repetitive tasks, such as welding and assembling, in a controlled and predictable environment.

There are four major problems in autonomous mobile robotics: perception, localization and mapping, path planning and motion control (locomotion) [1]. Their interdependence is illustrated in Fig. 1.4.

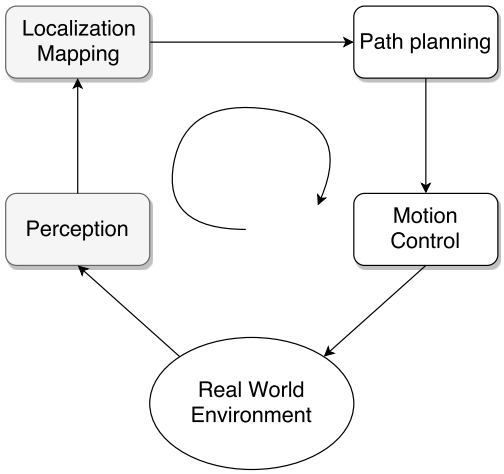


Figure 1.4: Diagram of an autonomous mobile robot system

The goal of the motion control problem is to study the kinematics of the robot and low-level details of the locomotion system. Perception is responsible for extraction and interpretation of raw sensor data. Localization, mapping and path planning are higher-level tasks that need robust techniques to be achieved [1]. This work is focused on the perception, localization and mapping problems.

1.1.1 SLAM

Autonomous mobile robots need an accurate map of the environment, as well as an estimation of its own pose with respect to the global coordinate system, in order to be able to navigate autonomously. However, usually both map and pose are unknown and only sensor information is available. Also, a map is needed to perform localization, and the robot needs a pose estimation to build an accurate map [2]. Therefore, this problem consists in simultaneously creating a map of the environment using raw sensor information and estimating the pose of the robot in this map, and it is known as Simultaneous Localization and Mapping (SLAM).

SLAM is a fundamental step to achieve full autonomy [3], therefore it is one of the most researched topics in mobile robotics. However, several problems remain open in SLAM [4] [5], such as dealing with fast robot dynamics

or highly dynamic scenarios, and achieving persistent autonomy, high-level understanding of the environment, and a robust performance.

SLAM has been implemented in different applications and types of environments. For example, in oil industry, inspection and maintenance robots need to perform localization, and need a globally consistent 3D metric representation of the environment for structural inspection, or to perform physical interventions, such as control panel operation or valve turning.

1.1.2 RGB-D Sensors

For many years the laser range-finders were the most broadly used sensors for SLAM, due do its high range and precision. However, their cost is a considerable limitation to research and commercial projects. Cameras are a more cheap solution and provide rich information about the environment. Tesla Motors, for example, has opted to use cameras and radars instead of laser range-finders to produce more afordable autonomous cars. However, the absence of a direct depth measurement increases the level of complexity of the algorithms to solve the SLAM problem.

RGB-D sensors are an interesting alternative due to their low cost and weight, and to the fact that they provide both color and depth information, despite having more limited measurement range than laser scanners. The Microsoft Kinect [6], shown in Fig. 1.5, is a popular RGB-D sensor created for the gaming industry, and has been incorporated into computer science and robotics research for development of SLAM systems, object recognition, machine learning and 3D reconstruction at an affordable price.

Figure 1.5: Microsoft Kinect v2

The first version of the Kinect, with a structured-light technology, had range and resolution limitations, and was also unsuitable for outdoor applications due to sunlight [7]. The second generation, the Microsoft Kinect v2 [6], is a time-of-flight camera with a higher resolution and a wider field

of view [8]. The new kinect has opened new possibilities for research and development of mobile robotics. Hernández-Aceituno et al. [9], for example, made a comparison between the accuracy of a kinect v2, laser range-finders and stereo cameras for outdoor close range obstacle detection in a self-driving car. The kinect outperformed the other sensors.

1.1.3 Robot Operating System

One of the problems in writing robot software is to deal with different types of hardware and integration of multiple tasks and systems. The Robot Operating System (ROS) is an open source framework for writing robot software that promotes collaborative development through a modular system that facilitates code reuse. ROS provides a set of tools and libraries for robotics applications and integrates multiple tasks, devices and low-level control through a graph-based architecture [10].

ROS has a large and active community with several collaborators, and can be used with multiple languages, such as C++ or Python. Beside the large use in research, several companies and organizations adopted ROS in their robots. The robot Guardian from Robotnik, for example, shown in Fig. 1.6, is a modular teleoperated mobile robot for outdoor operations equiped with a Hokuyo laser range-finder, an IMU and a GPS [11]. The robot Robonaut, shown in Fig. 1.7, is a humanoid robot designed and developed by NASA and General Motors to work at the International Space Station, which debuted the use of ROS in space [12].

Figure 1.6: Guardian robot

Figure 1.7: Robonaut

Other examples of ROS-driven robots include Warthog, an amphibious unmanned ground vehicle from Clearpath Robotics [13], MPO-700, an omnidirectional robot from Neobotix [14], and Baxter, an industrial robot from Rethink Robotics [15].

1.2 Problem Definition

As stated before, mobile robotics is inherently uncertain, especially due to unstructured environments, robot motion and sensor measurements. If a robot is performing localization deterministically using wheel encoders, for example, the estimation will be completely wrong after some motion [5]. Probabilistic algorithms deal with this uncertainty explicitly representing it using probability theory [3].

There are three major probabilistic paradigms for SLAM: Kalman Filters, Particle Filters and Graph-based approaches. The first two are called online SLAM methods, because the estimated state is the current position of the robot. The third one is also called full SLAM, because the complete trajectory of the robot is estimated [16].

This implementation uses the Graph-SLAM approach. It consists in a representation of the states of the robot with a graph that is optimized using a non-linear least squares minimization to generate a maximum likelihood solution for the trajectory.

The SLAM problem is divided into two main steps: front-end and back-end. The front-end processes sensor information and creates the graph using a motion estimation method. The back-end uses probability theory to optimize the graph given the measurement errors. In Fig. 1.8 is shown a typical Graph-SLAM system. In contrast to the front-end, the back-end is completely sensor-independent. It only depends on the right graph construction.

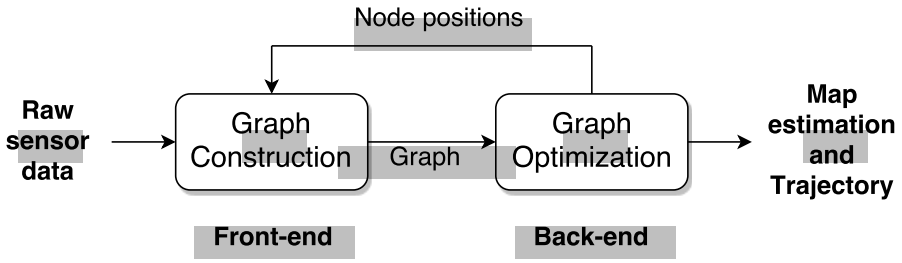


Figure 1.8: Graph-SLAM system

A major step in this implementation is known as the Loop Closure problem, when the robot revisits a previous known location, followed by the corresponding correction of the map. The quality of the implementation depends on the capability of the system to detect loop closures with enough accuracy [17].

1.3 Literature Review

In 1986, probabilistic methods were beginning to be used in robotics research. At the 1986 IEEE Robotics and Automation Conference in San Francisco, several researches recognized that probabilistic mapping was a fundamental problem in robotics [4]. According to Durrant-Whyte et al.[4], the coining of the term "SLAM" occurred in 1995 in his work about AGVs at the International Symposium on Robotics Research[18], but the SLAM problem was introduced in 1986 by Smith and Cheeseman [19], who used the Extended Kalman Filter to deal with geometric uncertainties and to incrementally use new information about locations of different landmarks in a map and their correlations, estimating the posterior probability distribution of the pose of the robot and landmark positions.

1.3.1 Filtering Approaches

The Kalman Filter is a Bayesian filter created by Rudolph Kalman in the 1950s for filtering and prediction in linear systems. It assumes linear state transitions, linear measurements and Gaussian noise. However, a mobile robot is a non-linear system, therefore the Kalman Filter is not applicable. The Extended Kalman Filter uses a local linearization procedure with a first order Taylor expansion to overcome the linearity assumption [3].

After the mathematical introduction to the SLAM problem with EKF by Smith and Cheeseman [19], the method was implemented in 1989 by Mountarlier and Chatila [20], using a mobile robot with a laser range-finder and odometry information. For several years the EKF was the dominant approach for SLAM, with many other implementations and variations. In 1991, Leonard and Durrant-Whyte [21] applied EFK-SLAM in a mobile robot navigation problem with sonar sensors in a known environment. Dissanayake et al. [22], in 2001, showed the monotonic convergence behaviour of the uncertainty of landmark locations in EKF-SLAM, and also implemented the method in a vehicle with a radar in an outdoor environment.

However, the EKF-SLAM suffer from a major drawback. In the EFK-SLAM the map is composed of point landmarks. The computational cost of the implementation is dominated by the number of landmarks in the map, because the $O(n^2)$ elements of the covariance matrix have to be updated every time a landmark is observed, which leads to a scalability limitation and the problem becomes computationally intractable for large maps [23]. This issue

was recognized by Leonard and Feder in 1999 [24], and by Guivant and Nebot in 2001 [25].

To overcome this problem, Montemerlo at al. presented in 2002 the FastSLAM algorithm [26], that is also a Bayesian method to estimate the posterior probability distribution of the pose and landmark positions. However, this algorithm uses Rao-Blackwellized Particle Filters to obtain a logarithmic computational requirement with the number of landmarks in the map.

Particle Filter, also known as sequential Monte Carlo method, is a non-parametric posterior probability estimator based on Bayes' Theorem and Monte Carlo techniques [27]. Instead of using a parametric representation of the uncertainty, as the Kalman filter does, the Particle Filter draws samples from a distribution to represent it [3]. Therefore, Particle filters have the advantages of being capable to deal with non-Gaussian noise, represent complex multimodal beliefs, and be applied to global localization problems, such as the kidnapped robot problem (when the robot is suddenly carried to another location), as opposed to Kalman filters [25].

First introduced for SLAM by Murphy in 1999 [28], the Rao-Blackwellized Particle Filter is a extension of the Monte Carlo Localization algorithm [29], which is an application of the Particle Filter to the robot localization problem [26]. The idea of FastSLAM is to use the Rao-Blackwellized Particle Filter to model the pose of the robot by a set of particles sampled from the probabilistic motion model. Each particle has K independent Kalman filters for its own K landmark locations [30] [23]. It results in a $O(Nlog(K))$ computational requirement. Applications of FastSLAM include the work of Neto et al. [31], in which they performed FastSLAM extracting visual features from the environment.

Another SLAM implementation that used Rao-blackwellized Particle Filter is the DP-SLAM [32], presented in 2003 by Eliazar and Parr. Unlike FastSLAM, the DP-SLAM makes no landmark assumptions and is specific for laser range-finders. Canchumuni and Meggiolaro [33], for example, performed DP-SLAM using a single laser range-finder with a Genetic Algorithm to obtain robot displacement.

1.3.2 Graph-based approaches

Despite the efforts to overcome the scalability problems from the early approaches to SLAM, it was found that filtering methods have an inconsistency issue if applied to SLAM problems [34]. This issue arises from one of the major disadvantages of filtering approaches, which is that data is discarded after

processed [35]. Smoothing, or Graph-based methods, have better performance and accurary than filtering methods by saving all data available [36][16]. Due to this characteristic, they are also called full SLAM methods.

The Graph-based approach for SLAM was proposed in 1997 by Lu and Milios [37]. They presented a method to maintain all robot poses and spatial relationships between them, generated from odometry and scan matching, creating a network and further optimizing it at once. However, their approach was infeasible to perform in real-time [38] [16], and it was heavily dependent on the initial estimate [39]. In 1999, Gutmann and Konolige [39] proposed a method for global pose estimation based on the work of Lu and Milios, but with an incremental approach, specifically designed to work for long cycles and to avoid local minimum convergence. Other important contributions were made afterwards. Howard et al.[40], in 2001, modeled the spatial constraints between poses as springs and apllied a relaxation method for localization and mapping. In 2002, Duckett et al. [41] used Gauss-Seidel relaxation for constraint minimization, assuming a known orientation, a place recognition system and odometry information. In 2005, Frese et al. [42] developed a improved relaxation implementation.

Another Graph-SLAM approach was presented by Olson et al. in 2006 [43], using stochastic gradient descent to optimize the graph. This method was a considerable improvement for its computational efficiency, and also for the possibility to recover the robot trajectory from poor initial estimates. In 2007, Grisetti et al. [44] extended the algorithm presented by Olson using a tree-based parameterization with a better rate of convergence.

All previous methods were developed for a two-dimensional space. One of the first implementations that could be applied in both 2D and 3D systems was the $\sqrt{SAM}$ (*square root SAM*) developed by Dellaert and Kaess [35] in 2006. They took advantage of the sparse structure of the matrices associated with the Graph-SLAM problem, performing a sparse factorization of the information matrix into a square root form. In 2008, Kaess et al. [45] presented a variant of the previous work, called iSAM, performing incremental updates of the square root information matrix.

In 2011, Kümmerle et al. [38] presented the g^2o framework, a general Graph-SLAM formulation that can be applied to Bundle Adjustment, 2D and 3D Graph-SLAM problems. g^2o is an open-source C++ framework that is currently one of the state-of-the-art formulations for graph optimization, and it has been used as a back-end in several monocular, stereo and RGB-D SLAM implementations. g^2o can use different solvers such as Cholesky, Preconditioned Conjugate Gradient (PCG) and Levenberg-Marquardt (LM).

Other important Graph-SLAM formulations for 2D and 3D problems include TORO, MTKM and HOG-Man. TORO [46], presented by Grisetti et al. in 2009, is an open source C++ standalone implementation that uses a tree-based parameterization. MTKM [47], developed by Wagner et al., is an open-source least-squares optimization tool for MATLAB, that can be used for multi-sensor calibration and Graph-SLAM problems. HOG-Man (Hierarchical Optimization for Pose Graphs on Manifolds) is an open source C++ implementation of the approach developed by Grisetti et al. [48], which has a hierarchical optimization solution to Graph-SLAM.

To overcome parameterization problems in 3D optimization, Hertzberg et al. [49] proposed in 2008 the ⊞-method, to perform optimization on manifolds, which is used in previous cited frameworks, such as g2o, MTKM and HOG-Man. The ⊞-method and the manifold optimization theory are discussed in details in chapter 3.

1.3.3 Visual SLAM

The use of cameras in robotics increased with the development and improvement of computer vision and image processing techniques, such as feature detectors. The work of Davison and Murray [50], in 1998, was the first visual SLAM system with real time processing [51].

The visual odometry and SLAM research had a considerable improvement in the past years, with several important open source implementations, such as: Semi-direct Visual Odometry (SVO) [52], Large Scale Direct monocular SLAM (LSD SLAM) [53] and ORB SLAM [54].

1.3.4 RGB-D SLAM

The use of RGB-D cameras for robotics research is relatively recent. Henry et al. [17] developed in 2012 a RGB-D SLAM system using sparse visual features combined with ICP. Another early system was developed by Endres et al. [55] in 2012.

In 2011, Huang et al. [56] implemented a visual odometry system, named FOVIS, in a micro air vehicle using an RGB-D camera for localization and mapping using sparse visual features, but without a probabilistic formulation. Another visual odometry formulation for RGB-D sensors was developed by Whelan et al. [57] in 2013, integrating FOVIS and other visual odometry estimation methods with a GPU-based implementation.

Newcombe et al. [58] presented, in 2011, the KinectFusion algorithm, performing accurate real-time mapping with a volumetric representation and a GPU-based implementation.

In 2012, Sturm et al. [59], from the Technical University of Munich, presented a benchmark for the evaluation of RGB-D SLAM system, with several sequences of depth and color images recorded from a Microsoft Kinect, and their respective ground-truth camera poses obtained from a motion-capture system.

The first popular RGB-D SLAM system was implemented in 2014 by Endres et al. [60]. It uses matched sparse visual features to estimate motion between frames in the front-end, and the g^2o framework as back-end.

In 2015, Whelan et al. [61] presented Elastic Fusion, a RGB-D SLAM system without a pose graph, using dense camera tracking and GPU.

In 2017 the ORB-SLAM2 [62] was presented by Mur-Artal and Tardós as an extension of their previous work. It is an open-source hybrid formulation that can be used in systems with monocular cameras, stereo cameras and RGB-D sensors. The ORB-SLAM2 formulation uses ORB features [63], a place recognition methodology, loop closure detection and the g^2o framework with Levenberg-Marquadt for non-linear optimization of the pose-graph.

1.4 Objectives

- Develop an algorithm for simultaneous localization and mapping of mobile robots in indoor environments using only RGB and Depth information provided by an RGB-D camera, assuming a 6-DOF system and a static environment.
- Use a graph-based probabilistic formulation to deal with the uncertainties of sensor information and robot motion.
- Use ROS as a middleware of the system.
- Evaluate the algorithm using a benchmark dataset.
- Test the implementation in a low cost system composed by a commercial differential drive robot (iRobot Create [64]), a Microsoft Kinect v2 and a notebook.
- Develop a standalone pose-graph optimization tool for MATLAB, that can operate as back-end for a SLAM system.

1.5 Dissertation Outline

This dissertation is divided into 6 chapters that are structured as follows:

Chapter 2 presents the fundamental concepts and theoretical background used in this implementation, including basic probability theory concepts, types of orientation representations, the camera model, visual features, types of maps and ROS concepts.
Chapter 3 presents the development of a back-end pose-graph optimization tool for MATLAB with the respective dataset evaluation.
Chapter 4 details the hardware, the open source libraries used, the algorithms and steps of the SLAM system implementation .
Chapter 5 shows the results obtained from the experiments using the robot platform with a kinect, and the results of a numerical evaluation of the SLAM system using a benchmark dataset.
Chapter 6 presents the conclusions and suggestions for future work.

2
Theoretical Background

2.1
Probability Theory

In probabilistic robotics, the robot states, sensor measurements and control inputs are modeled as random variables [3]. If X is a random variable, $p(x)$ denotes the probability of X take a specific value x, which is stated in Eq. (2-1).

$$p(x) = p(X = x) \tag{2-1}$$

The Eq. (2-1) is defined for a discrete random variable. In SLAM algorithms the variables are continuous. A continuous random variable depends on a non-negative function called probability density function (PDF), which gives the probability of the variable assume a value in a specific interval [65].

2.1.1
Gaussian Distribution

The majority of SLAM formulations represent uncertainty using a multivariate Gaussian distribution. The Gaussian distribution, also called normal, is defined through its parameters, which are mean μ and the covariance matrix Σ. The probability density function of a multivariate normal distribution is:

$$p(x) = \mathcal{N}(x; \mu, \Sigma) = det(2\pi\Sigma)^{\frac{1}{2}} exp\Big\{ -\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\Big\} \tag{2-2}$$

where the mean μ is a vector and the covariance matrix Σ is a positive semidefinite symmetric matrix. This is also called the moments representation [3].

The Gaussian distribution can be parameterized in a different form, using the information matrix Ω instead of the covariance matrix, and the information vector ν instead of the mean vector. They are obtained according to Eqs. (2-3) and (2-4).

$$\nu = \Sigma^{-1}\mu \tag{2-3}$$

$$\Omega = \Sigma^{-1} \tag{2-4}$$

which leads to Eq. (2-5):

$$p(x) = \mathcal{N}(x; \nu, \Omega) = \eta\, exp\Big\{ -\frac{1}{2}x^T\Omega x + x^T\nu \Big\} \tag{2-5}$$

where η is a constant. This is also called the canonical representation and it is used in information filters and in the graph-based formulation. The canonical representation has several advantages in robotics applications, in comparison with the moments representation. For instance, it is just needed to set $\Omega = 0$ to represent global uncertainty in the information form. However, it becomes infinity in a covariance matrix [3].

2.1.2 Conditional Probability

The conditional probability $p(x|y)$ expresses the change in the belief of x when there is knowledge about another related random variable y. The conditional probability is defined by Eq. (2-6).

$$p(x|y) = \frac{p(x,y)}{p(y)} \tag{2-6}$$

where $p(x,y)$ is called the joint distribution.

The Bayes' rule, stated in Eq. (2-7), relates two conditional probabilities.

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{2-7}$$

Bayes' rule is the base of the probabilistic formulations for SLAM. The probability $p(x)$ is called prior probability, which is the knowledge about X before incorporating the knowledge about Y, and $p(x|y)$ is called the posterior probability distribution of X. The Bayes' rule allows to infer $p(x|y)$ using the prior knowledge and $p(y|x)$, which is usually easier to obtain in probabilistic robotics problems [3].

The theorem of total probability, stated in Eq. (2-8) for the continuous case, is used to calculate probabilities with a conditional dependence.

$$p(x) = \int p(x|y)p(y)dy \tag{2-8}$$

2.1.3 Independence

The independence between two random variables X and Y arises when the knowledge about Y does not alter the probability of X, which is stated in Eq.(2-9).

$$p(x|y) = p(x) \tag{2-9}$$

The concept of independence is very important in SLAM because it is related to one of the assumptions made in the probabilistic formulation.

2.2 Probabilistic Formulation of SLAM

In the SLAM problem, the robot moves in an unknown environment and does not have prior knowledge of it own poses x, modeled as random variables $x_{1:T} = [x_1, ..., x_T]$. Instead, only odometry information $u_{1:T} = [u_1, ..., u_T]$ and sensor measurements $z_{1:T} = [z_1, ..., z_T]$ are available. Thus, the problem consists in the estimation of the posterior probability distribution of the trajectory of the robot and the map m, given the measurements of the environment, which is stated in Eq. (2-10).

$$p(x_{1:T}, m|z_{1:T}, u_{1:T}) \tag{2-10}$$

There are several map representations, such as landmarks, occupancy grids, surface maps and pointclouds. The poses and odometry information can be represented as 3D rigid transformations in SE(3). In this work they are obtained using color and depth information provided by the kinect, and performing a visual odometry estimation method.

Two important assumptions are made to estimate the posterior probability distribution in (2-10). The world is assumed to be static and the measurements are considered independent.

The SLAM problem can be represented as a Dynamic Bayesian Network (DBN), shown in Fig. 2.1. In DBNs, the nodes correspond to the random variables of the problem, connected if there is a conditional dependence [16]. DBNs are used to describe filtering processes, with the motion model $p(x_t|x_{t-1}, u)$, and the measurement model $p(z_t|x_t, m_t)$. The motion model represents the probability that the robot is in the state x_t at time t given the previous state and the odometry information. The measurement model

represents the probability of the measurement z_t is made give the current state of the robot [16].

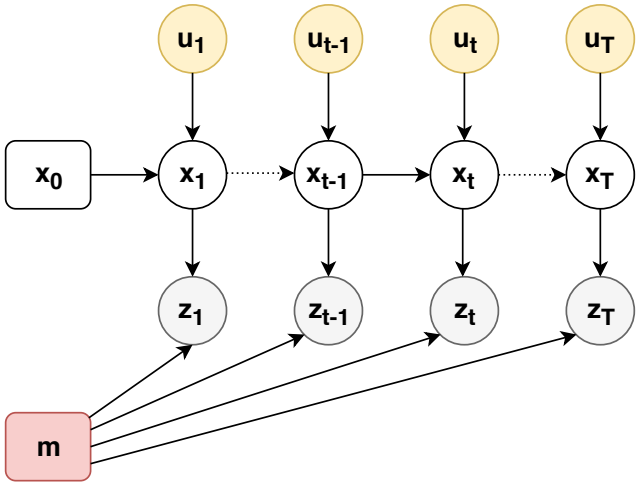


Figure 2.1: SLAM problem as a Dynamic Bayesian Network

Another representation of the SLAM problem is the Graph-based formulation, exemplified in Fig. 2.2. The nodes represent the poses of the robot, and the edges represent spatial constraints between two poses, locally affected by Gaussian noise, and resulted from measurements and odometry information [16]. In Fig. 2.2, e_{ij} is an error function that measures how well the states x_i and x_j satisfy the constraints [38]. The objective is to minimize this errors, finding the configuration that best satisfies the constraints.

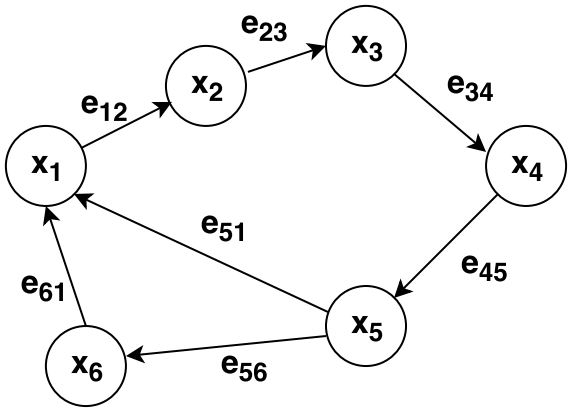


Figure 2.2: Pose-graph representation of the SLAM problem

Unlike filtering approaches, in graph-slam there is no requirement to distinguish motion and measurement models, both are considered factors in the graph [5].

2.3 Least Squares Problem

The Graph-SLAM formulation has an intrinsic connection with the Least Squares problem. The Least Squares is an optimization problem, which is generally defined as finding the minimum value x^* of an objective function $F(x)$, as stated in Eq. (2-11).

$$x^* = \underset{x}{\operatorname{argmin}} F(x) \tag{2-11}$$

The necessary condition for a value x^* to be a local minimizer of the cost function $F(x)$ is that the derivative of F in x^* should be equal to zero [66], as stated in Eq. (2-12).

$$F'(x^*) = 0 \tag{2-12}$$

According to Nocedal and Wright [67], the objective function for the general least squares problem can be defined as Eq. (2-13).

$$F(x) = \frac{1}{2}\sum_{i=1}^{m}(f_i(x))^2 \tag{2-13}$$

Thus, rewriting in vector formulation, the least squares problem can be stated as:

$$x^* = \underset{x}{\operatorname{argmin}} \frac{1}{2}\mathbf{f}(x)^T\mathbf{f}(x) \tag{2-14}$$

where $\mathbf{f}$ is a vector function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$.

2.3.1 Linear Least Squares

In a linear least squares problem, $\mathbf{f}$ is linearly dependent on x and is stated in Eq. (2-15).

$$\mathbf{f}(x) = b - Ax \tag{2-15}$$

where $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{mxn}$. The derivative of the cost function is given by:

$$F'(x) = -A^T(b - Ax) \tag{2-16}$$

which leads to:

$$(A^T A)x = A^T b \tag{2-17}$$

Eq. (2-17) can be solved using QR or Cholesky decomposition, for example [67]. In Fig. 2.3 is shown the solution of a linear least squares problem of line fitting.

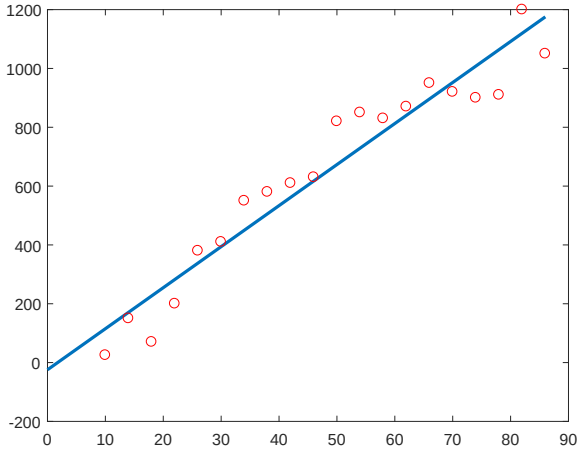


Figure 2.3: Linear least squares solution

2.3.2 Non-linear Least Squares

For non-linear problems, the cost function F(x) can be linearized by the Taylor expansion [66], stated in Eq. (2-18).

$$\mathbf{f}(x + \delta_x) = \mathbf{f}(x) + J(x)\delta_x + \mathcal{O}(||\delta_x||)^2 \tag{2-18}$$

where J is the Jacobian matrix, containing the first partial derivatives of the function, as shown in Eq. (2-19).

$$J(x) = \left[\frac{\partial f_j}{\partial x_i}\right]_{\substack{1<j<m \\ 1<i<n}} \tag{2-19}$$

Gauss-Newton is a popular method for solving non-linear least squares problems, that only consider the first order terms of the Taylor expansion (2-18). It has the advantage of not requiring the calculation of the second derivatives, which can be computationally expensive [67]. The solution is found by solving the following system:

$$(J(x)^T J(x))\delta_x = -J(x)f \tag{2-20}$$

with the incremental step:

$$x_{k+1} = x_k + \delta_{x_k} \tag{2-21}$$

The Levenberg-Marquadt is variant of Gauss-Newton, with an added damping factor to control convergence [66].

2.4 Rigid Motion in $\mathbb{R}^3$ and Attitude Representations

A robot movement is considered a rigid body motion, which is defined as a motion that preserves distance between points and angles between vectors [68]. A rigid body motion is composed by translation and rotation. The pose of a rigid body in three-dimensional space is described by its position and orientation. The position of the body is usually defined by the position of the origin that describes body coordinates $[x', y', z']$, expressed in fixed world coordinates $[x, y, z]$ [69], as shown in Fig. 2.4. For orientation, or attitude, there are several possible parameterizations in three-dimensional space.

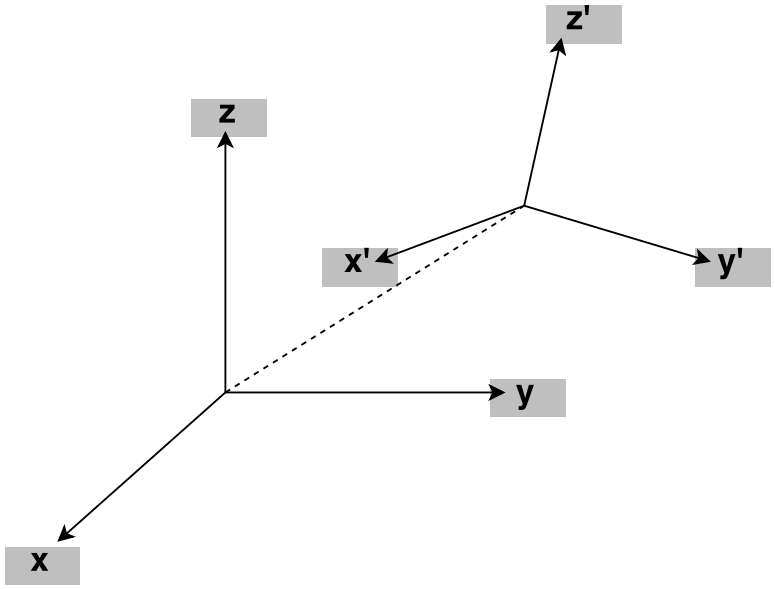


Figure 2.4: Position of the body in world coordinates

2.4.1 Rotation Matrices

The orientation of a rigid body can be described as a matrix transformation between a body coordinate frame and the fixed world coordinate frame [68]. This matrix has the following form:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2-22}$$

R is a 3x3 matrix whose columms are mutually orthogonal. The set of all 3x3 rotation matrices is called SO(3), the Special Orthogonal group [70]. This set is more formally defined by Eq. (2-23).

$$SO(3) = \{R \in \mathbb{R}^3 : RR^T = I, detR = +1\} \tag{2-23}$$

A set G is called a group if it satisfies the axioms of closure, identity, inverse and associativity. Therefore, $SO(3)$ is a group under the operation of matrix multiplication, with the identity matrix as the identity element. It is also called the rotation group of $\mathbb{R}^3$ [68].

As an example, Eq. (2-24) defines a rotation matrix that represents a rotation about the x-axis by an angle θ.

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & sin(\theta) \\ 0 & -sin(\theta) & cos(\theta) \end{bmatrix} \tag{2-24}$$

To represent a rigid motion in 3D composed by a translation and a rotation, it is used a 4x4 transformation matrix, written in homogeneous coordinates:

$$T = \begin{bmatrix} & & & x \\ & R & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2-25}$$

where R is a 3x3 rotation matrix and x, y and z are the coordinates of a translation vector. The set of all transformations that can be applied to a rigid body is called the Special Euclidean Group [68], as stated in Eq. (2-26).

$$SE(3) = \{(t, R), t \in \mathbb{R}^3, R \in SO(3)\} \tag{2-26}$$

where t represents the translation and R represents the rotation.

Given a pose P_{bc} in frame c, relative to frame b, and a pose P_{ab} in frame b, relative to frame a, then the pose in frame c relative to frame a is given by:[68]

$$P_{ac} = P_{ab}P_{bc} = \begin{bmatrix} R_{ab}R_{bc} & R_{ab}t_{bc} + t_{ab} \\ 0 & 1 \end{bmatrix} \tag{2-27}$$

2.4.2 Euler Angles

The most commom and intuitive rigid body attitude representation is the Euler angle parameterization [69]. It corresponds to a sequence of three different rotations: yaw (ϕ), pitch (χ) and roll (ψ). In Fig. 2.5 is shown one of the conventions that can be employed.

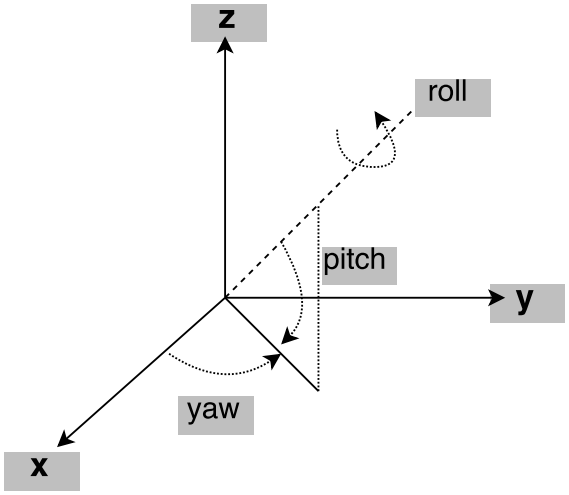


Figure 2.5: A yaw, pitch and roll convention

This is the most compact rotation representation in three-dimensional space, because it only requires three variables to describe the orientation. However, this representation has a major disadvantage. The minimal representation generates singularities, which is called the Gimbal Lock problem [71]. It occurs when two axes are aligned, leading to a loss of degree-of-freedom. In the convention exemplified in Fig. 2.5, it happens when pitch approaches $\pm\pi/2$, when a change in roll implies a change in yaw.

The Eq. (2-28) defines a pose described as a 3D translation plus a Euler angle representation of the orientation.

$$\mathbf{p} = [x, y, z, \phi, \chi, \psi] \tag{2-28}$$

2.4.3 Quaternions

Quaternions were first described by W. R. Hamilton in 1843 [72], and can be seen as a generalization of complex numbers, with three different imaginary parts [73]. In Eq.(2-29) is shown a general form of a quaternion.

$$q = q_x i + q_y j + q_z k + q_w \tag{2-29}$$

where

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2-30}$$

and $q_x, q_y, q_z, q_w \in \mathbb{R}$. Thus, it is a sum of a scalar q_w and a vector part $q_v = [q_x, q_y, q_z]$.

The complex conjugate q^* of a quaternion q is given by Eq. (2-31).

$$q^* = -q_x i - q_y j - q_z k + q_w \tag{2-31}$$

The set of unit-length quaternions, also called unit quaternions, are a sub group of quaternions that are used to represent rotations. They satisfy the condition $\|q_u\| = 1$. The unit quaternion is given by Eq. (2-32).

$$q_u = \frac{q}{\|q\|} = \frac{1}{\sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}} q \tag{2-32}$$

A full quaternion belongs to the $\mathbb{R}^4$ space. However, the unit quaternion belongs to a subspace of $\mathbb{R}^4$ called $\mathbb{S}^3$, which represents the unit sphere in $\mathbb{R}^4$ [74][75].

There are several advantages of using the unit quaternion notation, specially for representing rotations [73]. These advantages will be explained in details in chapter 3.

Thus, a pose in $\mathbb{R}^3$ can be represented by a translation vector and a unit quaternion, as stated in Eq. (2-33).

$$\mathbf{p} = [x, y, z, q_x, q_y, q_z, q_w] \tag{2-33}$$

The quaternion parameters can also be written in terms of unit vectors u_x, u_y, u_z and an angle θ, which is stated by Eqs. (2-34).

$$\begin{aligned} q_r &= cos(\theta/2) \\ q_x &= sin(\theta/2)u_x \\ q_y &= sin(\theta/2)u_y \\ q_z &= sin(\theta/2)u_z \end{aligned} \tag{2-34}$$

Another important property of quaternions is the inverse, which is defined in Eq. (2-35).

$$q^{-1} = \frac{q^*}{\|q\|^2} \tag{2-35}$$

If it is a unit quaternion, the inverse is the conjugate. One advantage of the quaternion notation is the facility to perform inversion, in comparison to rotation matrices.

The quaternion multiplication is similar to a polynomial multiplication. The multiplication between two quaternions p and q is given by:

$$\begin{aligned} pq =& (p_w + ip_x + jp_y + kp_z)(q_w + iq_x + jq_y + kq_z) = \\ & (p_w q_w - p_x q_x - p_y q_y - p_z q_z) \\ & + i(p_w q_x + p_x q_w + p_y q_z - p_z q_y) \\ & + j(p_w q_y + p_y q_w + p_z q_x - p_x q_z) \\ & + k(p_w q_z + p_z q_w + p_x q_y - p_y q_x) \end{aligned} \tag{2-36}$$

An important characteristic of quaternion multiplication is that commutativity is not preserved [76], in order words:

$$pq \neq qp \tag{2-37}$$

2.5 Camera Model

In order to correctly use the sensor information, this section defines the camera model used in this implementation, as well as its parameters. The camera model is used to map information from world coordinates to image coordinates, and to define the position of the camera in world coordinates. In Fig. 2.6 is shown the camera model used. It is named as the pinhole camera model [77].

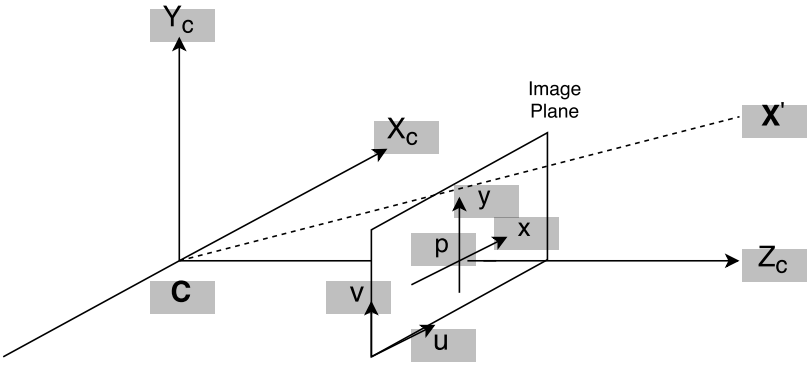


Figure 2.6: Camera Model

The center C of the euclidean coordinate system $[X_c, Y_c, Z_c]$ is also the center of the camera. The image plane is located at $Z_c = f$, where f is called the focal length of the camera. P is called the principal point, where the Z

axis meet the image plane. The origin of coordinates in image plane usually is not at the principal point. The coordinates of the principal point in the image plane are called c_x and c_y.

Using this model, a point in space with coordinates $\mathbf{X'} = [X', Y', Z']$ can be mapped to a point $[fX'/Z' + c_x, fY'/Z' + c_y]$ on the image plane [77]. This mapping process can be rewritten as stated in Eq. (2-38).

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} \tag{2-38}$$

where $[u, v, 1]$ are the coordinates of the mapped point in the image plane, written in homogeneous coordinates, and K is the matrix defined by Eq. (2-39).

$$K = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2-39}$$

K is called calibration matrix or intrinsic matrix, because it contains the intrinsic parameters of the camera. The extrinsic parameters express where the camera is located in world frame coordinates. These parameters are composed by a 3x3 rotation matrix and a translation vector [77], defined in Eq (2-40).

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2-40}$$

where $[X, Y, Z, 1]$ is a point in world frame homogeneous coordinates, and $[X_c, Y_c, Z_c, 1]$ is the same point in the camera coordinate frame.

2.6 Visual Features

In order to estimate motion between frames, the robot needs to detect and track important parts of the scene observed by the kinect, using the color images, which is done by a process called feature extraction. There are two main steps in feature extraction: keypoint detection and feature description [78].

The keypoint detector searches for a region in the image with a strong intensity variation, such as a corner. The keypoint is at the center of this region. The feature descriptor is a multidimensional vector that describes the region around the keypoint, obtained computing the orientations of the surrounding points [78].

These features need to be invariant to scale, rotation and translation, so they can be matched in sequential frames during camera motion. The Scale Invariant Feature Transform (SIFT) features were developed by Lowe in 2004 [79]. It is one of the most popular methods of feature detection and description in robotics and computer vision.

2.6.1 ORB Features

Despite of its popularity in computer vision applications, including object recognition and visual mapping, SIFT features are computationally expensive and can be restrictive to real-time applications, such as SLAM. ORB features, developed by Rublee et al. [63], overcome this problem, having similar accuracy to SIFT, with a speed performance almost two orders of magnitude faster.

ORB is based on the FAST [80] keypoint detector and the BRIEF [81] descriptor. SIFT uses histograms of gradient computations to describe the orientation in a keypoint, which is computationally expensive. ORB, alternatively, uses an intensity centroid approach to describe orientations, which gives a single dominant result for each keypoint [63].

2.7 Map representations

Aside from its importance in the SLAM problem solving, an accurate map has a major influence in the ability of the robot to perform other tasks, such as motion planing and collision avoidance.

Formally, a metric map is a structure that symbolically encodes the geometrical aspects of the environment [5]. There are two main metric representations in two-dimensional problems: landmark-based maps and occupancy grid maps.

A landmark map is a set of sparse point locations that are assumed to be distinguishable [82]. In Fig. 2.7 is shown a map of landmarks in an EKF-SLAM problem. The black crosses represent the landmarks and the red circle represent the position of the robot. It has the advantage of being a compact representation. The main problem regarding landmark maps is to perform

data association, in other words, to be able to distinguish different landmarks at different positions.

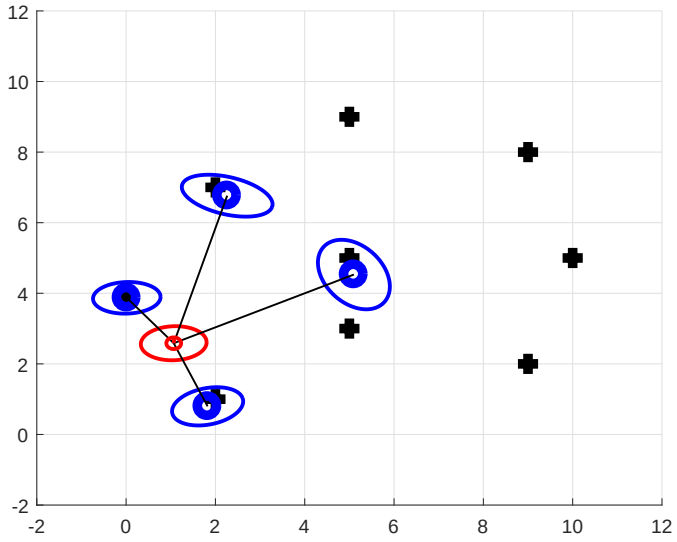


Figure 2.7: Map of Landmarks

Occupancy grid is a map representation developed by Elfes and Moravec [83][84]. It consists in a discretization of the environment into a regular grid square. Each cell is considered either occuppied or free. If the map is probabilistic, the cells have a percentual occupancy belief. In Fig. 2.8 is shown an example of mapping with known poses using an occupancy grid map. The green lines represent the trajectory of the robot.

For many years two-dimensional maps were widely used in mobile robots localization and navigation tasks. However, several problems in mobile robotics require a three-dimensional model. Complex environments, such as aerial and underwater, cannot rely on 2D maps. 3D mapping is also required in certain indoor applications. For instance, a reliable colision avoidance system requires a 3D map, because only 2D information cannot prevent colision into objects with irregular shapes. Furthermore, if the mobile robot has an arm and needs to identify and manipulate objects in a scene, it would need a 3D map.

There are several 3D map representations. They are usually subcategorized into sparse or dense maps. The sparse representation is also called feature-based, as it represent the environment as a set of sparse 3D landmarks corresponding to features [5]. Figure 2.9 shows a feature-based map generated by ORB-SLAM [54]. The camera poses are represented in blue, and the red points are the features.

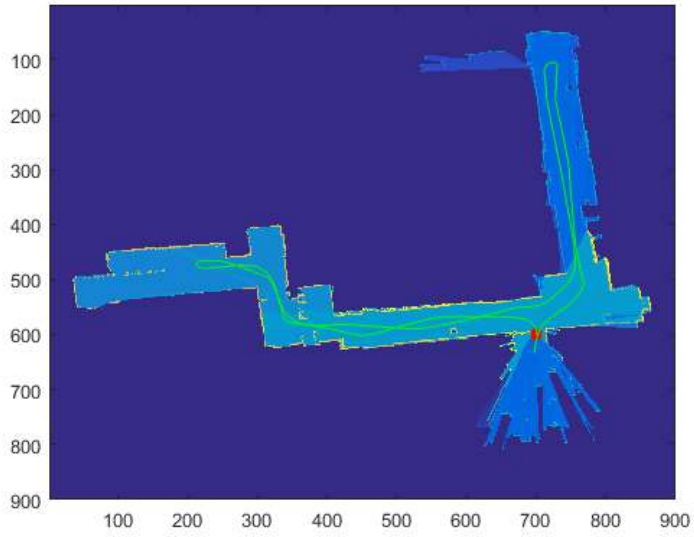


Figure 2.8: Grid Map

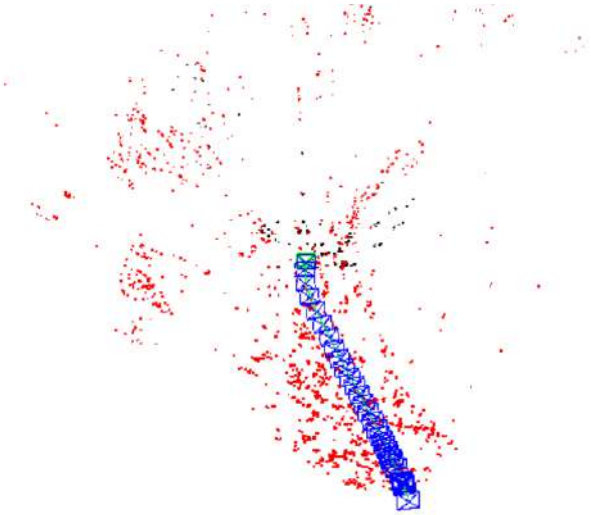


Figure 2.9: Feature Map

In contrast to sparse representations, dense representations aim to provide more detailed information about the environment. There are several types of dense representations. For instance, surfel maps [85], planar colored surfaces used by Henry et al. [17]. Also, voxel grids, presented by Roth-Tabak and Jain (1989) [86] and Moravec [87], which discretize the environment in cubic volumes. Furthermore, octree-based maps, such as OctoMap developed by Hornung et al. [88], with a probabilistic occupancy estimation.

However, this work uses point clouds, one of the most popular 3D representations in robotics.

2.7.1 Point Cloud

A Point Cloud is a structure used to represent 3D raw data with geometric coordinates and color information of a collection of points. The point cloud representation has an extensive history of usage in robotics applications, with stereo cameras, RGB-D sensors and 3D laser scanners [5]. In Fig. 2.10 is shown an example of a point cloud generated from depth and color information provided by a Kinect v2.

The Point Cloud Library (PCL) [89] was presented in 2010 as an open source library for point cloud processing. It is one of the greatest initiatives in open 3D perception, containing several state-of-the-art algorithms for point cloud registration, filtering and visualization. The PCL is used to create, vizualize, manipulate and store the pointclouds.

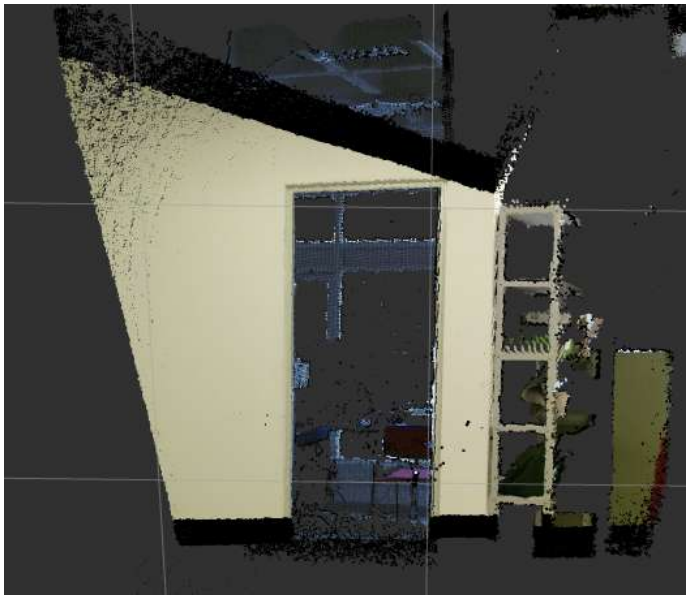


Figure 2.10: Point Cloud

The point cloud representation has two major drawbacks. First, it does not provide a direct representation of free or unknown space, as opposed to the octomap representation [88]. Thus, it is not feasible to perform a collision-free navigation using a point cloud representation. Second, there is no upper bound for memory consumption [88]. Figure 2.11, from Hornung et al. [88], shows a visual comparison between a point cloud, on the left, and a OctoMap, on the right.

Despite these problems, the point cloud is a practical representation, with an easy implementation, several state-of-the-art tools, and can be further post-processed to another representation, such as OctoMaps, to allow navigation and other tasks. However, this post-process is not in the scope of this work.

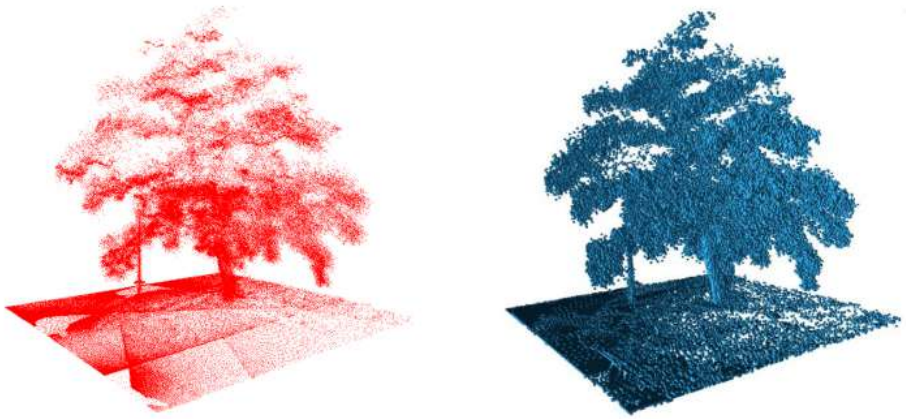


Figure 2.11: Comparison between a point cloud and a OctoMap

2.8 Iterative Closest Point

One technique to estimate the motion between two frames is the Iterative Closest Point (ICP) algorithm. The ICP technique has several variations and implementations. The most cited work about ICP was presented by Besl and McKay [90]. The main ideia of ICP is to iterativelly compute the rigid transformation between two sets of points which minimizes the distance between the pairs [91].

Given two point clouds X and Y and a initial guess T_0 for the transformation between them, the first step is to compute the correspondences between the two sets of points. In other words, to find, for every point in the first set, the closest point in the other set [92]. This correspondence is evaluated with a given threshold d_{max}. The second step is to find the new transformation by minimizing a error metric that measures the distance between the corresponding pairs [91]. These two steps are iterated until the convergence criteria is reached, which is when a change in the solution falls below a given threshold ϵ.

There are several types of error metrics used in ICP and also several methods to minimize it. One classic metric is the sum of squared distances between corresponding points [93], stated by Eq. (2-41).

$$E(R,t) = \frac{1}{N_p}\sum_{i=1}^{N_p} \|y_i - (R\,x_i + t)\|^2 \tag{2-41}$$

where N_p is the number of point correspondences, R is the rotation matrix and t is the translation vector of the transformation matrix.

One method to solve it is to use singular value decomposition (SVD), as presented by Umeyama [94] in 1991. According to Umeyama, the minimal value of E is found applying the SVD in the covariance matrix of X and Y:

$$C_{xy} = \frac{1}{N_p}\sum_{i=1}^{N_p}(y_i - \mu_y)(x_i - \mu_x)^T \tag{2-42}$$

where μ_x and μ_p are the mean vector of the two sets of points, given by Eqs (2-43) and (2-44).

$$\mu_x = \frac{1}{N_x}\sum_{i=1}^{N_x} x_i \tag{2-43}$$

$$\mu_y = \frac{1}{N_p}\sum_{i=1}^{N_y} y_i \tag{2-44}$$

The SVD of C_{xy} is given by:

$$\mathrm{SVD}(C_{xy}) = USV^T \tag{2-45}$$

Thus, R and t are given by:[95]

$$R = U\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & det(UV^T)\end{bmatrix}V^T \tag{2-46}$$

$$t = \mu_y - R\mu_x \tag{2-47}$$

The standard steps of ICP are described in algorithm 1, according to Segal et al. [91].

Algorithm 1 ICP

1: $X \leftarrow$ point cloud 1
2: $Y \leftarrow$ point cloud 2
3: $T \leftarrow T_0$
4: **while** not converged **do**

for $i \leftarrow 1$ **to** N_p **do**
  $b_i \leftarrow \text{FindClosestPointInY}(T \cdot x_i)$
  if $\|b_i - T \cdot x_i\| \leq d_{max}$ **then**
    $\alpha_i \leftarrow 1$
  else
    $\alpha_i \leftarrow 0$
  endif
endfor
$T \leftarrow \underset{T}{\text{argmin}}(\sum_{i=1}^{N_p} \alpha_i \|T \cdot x_i - b_i\|^2)$
endwhile

2.9 Random Sample Consensus

The Random Sample Consensus (RANSAC), presented by Fischler and Bolles [96] in 1981, is a robust estimation algorithm that fits a model to data in the presence of outliers [77]. For example, the line fitting problem from section 2.3.1, in which, given a set of 2D points, a line must be estimated by minimizing the sum of squared perpendicular distances from the points. If one of the points is incorrect, in other words, too far from the others, it can ruin the estimation. The RANSAC algorithm can overcome this problem rejecting this point, called outlier, given a threshold.

The general procedure for RANSAC estimation is described in algorithm 2. Given a data set S, a model M and a threshold, the minimum set k of data required to estimate the model is sampled from S and a hypothesis model is created. The rest of the data is tested with this hypothesis and evaluated using the threshold. If the mean error of these evaluated points is within the hypothesis threshold, a new hypothesis is evaluated using the inliers. This process is repeated until the best model is found.

Algorithm 2 RANSAC

$S \leftarrow$ data
$inliers \leftarrow 0$
for N iterations **do**
  $s \leftarrow$ sample k random points from S
  compute hypothesis from s
  for points **do**
    $error \leftarrow$ evaluate point using hypothesis
    if $error < threshold$ **then**
      $inliers \mathrel{+}= point$
    endif
  endfor

```
    compute mean error
    if mean_error < hypothesis_threshold then
        recompute hypothesis from inliers
        if hypothesis better than model then
            bestModel ← hypothesis
            bestInliers ← inliers
        endif
    endif
endfor
```

2.10 ROS

ROS is built as a large number of small programs that communicate one another through messages. These messages can be sensor input, debug messages or control output. This setup creates a graph-based structure where the nodes are the programs and the edges are the messages, carried by topics. These programs can publish or subscribe to topics, depending on its use [97]. The master node, part of roscore, is the node that connects all nodes of the ROS system, as shown in Fig. 2.12.

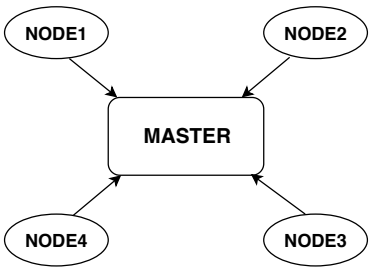


Figure 2.12: ROS Master

If a node is subscribed to more than one topic, they can be syncronized so the algorithm does not proceed until all specified messages with the same timestamp arrive. In Fig. 2.13 is shown an example of a topic subscribing and advertising connection.

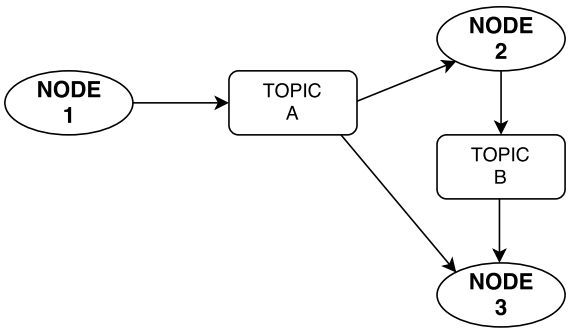


Figure 2.13: Node communication

In Fig. 2.14 is shown an example of a ROS graph that corresponds to a communication with a image topic between the kinect driver and a image processing code. The driver is publishing the image topic and the calibration node is subscribing to it.

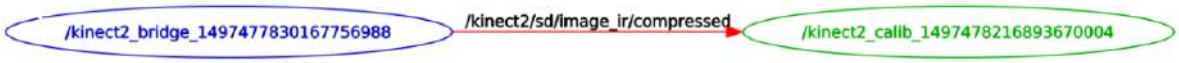


Figure 2.14: ROS Graph

2.10.1 rviz

ROS has a 3D vizualization environment called rviz that displays sensor data in real time with custom vizualization markers. It is a very useful tool for code debugging and data verification. In Fig. 2.15 is shown the rviz interface with a point cloud and its corresponding color image.

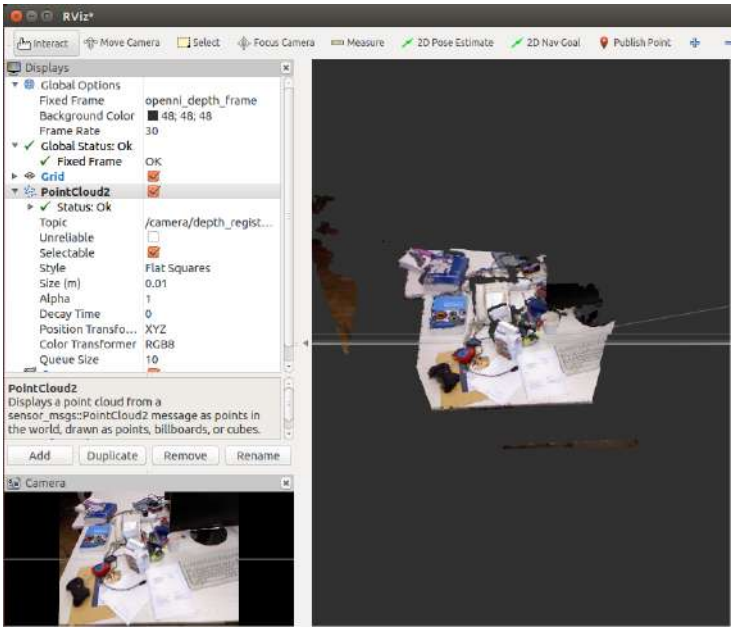


Figure 2.15: Point Cloud in rviz

2.10.2 Rosbag

ROS has a special data format called ".bag". The rosbag package has a set of tools to read and write bag files. The bag format is very useful to save ROS topics and sensor data and reproduce the same experiment in

different moments. Figure 2.16 shows a graphical example of a rosbag file publishing several kinect topics, such as depth images, color images and camera information. The topics are represented by the rectangular boxes.

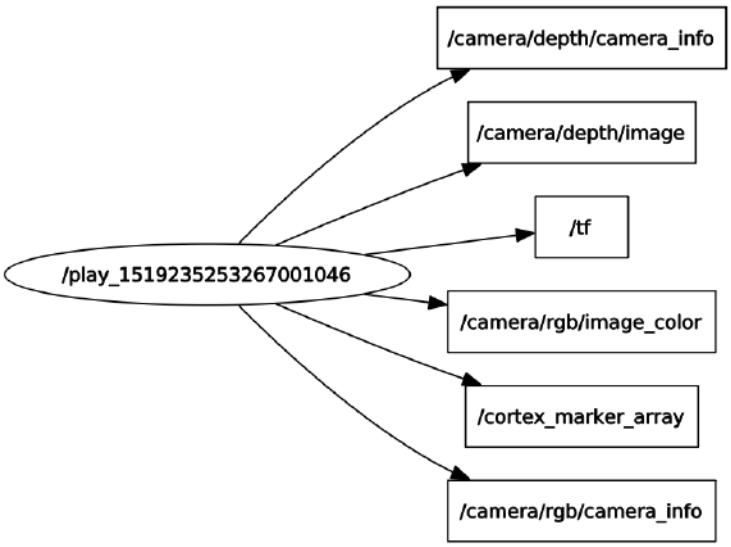


Figure 2.16: Rosbag publishing kinect data

3
Pose-Graph Optimization tool for MATLAB

This chapter details the theoretical framework of the Graph-based optimization, and presents the implementation of a pose-graph optimization tool for MATLAB that can operate as an effective SLAM back-end, based on the works of Grisetti et al. [16], Kümmerle et al. [38] and Wagner et al.[47].

3.1
Pose-Graph

The graph is composed by a set of nodes $x = [x_1, \ldots, x_T]$ that represent the positions of the robot in a plane or in space, depending if the problem is two-dimensional or three-dimensional. It is assumed that the robot has odometry information and range measurements. The relative transformation between two poses x_i and x_j is called the predicted measurement $\hat{z}_{ij}$. The real observations are represented by z_{ij}. The error function is computed by the difference between the measurement prediction and the real measurement [16], as stated in Eq. (3-1).

$$e_{ij}(x_i, x_j) = z_{ij} - \hat{z}_{ij} \tag{3-1}$$

When the robot performs a movement, going from position i to position j, a node x_j is created and also an edge e_{ij} between x_i and x_j. An edge is also created if the robot revisits a previous known location, a process called loop closure.

The processes of graph creation and loop closure detection are implemented in the front-end of a SLAM system, and are detailed in Chapter 4, for the particular case of RGB-D SLAM.

3.2
Graph Optimization as a Non-linear Least Squares Problem

The objetive of the graph-based SLAM problem is to find the trajectory of the robot that best explains the constraints between poses, imposed by the measurements. This is done by estimating the state $\hat{x}$ that maximizes the posterior belief [98], stated by Eq. (3-2).

$$\hat{x} = \underset{x}{\operatorname{argmax}}\, p(x|z) \tag{3-2}$$

However, usually $p(x|z)$ is hard to obtain due to the non-linearities associated with the map between the measurements and the states. Using Bayes' Theorem Eq. (3-2) can be rewritten as:

$$\hat{x} = \underset{x}{\operatorname{argmax}}\, \frac{p(z|x)p(x)}{p(z)} \tag{3-3}$$

Since $p(z)$ is independent of x and there is no knowledge about the prior state, $p(x)$ and $p(z)$ can be dropped [5], and the problem becomes a maximum likelihood estimation:

$$\hat{x} = \underset{x}{\operatorname{argmax}}\, p(z|x) \tag{3-4}$$

Assuming that the measurements are conditionally independent, the problem factorizes into:

$$\hat{x} = \underset{x}{\operatorname{argmax}} \prod_{k=1}^{n} p(z_k|x) \tag{3-5}$$

With the assumption of locally Gaussian measurements, the likelihood of the measurements will also be Gaussian:

$$\hat{x} = \underset{x}{\operatorname{argmax}} \prod exp(-e_{ij}^T(x_i, x_j, z_{ij})\Omega_{ij}e_{ij}(x_i, x_j, z_{ij})) \tag{3-6}$$

where Ω is the information matrix associated with each measurement. Taking the logarithm to transform the product into a sum:

$$\hat{x} = \underset{x}{\operatorname{argmax}} \sum_{ij} -e_{ij}^T(x_i, x_j, z_{ij})\Omega_{ij}e_{ij}(x_i, x_j, z_{ij}) \tag{3-7}$$

Removing the negative signal, it becomes a minimization problem:

$$\hat{x} = \underset{x}{\operatorname{argmin}} \sum_{ij} e_{ij}^T(x_i, x_j, z_{ij})\Omega_{ij}e_{ij}(x_i, x_j, z_{ij}) \tag{3-8}$$

which has the same structure of a non-linear least squares problem:

$$x^* = \underset{x}{\operatorname{argmin}}\, F(x) \tag{3-9}$$

where

$$F(x) = \sum e_{ij}^T \Omega e_{ij} \tag{3-10}$$

and can be solved using standard optimization methods such as Gauss-Newton or Levenberg-Marquardt, described in chapter 2.

A solution to the non-linear least squares problem is to use the first order Taylor expansion around the initial guess $\check{x}$ to approximate the error function, as stated in eq. (3-11).

$$e_{ij}(\check{x} + \delta x) \approx e_{ij} + J_{ij}\delta_x \tag{3-11}$$

where J_{ij} is the Jacobian of the error function computed in $\check{x}$. Thus, the cost function F of an observation between nodes i and j can be obtained rewriting a parcel of the sum in Eq. (3-8) with the local approximation of Eq. (3-11).

$$F_{ij}(\check{x} + \delta x) \approx (e_{ij} + J_{ij}\delta_x)^T \Omega_{ij}(e_{ij} + J_{ij}\delta_x) \tag{3-12}$$

The global cost function can be found with the sum of all local approximations:

$$F(\check{x}+\delta x) = \sum F_{ij}(\check{x}+\delta x) \approx \sum (e_{ij}^T \Omega_{ij} e_{ij} + 2e_{ij}^T \Omega_{ij} J_{ij}\delta_x + \delta_x^T J_{ij}^T \Omega_{ij} J_{ij}\delta_x) \tag{3-13}$$

Eq. (3-13) can be minimized solving the following linear system:

$$H\delta_x = -b \tag{3-14}$$

where

$$H = \sum J_{ij}^T \Omega_{ij} J_{ij} = J^T \Omega J \tag{3-15}$$

$$b = \sum J_{ij}^T \Omega_{ij} e_{ij} = J^T \Omega e \tag{3-16}$$

The solution for one iteration is then obtained by adding the increments to initial guess, as stated in Eq. (3-17).

$$x^* = \check{x} + \delta_x \tag{3-17}$$

The formulation is described using pseudocode notation in algorithm 3.

Algorithm 3 Pose Graph Optimization

```
1: procedure READ GRAPH
2:     x ← vertices
3:     z ← edges
4:     Ω_ij ← information matrices
5: endprocedure
6: while not converged do
7:     preallocate H and b
8:     for all measurements do
9:         compute error function e_ij
10:        compute Jacobians of the error function with respect to the nodes i and j
11:        compute the contribution of this measurement to H
12:        compute b
13:    endfor
14:    δ_x ← solve(Hδ_x = −b)
15:    x += δ_x
16: endwhile
```

A 1D numerical example was elaborated for a better comprehension of the methodology.

3.2.1 1D Example

Considering a robot traveling in a 1D environment, a loop closure occur when the robot returns to the initial point, as shown in Fig. 3.1. The edges e_{01}, e_{12}, e_{23} and e_{34} are obtained with visual odometry, and the edge e_{40} is the loop closure.

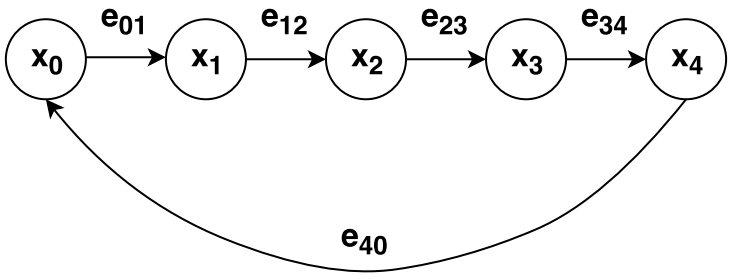


Figure 3.1: 1D pose-graph

The real initial state of the system is:

$$x_{groundtruth} = [0, 0.5, 2, 1.5, 0] \tag{3-18}$$

In table 3.1 is shown the measurements in comparison with the ground truth data. It is assumed that the visual odometry is not very accurate, but the system has a good loop closure detector, given that it was able to correctly estimate the initial position.

Table 3.1: Parameters of 1D Graph optimization example

Measurement	**Measured**	**Ground Truth**
z_{01} (visual odometry)	0.6	0.5
z_{12} (visual odometry)	1.6	1.5
z_{23} (visual odometry)	-0.5	-0.5
z_{34} (visual odometry)	-1.4	-1.5
z_{40} (loop closure)	0	0

According to the visual odometry, the predicted state is:

$$x_{predicted} = [0, 0.6, 2.2, 1.7, 0.3] \tag{3-19}$$

Comparing to the ground truth, the initial RMSE error is 18.97%.
The error vector is given by Eq. (3-1).

$$e(x) = \begin{bmatrix} z_{01} - (x_1 - x_0) \\ z_{12} - (x_2 - x_1) \\ z_{23} - (x_3 - x_2) \\ z_{34} - (x_4 - x_3) \\ z_{40} - (x_0 - x_4) \\ z_0 - x_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0.30 \\ 0 \end{bmatrix} \tag{3-20}$$

The last constraint is to assure that the pose x_0 is at the origin of the coordinate system. The Jacobian matrix is obtained calculating the derivative of the error function in terms of each pose.

$$J = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3-21}$$

The information matrix is given:

$$\Omega = \begin{bmatrix} \Omega_{01} & 0 & 0 & 0 & 0 & 0 \\ 0 & \Omega_{12} & 0 & 0 & 0 & 0 \\ 0 & 0 & \Omega_{23} & 0 & 0 & 0 \\ 0 & 0 & 0 & \Omega_{34} & 0 & 0 \\ 0 & 0 & 0 & 0 & \Omega_{45} & 0 \\ 0 & 0 & 0 & 0 & 0 & \Omega_0 \end{bmatrix} = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix} \tag{3-22}$$

The H matrix and b vector are obtained using Eqs (3-15) and (3-16).

$$H = J^T \Omega J = \begin{bmatrix} 210 & -10 & 0 & 0 & -100 \\ -10 & 20 & -10 & 0 & 0 \\ 0 & -10 & 20 & -10 & 0 \\ 0 & 0 & -10 & 20 & -10 \\ -100 & 0 & 0 & -10 & 110 \end{bmatrix} \tag{3-23}$$

$$b = J^T \Omega e = \begin{bmatrix} -30 \\ 0 \\ 0 \\ 0 \\ 30 \end{bmatrix} \tag{3-24}$$

With the matrix H and vector b is possible to find the solution to Eq. (3-14) using a linear solver, such as QR decomposition.

$$\delta_x = [0.0000, -0.0732, -0.1463, -0.2195, -0.2927] \tag{3-25}$$

Thus, the optimized trajectory after one iteration will be:

$$x_{optimized} = x_{predicted} + \delta_x = [0, 0.5268, 2.0537, 1.4805, 0.0073] \tag{3-26}$$

with a RMSE error of 2.84% is comparison with the ground truth.

It is noticeable that the value of Ω_5 is larger than the others. The information matrix encodes the uncertainty of each measurement. If a certain value is larger, it is more important to the optimization because the information about this constraint is more reliable. The front-end is responsible for the construction of the graph. If the front-end was not able to establish the correct information about the loop closure, the values of the elements of the information matrix would be more similar, which would lead, in this case, to a more slow optimization. For example, if the information matrix was given by:

$$\Omega = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix} \tag{3-27}$$

Then, the optimized trajectory would be:

$$x_{optimized} = [0, 0.54, 2.08, 1.52, 0.06] \tag{3-28}$$

with a RMSE error of 4.90% is comparison with the ground truth.

3.3 2D Pose-Graph Optimization

In the two-dimensional problem, the state of the robot is given by:

$$x_i = [t_i^T, \theta_i] \tag{3-29}$$

where t_i is a 2D vector, corresponding to the x and y coordinates of the position of the robot in a plane, and θ_i corresponds to the orientation of the robot at the node i. It is important to normalize the angle θ between π and $-\pi$ after every iteration.

Each measurement between the nodes i and j is given by z_{ij}:

$$z_{ij} = [t_{ij}^T, \theta_{ij}] \tag{3-30}$$

The information matrix of each measurement is a 3x3 matrix.

The rotations of the robot are expressed with 2x2 rotation matrices, as shown in Eq. (3-31).

$$R_i = \begin{pmatrix} cos(\theta_i) & -sin(\theta_i) \\ sin(\theta_i) & cos(\theta_i) \end{pmatrix} \tag{3-31}$$

The error function is expressed by Eq. (3-32):

$$e_{ij} = z_{ij}^{-1} \begin{pmatrix} R_i^T(t_j - t_i) \\ \theta_j - \theta_i \end{pmatrix} \tag{3-32}$$

The Jacobian matrix is composed by the derivate of the error function in terms of each pose. However, as the error function of each measurement only depends on the values of two nodes, the Jacobian has the folowing structure:

$$J_{ij} = \begin{pmatrix} 0 \dots 0 & \dfrac{\partial e_{ij}}{\partial x_i} & 0 \dots 0 & \dfrac{\partial e_{ij}}{\partial x_j} & 0 \dots 0 \end{pmatrix} \tag{3-33}$$

The Eq. (3-14) can be solved using different numerical methods. In MATLAB, the mldivide command ($\delta_x := H \backslash -b$) is a efficient tool to solve this system, as it uses optimized linear solvers, such as QR, LU or Cholesky, depending on the structure of the matrices.

3.3.1 2D Dataset Evaluation

The implementation is evaluated using several datasets available in the literature. There are two main formats to represent graph files: TORO and g^2o [99]. They differ with respect to the ordering of the elements of the information matrix. The datasets in the 2D evaluation are in the TORO format. All nodes are represented by an ID, x, y and θ values, which correspond to the initial odometry poses.

All edge lines have the format: "IDfrom IDto x y θ I11 I12 I22 I33 I13 I23". The first two numbers "IDfrom IDto" correspond respectively to the ID of observing and observed nodes i and j. The x and y values compose the translation vector between nodes, and θ correspond to the rotation angle between nodes. The numbers "I11 I12 I22 I33 I13 I23" are the 6 top triangular elements of the 3x3 information matrix correspondent to each measurement. As the information matrix is symmetric, it becomes:

$$\Omega_{ij} = \begin{bmatrix} I11 & I12 & I13 \\ I12 & I22 & I23 \\ I13 & I23 & I33 \end{bmatrix} \tag{3-34}$$

The first evaluation is with the Intel dataset, a benchmark dataset with real data acquired at the Intel Research Lab in Seattle, consisting of raw measurements from wheel odometry and laser range finder. Its graph contain 1228 poses 1505 constraints. Figure 3.2 shows the initial pose-graph corrupted by drift in odometry estimation and measurement errors. The blue dots are the poses of the robot, and the red lines are the measurement constraints, derived from scan matching.

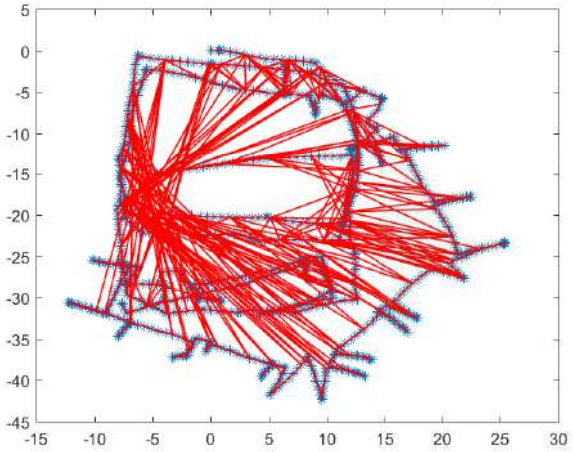
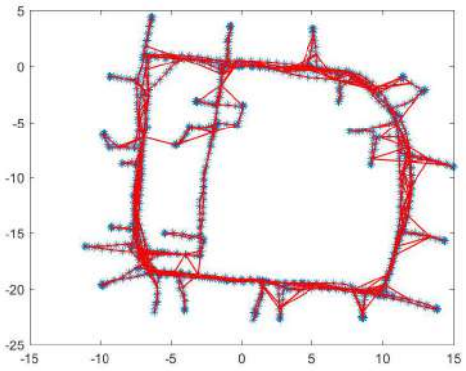


Figure 3.2: Intel - Initial corrupted pose-graph

In Fig. 3.3 is shown the optimized graph that corresponds to the optimized trajectory of the robot. In Fig. 3.4 is shown a comparative image of the same dataset optimized by a method called MOLE2D [100], developed by Carlone and Censi.



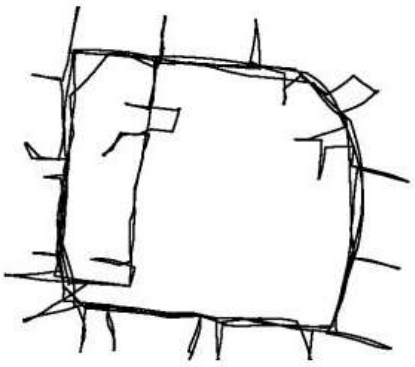


Figure 3.3: Intel Optimized pose-graph Figure 3.4: MOLE 2D Optimization

In Fig. 3.5 is shown the logarithmic global error per iteration. In only four iterations the system was able to optimize the entire graph, which shows the robustness of this implementation.

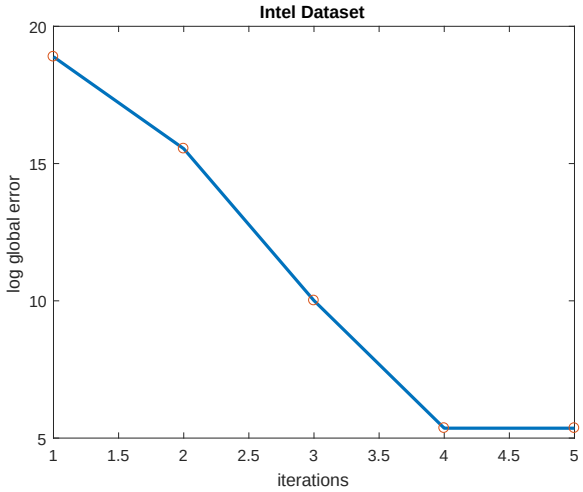


Figure 3.5: Intel dataset - Global error per iteration

Another evaluation was made using the Manhattan world dataset with 3500 poses and 5453 constraints, created by Olson et al. [43]. In Fig. 3.6 is shown the initial corrupted graph.

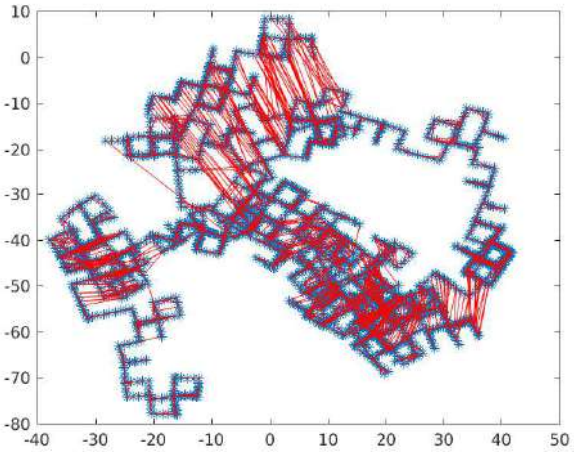


Figure 3.6: M3500 Initial corrupted pose-graph

In Fig. 3.7 is shown the optimized graph that corresponds to the optimized trajectory of the robot. In Fig. 3.8 is shown a comparative image of the same dataset optimized by Olson et al. The system achieved the final desired trajectory.

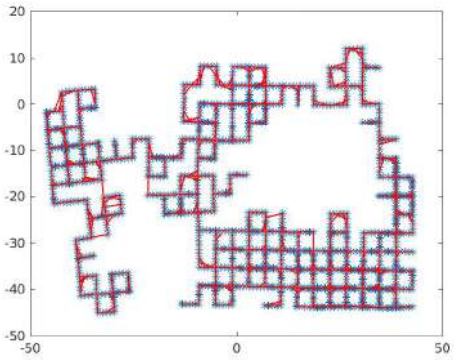


Figure 3.7: M3500 Optimized pose-graph

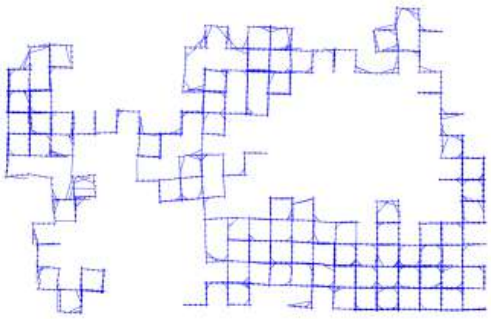


Figure 3.8: Olson's pose-graph

The global error is shown in Fig. 3.9. The initial global error is two orders of magnitude larger than the previous dataset. However, the optimization was made in three iterations.

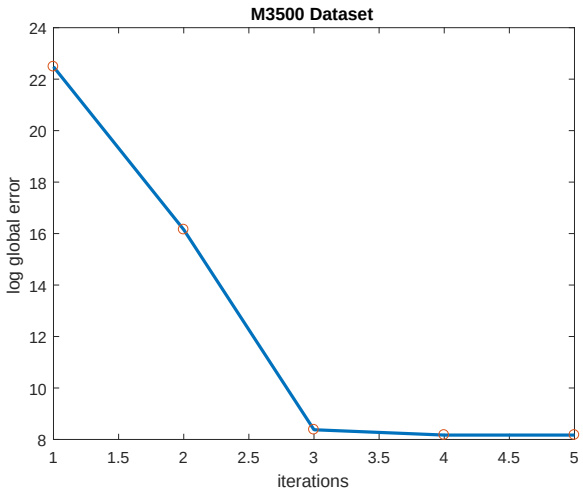


Figure 3.9: M3500 Global error per iteration

The objective of the last 2D test is to evaluate if the system is able to optimize a graph with oversized number of constraints. This dataset contains 10000 poses and 64311 constraints. The initial corrupted configuration is shown in Fig. 3.10. In Figs. (3.11) and (3.12) is shown a comparison between the result of the present work and the result obtained with LAGO, an algorithm developed by Carlone et al. [101].

The global error is shown in Fig. 3.13. Even with a larger number of constraints, the system is able to optimize the graph. The initial global error is one order of magnitude larger than the previous dataset. The optimization is made in six iterations.

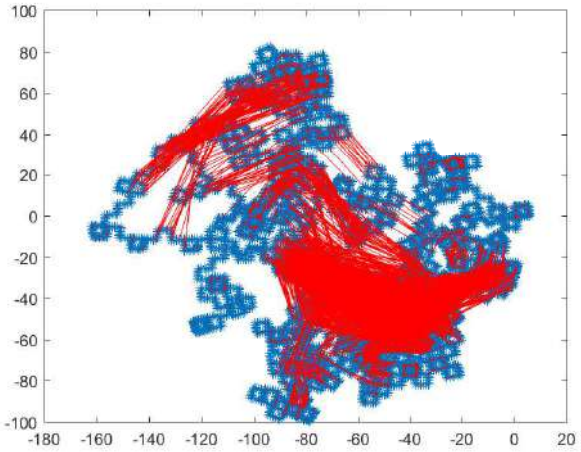


Figure 3.10: Initial corrupted pose-graph

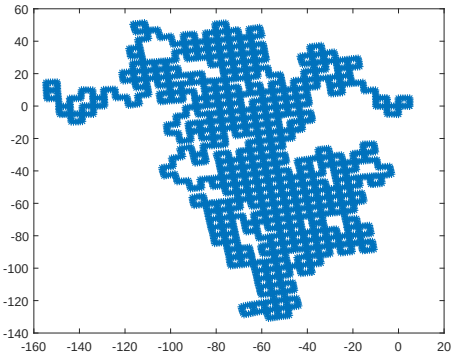


Figure 3.11: Optimized pose-graph

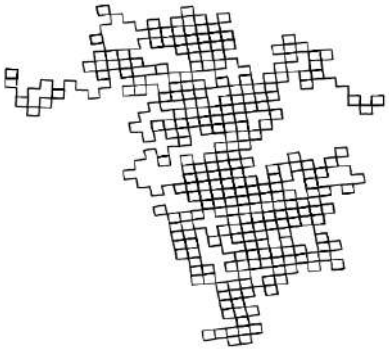


Figure 3.12: LAGO's pose-graph

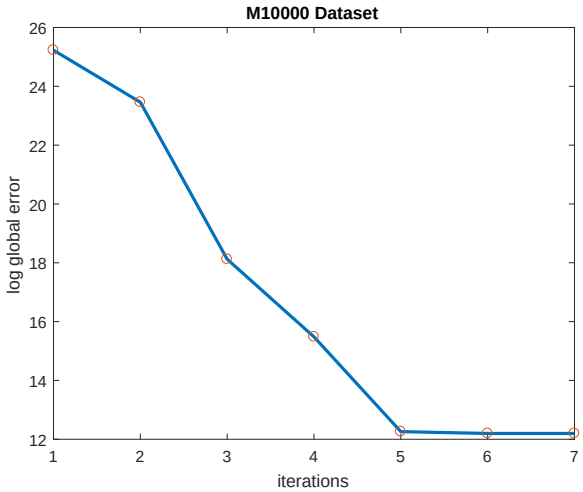


Figure 3.13: Global error per iteration

3.4 3D Pose-Graph Optimization

Opposed to 2D optimization, with a single normalized angle, the representation of orientation is problematic in 3D. In chapter 2, three parameterizations were presented: euler angles, rotation matrices and unit quaternions. However, all three suffer from considerable drawbacks in problems such as pose-graph optimization.

The use of Euler Angles is subject to singularities. As discussed in chapter 2, when two of the three rotation axes are aligned, a DOF is lost, which is called the gimbal lock problem. To overcome this problem, a solution would be the use of an over-parametrized representation, such as rotation matrices or unit quaternions.

The problem with the use of rotation matrices is the necessity to impose six non-linear constraints to ensure orthogonality and unit length of the columns. In other words, to ensure it remains in SO(3) [70].

Quaternions are more suitable for optimization problems than rotation matrices due to the number of constraints that need to be maintained at every iteration. A quaternion just need to maintain its unit lenght throughout the optimization process. However, the addition of this constraint degrades the performance of the algorithm [70]. The problems about the quaternion parameterization occur because rotations have three DOF and the quaternion can change in four directions.

Since estimation algorithms, in general, expect variables from euclidean vector spaces [49], the goal is to use a representation with three parameters, such as euler angles, but without singularities. However, there is no SO(3) parameterization with only three parameters that has no singularities [49].

To overcome these problems, the state is globally represented by a unit quaternion, but local perturbations around the current state have a minimal representation, ideally behaving as an euclidean space [49]. This parametrization is related to manifold theory and exponential maps.

3.4.1 Quaternion Exponential Map and Manifold Optimization

A manifold is a mathematical space that can be locally approximated by an euclidean space, but it is not on a global scale [102]. In other words, "every point of a manifold has a neighborhood that can be mapped bidirectionally to $\mathbb{R}^n$" [103].

The space of rotations and S^3, the unit quaternions, are manifolds and can be locally mapped to a euclidean space. Therefore, the parameterization problem discussed in the previous section can be dealt with using unit quaternions to represent the orientation of the state, and defining a operator $\boxplus$ that maps a local variation in the euclidean space to a variation on the manifold [16].

The analogous case for 2D is represented by Fig 3.14 from the work of Hertzberg [103]. The surface of the S^2 sphere is locally mapped into a plane in $\mathbb{R}^2$.

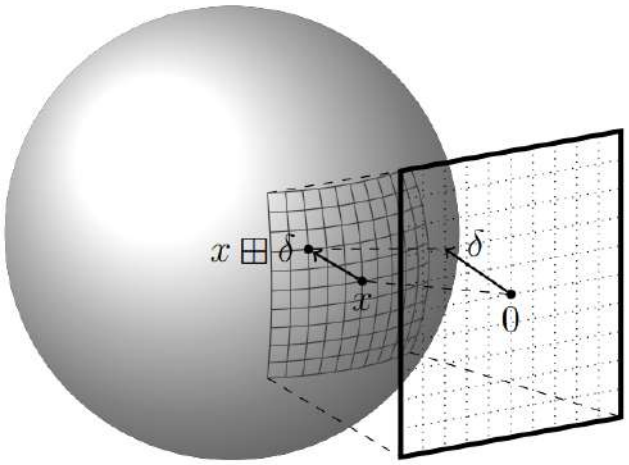


Figure 3.14: Mapping from $S2$ into $\mathbb{R}^2$

The $\boxplus$-method, developed by Hertberg [49], defines the mapping functions between the manifold and the euclidean spaces, which are called the exponential and logarithmic maps.

The operator $\boxplus$, stated in Eq. (3-35), represent the exponential map, which performs a rotation around axis δ with an angle $\|\delta\|$, accoding to Hertzberg et al. [103].

$$p \boxplus \delta = p \; exp\left(\frac{\delta}{2}\right) \tag{3-35}$$

The operator $\boxminus$, stated in Eq. (3-36), represent the logarithmic map, which computes the rotation from p to q. The global difference in manifold space is mapped to a local perturbation in euclidean space [16].

$$q \boxminus p = 2 \cdot log(p^{-1}q) \tag{3-36}$$

According to Ude [75] and Hertzberg [49], the exponential and logarithmic functions are given by the following equations, considering a quaternion with real part w and vector part u.

The exponential map function maps a vector v into a unit quaternion q:

$$exp: \mathbb{R}^3 \rightarrow \mathtt{S}^3 \tag{3-37}$$

$$exp(v) = q = \begin{cases} \left[sin(\|v\|)\frac{v}{\|v\|}, cos(\|v\|)\right] & \text{for} \quad \|v\| \neq 0 \\ [0,0,0,1] & \text{for} \quad v = 0 \end{cases} \tag{3-38}$$

The logarithmic map function maps a unit quaternion q into a vector v:

$$log: \mathtt{S}^3 \rightarrow \mathbb{R}^3 \tag{3-39}$$

$$log(q) = v = \begin{cases} 0 & \text{for} \quad u = 0 \\ \frac{atan(\|u\|/w)}{\|u\|}u & \text{for} \quad u \neq 0, w \neq 0 \\ \frac{\pi/2}{\|u\|}u & \text{for} \quad w = 0 \end{cases} \tag{3-40}$$

3.4.2 Implementation

This implementation is based on the work of Wagner et al. [47]. They managed to create a MATLAB framework for graph optimization with a manifold representation. However, their framework is more generic, extended to multi-sensor calibration problems, with an object-oriented implementation and an SO(3) exponential map. The present work is specifically designed to pose-graph optimization problems with a pose-quaternion representation. The implementation of Wagner et al. is based on the $\boxplus$-method, presented by Hertzberg [49].

The pose of the robot is represented by a translation vector and a full unit quaternion:

$$x_i = [t_i, q_i] \tag{3-41}$$

and the measurement functions are as well represented by a translation vector and a full unit quaternion:

$$z_{ij} = [t_{ij}, q_{ij}] \tag{3-42}$$

The expected measurement between two poses is:

$$\hat{z}_{ij} = (x_i^{-1} \cdot x_j) \tag{3-43}$$

Thus, the error function is defined by:

$$e_{ij} = \hat{z}_{ij} \boxminus z_{ij} \tag{3-44}$$

The rest of the implementation is analogous to the 2D case. For instance, Eq. (3-11) can be rewritten as Eq. (3-45), around $\delta_x = 0$.

$$e_{ij} = e_{ij}(\check{x} \boxplus \delta_x) \backsimeq e_{ij} + J_{ij}\delta_x \tag{3-45}$$

The Jacobian is given by Eq. 3-46.

$$J_{ij} = \frac{\partial e_{ij}(\check{x} \boxplus \delta_x)}{\partial \delta_x} \tag{3-46}$$

However, now the Jacobian matrix is computed numerically, according to Hertzberg [49]. A small perturbation is applied for each degree of freedom.

$$J_{ij} = \frac{e_{ij}(x \boxplus dv_j) - e_{ij}(x)}{d} \tag{3-47}$$

where e_{ij} is the error function, d is a small positive scalar and v_j is the unitary vector corresponding to the DOF.

Thus, the incremental addition to the initial guess is defined by the exponential map:

$$x^* = \check{x} \boxplus \delta_x \tag{3-48}$$

The operator $\boxplus$ first converts the rotational part of δ_x to a full quaternion and then apply the transformation to $\check{x}$ [16][104].

The formulation is described using pseudocode notation in algorithm 4.

Algorithm 4 3D Pose Graph Optimization

procedure READ GRAPH
  $x \leftarrow$ vertices
  $z \leftarrow$ edges
  $\Omega_{ij} \leftarrow$ inf matrices
endprocedure
while not converged **do**
  preallocate sparse J
  preallocate e
  given scalar d
  for all measurements m **do**
    compute error function e_{ij}
    $e_{ij} \leftarrow \Omega_{ij} e_{ij}$
    $z_{ij} \leftarrow$ measurement m
    procedure COMPUTE THE JACOBIAN
      for each dependant random variable rv **do**
        for $k = 1 : dof(e)$ **do**
          $x_{ie} \leftarrow x_i \boxplus de_k$
          $e_d \leftarrow (x_{ie}^{-1} x_j) \boxminus z_{ij}$
          $e_d \leftarrow \Omega_{ij} e_d$
          $J \mathrel{+}= \frac{e_d - e_{ij}}{d}$
        endfor
      endfor
    endprocedure
    $e \mathrel{+}= e_{ij}$
  endfor
  $H \leftarrow J^T J$
  $b \leftarrow J^T e$
  $\delta_x \leftarrow$ sparsesolve$(H\delta_x = -b)$
  $x = x \boxplus \delta_x$
endwhile

3.4.3 3D Dataset Evaluation

The 3D optimization is evaluated with datasets in the g^2o format. The nodes are listed in the format "ID x y z q_x q_y q_z q_w", which corresponds to the number of the pose and its respective 3D position and orientation in unit quaternion representation. However, in this case, the real part of the quaternion is the last one.

All edge lines have the format: "IDfrom IDto x y z q_x q_y q_z q_w I11 ... I66". The first two numbers "IDfrom IDto" correspond to the ID of observing and observed nodes i and j. The x, y and z compose the translation vector between nodes, and q_x, q_y, q_z, q_w is the unit quaternion rotation between the two nodes. The numbers I11 ... I66 are the 21 top triangular elements of the

6x6 information matrix, stated in Eq. (3-49). As the information matrix is symmetric, only 21 number are necessary to fill the entire matrix.

$$\Omega_{ij} = \begin{bmatrix} I11 & I12 & I13 & I14 & I15 & I16 \\ & I22 & I23 & I24 & I25 & I26 \\ & & I33 & I34 & I35 & I36 \\ & & & I44 & I45 & I46 \\ & \cdot\cdot & & & I55 & I56 \\ & & & & & I66 \end{bmatrix} \tag{3-49}$$

The first dataset represents the movement of a robot on a surface of a sphere, corresponding of 2500 poses and 4949 contraints. Fig. 3.15 shows the initial graph configuration, a sphere corrupted by noise. Figure. 3.16 shows that the system is able to correctly optimize the graph, displaying the optimized sphere. Figure 3.17 shows the global error per iteration of the evaluation in logarithmic scale. The system is able to optimize the graph in under 6 iterations.

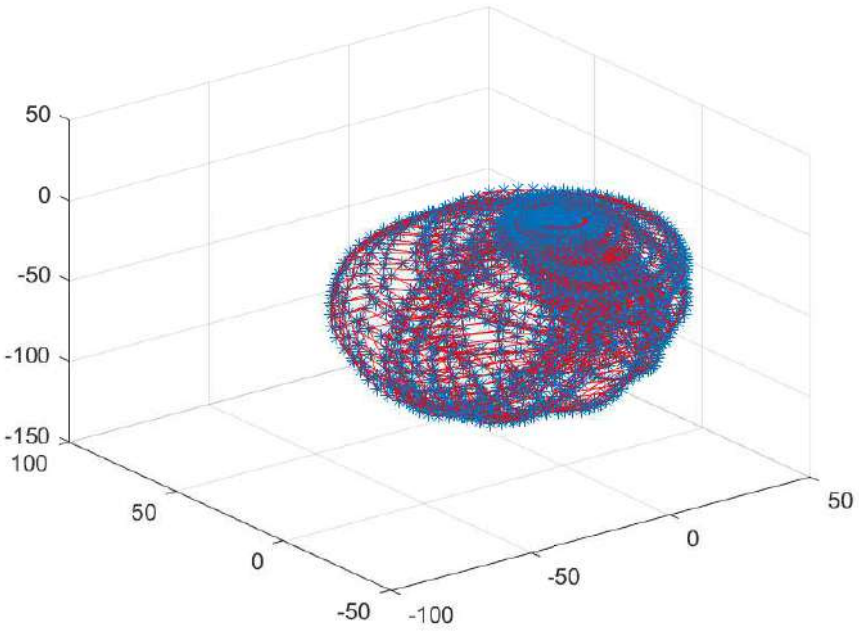


Figure 3.15: Initial sphere

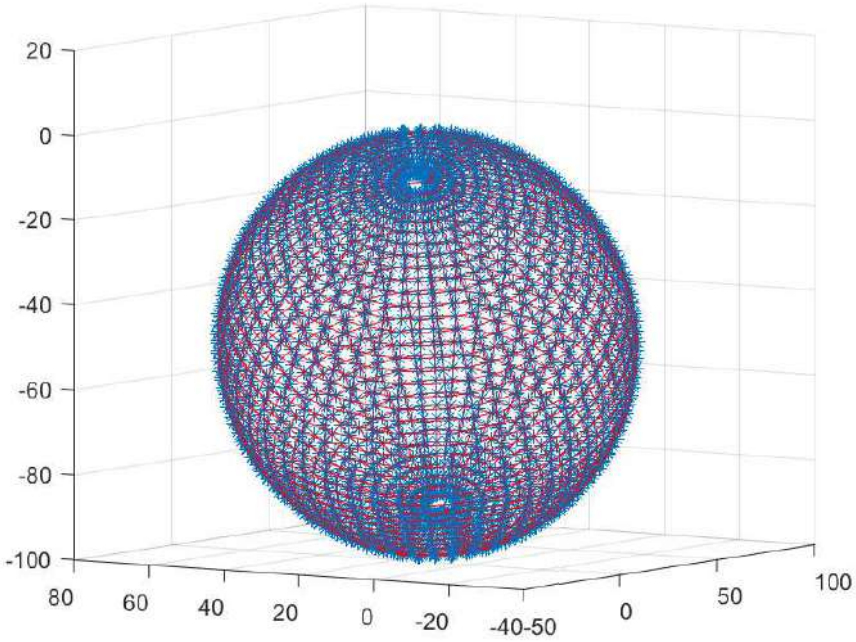


Figure 3.16: Optimized sphere

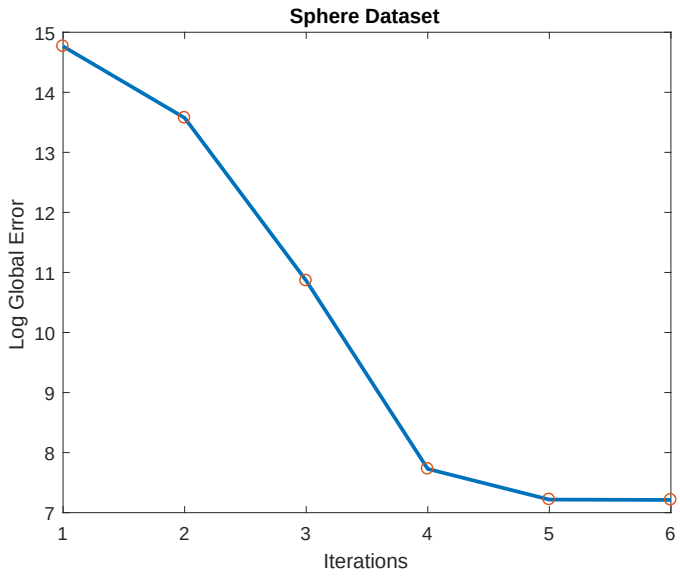


Figure 3.17: Global Error per Iteration - Sphere Dataset

The next evaluation uses real data aquired from an instrumented car at the Stanford parking garage. The parking garage dataset, provided by Carlone et al. [105], has 1661 poses and 6275 constraints. Figure 3.19 shows the place were data was aquired. In Fig. 3.18 is shown the initial corrupted graph, and Fig. 3.20 shows the optimized trajcetory of the robot.

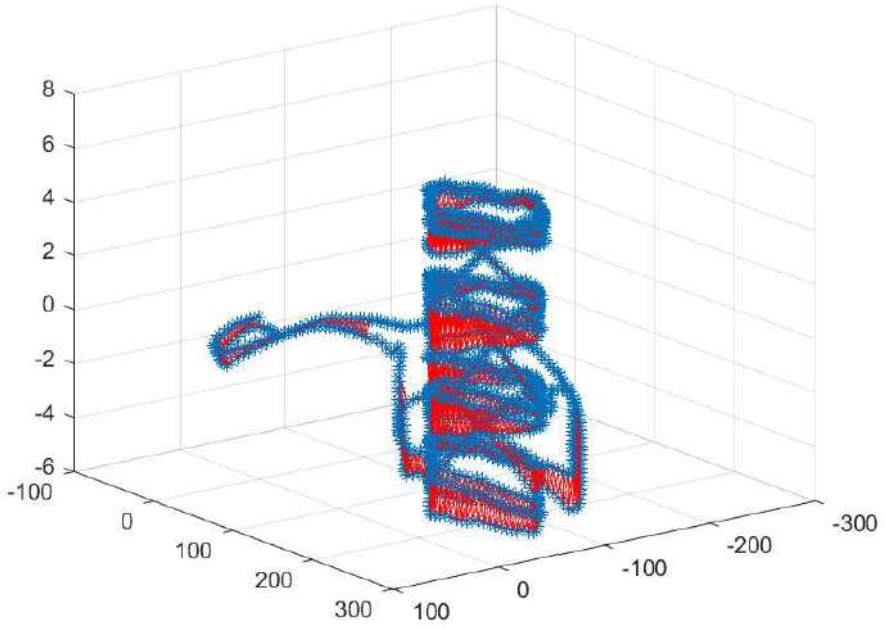


Figure 3.18: Garage - Initial Graph

Figure 3.19: Stanford garage

Figure 3.21 shows the global error per iteration of the evaluation in logarithmic scale. Despite having more constraints, it has a smaller initial global error, in comparison with the sphere dataset. The system is able to optimize the graph in under 4 iterations.

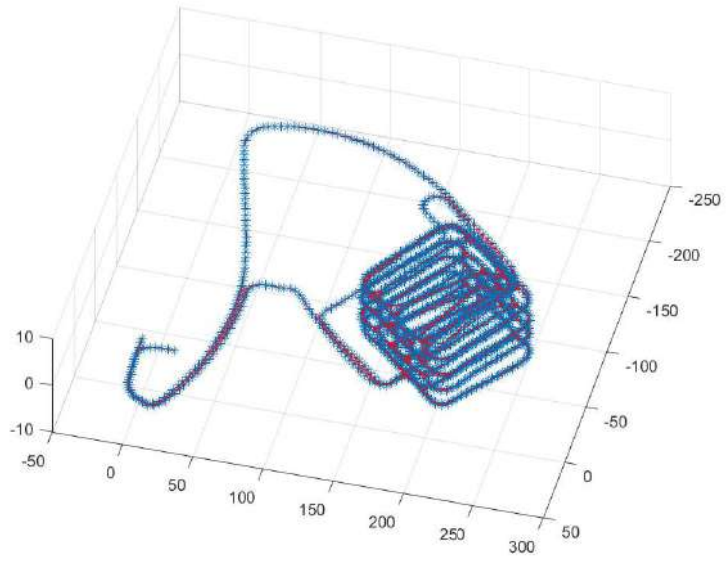


Figure 3.20: Garage - Optimized trajectory

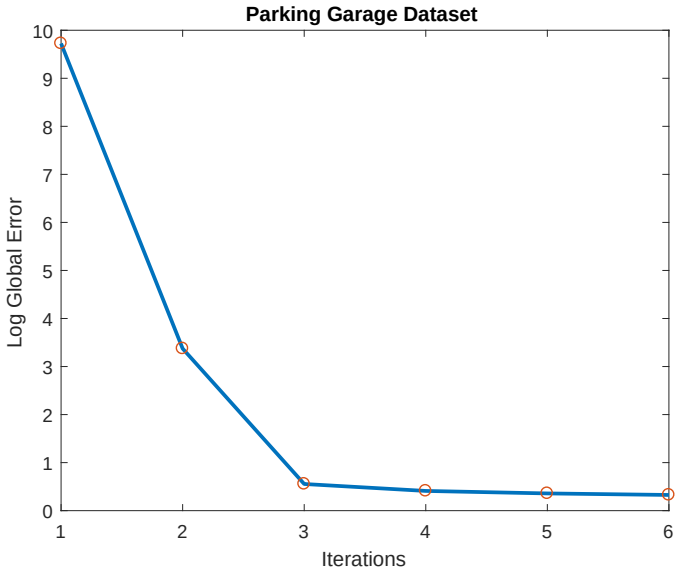


Figure 3.21: Global Error per Iteration - Parking Garage Dataset

4
SLAM Implementation

This chapter presents the implementation of a complete RGB-D SLAM system, with details of the hardware, software libraries, methodologies and algorithms used.

4.1
Hardware

4.1.1
Kinect v2

The Microsoft Kinect v2, shown in Fig. 4.1 [106], is composed by two sensors: a color camera and a depth sensor based on the time-of-flight principle. The scene is illuminated by a strobed infrared light that is reflected by obstacles. Then, the infrared (IR) camera register the time of flight for each pixel [8]. The work of Sell and O'Connor [107] provides a detailed explanation of the depth measurement methodology. Table 4.1 presents the specifications of the Microsoft Kinect v2, according to Fankhauser et al. [8].

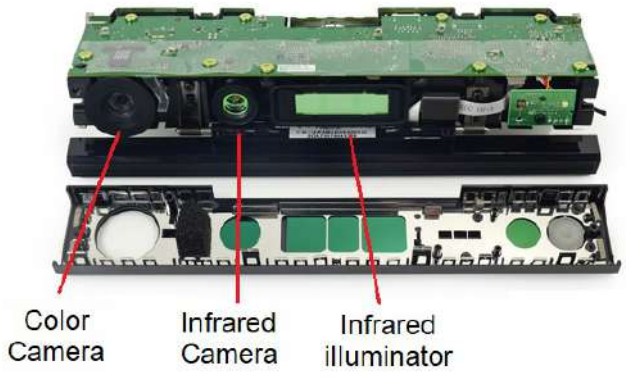


Figure 4.1: Microsoft Kinect v2

The kinect v2 has limitations that need to be observed. The minimum and maximum depth measurements have to be set in the driver. If the range is set out of the interval $0.5 - 4.5$, the measurement errors increase.

Table 4.1: Kinect v2 Specifications

Depth Camera	Resolution	512 x 424 px
	Field of View	70.6° x 60°
	Angular resolution	0.14°/px
	Operating range	0.5–4.5 m
Color Camera	Resolution	1920 x 1080 px
	Field of View	84.1° x 53.8°
	Frame rate	30 Hz
	Mass	970 g
	Connection	USB 3.0
	Voltage	12 V DC

The kinect driver used is the open source libfreenect2 [108], and the library iai_kinect is used as a link between libfreenect and ROS.

To allow a free locomotion of the robot, the kinect is powered with a lithium polymer (LiPo) battery of 12.6 V, shown in Fig 4.2. A Battery Eliminator Circuit (BEC), shown in Fig. 4.3 is a voltage regulator used to provide a constant supply of 12 V to the kinect.

Figure 4.2: LiPo Battery

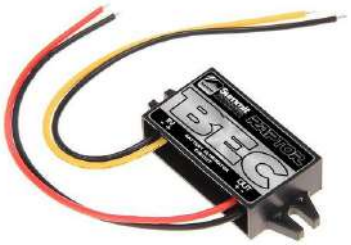


Figure 4.3: 12V BEC

4.1.2 Kinect Calibration

The kinect need to be calibrated in order to obtain the intrinsic and extrinsic parameters of the IR and RGB cameras. For this purpose, the calibration tool from iai_kinect2 [109] is used. The calibration is made using a chessboard calibration pattern. The follow steps are recommended to perform a correct calibration [109]:

- Print a calibration pattern and attach it to a flat surface, as shown in Fig. 4.4.
- Check with a caliper if the distance between squares in the pattern is correct.
- Assure that the image is clear and stable.
- Check if the pattern is detected, as shown in Fig. 4.5.
- Record images with different orientations and distances. It is recommended to record at least 100 images.

This process is performed for both color and IR cameras of the kinect v2. The iai_kinect2, then, register the depth images to the rgb images using the instrinsic and extrinsic parameters.

Figure 4.4: Kinect Calibration

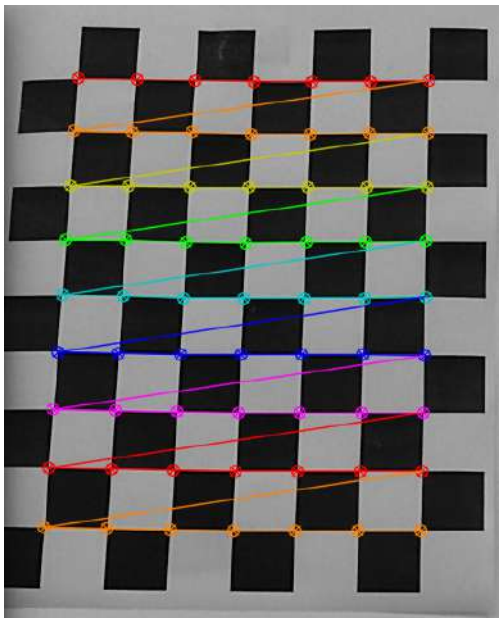


Figure 4.5: Image Pattern

4.1.3 iRobot Create

The iRobot Create [64], shown in Fig. 4.6 is a commercial differential drive mobile robot explicitly designed for robotics development and research. It has digital and analog input and output, approximately 90 minutes of battery autonomy, and can be controlled through an IR remote control, or using a driver available for ROS, called create_autonomy [110].

Aside from its two main wheels, the iRobot has two more to balance the robot with an added load: a castor wheel at the front of the robot and a simple roller wheel at the back, as shown in Fig. 4.7.

Figure 4.6: iRobot Create

Figure 4.7: iRobot wheels

4.1.4 Assembled Robot

Attached to the robot is a laptop, responsible to receive and process sensor data, store the map and trajectory, and send motion controls to the iRobot. The laptop has an Intel Core i7 6700 HQ processor with 2.60 GHz and 16 GB of RAM, running Ubuntu Linux 14.04 LTS and ROS Indigo. In Fig. 4.8 is shown the fully assembled robot, composed by iRobot Create, a Microsoft Kinect v2 and the laptop.

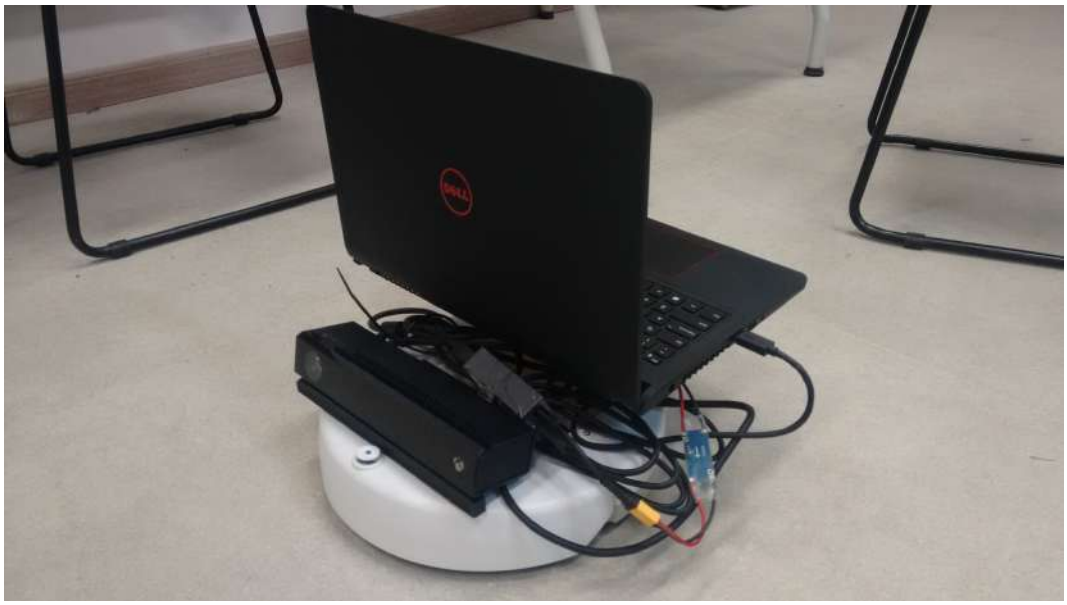


Figure 4.8: Robot fully assembled

4.2 System Overview

As stated in chapter 1, a SLAM system is composed of two main parts: front-end and back-end. The front-end comprises visual odometry and loop closure steps, which corresponds to the graph construction. After the graph is fully optimized, the global map is generated using the stored point clouds and the trajectory. This process is detailed in the following sections, and Fig. 4.9 illustrates the general pipeline. The final output of the system is the global map, the optimized trajectory in pose quaternion notation and the optimized graph in g^2o format. The system is written as a ROS package, composed by a main C++ ROS node and auxiliary header files.

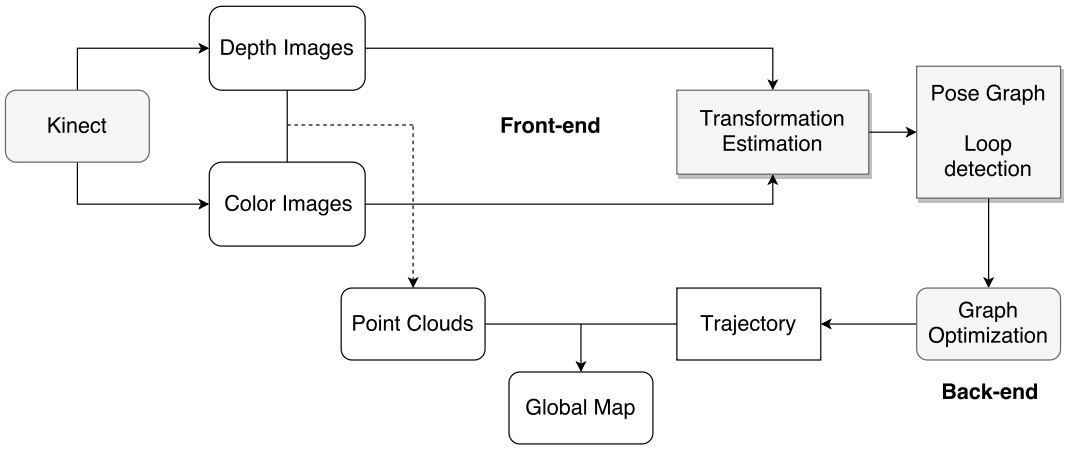


Figure 4.9: General Overview of the System

The following open-source libraries are used in this implementation:

- g^2o [38]: For graph construction and optimization
- FOVIS [56]: Visual odometry
- PCL [89]: For point cloud registration and visualization
- OpenCV [111]: Library for computer vision algorithms
- libfreenect2 [108]: The kinect v2 driver
- iai2_kinect [109]: Collection of tools for a ROS interface with kinect v2
- Eigen [112]: For operations with vectors, matrices and quaternions

4.3 Data Acquisition

The main node receives color and depth images provided by the kinect. All data is passed in form of ROS topics. The kinect driver grabs the images, the depth images are registered to the color images by iai_kinect2, and they are sent to the main node, as shown in Fig. 4.10.

Figure 4.10: Registered color and depth frames

4.3.1 Point Clouds from images

After the registration of the depth images, the point clouds are generated combining the color and depth data using the equations from the camera model, stated in Eqs. (4-1), (4-2) and (4-3). The library iai_kinect2 provides the point clouds for the main node.

$$Z = depth\_image[v, u] \tag{4-1}$$

$$X = (u - c_x)\frac{Z}{f_x} \tag{4-2}$$

$$Y = (v - c_y)\frac{Z}{f_y} \tag{4-3}$$

4.3.2 Downsampling

Due to the large amount of memory required by the point cloud representation, a voxel grid filter from PCL is applied to reduce the number of points used to create the global map. In fig. 4.11 is shown a point cloud with the full set of measured points and in Fig. 4.12 is shown the same point cloud after the filtering process.

Figure 4.11: Point Cloud

Figure 4.12: Downsampled Cloud

4.4 Visual Odometry

The FOVIS library [56] is used to obtain a camera motion estimate between frames. FOVIS is a robust visual odometry framework that uses sparse visual features and a quaternion absolute orientation estimation [113].

This work uses a ROS implementation of FOVIS. The FOVIS node subscribes to the corresponding topics: rgb camera information, depth camera information, depth image and color image. The camera information topics carry the camera matrices from calibration. The FOVIS node publishes the odometry topic, which is sent to the main node. At every iteration the transformation between the previous and the current FOVIS pose is computed.

4.5 Loop Closure

The incremental frame-to-frame alignment of FOVIS accumulates drift over time, due to sensor noise, which is the reason to use the Graph-SLAM

formulation. The loop closure detection is the most important step in Graph-SLAM, as it directly defines the errors of the trajectory.

Ideally, to detect a loop closure, it would simply require a comparison between the current frame and all past frames. However, it is computationally infeasible [17]. To overcome this problem, only a subset of frames is selected to be compared. They are called keyframes.

The first frame is selected as a keyframe and it is matched against the next frames. When the number of matched inliers starts to decrease and is below a chosen threshold, called keyframe inliers threshold, it means that the robot has made a significant movement and a new keyframe is chosen.

Every new keyframe is matched against the previous ones to search for a loop closure, if the poses of the previous keyframes are close enough to the current one, given a metric constraint.

If the number of matched inliers between the current keyframe and a past keyframe is above a threshold, called loop closure inliers threshold, then a loop is detected and a constraint is added to the graph. In Fig. 4.13 is shown a loop detection with the estimated trajectory and the corrected trajectory in green, after the loop closure and graph optimization.

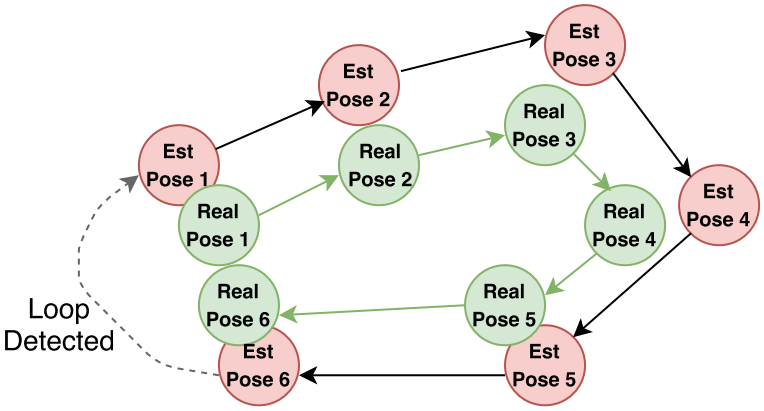


Figure 4.13: Loop Closure

The loop closure constraints are created using the ICP method with a RANSAC initial alignment, detailed in the next subsections. The information matrix is the identity matrix multiplied by the number of matched inliers between the two keyframes. This formulation is based on the work of Henry et al. [17], and on the implementation of Miguel Algaba [114].

4.5.1 Feature Detection

The OpenCV library has several feature detection implementations. The ORB features [63] are used in this work for their efficiency and computational performance. In Fig. 4.14 is shown an example of the feature detection implementation used. The corresponded feature descriptors are also obtained with the OpenCV implementation.

Figure 4.14: Feature Detection

4.5.2 Feature Matching

To match features from two images, the OpenCV implementation of the brute-force matcher is used. The brute-force matcher compares each descriptor of the first frame with all other features from the second frame using a distance metric. For each feature in the first frame, the closest one in the second frame is its match. In Fig. 4.15 it is shown two sequential color frames and its corresponding matched points.

The problem with feature matching is that false matches, also called outliers, can be detected, which can lead to a failure in the loop detection and, consequently, in graph optimization. Therefore, a outlier rejection technique must be applied.

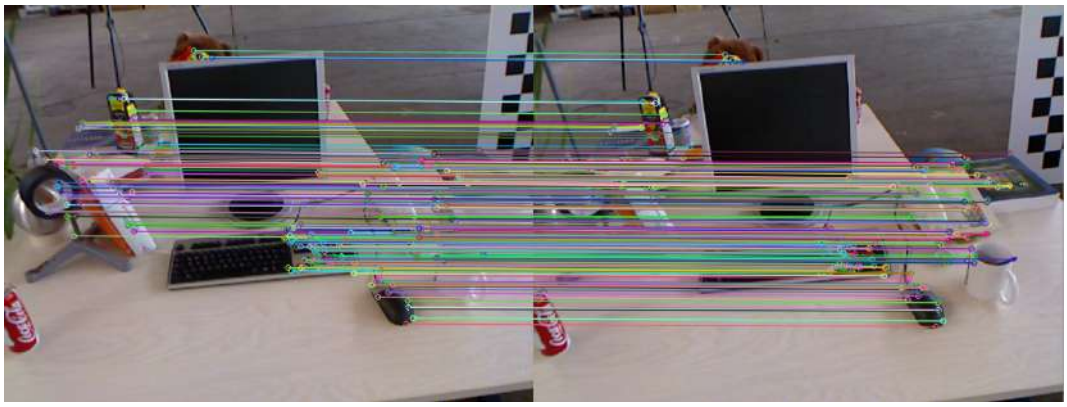


Figure 4.15: Feature Matching

4.5.3 Outlier Rejection

Considering the sets x and x' of the n matched points from two frames, the fundamental matrix is the 3x3 matrix that relates these two sets as stated in Eq. (4-4).

$$x_i' F x_i = 0,\ i = 1 \dots n \tag{4-4}$$

One method to reject outliers in feature matching is to find the fundamental matrix associated with the matched point of the two frames and apply Eq. 4-4 for each correspondence. If the result is below a threshold close to zero, then the correspondence is an inlier. Otherwise, the correspondence is rejected.

The fundamental matrix is found through the 8-point algorithm, detailed by Hartley and Zisserman [77]. To find the fundamental matrix, there must exist at least 8 point matches for the solution be unique [77]. In this work, the fundamental matrix is found with the OpenCV implementation, which uses the RANSAC method to improve robustness.

4.5.4 ICP

When a loop is detected, the transformation between the two frames need to be estimated in order to create the graph constraint. For this purpose, the PCL implementation of ICP is used. The transformation is estimated based on the SVD method, as described in chapter 2.

However, the ICP algorithm needs a initial guess for the transformation estimation, in order to avoid a local minimum. A initial alignment is made using RANSAC. In Tab. 4.2 is shown the chosen parameters for the ICP alignment.

Table 4.2: ICP Parameters

Parameter	Value
Max Iterations	150
Max Correspondance Distance	0.05
Transformation threshold	1e-4

4.5.5 Initial Alignment

With the matched points from two frames and their respective values, it is possible to apply the RANSAC algorithm to estimate the transformation between the frames. The PCL RANSAC initial alignment implementation is used in this step, and the obtained transformation is used as an initial guess in the ICP algorithm.

Table 4.3: Initial Alignment Parameters

Parameter	Value
Max Iterations	150
Inlier RANSAC threshold	0.01

4.5.6 Loop Closure Parameters

In Tab. 4.4 is shown the parameters used in the loop closure detection. The number of matched features is filtered using the main outlier threshold in the main code, to select the keyframes, and with the loop outlier theshold to detect a loop closure.

Table 4.4: SLAM Parameters

Parameter	Value
Number of features	2200
Keyframe Inliers Threshold	200
Loop Closure Inliers Threshold	100
Loop Outlier Rejection Threshold	0.1
Main Outlier Rejection Threshold	3.0

4.6 Graph Optimization

The g^2o framework is used to construct and optimize the graph. The graph can be optimized in real time, after every loop closure, or at the end of the procedure, before the map construction. The poses and constraints are represented in the g^2o format. The information matrix for every measurement is the identity matrix multiplied by the number of inliers.

4.7 Map Construction

The global map is constructed through the alignment of all stored point clouds. If the system is only performing visual odometry, the map can be incrementally constructed aligning the current point cloud to the global map, given the estimated transformation between the previous and the current pose. In this implementation, the map is constructed at the end of the procedure, after the graph is completed optimized, to improve performance. At the end of the optimization, all point clouds are stored, each one with a respective pose.

The PCL has a function "transformPointCloud" that applies a given transformation to a point cloud. This function is used to align all keyframe point clouds with their respective optimized poses. Finally, each one is added to the global map at the same coordinate system. This formulation is described using pseudocode notation in algorithm 5.

Algorithm 5 Global Map Construction

```
initialize global map
keyframes ← get keyframes
poses ← get poses
for all keyframes do
    pointcloud_k ← get keyframe pointcloud
    aligned_cloud = transform(keyframe_k, pose)
    global map += aligned_cloud
endfor
```

4.8 Summary

The algorithm 6 details the whole procedure of graph construction and optimization using the previously explained techniques. The "ROS-OK" condition returns "false" if the node is shut down.

Algorithm 6 RGB-D SLAM

```
initialize node
while ROS-OK do
    save first frame
    detect features
    pc1 ← point cloud 1
    dpc1 ← downsample pc1
    P_1 ← first fovis pose
    graph vertex 1 ← P_1
    for every new frame i do
        P_i ← i^th fovis pose
        save current frame i
        detect features
        feature match(features i, features i-1)
        n_inliers ← outlier rejection
        if n_inliers < threshold then
            keyframe_j ← currentframe_i
            graph vertex j ← P_j
            T_{j-1,j} ← compute transformation between keyframes
            add edge
            detect loop between current and past keyframes
            if loop detected then
                add edge
            endif
        endif
    endfor
endwhile
optimize graph
save graph
save trajectory
create global map
```

5
Results

This chapter presents the results obtained from real time experiments and real world dataset evaluation, using the proposed methodology. First, point cloud maps are created in real time experiments with the kinect and the iRobot. Next, the system is numerically evaluated using benchmark sequences and corresponded groundtruth trajectories.

To numerically analyze the proposed methodology with experimental data, it is necessary to have the ground truth trajectory of the robot in the environment. However, this would require motion detector systems or other external measurement device. Therefore, the numerical evaluation is made only with benchmark datasets.

5.1
Experiments

This section presents the qualitative results obtained with the proposed methodology, using the assembled robot presented in chapter 4. All the experiments were conducted in the Robotics Laboratory from the Pontifical Catholic University of Rio de Janeiro. In Figs. 5.1 and 5.2 is shown the assembled robot performing SLAM.

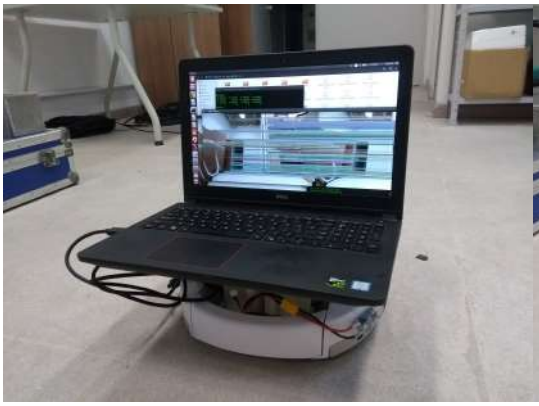


Figure 5.1: Robot performing SLAM

Figure 5.2: Robot performing SLAM

In the first two experiments, two separated parts of the room are mapped, one at a time, with slow movements and covering a small region. The maps are shown in Figs 5.3 and 5.4. The maps have a good quality with few misalignments.

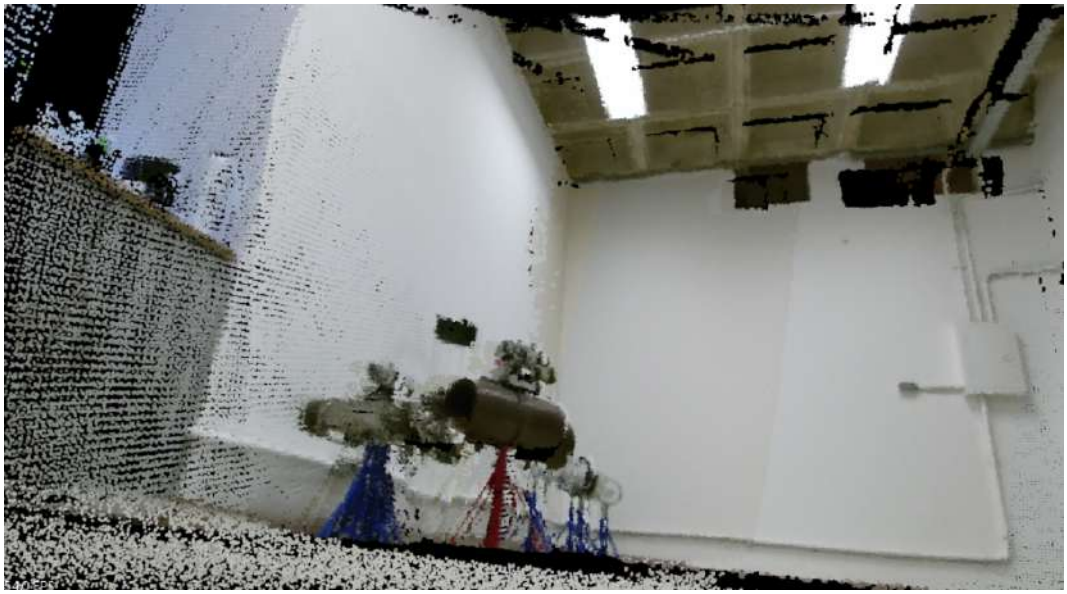


Figure 5.3: Experiment 1 - point cloud map

Figure 5.4: Experiment 2 - point cloud map

In the following experiment, the robot maps the entire room at once. The Figs. 5.5 and 5.6 are parts of the same map. The velocity of the robot is higher than the previous test.

Figure 5.5: Experiment 3 - point cloud map

Figure 5.6: Experiment 3 - point cloud map 2

This experiment clearly have an inferior quality than the first two experiments for two main reasons. First, the higher motion speed, which causes image blur and other effects. If the robot moves too fast or make a rough movement, it causes a considerable misalignment in the map. Second, a larger map implies a larger trajectory, which causes odometry drift. However, the graph optimization overcome major misalignment problems caused by odometry drift. Despite the misalignments, the map is still consistent due to the loop closure detection and optimization.

5.2 Dataset Evaluation

The numerical evaluation of the system is made using the RGB-D benchmark [59] from Technical University of Munich, which provides datasets of color and depth image sequences of a kinect sensor, under different conditions. All sequences have a corresponded ground-truth trajectory, obtained with a high precision motion capture system.

This evaluation employs the Absolute Trajectory Error (ATE) to compare the estimated trajectory with the provided ground-truth trajectory. The ATE compares absolute distances between both trajectories and evaluates the global consistency [59].

For each dataset is evaluated the mean, minimum, maximum and root mean square errors. Given the trajectory estimate with translational components $\hat{x} = [\hat{x}_1, \cdots, \hat{x}_n]$, and the ground truth trajectory with translational components $x = [x_1, \cdots, x_n]$, the root mean square error (RMSE) is given by Eq. 5-1.

$$RMSE = \left(\frac{1}{n} \sum_{i=1}^{n} \|\hat{x}_i - x_i\|^2 \right)^{1/2} \tag{5-1}$$

The ground truth trajectories have the format "timestamp tx ty tz qx qy qz qw", where timestamp is the time of each pose in unix epoch time, "tx, ty, tz" is the translation vector, and "qx qy qz qw" is a unit quaternion. The estimated trajectories and the ground truth trajectories are aligned using the timestamps of each pose [59].

5.2.1 Translation

The "freiburg1_xyz" is a sequence of 798 frames corresponding to a distance of $7.11m$ and a translational velocity of $0.24m/s$. It contains translatory motions along the axes x, y and z, with almost no rotational movements. In Fig. 5.7 is shown the comparison between the motion estimation and the ground-truth, and in Tab. 5.1 is shown the RMSE, mean, minimum and maximum errors of the estimated trajectory. In Fig. 5.8 is shown a color frame of the sequence and in Fig. 5.9 is shown the resulted point cloud map.

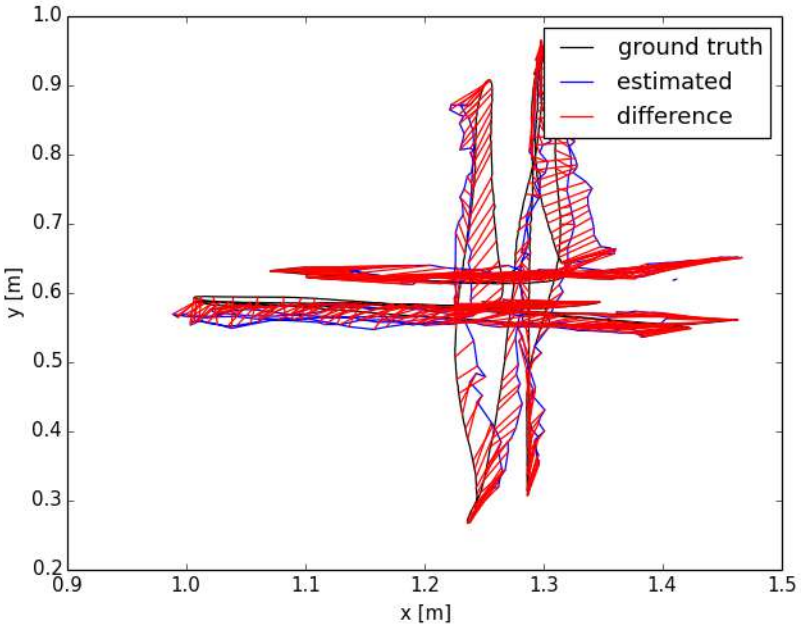


Figure 5.7: fr1-xyz - Trajectory

Figure 5.8: fr1-xyz - Color Frame

Table 5.1: ATE evaluation of the fr1 xyz dataset in meters

Error	**Value (m)**
RMSE	0.0569
Mean	0.0522
Min	0.0016
Max	0.1224

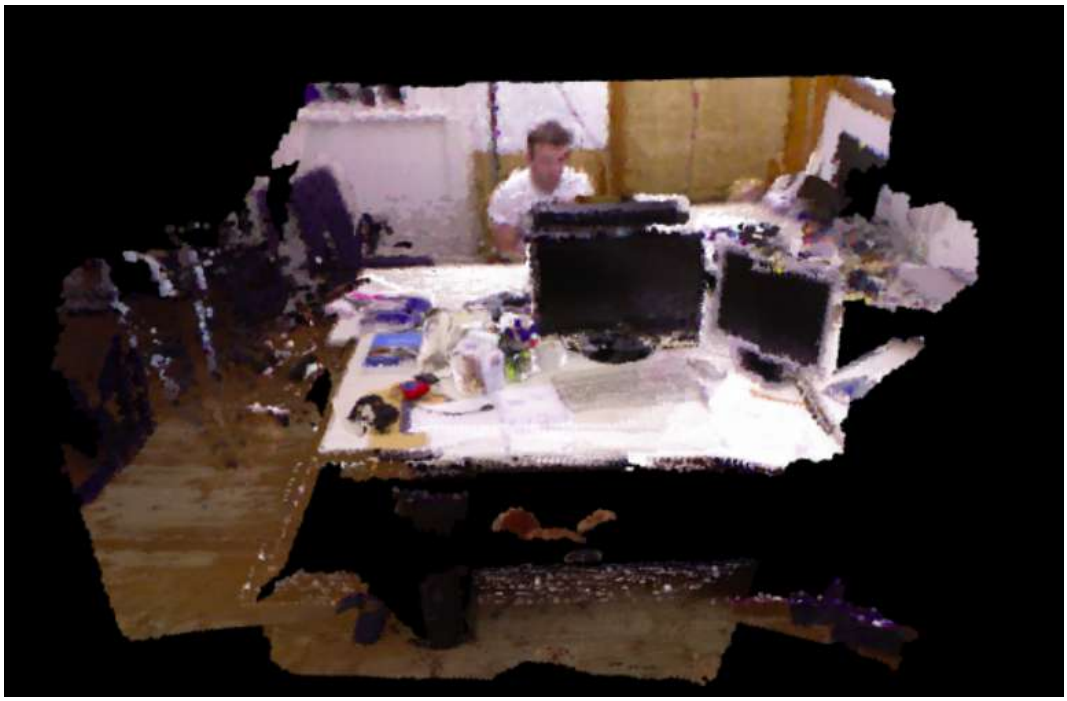


Figure 5.9: fr1-xyz - Point Cloud Map

5.2.2 Translation 2

The "freiburg2_xyz" is also a sequence of translatory motions along the axes x, y and z. However, it has a slower camera motion. The average translational velocity is $0.058m/s$ and an average angular velocity of $1.716deg/s$. The translational velocity is about 4 times slower than the previous sequence. Figure 5.10 shows the comparison between the estimated trajectory and the ground truth trajectory. In Tab. 5.2 is shown the RMSE, mean, minimum and maximum errors of the estimated trajectory. In Fig. 5.11 is shown the resulted Point Cloud map.

Table 5.2: ATE evaluation of the fr2 xyz dataset in meters

Error	**Value (m)**
RMSE	0.0174
Mean	0.0155
Min	0.0012
Max	0.0373

It is noticeable that the estimated trajectory is much more aligned with the groundtruth in comparison with the previous sequence, which shows that the motion velocity is an important factor in the SLAM process.

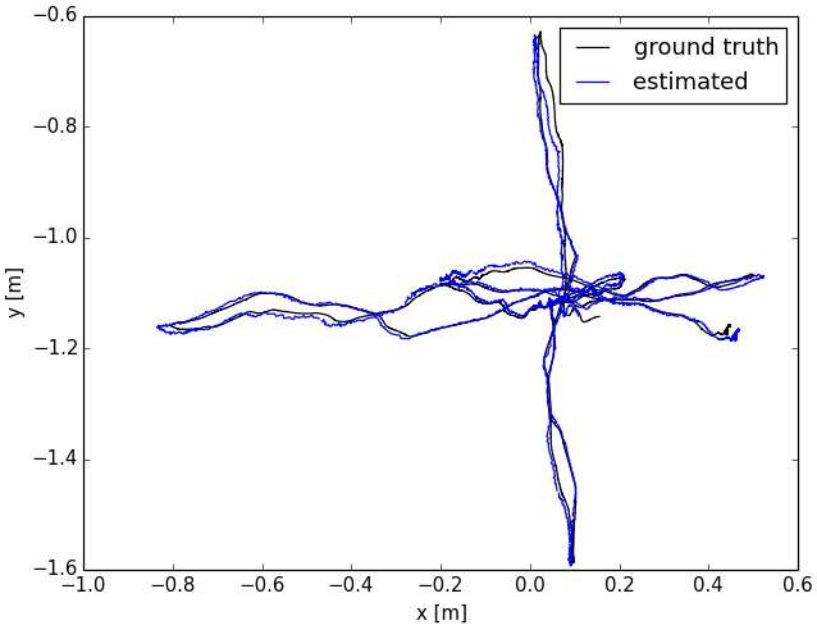


Figure 5.10: fr2-xyz trajectory

Figure 5.11: fr2-xyz - Point Cloud Map

5.2.3 Freiburg Room

The "freiburg1_room" is a larger sequence, with 15.989m of ground-truth trajectory length, 0.334m/s of average translational velocity and 29.882deg/s

of average angular velocity. The simulation is made with a frame rate of 3 Hz. The sequence is a movement through a whole office and is suited for evaluating loop closure detection. In Fig. 5.12 is shown the comparison between the ground truth trajectory and the estimated trajectory with only visual odometry. In Fig. 5.13 is shown the same comparison, but with graph optimization.

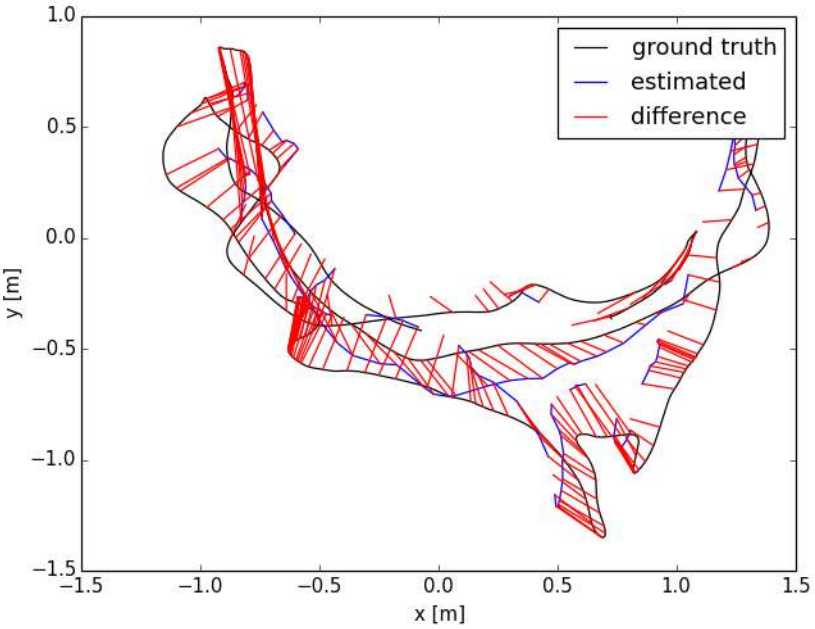


Figure 5.12: fr1-room - Visual Odometry

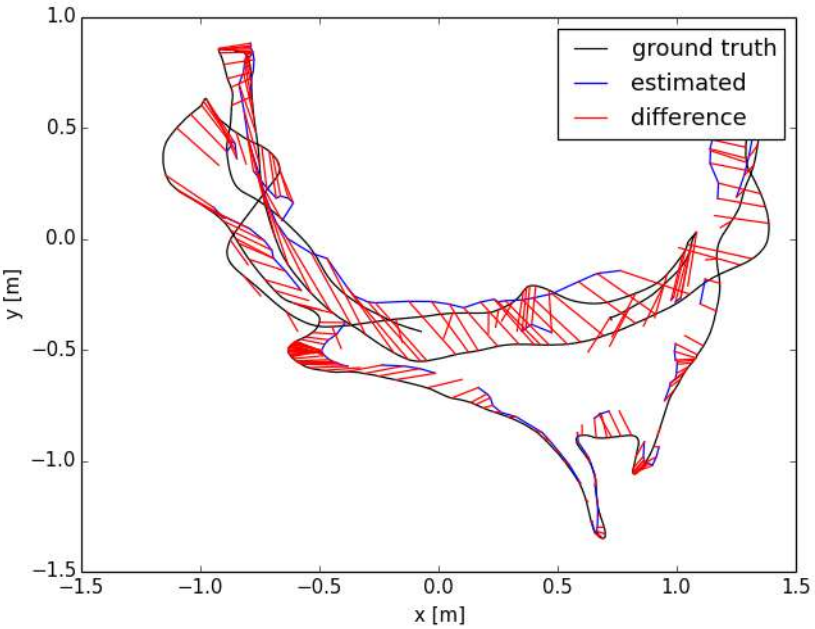


Figure 5.13: fr1-room - Optimized

Table 5.3 compares the results using only visual odometry with the ones with graph optimization, showning the RMSE, mean, minimum and maximum errors in meters. The overall errors are larger than the previous sequences due to faster motions and larger trajectory. However, the graph optimization provides a considerable improvement in the trajectory, and a cohesive map is produced.

Table 5.3: ATE evaluation of the freiburg1 room dataset in meters

Error	**Visual Odometry**	**Graph Optimization**
RMSE	0.2807	0.1987
Mean	0.2432	0.1722
Min	0.0256	0.0159
Max	0.6446	0.4072

In Figs. 5.14 and 5.15 is shown two views of the resulted global point cloud map of the sequence. It is noticeable that a person is standing in one corner of the map. As the person stood still during the mapping process, there was no problem. However, if the person had moved during mapping, it would violate the assumption that the world is static and the map would be corrupted.

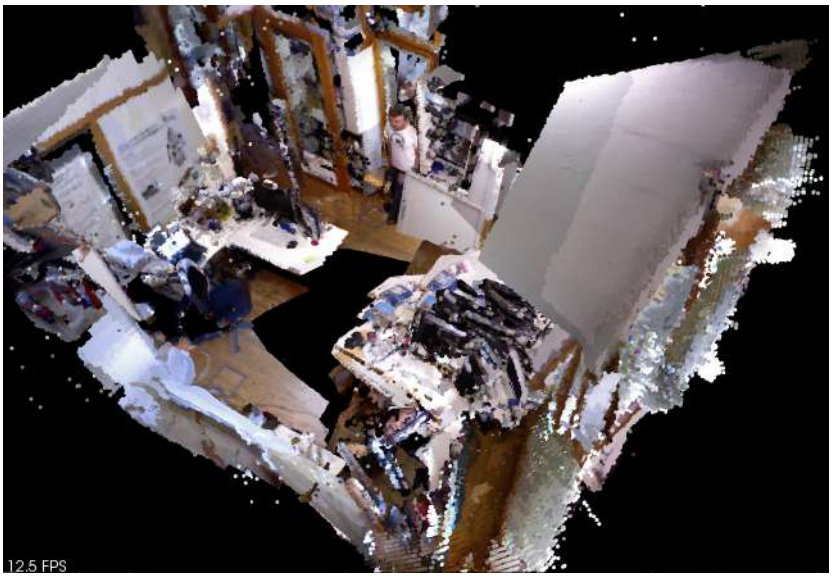


Figure 5.14: fr1-room - Point Cloud Map

Figure 5.15: fr1-room - Point Cloud Map 2

5.2.4 Long Office Household

The "freiburg3_long office" is an even larger sequence, with 21.455m of ground-truth trajectory length, and 0.249m/s of average translational velocity. This dataset has a loop closure near the end, which can also be used to evaluate the loop search system and graph optimization. This sequence is evaluated with a framerate of 3Hz. In Fig. 5.16 is shown the comparison between the ground truth trajectory and the estimated trajectory with only visual odometry. In Fig. 5.17 is shown the same comparison, but with graph optimization, stating that the optimization deacrease the error along the trajectory.

Table 5.4 compares the results using only visual odometry with the ones with graph optimization, showning the RMSE, mean, minimum and maximum errors in meters. The maximum error is reduced in 50% after the graph optimization.

Table 5.4: ATE evaluation of the fr3 long office household in meters

Error	**Visual Odometry**	**Graph Optimization**
RMSE	0.2717	0.1238
Mean	0.2329	0.1101
Min	0.0381	0.0156
Max	0.5375	0.2685

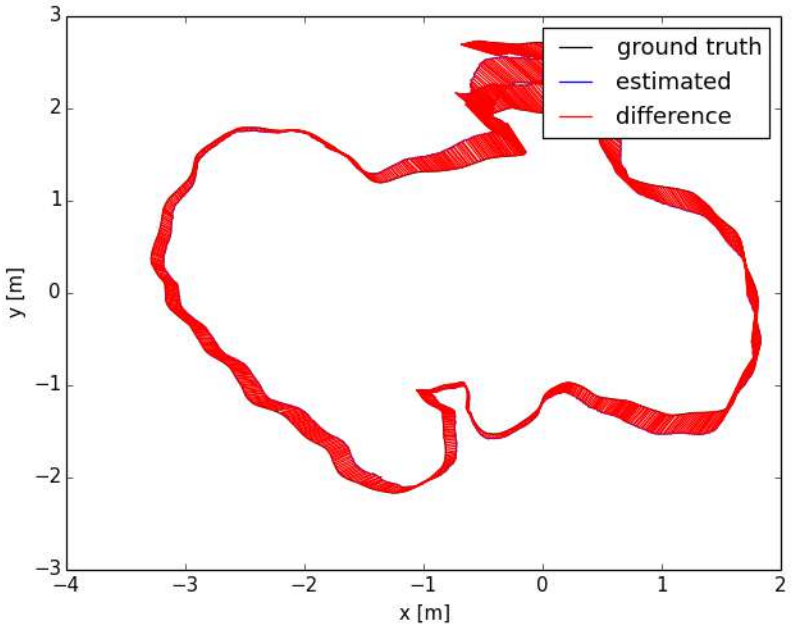


Figure 5.16: fr3-long-office - Visual Odometry

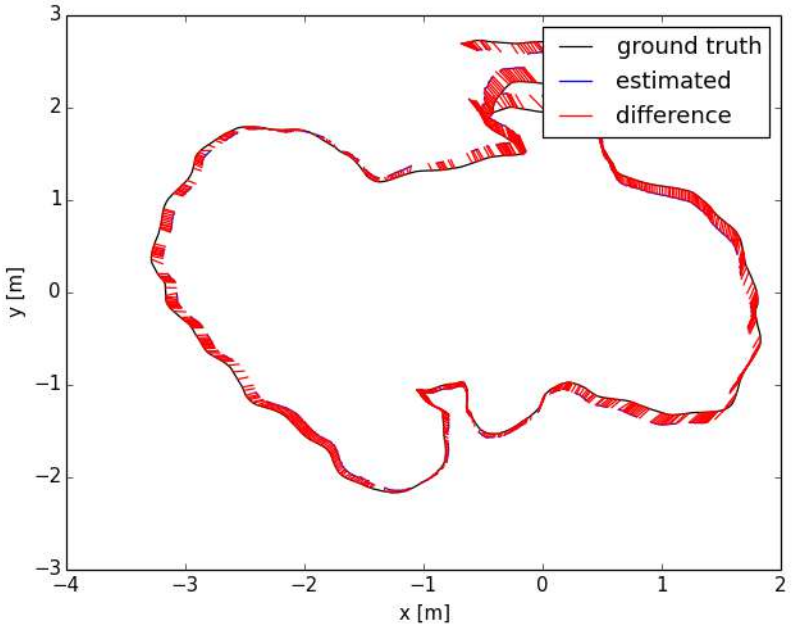


Figure 5.17: fr3-long-office - Optimized

In Fig. 5.18 is shown the initial point cloud of the dataset. In Figs. 5.19 and 5.20 is shown the progressive creation of the global map.

Figure 5.18: fr3-long-office: Initial Point Cloud

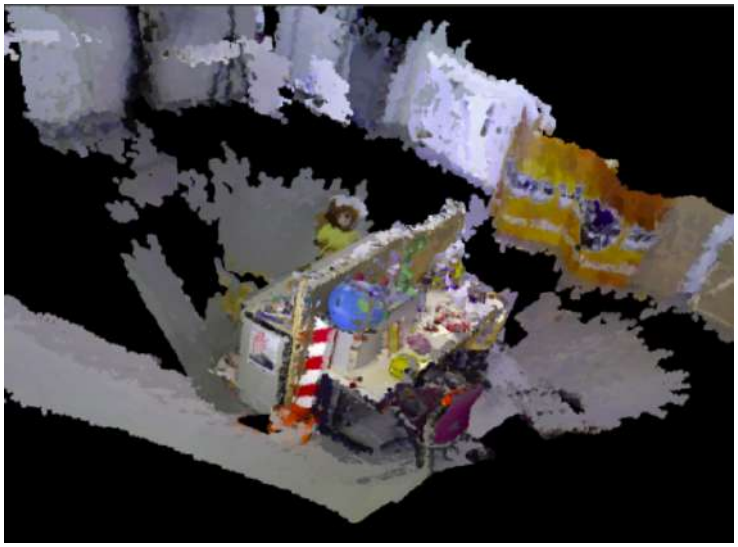


Figure 5.19: fr3-long-office: Point Cloud Map

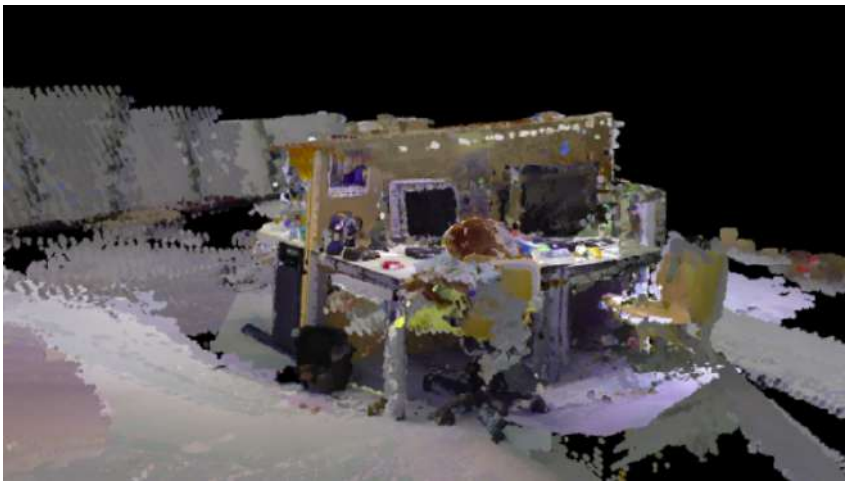


Figure 5.20: fr3-long-office: Point Cloud Map 2

This sequence is also evaluated with a framerate of 30 Hz, to test the performance of the system operating in real time in a long trajectory. In Fig. 5.21 is shown the trajectory with only visual odometry in real time. In Fig. 5.22 is shown the trajectory with graph optimization. 129 keyframes are selected in the process and a loop is detected between nodes 4 and 121, which allow the RMSE error to drop from $0.5144m$ to $0.3064m$, as shown in Tab. 5.5. Thus, the system is capable to detect loop closures in real time and perform graph optimization.

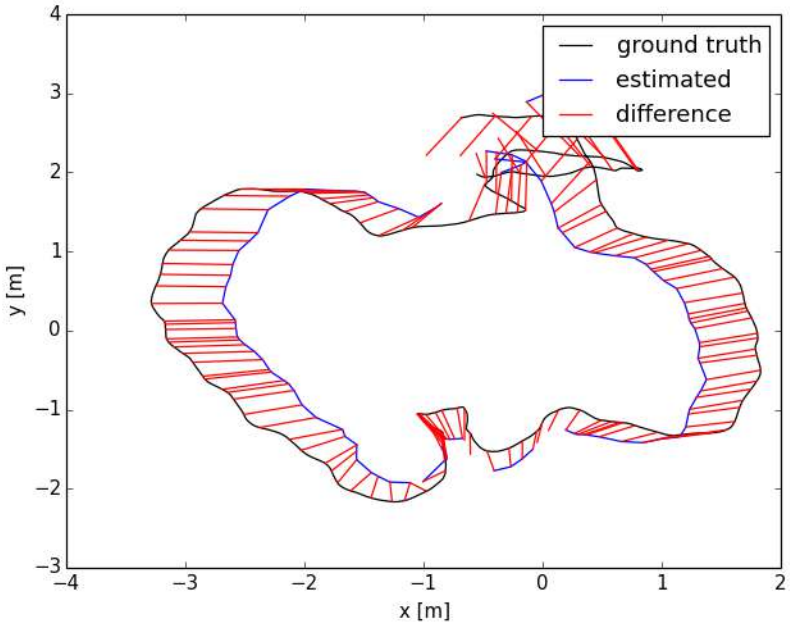


Figure 5.21: fr3-long-office 30Hz - Visual Odometry

Table 5.5: ATE evaluation of the freiburg3 long office household 30Hz in meters

Error	**Visual Odometry**	**Graph Optimization**
RMSE	0.5144	0.3064
Mean	0.4863	0.2842
Min	0.1799	0.0476
Max	1.1818	0.4654

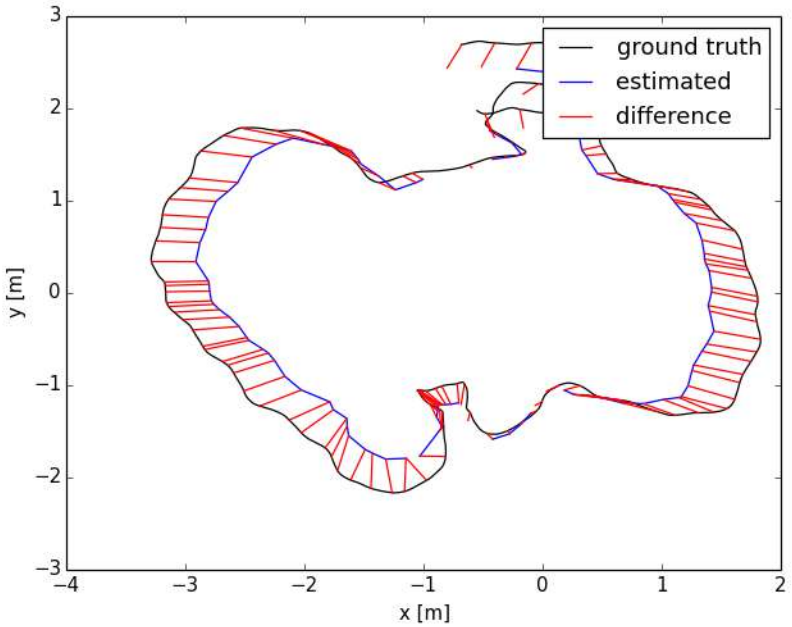


Figure 5.22: fr3-long-office 30Hz - Optimized

5.2.5 State-of-the-art Comparison

In Tables 5.6 and 5.7 are shown comparisons between the proposed implementation and state-of-the-art methods. The comparisons are made using the ATE RMSE metric and the datasets freiburg room, with 3Hz of frame rate, and freiburg long office household, with 30 Hz of frame rate.

For the freiburg room dataset, the comparison, shown in Tab. 5.6, is made with the first version of the rgbdslam method developed by Endres et al. [55] and FOVIS. This work outperformmed both methods.

Table 5.6: ATE RMSE Comparison between the present work and state-of-the-art implementations for the fr1 room dataset

Method	**ATE RMSE (m)**
This work	0.1987
RGB-D SLAM	0.2190
FOVIS	0.2807

For the freiburg long office household dataset, the comparison, shown in 5.7, is made with LSD-SLAM [53], a direct monocular SLAM algorithm, and ElasticFusion [61], the dense visual SLAM method mentioned in chapter 1.

Despite outperforming LSD-SLAM and FOVIS, this work has an inferior result compared to ElasticFusion for two main reasons. First, this work still need improvement in the loop closure implementation, which needs a more robust method for comparing images in the loop closure search, as the feature matching step is computationally expensive. Also, the ElasticFusion algorithm is a GPU-based method, which significantly improves speed.

Table 5.7: Comparison between the present work and state-of-the-art implementations for the fr3 office dataset

Method	**ATE RMSE (m)**
ElasticFusion	0.0170
This work	0.3064
LSD-SLAM	0.3853
FOVIS	0.5144

6
Conclusions

This work was subdivided into two main parts. First, it was proposed the development of an RGB-D SLAM system, capable of real-time performance, using a graph-based probabilistic framework to deal with the high uncertainty and non-linearity inherent to unstructured environments, sensor measurements, and robot motion. The second proposition of this work was to develop a MATLAB tool for pose-graph optimization.

The pose-graph optimization tool for MATLAB was presented in chapter 3. A methodology was proposed for bidimensional and three-dimensional problems. For the 2D case, the proposed methodology was evaluated with simulated and real world datasets, and compared with results from state-of-the-art implementations, achieving desired results. For the 3D problems, a methodology was presented to deal with the orientation representation problem discussed. The 3D implementation was also evaluated using datasets available in the literature. The system was able to optimize graphs with a large number of constraints and a considerable initial error in both cases.

The RGB-D SLAM implementation was presented in chapter 4. The system was evaluated in two situations. First, it was tested with real-time experiments using an iRobot and a kinect v2. The second evaluation was made using benchmark datasets for numerical analysis. The system was able to work in real time, and the results showed that the loop closure detection and graph optimization provided a considerable reduction of odometry drift, in both dataset evaluation and experiments.

The real time experiments showed that the system was able to detect loop closures in real time, which allowed the robot to perform SLAM through an entire room without serious misalignments in the map. The duration of the experiments showed that the system has scalability to handle real world problems.

The dataset evaluation was made using four sequences. The first two were composed by simple translational movements. The third and fourth sequences had a faster camera speed and a longer trajectory, but the system was able to achieve good results in localization and mapping.

However, the SLAM implementation faced some limitations. In both experiments and dataset evaluation, the system denoted sensitivity to rough movements and increased camera speed. Depending on the speed, the graph optimization was not able to correct the drift.

Also, despite its effectiveness, the loop closure system is not optimal in terms of computational performance. The need for feature matching with previous keyframes is expensive, and there are techniques of place recognition to improves this step that were not used in this work.

Furthermore, the assumption of static world prevents any moving object or person to be in the front of the robot whilst it is performing SLAM, which can be a problem for a factory or an office operation, for example.

Finally, the choice of point clouds for map representation is also not memory-efficient.

Despite these problems, the system was able to perform real-time SLAM, created consistent maps and achieve an acceptable global error in localization. This implementation has low cost, using only open source software and affordable hardware, and it has a vast applicability.

6.1 Future Works

For the pose-graph optimization tool, a C++ implementation is proposed to allow its use in real time SLAM implementations, such as the proposed RGB-D SLAM system.

For the RGB-D SLAM system, there are several open problems and challenges that can be explored in future implementations. The first suggested improvement is to use other map representations than point clouds, such as OctoMaps, for a more efficient memory usage, specially for long time operation, and to allow navigation tasks.

Another important improvement is to extend the implementation to the use in dynamic environments, which would improve considerably the applicability of the system.

Other interesting approach is to perform semantic mapping, in other words, to create maps with a level of information beyond metric, which can enhance the robustness of the system and open a path to novel applications.

Furthermore, the performance of the implementation can be increased using a more efficient loop detector, with a bag of words formulation for place recognition, for example.

Bibliography

[1] R. Siegwart and I. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, 2004.

[2] J. Leonard and H. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proc IEEE Int Workshop on Intelligent Robots and Systems*, Osaka, Japan, 1991.

[3] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[4] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13:99 – 110, 2006.

[5] C. Cadena, L. Carlone, H. Carrillo, Yasir Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32:1309–1332, 2016.

[6] Microsoft. Kinect v2. http://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx, 2015.

[7] R. El-laithy, J. Huang, and M. Yeh. Study on the use of microsoft kinect for robotics applications. In *Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium*, pages 1280–1288, 2012.

[8] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart. Kinect v2 for mobile robot navigation: Evaluation and modeling. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 388–394, 2015.

[9] J. Hernández-Aceituno, R. Arnay, J. Toledo, and L. Acosta. Using kinect on an autonomous vehicle for outdoors obstacle detection. *IEEE Sensors Journal*, 16(10):3603–3610, 2016.

[10] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, may 2009.

[11] Robotnik. Mobile robot guardian. https://www.robotnik.eu/mobile-robots/guardian/, 2014.

[12] B. Gerkey. Ros running on iss. http://www.ros.org/news/2014/09/ros-running-on-iss.html, 2014.

[13] Clearpath Robotics. Warthog. https://www.clearpathrobotics.com/warthog-unmanned-ground-vehicle-robot/.

[14] Neobotix. Omnidirectional robot mpo-700. http://www.neobotix-robots.com/omnidirectional-robot-mpo-700.html.

[15] Rethink Robotics. Baxter. http://www.rethinkrobotics.com/baxter/.

[16] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2:31 – 43, 2010.

[17] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31:647–663, 2012.

[18] H. Durrant-Whyte, D. Rye, and E. Nebot. Localization of autonomous guided vehicles. In *Robotics Research: The Seventh International Symposium*, pages 613–625. Springer London, 1996.

[19] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainly. *The International Journal of Robotics Research*, 5:56–68, 1986.

[20] P. Moutarlier and R. Chatila. An experimental system for incremental environment modeling by an autonomous mobile robot. In *1st International Symposium on Experimental Robotics*, 1989.

[21] J. Leonard and H. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics*, 7:376 – 382, 1991.

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

[23] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam. In *2003 IEEE International Conference on Robotics and Automation*, volume 2, pages 1985–1991, 2003.

[24] J. Leonard and H. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In *Robotics Research: The Ninth International Symposium*. Springer Verlag, 2000.

[25] J. Guivant and E. Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Transactions on Robotics and Automation*, 17:242–257, 2001.

[26] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Eighteenth National Conference on Artificial Intelligence*, pages 593–598, 2002.

[27] J. Handschin and D. Mayne. Monte carlo techniques to estimate the conditional expectation in multi-stage non-linear filtering. *International Journal of Control*, 9(5):547–559, 1969.

[28] K. Murphy. Bayesian map learning in dynamic environments. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pages 1015–1021, Cambridge, MA, USA, 1999. MIT Press.

[29] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proceedings 1999 IEEE International Conference on Robotics and Automation*, volume 2, pages 1322–1328 vol.2, 1999.

[30] G. Grisetti, G. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi. Fast and accurate slam with rao–blackwellized particle filters. *Robotics and Autonomous Systems*, 55(1):30 – 38, 2007.

[31] A. Neto, P. Rosa, T. Oliveira, and P. Pellanda. Efficiency of the visual fastslam technique with a common feature map for two vehicles in the integrated exploration of an indoor environment. *Journal of Control, Automation and Electrical Systems*, 4(24):450–469, 2013.

[32] A. Eliazar and R. Parr. Dp-slam: fast, robust simultaneous localization and mapping without predetermined landmarks. In *IJCAI'03 Proceedings of the 18th international joint conference on Artificial intelligence*, 2003.

[33] S. Canchumuni and M. Meggiolaro. Probabilistic simultaneous localization and mapping of mobile robots in indoor environments with a laser range finder. In *22nd International Congress of Mechanical Engineering (COBEM)*, Ribeirão Preto, São Paulo, Brazil, 2013.

[34] S. J. Julier and J. K. Uhlmann. A counter example to the theory of simultaneous localization and map building. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, volume 4, pages 4238–4243 vol.4, 2001.

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

[36] S. Thrun and M. Montemerlo. The graphslam algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research*, 25:403–430, 2005.

[37] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[38] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[39] J. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, 1999.

[40] A. Howard, M. J. Mataric, and G. Sukhatme. Relaxation on a mesh: a formalism for generalized localization. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium*, volume 2, pages 1055–1060 vol.2, 2001.

[41] T. Duckett, S. Marsland, and J. Shapiro. Fast, on-line learning of globally consistent maps. *Auton. Robots*, 12(3):287–300, May 2002.

[42] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localization and mapping. *IEEE Transactions on Robotics*, 21(2):196–207, 2005.

[43] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2262–2269, 2006.

[44] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *In Proc. of Robotics: Science and Systems (RSS)*, 2007.

[45] M. Kaess, A. Ranganathan, and F. Dellaert. isam: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008.

[46] G. Grisetti, C. Stachniss, and W. Burgard. Nonlinear constraint network optimization for efficient map learning. *Trans. Intell. Transport. Sys.*, 10(3):428–439, September 2009.

[47] R. Wagner, O. Birbach, and U. Frese. Rapid development of manifold-based graph optimization systems for multi-sensor calibration and slam. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3305–3312, 2011.

[48] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2d and 3d mapping. In *2010 IEEE International Conference on Robotics and Automation*, pages 273–278, May 2010.

[49] C. Hertzberg. A framework for sparse, non-linear least squares problems on manifolds, 2008.

[50] A. Davison and D. Murray. Mobile robot localisation using active vision. In *European Conference on Computer Vision*, pages 809–825. Springer, 1998.

[51] A. Davison, I. Reid, N. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[52] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, 2014.

[53] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II*, pages 834–849, 2014.

[54] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[55] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the rgb-d slam system. *2012 IEEE International Conference on Robotics and Automation*, pages 1691–1696, 2012.

[56] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Int. Symposium on Robotics Research (ISRR)*, Flagstaff, Arizona, USA, 2011.

[57] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald. Robust real-time visual odometry for dense rgb-d mapping. In *2013 IEEE International Conference on Robotics and Automation*, pages 5724–5731, 2013.

[58] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[59] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, 2012.

[60] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3-d mapping with an rgb-d camera. *IEEE Transactions on Robotics*, 30:177 – 187, 2014.

[61] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, and A. Davison. Elasticfusion: Dense slam without a pose graph. Robotics: Science and Systems, 2015.

[62] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[63] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[64] iRobot. irobot create. http://www.irobot.com/filelibrary/create/Create Manual_Final.pdf, 2006.

[65] D.P. Bertsekas and J.N. Tsitsiklis. *Introduction to Probability*. Athena Scientific books. Athena Scientific, 2002.

[66] K. Madsen, H. B. Nielsen, and O. Tingleff. Methods for non-linear least squares problems (2nd ed.), 2004.

[67] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[68] R. Murray, S. Sastry, and L. Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1994.

[69] J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors, 2006.

[70] F. Grassia. Practical parameterization of rotations using the exponential map. *J. Graph. Tools*, 3(3):29–48, 1998.

[71] J. Blanco. A tutorial on se(3) transformation parameterizations and on-manifold optimization, 2010.

[72] Y. Jia. Quaternions and rotation. 2017.

[73] B. Horn. Some notes on unit quaternions and rotation. 2001.

[74] J. Gallier. Notes on differential geometry and lie groups. 2012.

[75] A. Ude. Filtering in a unit quaternion space for model-based object tracking. *Robotics and Autonomous Systems*, 28(2-3):163–172, 1999.

[76] L. Vicci. Quaternions and rotations in 3-space: The algebra and its geometric interpretation. Technical report, Chapel Hill, NC, USA, 2001.

[77] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[78] V. Högman. Building a 3d map from rgb-d sensors. Master's thesis, Royal Institute of Technology (KTH), 2012.

[79] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.

[80] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[81] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[82] S. Thrun et al. Robotic mapping: a survey. *Exploring artificial intelligence in the new millennium*, 1:1–35, 2002.

[83] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Pittsburgh, PA, USA, 1989.

[84] H. Moravec. Sensor fusion in certainty grids for mobile robots. *AI magazine*, 9(2):61, 1988.

[85] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342. ACM Press/Addison-Wesley Publishing Co., 2000.

[86] Y. Roth-Tabak and R. Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, 1989.

[87] H. Moravec. Robot spatial perceptionby stereoscopic vision and 3d evidence grids. 1996.

[88] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Auton. Robots*, 34(3):189–206, apr 2013.

[89] R. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[90] P. Besl and N. McKay. Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–607. International Society for Optics and Photonics, 1992.

[91] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[92] T. Zinßer, J. Schmidt, and H. Niemann. A refined icp algorithm for robust 3-d correspondence estimation. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 2, pages II–695. IEEE, 2003.

[93] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001.

[94] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(4):376–380, April 1991.

[95] D. Eggert, A. Lorusso, and R. Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Machine vision and applications*, 9(5-6):272–290, 1997.

[96] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commum ACM*, 24:381 – 395, 1981.

[97] M. Quigley, B. Gerkey, and W. Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media, Inc., 1st edition, 2015.

[98] F. Endres. *Robot Perception for Indoor Navigation*. PhD thesis, University of Freiburg, 2015.

[99] L. Carlone. g2o versus toro: Format and cost functions.

[100] L. Carlone and A. Censi. From angular manifolds to the integer lattice: Guaranteed orientation estimation with application to pose graph optimization. *CoRR*, abs/1211.3063, 2012.

[101] L. Carlone, R. Aragues, J. Castellanos, and B. Bona. A fast and accurate approximation for planar pose graph optimization. *The International Journal of Robotics Research*, 33(7):965–987, 2014.

[102] J. Lee. *Introduction to Smooth manifolds*. 2001.

[103] C. Hertzberg, R. Wagner, U. Frese, and L. Schröder. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, 14(1):57–77, 2013.

[104] A. Ude. Nonlinear least squares optimisation of unit quaternion functions for pose estimation from corresponding features. In *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 1, pages 425–427. IEEE, 1998.

[105] L. Carlone, R. Tron, K. Daniilidis, and F. Dellaert. Initialization techniques for 3d slam: a survey on rotation estimation and its use in pose graph optimization. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4597–4604. IEEE, 2015.

[106] iFixit. "xbox one kinect teardown". https://www.ifixit.com/Teardown/Xbox+One+Kinect+Teardown/19725.

[107] J. Sell and P O'Connor. The xbox one system on a chip and kinect sensor. *IEEE Micro*, 34(2):44–53, 2014.

[108] J. Blake, F. Echtler, C. Kerl, and L. Xiang. libfreenect2: Open source drivers for the kinect for windows v2 device. https://github.com/OpenKinect/libfreenect2.

[109] T. Wiedemeyer. IAI Kinect2. `https://github.com/code-iai/iai_kinect2`, 2014 – 2015. Accessed June 12, 2015.

[110] J. Perron. create autonomy: a ros driver for irobot create. https://github.com/AutonomyLab/create_autonomy.

[111] G. Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[112] B. Jacob and G. Guennebaud. Eigen. http://eigen.tuxfamily.org/.

[113] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *JOSA A*, 4(4):629–642, 1987.

[114] M. Algaba. kinect6dslam. https://github.com/MiguelAlgaba/KinectSLAM6D, 2012.