Retail Store Visual Structure-from-Motion

Valter André Ribeiro dos Santos Piedade

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Dr. Pedro Daniel dos Santos Miraldo

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Dr. Pedro Daniel dos Santos Miraldo
Member of the Committee: Prof. Alexandre José Malheiro Bernardino

October, 2021

Declaration:

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the *Universidade de Lisboa*.

Declaração:

Declaro que o presente documento é um trabalho original da minha autoria e que cumpre todos os requisitos do Código de Conduta e Boas Práticas da Universidade de Lisboa.

Instituto Superior Técnico

Retail Store Visual Structure-from-Motion

Valter André Ribeiro dos Santos Piedade

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervision team:
Dr. Pedro Daniel dos Santos Miraldo
Msc. Gonçalo José Dias Pais
Msc. André Gonçalves Mateus

October, 2021

Abstract

3D mapping technologies have received increasing attention over the last few years. Among these technologies are Simultaneous Localization and Mapping (SLAM) and Structure-from-Motion (SfM) systems. One of the many applications of these systems is found in retail stores where robots are used to, *e.g.* , autonomously navigate through the store to stock the shelves or detect missing products. In this thesis a SfM system to work in a retail store was implemented from scratch in C++, using the structure of the well-known ORB-SLAM2, proposed in [35], as a baseline for the developed method. The system receives images from a moving stereo camera and uses them to estimate visual odometry and to build a map of the traveled region. The pipeline is modular so that various feature detectors and several methods of visual odometry estimation can be used. Two hybrid methods for visual odometry estimation are further proposed in this thesis. One of them aims at exploiting the typical geometric structure of retail stores. Results of the developed system were obtained using acquired datasets from a retail store and the KITTI dataset, and show some ablation studies for each module of the pipeline.

Keywords: Structure-from-Motion, visual odometry, mapping, keyframe, keypoint.

Resumo

Ao longo dos últimos anos, tecnologias de mapeamento 3D têm vindo a receber uma crescente atenção. Entre estas tecnologias estão sistemas de *Simultaneous Localization and Mapping* (SLAM) e de *Structure-from-Motion* (SfM). Uma das diversas utilizações deste tipo de sistemas encontra-se em supermercados onde robots são utilizados para, *p.e.*, navegar autonomamente pela loja para repôr produtos nas prateleiras ou verificar quando estes se encontram em falta. Nesta tese foi implementado um sistema de SfM de raiz em C++ que será utilizado em supermercados. A estrutura do conhecido sistema ORB-SLAM2, proposto em [35], foi utilizada como base para o método desenvolvido. O sistema recebe imagens de uma câmara *stereo* em movimento e utiliza-as para estimar o movimento da câmara e construir um mapa da região percorrida. A estrutura do sistema é modular de forma a permitir a utilização de vários tipos de detetores de pontos-chave e diversos tipos de métodos para estimação da odometria visual. São ainda propostos nesta tese dois métodos híbridos de estimação de odometria visual, tendo um deles como objetivo utilizar a estrutura geométrica típica dos supermercados. Foram obtidos resultados para o sistema desenvolvido utilizando conjuntos de dados adquiridos em supermercados e o conjunto de dados do KITTI, com os quais se realizaram estudos dos resultados obtidos para cada modulo do sistema.

Palavras-chave: *Structure-from-Motion*, odometria visual, mapeamento, *keyframe*, ponto-chave.

Acknowledgements

I would like to thank first of all my supervisor Dr. Pedro Miraldo, as well as his Ph.D. students André Mateus and Gonçalo Pais for all the help, availability, and knowledge they provided me during this 1-year project of which they were also part. A word of thanks also to my colleague Luis Lopes who joined the project at a later stage.

A word of appreciation to all the friends I made during the five years of university, with whom I accomplished many projects, had good experiences, and thanks to them I had a great time since the first day of this journey.

To my girlfriend, a special word of thanks for all the motivation she has given me and for always believing in my capabilities.

Last but not least, I would like to thank my family. It was through their efforts that I started this journey and it is also thanks to their help that I am completing it.

Contents

List of Figures

List of Tables

Acronyms

BA Bundle Adjustment. 18, 20, 39, 41, 43, 45

BoW Bag-of-words. 41, 42, 55

FPS Frames per second. 38, 47

GPS Global Positioning System. 3, 10

IMU Inertial Measurement Unit. 3, 10, 21, 59

LiDAR Light Detection and Ranging. xi, xiii, 3, 7, 8, 9, 16

RANSAC Random sample consensus. xi, 13, 14, 15, 17, 24, 26, 34, 49

SfM Structure-from-Motion. iii, v, xi, 3, 5, 6, 7, 18, 21, 29, 59

SLAM Simultaneous Localization and Mapping. iii, v, 3, 4, 5, 6, 7, 19, 21, 59

Chapter 1

Introduction

Simultaneous Localization and Mapping (SLAM) and Structure-from-Motion (SfM) have received increasing attention over the last years due to their applications in several industries like robotics, autonomous vehicles, mapping, security, and more.

SLAM systems have the goal of estimating the position of a robot and its trajectory, while simultaneously building a map of the traveled region. To achieve this, sensors such as cameras, LiDAR, sonar, GPS, IMU, among others are used. Although it can be used offline, the main focus of this type of system is its use in real-time. SfM systems, on the other hand, aim to reconstruct the 3D environment from a series of 2D images taken from different positions. Unlike SLAM systems, these systems work offline, so it is necessary to get the data and only then solve the problem. In some situations, SfM and SLAM systems work similarly, namely when both use cameras and when the different camera positions in the region are captured by a robot moving on a particular trajectory. In this case, the estimation of the robot's trajectory is necessary for the construction of the map. There are therefore concepts and methods common to both systems, such as odometry estimation, which is the estimation of the motion of the camera, and bundle adjustment (that is used to correct the position of points on the map).

Figure 1.1 exemplifies the operation of an SfM system. Throughout the movement made by the camera, images are captured. In these images, keypoints are detected and a search for these points is made throughout the various images to find common points between images taken from different positions. With this information, it is possible to estimate the camera movement between images, compute 3D points and also correct the map based on multiple views of the same point from different positions.

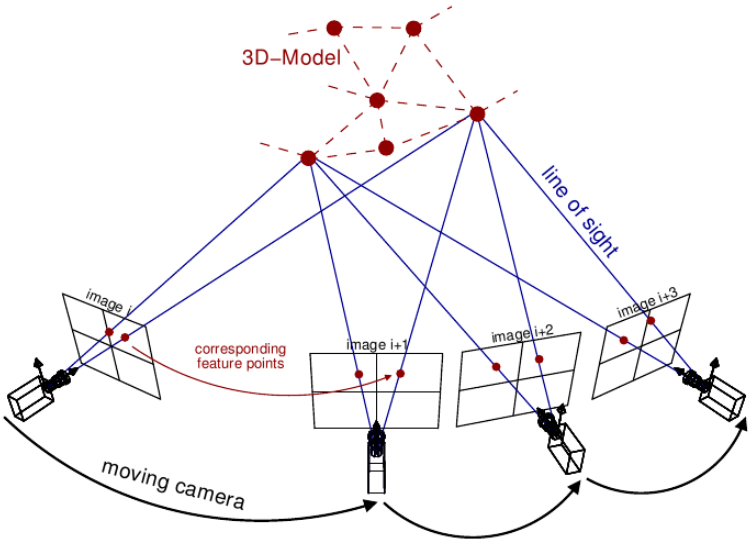


Figure 1.1: Operating scheme of a Structure-from-Motion system. Source: theia-sfm.org.

One particular use of these two types of systems can occur in retail store. This use has been increasingly adopted to have robots that can, *e.g.* , store, pick up, or detect products on the shelves. It is, therefore, crucial to have a correct map of the store so that the robot knows its location in the environment so that it can efficiently perform its function. A particular characteristic of these types of environments is their geometric structure. Retail stores have a lot of boxes and shelves, and are composed mostly of corridors, so it can be assumed that the environment can be characterized by having three orthogonal vanishing directions (Manhattan World assumption). These characteristics can thus be exploited for better environment-specific results.

1.1 Motivation

The work carried out in this dissertation is included in a research project that aims at developing and implementing a vision-based SLAM module for an autonomous mobile robot in a retail store environment. The research project working plan is divided into three stages. Stage I is the development and implementation of algorithms for the localization of an autonomous mobile robot working in the salesroom of a retail store. Stage II is the development of algorithms for map update and simultaneous location and mapping for the autonomous mobile robot described above, using the same set of sensors

as in Stage I. Lastly, Stage III are tests, adaptation, and integration of the system on the robot platform built as part of the project. The system must be designed to operate in the sales store of a retail store and the constructed map have to enable the robot to identify its location based on the images of the robot's operating space. The map format will allow its use in motion execution and planning systems that will be created in parallel.

After reviewing the literature and testing some state-of-the-art SLAM and SfM methods using a typical sequence of images (*e.g.* KITTI dataset [18]) and images from a retail store environment, it was decided to use as reference ORB-SLAM2, proposed in [35]. However, since the project is intended for commercial purposes and the scripts available from ORB-SLAM2 are under a license that does not allow such uses, the entire pipeline had to be implemented from scratch. We took this opportunity to make significant changes in the original pipeline, including making it more modular, i.e., it allow us to run proper ablation studies and is open to future research.

This dissertation fits in Stages I and II of the project's work plan and is the work that has been done so far in each of the stages. The work results in an SfM system that differs from the SLAM system by not having, mainly, a re-localization module nor running in real-time. The developed system is capable of, based on images captured by a stereo camera, estimating the movement of the camera along its trajectory, while creating a map of the region based on the 3D points computed along the path and adjusting their positions using bundle adjustment. The system was also developed to be modular in several aspects, such as the type of image features to extract and odometry estimation methods to use.

1.2 Objectives

The goals to be achieved with this thesis are:

- Implement a Structure-from-Motion pipeline in C++, based on a known pipeline;
- Integrate the use of several types of keypoint extractors and several conventional odometry estimation methods;
- Create an odometry estimation method that takes advantage of the structured retail store environment.

The desired result is to obtain a map of the environment in which the robot has traveled, together with its trajectory.

1.3 Thesis Outline

This report is divided into six chapters. Chapter 2 presents SfM systems and the conditions in which they resemble SLAM systems, thus dividing these systems into odometry estimation and mapping. For odometry estimation, featureless, feature-based, and deep learning methodologies are discussed. For mapping some methodologies are also discussed, namely bundle adjustment when loop closure occurs. Finally, the ORB-SLAM2 system is described, whose framework served as the basis for the development of the proposed SfM system. Chapter 3 presents the two proposed methods for odometry estimation that combine a rotation-only and a translation-only estimation method. Chapter 4 describes the developed SfM pipeline, highlighting each module and its functionalities. Chapter 5 reports the experimental results obtained throughout the evolution of the pipeline. Initially, results are presented only for the Tracking module, followed by the addition of Local Mapping and finally the addition of Loop Closure. Results of using various types of keypoint detectors and odometry estimation methods are also presented. Finally, in Chapter 6, a conclusion is made about the work accomplished and the results obtained, and whether they met the proposed objectives. Some future improvements to be made are also mentioned.

Chapter 2

Background

In recent years, 3D mapping technologies like Simultaneous Localization and Mapping (SLAM) or Structure-from-Motion (SfM) have received increased attention in several areas such as robotics, autonomous vehicles, and others. While SfM systems work offline and use only images from monocular, stereo, or RGB-D cameras to make a 3D reconstruction of the environment, SLAM systems operate in real-time and use sensors such as LiDAR or sonar in addition to cameras. The large number of applications of these technologies is partly due to the wide variety of sensors that can be used. This variety is important since each sensor has advantages and disadvantages depending on the type of use.

SfM systems require several observations of the same region to reconstruct the environment. For this, they can use either several static cameras pointing at the same area or moving cameras. When the camera is in motion, the functioning of this system resembles that of SLAM systems, since it is required to estimate the position of the camera along its path. Two steps can then be considered in the operation of these systems, odometry estimation, and mapping.

Odometry estimation, or ego-motion, consists of estimating the motion of the camera. This movement is described by a rotation and a translation typically between consecutive frames, which is used to convert the referential of each received image into a global referential common to the whole map. It is common to have errors in the estimation of odometry, which causes the estimated trajectory and map to deviate from the ground truth data. The greater the distance traveled, the greater the accumulated error. The mapping step consists of correcting the accumulated map error. This can be done by

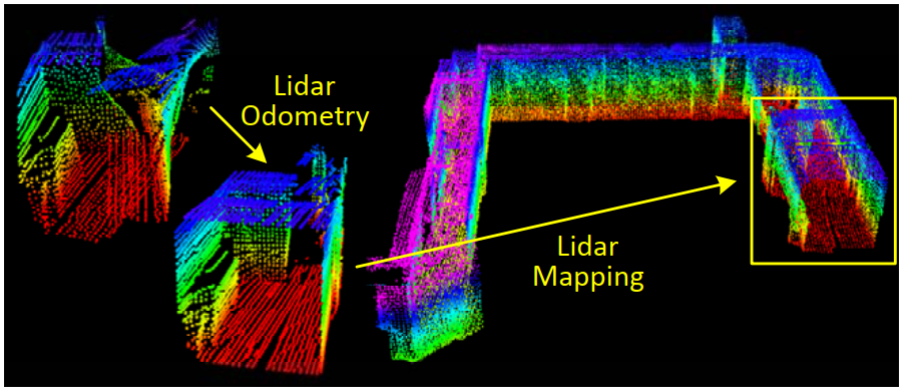


Figure 2.1: Example of odometry correction and mapping using a LiDAR sensor. Source: [58].

performing local optimizations based, *e.g.* , on points that are common to multiple consecutive frames. The most common process is loop closure detection, in which it is detected that the camera has returned to a known position, so it is possible to estimate the accumulated error and optimize the whole map based on that error and on common points between the images in which the loop was detected.

Figure 2.1 shows an example of the effect of the odometry estimation and mapping steps. On the left side of the figure are shown two point clouds from consecutive frames that are misaligned. By estimating the odometry between the two point clouds, it is possible to align them by transforming them into the same referential. By executing this process of estimating the movement along a region, a map of the region is created, as shown on the right side of the figure.

The following sections analyze the different types of sensors and approaches that can be used to solve the odometry estimation problem. These approaches are divided into featureless, feature-based, and deep learning registration methods. Some common techniques used for mapping and the well known ORB-SLAM2 system will also be analyzed.

2.1 Sensor Comparison

From the various types of cameras (stereo, RGB-D, monocular) to laser sensors, there are lots of scientific instruments available that can be used to perform 3D odometry

Sensors	Advantages	Disadvantages
Stereo Camera	Color information; Depth computed based on only one stereo pair; Images with low noise and typically easier to process;	Rely on the environment lighting conditions;
RGB-D Camera	Color information; Depth information; Images with low noise and typically easier to process;	Rely on the environment lighting conditions; Rely on having depth available;
Monocular Camera	Color information; Images with low noise and typically easier to process;	Rely on the environment lighting conditions; At least two images are needed to compute depth.
LiDAR Sensor	High range measurements; Aren't affected by the lighting conditions;	Data sparse and noisy; No color information;

Table 2.1: Advantages and disadvantages of stereo cameras, RGB-D cameras, monocular cameras, and LiDAR sensors.

estimation and mapping. The most common are stereo cameras, RGB-D cameras, and Light Detection and Ranging (LiDAR) sensors.

Stereo cameras as used in [35, 23, 38, 15] can provide images that are easier to treat than large 3D sparse point clouds. However, they heavily rely on the environmental lighting conditions, since their methods use visual features that became unavailable in low light conditions. Methods that only use LiDAR sensors, [58, 16, 8, 36], are also common since LiDAR sensors can perform range measurements at a high frequency and can reach great distances. They also ensure that measurement errors do not depend on the measured distance and aren't affected by the lighting conditions of the environment. As used in [28], since each LiDAR scan is taken from a single viewpoint, the 3D point cloud obtained can be converted into a 2D image without loss of information. This property of LiDAR sensors increases the amount of use given to LiDAR sensors data. Overall, the major downside is that the data received from these sensors is typically sparse and noisy and that we do not have color information. To balance the disadvantages of LiDAR sensors and cameras, methods like V-LOAM [59] use a combination of both a LiDAR and a monocular camera. Some methods using RGB-D cameras, [35, 25, 56, 24], are

also able to accomplish good results. However, they only utilize visual images from areas that have depth available, which often limits the number of images that these methods use. Table 2.1 shows the advantages and disadvantages of each of these sensors.

In addition to the previous sensors, sensors such as Global Positioning System (GPS) and Inertial Measurement Unit (IMU) can be used as a complement to improve mapping accuracy as shown in, for example, [36]. These sensors are capable of measuring the orientation, position, velocity, and acceleration of a moving object, which can be used directly or as an initial estimation when estimating sensor motion.

2.2 Visual Odometry

Visual odometry is the process of estimating the position and orientation of a camera along its trajectory, based on the sequence of acquired images. This is a crucial step to achieving a good result since the better the odometry estimation, the smaller the error accumulated along the path. Several approaches have been developed to solve the problem of visual odometry. These approaches either use 2D points, 3D points, or a combination of both.

In the following subsections, three different approaches to obtain a solution to the visual odometry problem will be seen. These approaches are divided into methods that use visual features (feature-based), methods that do not use visual features (featureless), and deep learning methods.

2.2.1 Featureless Registration

Concerning 3D registration methods without resorting to feature points, the Iterative Closest Point (ICP), [6], and its variants (some of them are presented in [12, 50, 51]) are the standard approach. These methods, however, have some disadvantages, the main one being the convergence in local minimums. To solve this problem, more ICP variations were developed, [39, 57], in an attempt to obtain a solution that guarantees global optimality. Another method that does not require feature points is the NDT, [7], which computes a normal distribution of the data from the received scans and estimates the best transformation based on an estimated score value from the computed distributions.

The standard Iterative Closest Point (ICP) algorithm estimates a rotation $\mathbf{R} \in SO(3)$

and a translation $\mathbf{t} \in \mathbb{R}^3$ that minimize

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{M} \|\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_{j^*}\|^2, \tag{2.1}$$

where $E(\mathbf{R}, \mathbf{t})$ corresponds to a l_2 error and $\mathbf{y}_{j^*}$ to the best match for $\mathbf{x}_i$. The best match corresponds to the point closest to $\mathbf{x}_i$ after the transformation, *i.e.*

$$\mathbf{j}^* = \arg\min \|\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_j\|^2. \tag{2.2}$$

The algorithm iteratively varies between estimating the transformation and searching for the nearest points. In the solution proposed in [6] the sets of points are selected in a point-to-point manner, while in [12], the sets of points are selected in a point-to-plane manner. The main problem with ICP is that it does not guarantee an optimal global solution, often generating solutions that are only local minimal. To try to reach the global maximum, it is necessary to provide a good initial estimate of the transformation, which is not always possible. Another important problem is that this algorithm is greatly affected by the presence of outliers.

Generalized-ICP (GICP), [50], combines the previous methods in a probabilistic framework. This method analyzes flat local surfaces in both received sets of data which allows using both sets to obtain correspondences, while in the previous methods, it was taken only one data set and tried to fit in the other. This way, the method can be considered as doing plane-to-plane matching. Using this approach, the simplicity and speed of the previous methods are maintained and the accuracy increases.

Normal Distributions Transform (NDT), [7], can be seen as a variant of the ICP applied to laser data. It uses data received from a 2D laser sensor and converts each received scan (that corresponds to a 2D plane) into cells of constant size. For each cell, a normal distribution is calculated which expresses the probability of evaluating a sample. This way, it is obtained a probability density of the points received, which is differentiable and piece-wise continuous. To perform the correspondence, it is used the 2D transformation $\mathbf{T}$ defined by

$$\mathbf{T} : \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}, \tag{2.3}$$

that describes the rotation ϕ and translation $\begin{bmatrix} t_x & t_y \end{bmatrix}^T$ between the two scans. To perform scan alignment first it is calculated an NDT of the initial scan. Then, using initial values

for **T**, the second scan is mapped to the coordinates of the first scan. A score is assigned to the result obtained with **T** by adding the NDT values of the initial and the mapped scans. Using Newton's algorithm, new parameters for **T** are estimated to improve the score obtained until a certain convergence criterion is reached.

Although the previous ICP variants increase the performance of the original method, they still do not guarantee that the global minimum will be obtained. To solve this problem, Globally Optimal ICP (Go-ICP), [57], proposed the first global optimum solution for Euclidean registration. It uses a nested branch and bound (BnB) structure, in which there is an outer BnB that performs a search in the rotation space $SO(3)$, which is associated with an inner BnB that gives the optimal translation. Using this combination, it is possible to jump over the local minimal, thus generating only the optimal global solution. Despite guaranteeing, in any case, the optimal solution, this method is mainly useful in cases in which real-time performance is not the most important.

2.2.2 Feature-based Registration

Feature-based registration methods require the use of visual features extracted from images and their correspondence between different images. This type of method is divided into several steps. The first is to extract keypoints from the two images. These points usually correspond to edges, corners, blobs, or ridges. Next, it is necessary to find matches between the keypoints of the two images. To do so, descriptors are extracted for each detected keypoint. The descriptors characterize the region in the image from which the keypoint was extracted and it is by comparing the descriptors that the correspondences between the points are found. The correspondences indicate that the same point is seen from two different perspectives. The last step is to compute the rotation and translation between the two cameras, based on the points they have in common.

Feature types

Feature extraction can be done from both 2D images and 3D point clouds. For 2D images, methods like [29, 5, 45, 20, 4, 3] are among the most common. Scale-Invariant Feature Transform (SIFT), [29], can extract invariant features from images that describe their local structure. It uses a Gaussian function to detect candidate locations from where feature points will be extracted. Through the local structure of a

feature, it becomes easier to perform matching of the features since they are more robust to translation, rotation, scaling, distortion, and light changes. Speeded Up Robust Features (SURF), [5], is both a feature detector and a descriptor, and partially inspired by SIFT. It relies on a Hessian matrix to detect blobs on an image. Oriented FAST and Rotated BRIEF (ORB), [45], consists of a FAST key-point detector, [44], and a BRIEF descriptor, [9], and is an efficient alternative to both SIFT and SURF. Unlike SIFT and SURF features, which exploit the Gaussian scale space, KAZE ([3]) features exploit the non-linear scale space by using non-linear diffusion filtering. To speed up the detection of features, AKAZE ([4]) features were created based on KAZE using a more efficient algorithm. For applications where specific features like edges or corners are important, Harris Corner Detector, [20], can be an alternative.

Although the methods of extracting 2D image features can be applied to point clouds if they are converted to 2D images, there are methods such as Fast Point Feature Histograms (FPFH), [46], which is a fast variant of Point Feature Histograms (PFH) proposed in [48, 47] that characterize the geometry of local feature 3D points, and can thus be used to match the same position in two different point clouds.

Registration methods

After having the feature points of both point clouds, the next step is to find matches between them. Many feature matching methods use brute force in this process and then use an outlier removal method to eliminate the bad matches (outliers) and estimate the best transformation based on the inliers. One popular solution to perform outlier removal is RANdom SAmple Consensus (RANSAC), [17]. RANSAC is an iterative method that selects random subsets of features, fits a model to the selected subset, and classifies the remaining points in inliers and outliers, according to a predetermined threshold. This process is repeated for a specific number of iterations to find the set of points that better suit the model. In its use to estimate odometry, this method selects sets of matches and uses methods like [49, 53] to estimate the transformation parameters between the matched points, and based on the error obtained, the points are classified as outliers and inliers. In the end, the set of inliers that produced the least error in the transformation is obtained.

The main problems of RANSAC are that it cannot yield good results in cases with a high percentage of outliers and that it does not guarantee optimal convergence. However,

due to its capacity to tolerate outliers and its simplicity of implementation, RANSAC is a common technique.

Over the years, variants of RANSAC have emerged that have adapted it to match points and lines, or a combination of both, and further improve its robustness and adaptability for real-time uses. Some of these variants are analyzed in [42].

One of the referred methods that is capable of estimating the transformation parameters (rotation and translation matrices) between sets of points is the Orthogonal Procrustes problem, [49]. This method solves a least-squares problem that estimates a orthogonal rotation $\mathbf{R} \in SO(3)$ by

$$\mathbf{A} = \mathbf{BR} + \mathbf{E} \tag{2.4}$$

$$\mathbf{RR}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I} \tag{2.5}$$

$$\min \operatorname{tr}(\mathbf{E}^T\mathbf{E}) = \min \sum \|\mathbf{A} - \mathbf{BR}\|_{\text{frob}}, \tag{2.6}$$

where matrices $\mathbf{A}$ and $\mathbf{B}$ have the matching feature points between the point clouds and $\mathbf{E}$ is the residual matrix. $\|.\|_{\text{frob}}$ denotes the matrix's Frobenius norm. The translation can be removed from the transformation by considering that the reference frame of each set of points is placed on the average distribution point. This way there is only rotation between the two sets of points.

Other solution to align the data is the proposed by Umeyama in [53]. His solution is also a least squares estimation of the transformation parameters between two provided point patterns $\mathbf{A}$ and $\mathbf{B}$. The transformation is just a rotation $\mathbf{R} \in SO(3)$, and the estimation of the parameters are defined by

$$\min_{\mathbf{R}} \|\mathbf{A} - \mathbf{BR}\|_{\text{frob}} = \|\mathbf{A}\|_{\text{frob}} + \|\mathbf{B}\|_{\text{frob}} - 2 \cdot \operatorname{tr}(\mathbf{DS}), \tag{2.7}$$

where

$$\mathbf{S} = \operatorname{diag}(1, 1, \det(\mathbf{AB}^T)). \tag{2.8}$$

An alternative to using RANSAC is Fast Global Registration, [60]. It aims at minimizing a rigid transformation $\mathbf{T}$ that aligns points from the point sets $\mathcal{P}$ and $\mathcal{Q}$, using correspondences between their points. The objective function for this problem is defined by

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \rho(\|\mathbf{p} - \mathbf{Tq}\|^2), \tag{2.9}$$

where ρ is a robust loss function, $\mathbf{p} \in \mathcal{P}$, $\mathbf{q} \in \mathcal{Q}$ and $\mathcal{K} = (\mathbf{p}, \mathbf{q})$ is the set of correspondences between points of $\mathcal{P}$ and $\mathcal{Q}$, obtained using Fast Point Feature Histogram, [46].

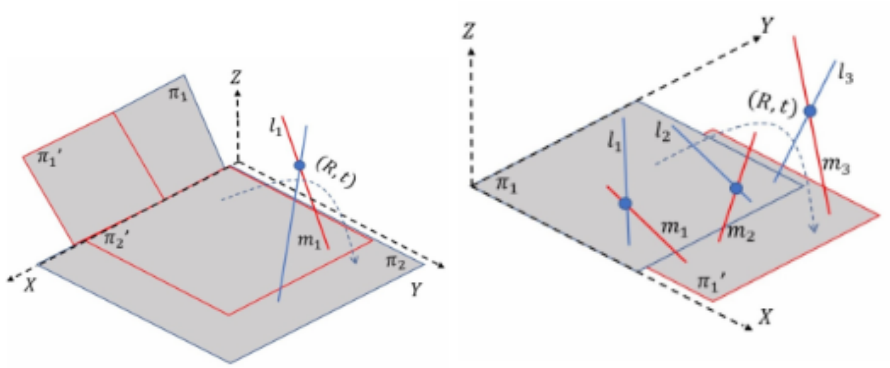
This method of registration thus corresponds to a direct approach, in which only one process is performed (single-stage), which corresponds to the minimization of the objective function. Unlike other methods, it does not require an initial estimate nor involve model fitting or local refinement which are iterative and computationally expensive processes, and it is also able of aligning surfaces with partial noise.

4PCS, [2], presents a solution more focused on data geometry by extracting all coplanar sets of 4-points that are roughly congruent. Congruent sets of points are related by a rigid transformation and are used since some ratios between points maintain invariant under this transformation. This method does not require any assumption over the initial transformation parameters and is robust even to data contaminated with outliers. Super4PCS, [32], is a fast registration algorithm that reduces the quadratic complexity of 4PCS to a linear complexity, that is more suitable to real-world application. Even with the reduced complexity, it maintains robustness to outliers and effectiveness in low overlap scans.

Finally, another type of solution-focused on data geometry is proposed in [43, 31]. These methods use minimal solver approaches combining different types of constraints over line intersections, point matches, and plane matches to align point clouds.

In [43] the following combinations between constraints are proposed: the intersection of a line and the correspondence of two planes, and the intersection of three lines and the correspondence of a plane, as shown in the Figure 2.2. [31], in addition to the combinations of constraints presented in [43], also present combinations between the intersection of three lines and the correspondence of one point, the intersection of one line and correspondence of two points, and, finally, the intersection of a line and the correspondence of a point and a plane. This type of approach using geometry constraints is better to deal with sparse 3D point cloud than methods using only point correspondences since specific feature points can disappear between scans, thus losing correspondences.

Both methods referred to use RANSAC frameworks to estimate, from the different combinations of match points, line intersections, or match planes, the pose variation between the two scans.



(a) (b)

Figure 2.2: Representation of two minimal solvers strategies: (a) 1 line intersection and 2 plane correspondences; (b) 3 line intersections and 1 plane correspondence. Source: [43].

2.2.3 Deep Learning Registration

Due to the recent increase in studies involving machine learning, namely the study of CNN (Convolutional Neural Networks), several signs of progress have been achieved in the area of odometry and mapping estimation. These new learning-based methods can surpass or complement the classical methods and achieve state-of-the-art results. The versatility of neural networks enabled the creation of deep learning methods that solve problems such as feature extraction ([14]), feature matching ([28]), registration of point clouds ([40, 54]) and estimation of odometry ([27, 13]).

LO-Net, [27], is considered to be the first successful method to perform learning-based LiDAR odometry. It uses the fitting ability of CNN to determine in an end-to-end manner the relative motion between a given pair of consecutive scans. This novel LiDAR odometry estimation network stands out for estimating the normal and a mask, which improves feature learning in dynamic regions, without using geometric constraints. This learned information is used in a mapping module to improve the accuracy of the estimation. It is also included in the network a geometry consistency constraint to regularize the learning. This network, however, is trained using ground truth data, becoming likely to overfitting, and it cannot take complete advantage of the geometric constraints of the data.

Traditional feature-based methods suffer to obtain high matching accuracy since obtaining accurate correspondences between points in sparse and noisy LiDAR data continues challenging. An alternative to these traditional methods is Deep Matching LiDAR Odometry (DMLO), [28], which is a learning-based framework to perform feature matching in the estimation of odometry tasks. DMLO is divided into two components. First, it uses a learning-based network to provide accurate matches between points from two scans. Before using the network to find matches, the data from the LiDAR scans are encoded into 2D images. Then, the network uses CNN to extract features and compare similarities between both images in local regions. For each pair of matches, it is calculated a confidence level to facilitate the final selection of matches. Finally, using Singular Value Decomposition (SVD) the distances between the matched pairs are minimized to obtain motion estimation.

In the registration of 3D scans, the main problem is often related to the consistency of outlier removal. 3DRegNet, [40], offers a deep learning solution that classifies matched points from consecutive scans in inliers or outliers and uses the inliers to do a regression of the motion parameters that aligns both scans. There are presented two approaches for the regression. The first approach uses a Deep Neural Network (DNN) and the second solves the Procrustes problem using SVD. It is shown that despite the run time being lower by solving the Procrustes problem, the difference is small and the DNN compensates by achieving higher accuracy. To improve the accuracy results it is also proposed the inclusion of a refinement network corresponding to a smaller 3DRegNet that will improve even further the initial registration.

Another method related to 3D point cloud registration is Deep Closest Point (DCP), [54]. DCP was created to solve the problems of the ICP method. Similar to other learning-based registration methods, it receives two point clouds and estimates the motion between them in the form of a rigid transformation. The method is divided into three parts. The first find correspondences between points using DGCNN, [55], that capture local geometric features. DGCNN was compared to PointNet, [41], which learns a global descriptor of the entire point cloud, however, DGCNN provided more consistent results. The second part finds matches between the point clouds, and the final part computes the desired rigid transformation through a layer that uses SVD.

Finally, when it comes to feature extraction, Fully Convolutional Geometric Features (FCGF), [14], can compute geometric features from a 3D point cloud by using a fully

convolutional network that quickly extracts features. These features are compact, obtained even in large scenes, and are used mainly to find correspondences between points, using, *e.g.* , a method like RANSAC.

2.3 Mapping

The 3D mapping problem consists of joining several sets of point clouds acquired at different times and in different positions, either by several static sensors looking over the same region or by sensors moving along a path, thus creating a map of a region. In any case, to ensure that there is no distortion in the final map, it is necessary to estimate the transformations between the coordinates systems where the point clouds are obtained and the reference coordinate system in which the map is being made. The odometry estimation problem is thus part of the mapping problem as it helps to estimate these transformations between the different scans acquired over time in different positions. Various methods allow these transformations to be computed not only in the case where there is movement but also in cases where it is intended to reconstruct a map from several static sensors that have different perspectives from the same region. The methods to be analyzed are Bundle Adjustment and Rotation Averaging.

Bundle Adjustment (BA) is an essential component in solving SfM problems, as shown in [1] and explained in survey [52]. It is capable of making an optimal visual reconstruction of a 3D structure and estimating the position of the camera or its calibration parameters using feature points and their correspondences. To obtain an optimal solution, the problem is defined as an optimization problem that minimizes the difference between a given point and its 3D projection on the image plane. This minimization is usually done using the l_2 norm, and the problem is typically solved by Levenberg–Marquardt's algorithm. This method, however, requires a good initialization of its parameters and has a long execution time for large data sets. To ensure a correct mapping using bundle adjustment, it is also necessary to consider loop closure. This way, the map is coherent even when it returns to a known position. An example of the effect of considering loop closure is shown in Figure 2.3. In red, the estimated odometry is represented, and in pink, the optimized trajectory using loop closure (represented in blue). The optimized trajectory is much closer to the ground truth than the trajectory using only the odometry estimation.

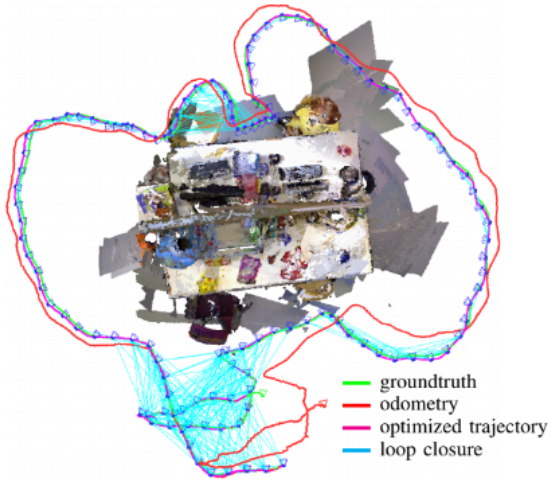


Figure 2.3: Trajectory optimization effect with loop closure. Source: [24].

Rotation averaging is a common alternative to bundle adjustment that has multiple strands as explained in the survey [21]. One of the strands is the problem of single rotation averaging, in which the same rotation $\mathbf{R}$ is calculated with data from various measurements, the final result being the average of the estimated rotations. If there are noise measurements that cause wrong rotations, some of the effects of these rotations will be removed when averaging. Another strand corresponds to Conjugate Rotation Averaging, which estimates a rotation between two pairs of rotations that have two different coordinate frames. The final strand is Multiple Rotation Averaging which takes several relative rotations $\mathbf{R}_{ij}$ and estimates the rotation $\mathbf{R}_i$ which satisfies $\mathbf{R}_{ij}\mathbf{R}_i = \mathbf{R}_j$, where the rotation matrices belong to the closed group $SO(3)$. This solution is the most common in SfM systems.

As shown in [19], if the rotations belong to the $SO(3)$ group, it is possible to use them in averaging relative motion estimates methods, since it is easier to join data from different sensors. This method can calculate the overall rotation of all sensors efficiently using the l_2 norm. However, this solution is not robust since it estimates wrong rotations in the presence of outliers. Using a methodology similar to [19], in [11] the l_1 norm is used instead of l_2. This way the method becomes less susceptible to outliers since the data is not squared. To try to further reduce the effect of these, a weight to the l_1 norm is inserted iteratively. The result produced by this approach is thus efficient, accurate, and scalable for larger problems.

2.4 Baseline Visual SLAM System

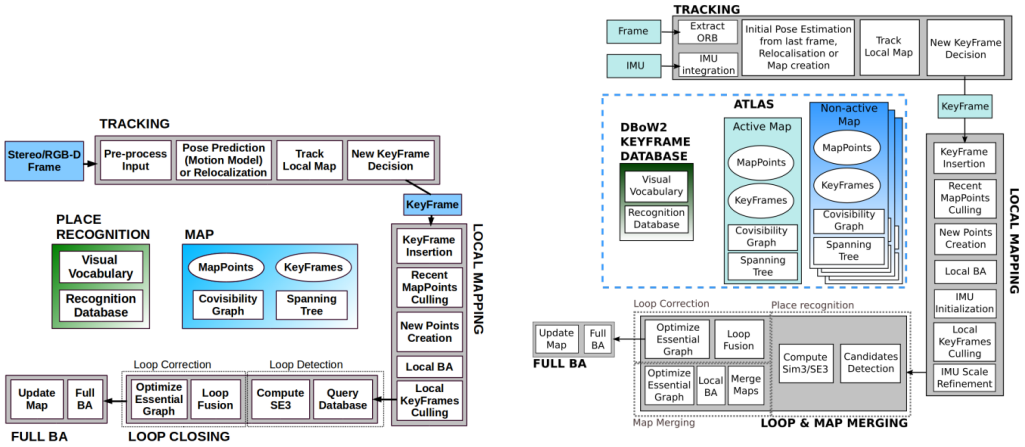
ORB-SLAM was first proposed in [34] and is a real-time visual SLAM system based on ORB features. This system has been developed and improved over the years from ORB-SLAM to ORB-SLAM2 [35] and more recently to ORB-SLAM3 [10].

The initial ORB-SLAM only works with monocular cameras. The advantages of using a monocular camera are that they are cheaper than other alternatives (e.g. stereo and RGB-D cameras) and have smaller dimensions (which facilitates its integration in robots). However, they have disadvantages, one of them being that it is not possible to estimate depth values based on a single frame, which does not allow the triangulation of 3D points. The initialization of the map thus requires several frames with a view of the same area to be used. Furthermore, odometry estimation is more prone to failure and the map scale to drift.

Despite some good results with ORB-SLAM, it was the introduction of ORB-SLAM2 that made this method well known and able to be used in different environments. ORB-SLAM2 maintains the basic structure of the first method but adds the use of stereo and RGB-D cameras in addition to monocular. Using stereo cameras takes advantage of both stereo and monocular points for more accurate results. Improvements have also been made to trajectory optimization, relocation, and other pipeline components.

The system ORB-SLAM2 is divided into four modules: tracking, local mapping, loop closure, and re-localization. The pipeline is shown in Figure 2.4(a). Tracking estimates the position of the camera along its trajectory using correspondences between image points of consecutive frames. Local mapping builds the map along the trajectory, manages its dimension, and locally optimizes the generated keyframes to reduce the odometry estimation error. Loop closing reduces the error of the entire map by performing a full BA when the camera returns to a previously mapped position. The relocation operates alternately to the construction of the map. It works as an extra module that is only used after the map has been built. For that, it uses Bag of Words to create an image database which is used to compare new images with the already processed images. This is used to detect areas being revisited or to assist the loop closure.

The main processes of this system like pose estimation, local BA optimization, and full BA optimization are performed using graph optimization. In the case of pose estimation, points with correspondences between consecutive frames are inserted in the graph



(a) ORB-SLAM2 (b) ORB-SLAM3

Figure 2.4: ORB-SLAM2 and ORB-SLAM3 frameworks. Sources: [35], [10].

with fixed positions and only the positions of the frames are optimized, from which the movement between frames is estimated. In the case of BA, the positions of keyframes and 3D points are optimized based on the points that the keyframes have in common. For the local BA, only the keyframes and their respective points close to the current keyframe are used, while in the full BA all keyframes and 3D points are used. These procedures have been shown to yield good results, however, they require correct tracking of points over several frames.

Recently, ORB-SLAM3 has further expanded ORB-SLAM2 by introducing a multi-map structure, the use of fisheye cameras, and the integration of IMU sensors. Figure 2.4(b) shows the ORB-SLAM3 framework. IMU integration assists not only in odometry estimation but can also be used in situations where there are few points or few correspondences between frames. The multi-map is intended to hold several maps simultaneously that may or may not be linked. In ORB-SLAM2 when there were failures in the estimation, the system would enter the relocation module, and would only map again when an already mapped zone was found, thus losing the information of the zones navigated during those instants. With a multi-map, when the robot track is lost, a new map is created and when it is verified that two maps come to the same zone, the maps are joined, not losing information.

2.5 Summary

This chapter first described SfM systems, its resemblance to SLAM systems, and the various types of sensors that are typically used in both systems. Next, the systems were divided into odometry estimation and mapping. For odometry estimation, both traditional and more recent methods were seen. The methods seen are divided into featureless, feature-based, and deep learning. For mapping, two typically used methods were seen and the loop closure problem was described. Finally, the ORB-SLAM system and its versions were described according to their structure, functionalities, and improvements between versions.

Chapter 3

2-step Visual Odomety

As an alternative to the methods presented in Section 2.2 for odometry estimation, two hybrid methods are proposed. The methods are hybrids since they combine two different methods, one for rotation estimation and one for translation estimation. Each hybrid approach has its method for rotation estimation but they share the method for translation estimation.

In the following sections, the two rotation estimation methods and the translation estimation method will be described.

3.1 Rotation Estimation

The two proposed methods for estimating the rotation are presented in this section. The first method uses 2D-2D correspondences to estimate the essential matrix. By decomposing this matrix using SVD the rotation and translation at less than a scale factor are obtained. The second method uses lines detected in the image, instead of keypoints. From the detected lines, three vanishing points are estimated with which the orientation of the camera is obtained. This method aims to exploit the structured environment of retail stores.

3.1.1 Essential Matrix Estimation

The first method used for estimating the rotation between two consecutive frames uses the essential matrix. The essential matrix represents the geometrical relationship between matching points of two images. Its estimation uses the detected keypoints (2D

points of the image coordinates), instead of 3D points. The desired rotation can be estimated by decomposition of the essential matrix.

The estimation of the essential matrix requires two sets of points that have correspondences with each other. The relation between the sets of points is given by

$$\mathbf{x'}^T\mathbf{E}\mathbf{x''} = 0, \tag{3.1}$$

as used in [22], where $\mathbf{x'}$ is the set of points of the current frame, $\mathbf{x''}$ is the set of points of the previous frame, and $\mathbf{E}$ is the essential matrix. (3.1) defines the coplanarity constraint between the two sets of points.

The essential matrix is then computed from (3.1) based on the five-point algorithm solver described in [37], that uses a RANSAC framework to remove outliers from the estimation. The matrix obtained from this algorithm has the following format

$$\mathbf{E} = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}. \tag{3.2}$$

Having the essential matrix, the rotation and translation are obtained by decomposing it using SVD according to

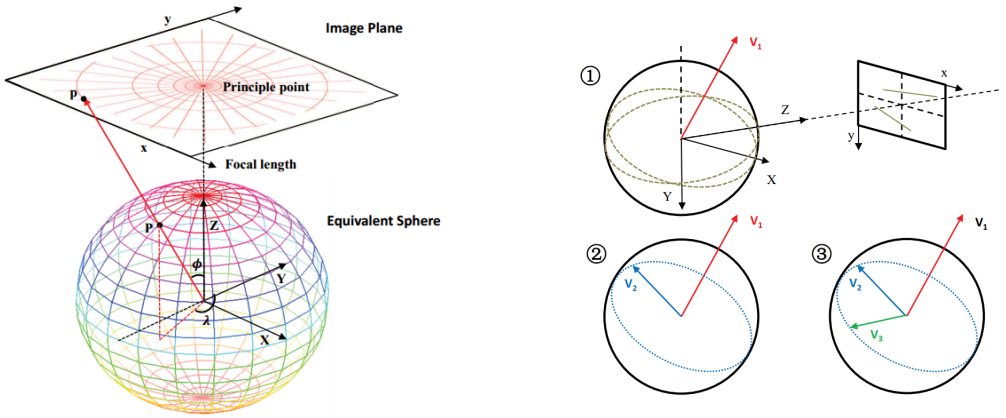
$$\mathbf{E} = \mathbf{U}\mathbf{S}\mathbf{V}^T, \tag{3.3}$$

where $\mathbf{U}$ and $\mathbf{V}$ are 3×3 orthogonal matrices with the singular vectors of $\mathbf{E}$ and $\mathbf{S}$ is a 3×3 diagonal matrix with the singular values of $\mathbf{E}$. With the matrices obtained from the decomposition, the rotation matrix $\mathbf{R}$ and the translation vector $[\mathbf{t}]_\times$ are computed by

$$\mathbf{R} = \mathbf{U} \begin{bmatrix} 0 & \pm 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{V}^T \tag{3.4}$$

$$[\mathbf{t}]_\times = \mathbf{U} \begin{bmatrix} 0 & \pm 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{S}\mathbf{V}^T. \tag{3.5}$$

This process gives four solutions, but only one of them places the points in front of both cameras, being that the desired solution. Only the rotation matrix $\mathbf{R}$ is used since the translation is obtained at less than a scale factor, so only information about its direction is available.



(a) (b)

Figure 3.1: Vanishing points estimation: (a) Relationship between the image plane and the equivalent sphere; (b) Procedure of generating orthogonal vanishing points. Source: [30].

3.1.2 Vanishing Points Estimation

The second method used to estimate a rotation matrix between two images of consecutive frames requires the estimation of vanishing points. For this the method presented in [30] was used. This method uses lines detected in the images to estimate vanishing points, assuming the environment is under the Manhattan World assumption. This assumption states that all surfaces in the environment are aligned along three dominant directions. Each dominant direction is thus described by one vanishing point. This method only works with images taken by pinhole cameras.

In order to be efficient for real-time applications, a polar grid is created by expanding the unit vectors on the equivalent sphere to intersect the image plane. This will be used to store the response of each line segments. Figure 3.1(a) shows the relationship between the image plane and the equivalent sphere. A point $(x, y)^T$ on the image is converted to a 3D point $\mathbf{P} = (X, Y, Z)^T$ on the equivalent sphere according to

$$\begin{cases} X = x - x_0 \\ Y = y - y_0 \\ Z = f \end{cases}, \tag{3.6}$$

where (x_0, y_0) are the principal point and f is the focal length. The longitude and

latitude is further computed using

$$\begin{cases} \phi = \text{acos}(Z/\sqrt{X^2+Y^2+Z^2}) \\ \lambda = \text{atan2}(X,Y) + \pi \end{cases}, \tag{3.7}$$

The first vanishing point $\mathbf{v}_1 = (X_1, Y_1, Z_1)^T$ is computed iteratively by randomly choosing two line segments and computing their intersection point (step 1 of Figure 3.1(b)). Since the three vanishing points are orthogonal, the second vanishing point will belong to the great circle of $\mathbf{v}_1$ (step 2 of Figure 3.1(b)). The circle is divided into fractions of 360° (1% accuracy), and for each fraction a vanishing point $\mathbf{v}_2 = (X_2, Y_2, Z_2)$ is computed according to

$$\begin{cases} X_2 = \sin(\phi)\ \sin(\lambda) \\ Y_2 = \sin(\phi)\ \cos(\lambda) \\ Z_2 = \cos(\phi) \end{cases}, \tag{3.8}$$

and

$$X_1\ X_2 + Y_1\ Y_2 + Z_1\ Z_2 = 0, \tag{3.9}$$

where ϕ is the latitude and λ is the longitude. Finally, the third vanishing point $\mathbf{v}_3$ is the cross product of $\mathbf{v}_1$ and $\mathbf{v}_2$

$$\mathbf{v}_3 = \mathbf{v}_1 \times \mathbf{v}_2, \tag{3.10}$$

since the three vanishing points must be orthogonal to each other (step 3 of Figure 3.1(b)).

Several hypotheses are thus produced for the set of the three vanishing points, so it is necessary to validate and choose the best set. The validation is done by computing the response of the detected line segments to each hypothesis, and the vanishing points that produce the best response are selected.

Once the vanishing points are estimated, they are combined to get the rotation matrix that describes the rotation of the image. With this approach, however, it is not possible to estimate the translation.

3.2 Translation Estimation

Using the rotations obtained with each of the methods in the previous section, a translation estimation method is proposed that combines RANSAC for outlier removal and

the least-squares method for translation estimation. As seen in Section 2.2.2, RANSAC is an iterative method that estimates from a random set of points the model that best fits the given dataset. The best model is selected as the model that best fits the dataset, *i.e.* has the most inliers. Points that are considered outliers are removed from the dataset and are not used to estimate the movement between consecutive frames.

Having correspondences between points, to estimate the rotation and translation that align a point ${}^{\mathrm{l}}\mathbf{p} = [{}^{\mathrm{l}}x, {}^{\mathrm{l}}y, {}^{\mathrm{l}}z]^T$ in the last frame and a point ${}^{\mathrm{c}}\mathbf{p} = [{}^{\mathrm{c}}x, {}^{\mathrm{c}}y, {}^{\mathrm{c}}z]^T$ in the current frame, at least 3 sets of matching points are required. Since the rotation has already been estimated using the methods described in Section 3.1.1 or Section 3.1.2, instead of needing a minimum of 3 sets of matching points, only 1 set is needed. A point is then chosen randomly, with which the translation is computed. Having the rotation and translation, the number of inliers is computed based on the squared error e between the point in the current frame and the point in the previous frame transformed to the current frame as follows

$$e = \|{}^{\mathrm{c}}\mathbf{p} - (\mathbf{R} \cdot {}^{\mathrm{l}}\mathbf{p} + \mathbf{t})\|^2. \tag{3.11}$$

The choice between inliers and outliers is established based on a predefined threshold value.

This process of randomly choosing a point, computing the error according to (3.11), and counting the number of inliers is done iteratively over a fixed number of iterations. In the end, the inliers of the iteration that produced the most inliers are obtained.

Since this is an iterative method, it will always perform the predefined number of iterations. And since the choice of points is random, it is necessary to choose a value that is not too low to ensure a good estimation. However, for real-time estimation cases increasing the number of iterations in order to guarantee a good estimation may increase in some cases the computation time unnecessarily. Therefore, a stopping criterion was defined that calculates the number of iterations needed to be performed based on the best result obtained so far. The number of iterations k is given by

$$k = \frac{\log(1-p)}{\log(1-w^n)}, \tag{3.12}$$

where p is the desired probability of getting a useful result, n is the number of randomly chosen points and w is the probability of selecting an inlier point in the data set and is

given by

$$w = \frac{\text{number of inliers in the selected model}}{\text{total number of points}}. \tag{3.13}$$

Using the inlier points obtained, the translation is estimated using the least squares method. The residual r_i of a point $i \in \{1...N\}$ is defined as the difference between the value of the point i in the current frame ${}^{\mathrm{c}}\mathbf{p}_i = [{}^{\mathrm{c}}x_i, {}^{\mathrm{c}}y_i, {}^{\mathrm{c}}z_i]^T$ with the value of the point i in the previous frame ${}^{\mathrm{l}}\mathbf{p}_i = [{}^{\mathrm{l}}x_i, {}^{\mathrm{l}}y_i, {}^{\mathrm{l}}z_i]^T$ transformed to the referential of the current frame, according to

$$r_i = {}^{\mathrm{c}}\mathbf{p}_i - (\mathbf{R} \cdot {}^{l}\mathbf{p}_i + \mathbf{t}), \tag{3.14}$$

where $\mathbf{R}$ is the previously estimated 3×3 relative rotation matrix and $\mathbf{t}$ is a 3×1 column vector corresponding to the desired translation.

The goal is to obtain the optimal value of the parameter $\mathbf{t} = [t_x, t_y, t_z]^T$ that minimizes the square sum of the residuals S, according to

$$S = \sum_{i=1}^{N} \|r_i\|^2 = \sum_{i=1}^{N} \|{}^{\mathrm{c}}\mathbf{p}_i - (\mathbf{R} \cdot {}^{\mathrm{l}}\mathbf{p}_i + \mathbf{t})\|^2. \tag{3.15}$$

Solving this problem results in

$$\frac{\partial S}{\partial \mathbf{t}} = 0 \Rightarrow \mathbf{t} = \frac{\sum_{i=1}^{N} {}^{\mathrm{c}}\mathbf{p}_i - \mathbf{R} \cdot {}^{\mathrm{l}}\mathbf{p}_i}{N}. \tag{3.16}$$

Chapter 4

Structure-from-Motion Pipeline

The developed Structure-from-Motion (SfM) system is inspired on the ORB-SLAM2 pipeline (Figure 2.4(a)), being divided into three modules: Tracking, Local Mapping, and Loop Closure. Based on a sequence of images received, the Tracking extracts important features from the images, finds matches between consecutive frames, and estimates the position of the camera along with its movement (visual odometry). It also creates keyframes (detection of important camera positions based on the available features over time). Local Mapping aims to optimize the position of new keyframes and their respective map points, taking into account the existing keyframes (and respective map points) close to the new keyframe. The final module, Loop Closure, optimizes the entire map when a loop is detected. That is when the camera returns to a position already known on the map. Figure 4.1 shows how the three modules are combined to create the SfM system.

The pipeline was implemented in C++, from scratch and the following external libraries were used: OpenCV[1], Point Cloud Library[2], g^{2}o [26], and DBoW3 [33]. OpenCV was used for image processing, notably feature extraction and matching using descriptors, among others. Point Cloud Library was used for visualization and for registering 3D point clouds from the 3D points estimated in each frame. The g^{2}o library was used to perform graph-based optimizations such as local and full bundle adjustment and pose optimization. The DBoW3 library is used for place recognition, and was mainly used for loop closure detection.

[1] https://opencv.org
[2] https://pointclouds.org

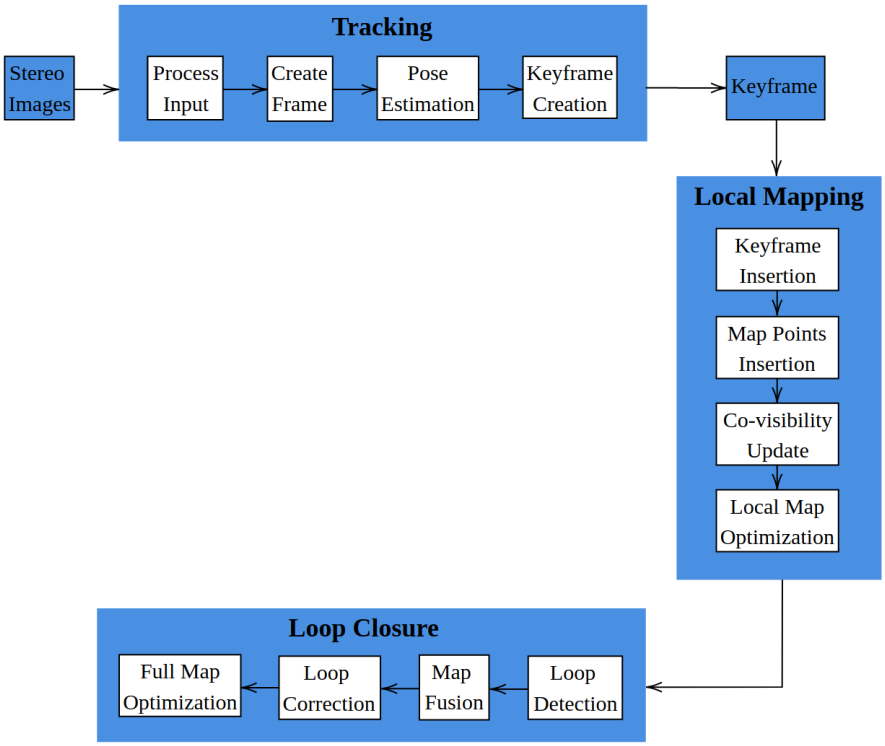


Figure 4.1: Developed SfM system pipeline.

In the following sections, each of the three implemented modules will be explained, respectively.

4.1 Tracking

The developed Tracking module, shown in Figure 4.2, is divided into four sub-modules. The module starts by receiving the stereo camera images captured at a certain time and processes them. The images can come from either a fisheye or a pinhole camera and are assumed to be rectified and their calibration is also assumed to be available. If not, they can be easily calibrated or rectified using calibration toolboxes. For every pair of stereo images received, a frame is created. For each frame, keypoints are detected and descriptors are extracted using several alternatives (*e.g.* SIFT, ORB, or AKAZE features). With the descriptors, the keypoints of the left and right images are matched, which allows the triangulation of 3D points. Using the computed 3D points or the 2D

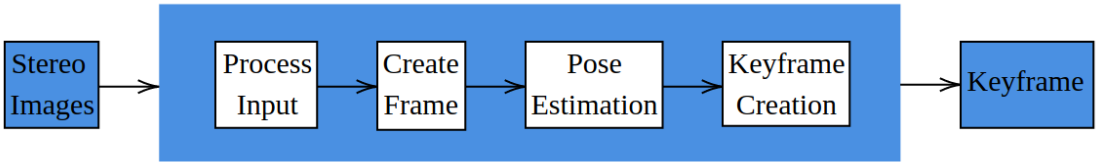


Figure 4.2: Tracking module pipeline.

keypoints with matches in consecutive frames, it is estimated the relative motion between the consecutive frames, using 3D-3D or 2D-2D correspondences. For that, it is required a new matching of the descriptors between the current and previous frames. The motion estimated is accumulated over time to obtain the camera movement from the current frame to the world reference frame. The final step of the Tracking module is the creation of keyframes. Keyframes represent similar sets of frames and prevent repeated information from being inserted into the map, which would only increase its size unnecessarily. New keyframes have the same information as the frame that created it and are created according to the conditions defined in Section 4.1.4.

4.1.1 Process Image Input

The Tracking module input is the pairs of images taken by a stereo camera. Images from two different types of cameras can be processed: pinhole and fisheye. Both types of images can be processed directly, however in fisheye images, the distortion can be removed thus converting the image to the pinhole model, as shown in Figure 4.3. Although this remapping removes the distortion, which is useful in certain applications, the edges of the image get blur, which affects feature detection in those areas.

The images also need to be rectified, *i.e.* the images from the two cameras need to be parallel (virtually create fronto parallel cameras). As this is not always the case it is necessary to estimate a transformation that remaps the right image so that corresponding points have the same y-coordinate in both images. This must be the case since the method of computing 3D points and the optimizations made in the Local Mapping and Loop closure modules depends on this characteristic.

(a) (b)

Figure 4.3: Example of possible system inputs: (a) Fisheye image; (b) Fisheye image with distortion removed.

4.1.2 Create Frame

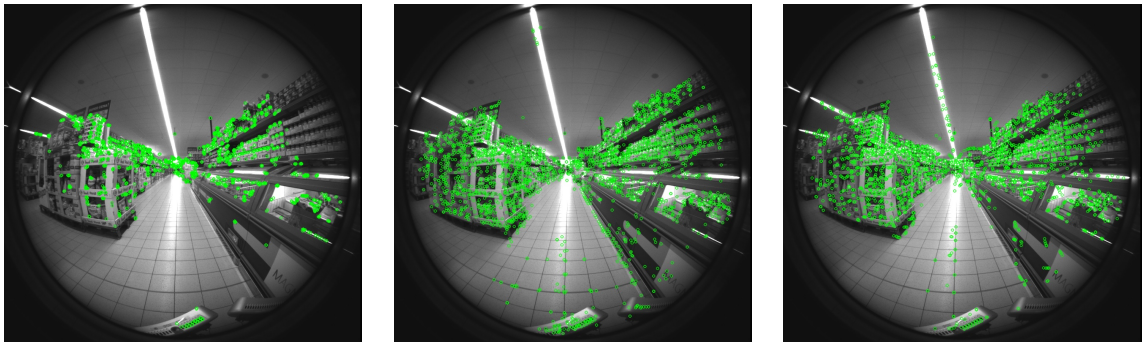
Frames describe each stereo image pair received. It contains all the information extracted from the images. This information is keypoints, descriptors, correspondences between keypoints and 3D points.

Keypoint detection and Descriptor extraction

Keypoints are points on the image that are differentiated either by color, intensity, texture, among others. Their differentiation is important to allow an easier match with other similar keypoints. Descriptors are calculated for each detected keypoint. The descriptors contain information about the region around the keypoint in the compact form of a vector and are used to determine whether two keypoints are similar.

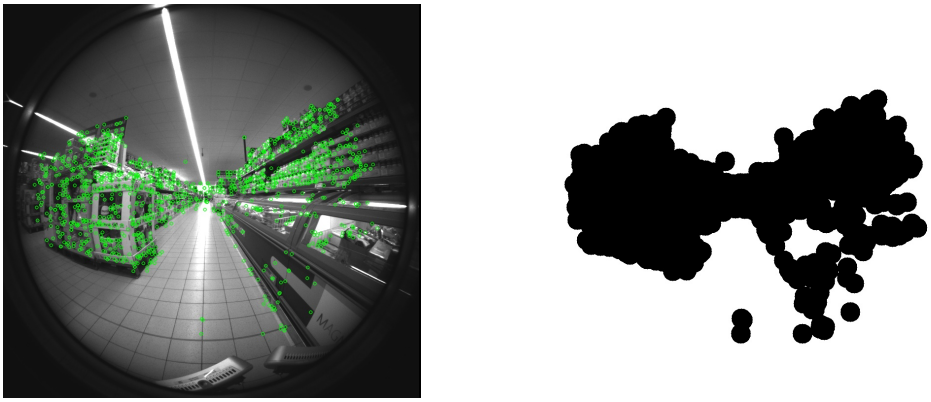
Three different types of features were used: ORB, SIFT, and AKAZE. Figure 4.4 shows the keypoints detected for each of these types of features for the same image from a fisheye camera.

To reduce the computation time for keypoint detection, a mask for the image is created. This mask corresponds to a binary matrix with the dimension of the image and indicates in which regions of the image to look for keypoints. The mask is constructed by selecting a circular region with a diameter of 20 pixels and a center on the keypoint for all keypoints that have a stereo match, in the previous image. This mask is reconstructed



(a) (b) (c)

Figure 4.4: Example of detection of keypoints on a fisheye image using different types of features: (a) ORB features; (b) SIFT features, (c) AKAZE features.



(a) (b)

Figure 4.5: Example of a mask created to delimit the region to search for keypoints: (a) Keypoints detected with stereo correspondence; (b) Mask created based on the keypoints with stereo correspondence.

whenever a new keyframe is created.

Figure 4.5 shows the left image of the stereo pair with the detected keypoints that have a match in the other image of the stereo pair and the respective mask created for that image. For all frames that belong to the keyframe created by this image, this mask is used to detect feature points. Only keypoints in the masked region are detected. This process, besides helping computationally since a search is made in a smaller area of the image, also removes potential outliers while allowing tracking of points over several frames.

Descriptor matching

The matching of keypoints between the stereo image pair is done based on the descriptors extracted for each keypoint of each image. The matching is done using a brute-force descriptor matcher that searches for k best matches of the right image for each descriptor of the left image. The best matches are those with the smallest distance, which are computed using the l_2 norm.

The default value used for k is 2. For values of $k \geq 2$ the two keypoints of the right image with the smallest distance to the keypoint of the left image may have close distances. In this case, Lowe's ratio test is applied, where the match is only valid if

$$d_{1^{\text{st}}\text{closest}} > 0.75 \cdot d_{2^{\text{nd}}\text{closest}}, \tag{4.1}$$

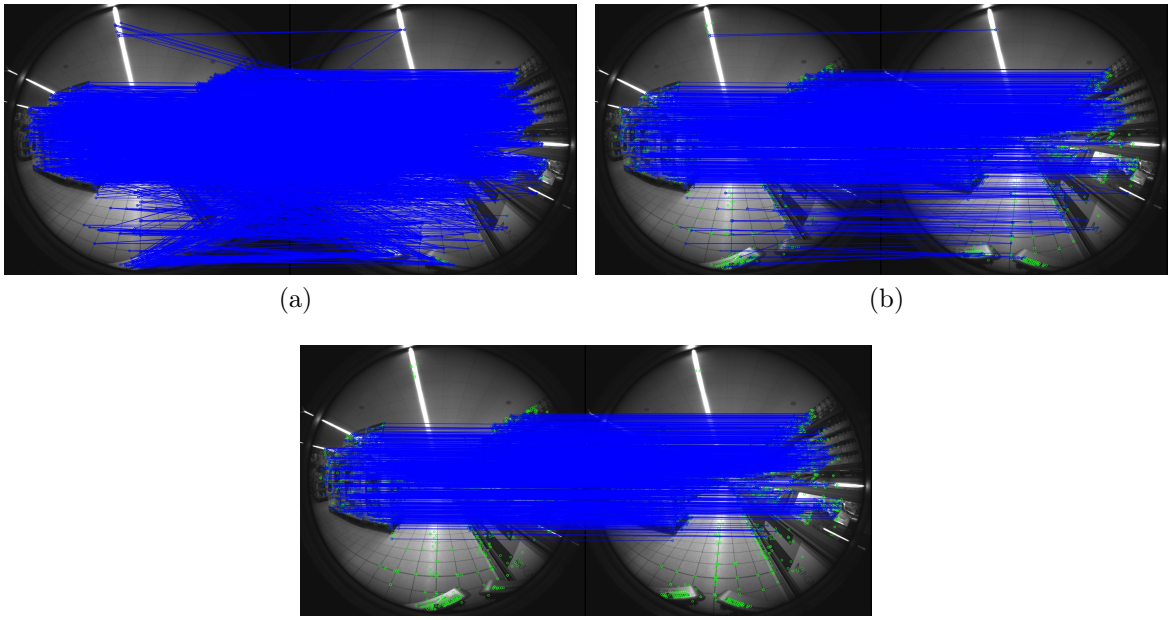
where $d_{1^{\text{st}}\text{closest}}$ and $d_{2^{\text{nd}}\text{closest}}$ are the distances computed for the closest and second closest match, respectively.

To further eliminate bad matches, the fundamental matrix is estimated. This estimation uses a RANSAC algorithm, which allows the exclusion of matches that are classified as outliers in the matrix estimation.

Figure 4.6 shows the initially estimated matches, the results after removing oulier matches using the Lowe's ratio test, and the final result that uses the Lowe's ratio test followed by a RANSAC algorithm to remove outliers.

3D points computation

The use of stereo cameras allows 3D points to be easily computed using only the images of a single frame. Since the images are rectified, both belong to the same plane, and there is only a translation on the X axis between the left and right images, which corresponds to the baseline of the camera as shown in Figure 4.7. The baseline value is a known input parameter since the cameras are calibrated.



(a) (b)

(c)

Figure 4.6: Example of the matches detected for each stage of their estimation: (a) Matches estimated using a brute-force descriptor matcher with $k = 2$; (b) Matches after removing outlier matches according to Lowe's ratio test; (c) Final matches, obtained by removing outlier matches from the matches obtained in (b) using a RANSAC algorithm.

A 3D point $\mathbf{X} = [x, y, z]^T$ in the left camera reference is estimated using

$$d = x_{\text{left}} - x_{\text{right}} \tag{4.2}$$

$$z = \frac{b \cdot f_x}{d}, \tag{4.3}$$

$$x = \frac{x_{\text{left}} \cdot z}{f_x} \tag{4.4}$$

$$y = \frac{y_{\text{left}} \cdot z}{f_y}, \tag{4.5}$$

where d is the disparity between the cameras, b is the stereo baseline and $f = (f_x, f_y)$ is the focal length of the left camera.

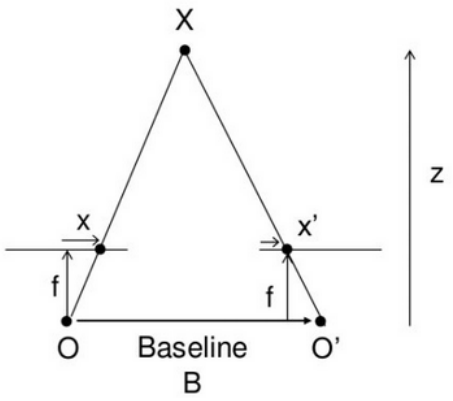


Figure 4.7: Depth estimation in parallel cameras. Source: https://docs.opencv.org.

4.1.3 Pose Estimation

The pose estimation sub-module aims at estimating the trajectory of the camera along its path in the world reference frame. The world reference is defined as being the reference of the first keyframe.

The movement of the current frame i in the world reference frame ${}^{W}\mathbf{T}_i$ is computed as the accumulation of the relative transformations between consecutive frames according to

$$ {}^{W}\mathbf{T}_i = {}^{W}\mathbf{T}_1 \cdot {}^{1}\mathbf{T}_2 \cdot \ldots \cdot {}^{i-1}\mathbf{T}_i, \tag{4.6} $$

where i is the current frame.

The relative transformations between frames ${}^{i-1}\mathbf{T}_i$ can be estimated using different methods. The two hybrid methods proposed in Chapter 3 can be used, as well as other known methods such as Umeyama, seen in Section 2.2.2.

Since the methods are based on 2D-2D or 3D-3D correspondences between frames, it is necessary to have a match between the keypoints of the left images of two consecutive frames. Only keypoints for which there is a stereo match, *i.e.* for which there is an estimated 3D point, are used. This is especially important for the case of pose estimation using 3D-3D correspondences.

Figure 4.8 shows in blue the computed stereo matches and in red the computed frame-to-frame matches between the left images of consecutive frames.

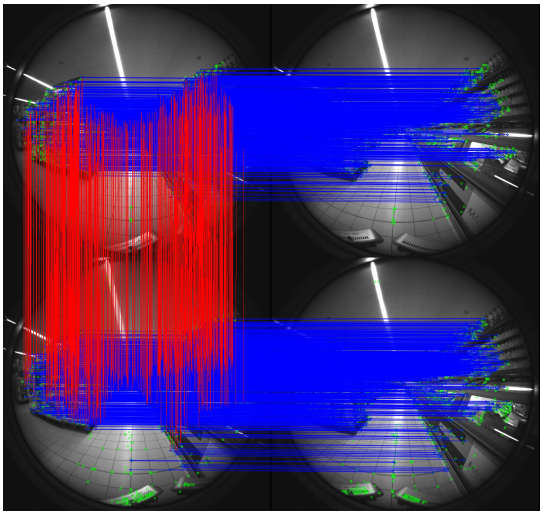


Figure 4.8: Example of the stereo matches (blue) and frame-to-frame matches (red) needed for relative motion estimation between consecutive frames.

4.1.4 New Keyframe Decision and Creation

The last task done in the Tracking module is the decision of creating new keyframes, and their creation when that decision is favorable.

A keyframe is defined as a frame that represents similar frames of a certain region. Due to the high frequencies of images captured by the cameras, consecutive images are often similar to each other, especially when the movement speed of the camera is reduced or when the camera is moving in straight lines. Although this high frequency of image capture guarantees a better tracking of features and consequently a better estimation of the camera movement, if all frames and their respective 3D points are sent to the local mapping module to be inserted into the map, the system would have a high computational cost both in terms of execution speed and memory usage. The importance of creating keyframes is thus related to the complexity that the map will have. It is, therefore, necessary to ensure that frames with new features of the environment will be given to the map, at the same time that frames that will only add features already in the map are only used to estimate the trajectory.

Two conditions were defined to decide when a new keyframe is created:

1. The current keyframe tracks less than 30% of the points of the current keyframe.

This condition ensures that new information about the environment is not lost by group-

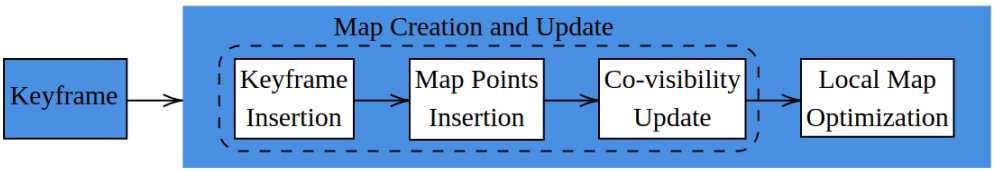


Figure 4.9: Local Mapping module pipeline.

ing frames that represent different areas of the environment. For this, a match between the keypoints of the current frame and the current keyframe is made. A keyframe is thus created when the current keyframe tracks less than 30% of the points of the current keyframe.

2. Each keyframe can only represent, at maximum, 30 frames.

A maximum limit of frames for each keyframe is set so that the complexity of the map is kept reduced while creating enough keyframes to guarantee a correct mapping. The default maximum number of frames is set to 30 because that is the amount of FPS of the used camera. This value can be changed via the input configuration file.

4.2 Local Mapping

The Local Mapping module is responsible for creating and managing the map. The map consists of the keyframes created in the Tracking module and the corresponding 3D points (map points). The creation and management of the map are divided into two sub-modules, as shown in Figure 4.9. When creating or adding data to the map, the module is responsible for adding keyframes and map points. However, since different keyframes can see the same map points, it is necessary to evaluate when to insert new map points or update existing map points. The second sub-module optimizes the poses of the current keyframe and the keyframes connected to it, as well as the map points belonging to those keyframes. This process aims to reduce the impact of trajectory estimation errors on the Tracking module.

In this section, the composition of the map will be described, as well as the sub-modules showed in Figure 4.9.

4.2.1 Map

The map has two types of data, keyframes and map points. Keyframes, as seen in Section 4.1, represent sets of similar consecutive frames and map points are the 3D points seen by the keyframes. The map thus has the keyframes created and the map points associated with each keyframe. Map points can belong to several keyframes and keyframes have connections between each other based on the map points they have in common, creating a co-visibility graph.

4.2.2 Map Creation and Update

The first step in Local Mapping is to add new keyframes to the existing keyframe map. This is a simple process since each keyframe is unique, so there is no risk of having two repeated keyframes in the map.

After inserting the new keyframe, the map points seen by it are inserted into the map. Unlike the keyframes, for the map points, it is necessary to check if those points already exist on the map (seen by other keyframes) or if they are new points. For that, matching is made between the map points seen by the new keyframe and by the previous keyframe. This matching, similarly to what is done in Tracking, is done based on the descriptors of the keypoints with which the map points were computed. Based on the matches, the points that did not have a match are considered new and are inserted in the map. The ones that had a match are not inserted, and the new observations for those points are added to the respective map points.

Finally, co-visibility connections are made between the keyframes. These connections are established based on the number of map points that the keyframes have in common. The value chosen for the minimum number of points in common can be chosen from an initial configuration file. The default value used is at least 10 points in common. These co-visibility connections are important for the optimization processes in the next section and in the loop closure module.

4.2.3 Local Map Optimization

To correct some of the drift that is inevitably obtained in this type of system, due to the continuous integration of odometry estimation errors, a local BA is performed.

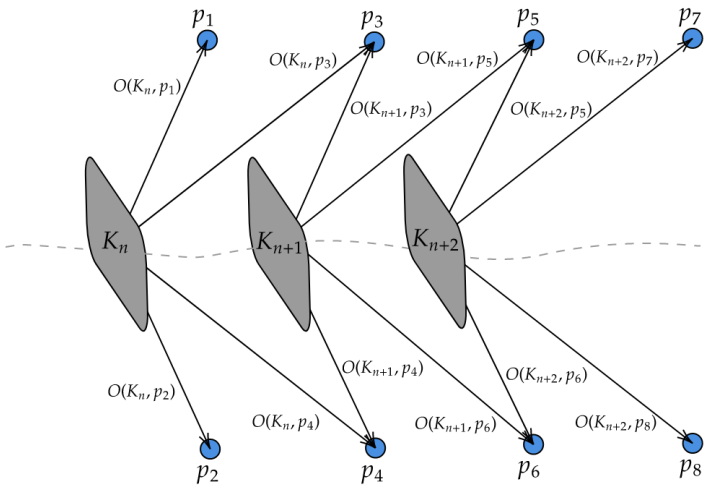


Figure 4.10: Scheme of the graph created to perform bundle adjustment.

This optimization will adjust the position of the current keyframe, the position of the keyframes connected to it according to the co-visibility graph, and the 3D points of all these keyframes, based on the points seen by each keyframe (observations). The observations correspond to the coordinates of the keypoints that originated the 3D points.

The optimization is done using the Levenberg–Marquardt method and aims to minimize the reprojection error between 3D points seen by multiple keyframes, according to observations $\mathbf{o} = [u_L, v_L, u_R]^T$, where (u_L, v_L) are the keypoint coordinates of the left image and u_R is the keypoint horizontal coordinate of the right image.

Let $\mathcal{K} = \{\mathbf{k_1}, ..., \mathbf{k_n}\}$ be the set of keyframes that includes the current keyframe and the keyframes connected to it by co-visibility connections, and defining $\mathcal{X}_{\mathbf{k_i}} = \{\mathbf{x_1^i}, ..., \mathbf{x_m^i}\}$ as the set of 3D points that are seen by the keyframe $\mathbf{k}_i \in \mathcal{K}$, the cost function to minimize is

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \rho(\|\mathbf{o}_j^i - \pi(\mathbf{R}_{\mathbf{k}_i}\mathbf{x}_j^i + \mathbf{t}_{\mathbf{k}_i})\|^2), \tag{4.7}$$

where ρ is the robust Huber cost function, $\mathbf{R}_k \in SO(3)$ and $\mathbf{t}_k \in \mathbb{R}^3$ are the orientation and position of the keyframe k, respectively, and π is the function that projects the 3D

points onto the image, according to

$$\pi\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} f_x \frac{X}{Z} + c_x \\ f_y \frac{Y}{Z} + c_y \\ f_x \frac{X-b}{Z} + c_x \end{bmatrix}, \tag{4.8}$$

where $f = (f_x, f_y)$ is the left camera focal length, $c = (c_x, c_y)$ is the left camera principal point and b is the stereo camera baseline.

In the optimization process, initially, only 5 iterations are performed, after which outlier observations are removed according to an established threshold value. After the outliers are removed, 10 iterations are performed and the positions of the keyframes and map points are updated according to the new optimized positions.

Figure 4.10 exemplifies the optimization process by showing the positions of three keyframes $\mathbf{K}_{n+i}$, $i \in \{0, 1, 2\}$ and eight 3D points $\mathbf{p}_j$, $j \in \{1, ..., 8\}$. Choosing $\mathbf{K}_{n+2}$ as the current keyframe, it has co-visibility connections only with $\mathbf{K}_{n+1}$ because it has points in common (which is not the case with $\mathbf{K}_n$). The optimization thus uses the keyframes $\mathbf{K}_{n+2}$ and $\mathbf{K}_{n+1}$ and their 3D points $\mathbf{p}_3$, $\mathbf{p}_4$, $\mathbf{p}_5$, $\mathbf{p}_6$, $\mathbf{p}_7$ and $\mathbf{p}_8$. The observations of the points for each keyframe are shown in Figure 4.10 as $O(\mathbf{K}_{n+i}, \mathbf{p}_j)$. The optimization is therefore based on the several 3D points seen by multiple keyframes from different observations.

4.3 Loop Closure

Loop closure aims to detect when the camera returns to a previously visited position that is already mapped. Throughout the motion, it is common for the trajectory error to increase mainly due to the accumulation of several small motion estimation errors. By returning to an already known position it is possible to eliminate some of that error so that it does not grow infinitely. Figure 4.11 shows the implemented module and its four sub-modules: loop detection, map fusion, loop correction and full map optimization.

Loop detection uses bag-of-words (BoW) to estimate similarities between images that are used to detect loop candidates and performs feature matching to validate the detected candidate keyframes. Map fusion joins the current and candidate keyframes by updating the observations of the map points seen by both keyframes and by creating connections in the co-visibility graph. Loop correction aims to distribute the loop closing

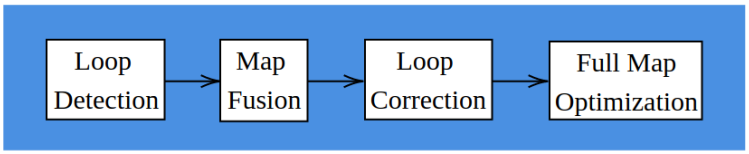


Figure 4.11: Loop Closure module pipeline.

error along the trajectory. Finally, the full map optimization performs a BA on all map points and keyframes. In the following sections, each of the modules will be described.

4.3.1 Loop Detection

The first step in correcting the loop closure error is to detect when the loop is closed. To do so, two procedures are performed. The first detects loop candidates and the second checks whether any of the candidates are valid.

The first step of loop candidate detection uses BoW place recognition. For this, whenever a new keyframe is created in the Tracking module, the left stereo image of the created keyframe is converted into a BoW word. This word is created based on the keyframes and descriptors detected in the image and acts as a frequency histogram of the features in the image. After creating the word, it is inserted into a database that associates the word with the id of its keyframe. In the loop closure module, a comparison of the word created for the current keyframe is made with the words in the database to compute a similarity score between the keyframes. The score is computed using the l_1 norm (here denoted as $|.|$) according to

$$s(\mathbf{w}_1, \mathbf{w}_2) = 1 - \frac{1}{2}\left|\frac{\mathbf{w}_1}{|\mathbf{w}_1|} - \frac{\mathbf{w}_2}{|\mathbf{w}_2|}\right|, \tag{4.9}$$

where $\mathbf{w}_1$ and $\mathbf{w}_2$ are the words of two different keyframes. The values of the scores are between $[0, 1]$. The keyframes that have a score greater than 0.9 times the maximum score among the keyframes connected to the current keyframe are considered candidates for loop closure.

The second step is to validate the detected candidates. Although BoW is a good solution that allows fast matches between all keyframes, perceptual aliasing can occur. When this occurs, images from different places originate similar words, which can cause false loop candidates. To avoid such situations, a validation of the geometric consistency

between the candidate and the current keyframes is performed through feature matches. For each candidate, a match is made between the descriptors of the candidate and current keyframe points. The candidates are only valid if at least 30% of the points of the current keyframe have matches in the candidate keyframe.

Since the optimizations performed to correct loop errors can be computationally high, new loops are only detected 10 keyframes after the last loop was detected.

4.3.2 Map Fusion

After validating the detection of a new loop closure, it is necessary to pass that information to the map. To do so, an update of the observations of the points that the current keyframe and the loop keyframe (keyframe where the loop was detected) have in common must be done. It is also necessary to create co-visibility connections between these keyframes and possibly other keyframes that share observations for the same points.

Similar to the process done in Section 4.2.2, the update of the observations of the map points that the current keyframe and the loop keyframe have in common is done based on the descriptor matching done in the previous section to validate the loop. Only the points seen by both keyframes are thus updated. As the points are updated, the number of points that the current keyframe has in common with other keyframes are counted. This count is necessary since other keyframes may observe the same points that the current keyframe and the loop keyframe have in common. Based on the number of points the keyframes have in common new co-visibility connections are established.

4.3.3 Loop Correction

Before performing the final step of optimizing the total map, an adjustment is made to the positions of the keyframes in order to distribute the loop closure error over the entire trajectory. Two ways of performing the error distribution have been implemented. The first is keyframe optimization and the second is transformation averaging. In both, it is necessary to first estimate the correct position of the current keyframe. Initially, an attempt was made to estimate odometry between the current keyframe and the keyframe that completes the loop, however, the estimation was not correct. Therefore, the position of the current keyframe was estimated by doing an optimization identical

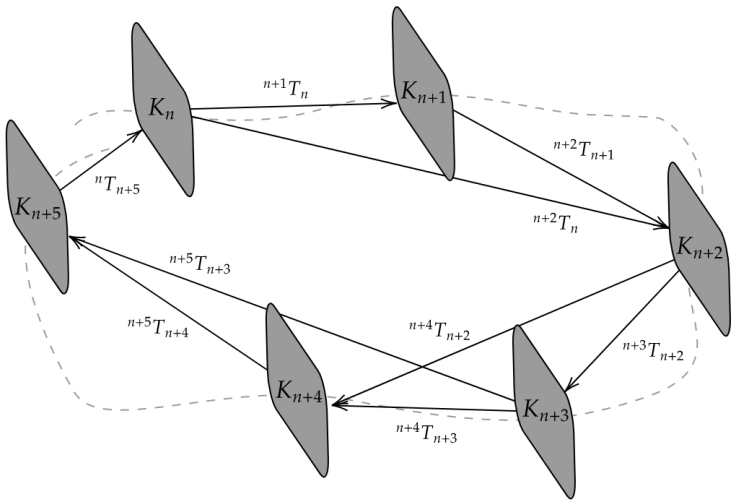


Figure 4.12: Scheme of the graph created to perform keyframe optimization.

to the one in Section 4.2.3. Note that after the fusion of the map the set of keyframes connected to the current one now has the keyframe where the loop closure was detected. The difference with the BA performed in Section 4.2.3 is that in this case the optimized positions are not kept, but are only used to estimate the relative transformation between the keyframes forming the loop.

Having the correct position for the current frame, one of two methods is applied: keyframe optimization or transformation averaging. The keyframe optimization corresponds to the optimization of a pose-graph that has as vertices the position of the keyframes and as edges between the keyframes the relative position between them. There are only edges between keyframes connected by the co-visibility graph. The relative position between keyframes is computed based on the positions of the keyframes in the world referential, with the exception of the edge that connects the loop where the previously estimated position is used. A scheme of the created graph is presented in Figure 4.12.

Let $\mathcal{K} = \{\mathbf{K}_1, ..., \mathbf{K}_n\}$ be the set with all keyframes positions and using the Levenberg– Marquardt method, the goal is to minimize the position error between keyframes according to the cost function

$$\sum_{i=0,j=0,i\neq j}^{n} \|\mathbf{K}_i - {}^{i}\mathbf{T}_j\mathbf{K}_j\|_{\text{frob}}, \tag{4.10}$$

where $\mathbf{K}_i$ and $\mathbf{K}_j$ correspond to the position of the keyframes i and j, respectively, and ${}^{i}\mathbf{T}_j$ is the relative position from keyframe j to keyframe i.

The second method implemented, transformation averaging, corresponds to the distribution of the average loop error over the keyframes. Based on the computed correct position for the current keyframe and its position with accumulated error, the loop error is estimated. From this error, the translation error and the rotation error are obtained separately and divided by the total number of keyframes. The obtained value is assumed to be the average accumulated error between each keyframe and is therefore distributed over all the keyframes.

4.3.4 Full Map Optimization

To complete the loop closure, all keyframes and map points are optimized. This is done with a BA similar to the one in Section 4.2.3, the difference being that all keyframes and map points are used instead of just the current keyframe and those connected to it with co-visibility connections and their 3D points.

Chapter 5

Experiments and Results

This chapter aim to highlight the experiments and the results obtained throughout the implementation of the pipeline. The results are divided into three sections: Tracking, Local Mapping and Loop Closure. The first section will present results of using different feature types and different visual odometry estimation methods, using only the Tracking module. The second section compares the Tracking results before integration of the Local Mapping module with the results after integration. Finally, the third section compares the results obtained with the Tracking module only, with Tracking and Local Mapping modules, and with Tracking, Local Mapping and Loop Closure modules.

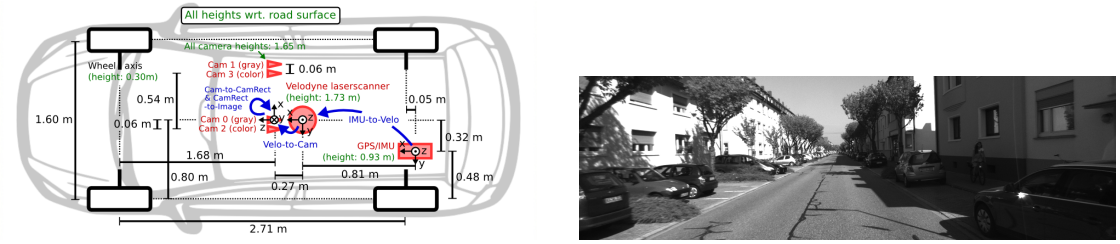
The datasets used to test the pipeline belong to two different environments using cameras with different models. The first type of data was acquired for the project and belongs to a retail store environment, which is the main environment in which the system will operate. The setup of the robot used for the project is shown in Figure 5.1(a). The camera used is the Intel® RealSense™ T265 fisheye stereo camera shown in Figure 5.1(b), which has an image capture rate of 30 FPS. Figure 5.1(c) shows a typical image of this type of environment captured with the described setup.

The second type of data is data from the KITTI dataset ([18]), which has several outdoor sequences obtained from a moving vehicle that has two pinhole cameras Point Grey Flea 2 (FL2-14S3M-C) arranged in parallel, forming a stereo pair. The setup of the car together with the cameras is shown in Figure 5.2(a), where the two cameras used are Cam 0 and Cam 1. The cameras capture new images at a rate of 10 FPS. An example image of sequence 00 from this dataset is shown in Figure 5.2(b).

Although one of the main objectives is to have a system that works in the indoor

(a) (b) (c)

Figure 5.1: Store environment datasets description: (a) Project's robot setup; (b) Stereo camera Intel® RealSense™ T265. Source: https://www.intelrealsense.com.; (c) Example image of the store environment.



(a) (b)

Figure 5.2: KITTI dataset description: (a) Vehicle sensor setup; (b) Example image of KITTI sequence 00. Source: http://www.cvlibs.net/datasets/kitti.

environment of a store, the KITTI dataset was chosen because it has ground-truth information that is useful for evaluating the error in the results obtained. The store datasets, despite being from the environment in which the system will work, have no ground-truth information available. Furthermore, one benefit of using two datasets from different environments and that have images from two different camera models is that it allows us to analyze the behavior of the different types of features and methods in each situation, which is one of the goals of the thesis.

Two datasets from the store environment and two datasets from KITTI are used. From the store environment, the two datasets will be referred to as STORE1 and STORE2 (example images of the STORE1 sequence are shown in Appendix A). From

Dataset	Distance [m]
STORE1	21.91
STORE2	34.23
KITTI00	217.06
KITTI07	694.70

Table 5.1: Distance traveled in each dataset.

the KITTI datasets the sequences 00 (only the first 300 images) and 07 are used, which will be referred to as KITTI00 and KITTI07. Table 5.1 shows the distance traveled in each of the datasets.

For error metrics, two were used, one for rotation errors and one for translation errors. The metrics are as follows:

$$e_R(\mathbf{R}) = \text{acos}\left(\frac{\text{trace}(\mathbf{R}^{-1}\mathbf{R}_{\text{GT}}) - 1}{2}\right) \tag{5.1}$$

$$e_t(\mathbf{t}) = \|\mathbf{t} - \mathbf{t}_{\text{GT}}\|^2, \tag{5.2}$$

where $\mathbf{R}_{\text{GT}}$ is the ground-truth rotation and $\mathbf{t}_{\text{GT}}$ is the ground-truth translation.

Using (5.1) and (5.2) both relative and absolute errors were estimated. The relative errors are computed between consecutive frames and express the error obtained in the estimation of the movement between consecutive frames. The absolute error is computed with the last frame of the sequence and expresses the total error of the trajectory.

5.1 Tracking

In this section the results obtained for the Tracking module alone will be shown and discussed. Being tracking only, the trajectories are just related to the quality of odometry estimation.

The results to be shown belong to the datasets STORE1 and KITTI00. For each dataset the three types of implemented features (SIFT, AKAZE, and ORB) were tested, as well as each of the following methods of odometry estimation methods:

- Method EM-R: Essential Matrix for rotation and 1-Point RANSAC for translation;
- Method VP-R: Vanishing Points for rotation and 1-Point RANSAC for translation;
- Method R-Um: 3-Point RANSAC and Umeyama;

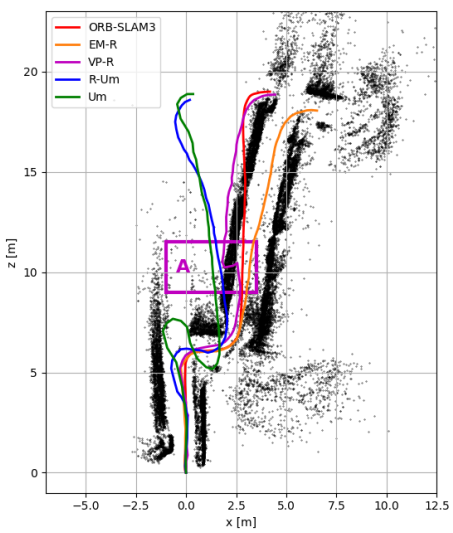
- Method Um: Umeyama.

EM-R and VP-R are the two methods proposed in Chapter 3.

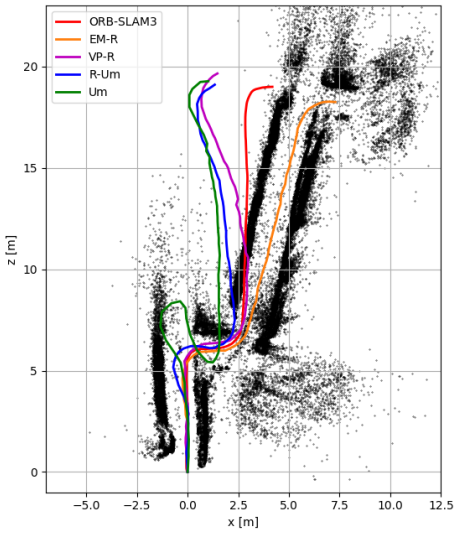
Figures 5.3(a) and 5.3(b) show the results obtained using various odometry estimation methods in the STORE1 dataset, using SIFT and AKAZE features, respectively. Since this is a dataset from a structured environment Method VP-R is used. As there is no ground truth available for this dataset, the results are only evaluated qualitatively and the result produced by ORB-SLAM3 for this dataset is given as reference. For this dataset the points on the map refer to the trajectory with the EM-R method.

Figures 5.4(a) and 5.4(b) also show the results obtained using various odometry estimation methods, but for Dataset KITTI00 using SIFT and AKAZE features, respectively. For this dataset the Method VP-R for odometry estimation is not used since it requires a structured environment, which is not the case for this outdoor dataset. The points on the map refer to the trajectory with Method EM-R. Since ground truth information is available, the relative and absolute errors of rotation and translation were computed. The relative error indicates the amount of error that is inserted in the map at each odometry estimation, and the absolute error indicates the total error between the ground truth and the method used at the end of the trajectory. The results obtained are presented in Table 5.2 and show both the mean value of the error (μ) and the standard deviation (σ). Figure 5.4(c) shows the results of using ORB features on this dataset with the EM-R method. As the trajectory results did not match the ground truth at all and had high errors from the beginning of the trajectory, this type of feature was no longer used.

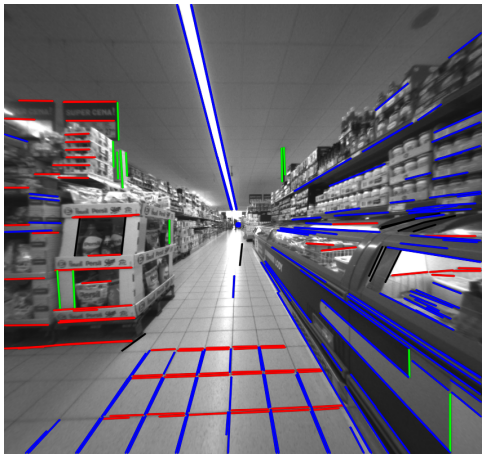
Analyzing the results obtained for the KITTI00 dataset using SIFT features, it can be seen that the results using method Um have the largest amount of error, and that by using an outlier removal method like RANSAC (method R-Um) it is possible to reduce some of this error and improve the results. However this result still has a substantial error in the trajectory. The proposed method EM-R is thus the one that produces the best result among the three methods for SIFT features. For AKAZE features there is also an improvement when introducing outlier removal by switching from the Um to the R-Um method. However, for these features, the proposed EM-R method is not the best among the three methods, coming second. One reason for this has to do with the amount of keypoints detected when using each feature type, where using SIFT produces more



(a) STORE1 - SIFT features



(b) STORE1 - AKAZE features

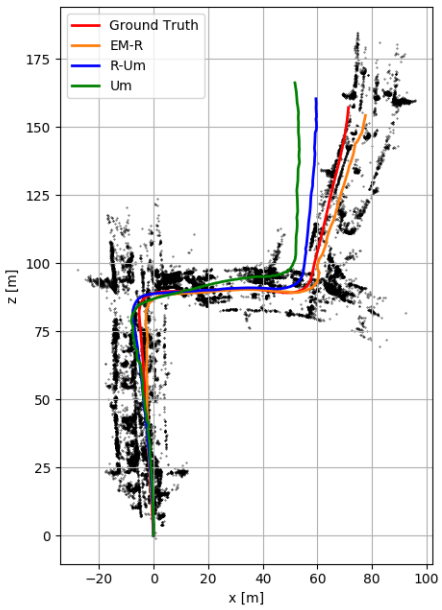


(c) STORE1 - Vanishing Points

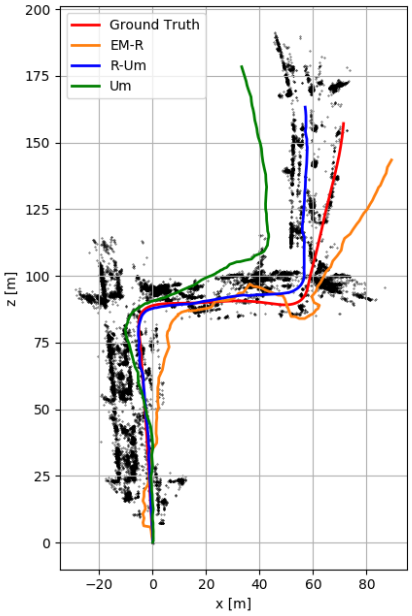
Figure 5.3: Tracking module results on STORE1 dataset: (a) Using SIFT features; (b) Using AKAZE features; (c) Detected lines and estimated vanishing points using VP-R method.

keypoints. In this dataset, it also occurs that in the right side of the images there are few points, which is noticeable when looking at the 3D points, which inevitably causes errors in odometry estimation. Overall, for this dataset, the method that gave the best results was EM-R using SIFT features.

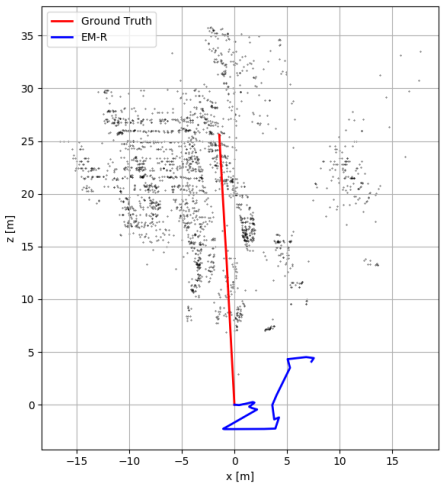
The results obtained with the STORE1 dataset are somewhat in line with what



(a) KITTI00 - SIFT features



(b) KITTI00 - AKAZE features



(c) KITTI00 - ORB features

Figure 5.4: Tracking module results on KITTI00 dataset: (a) Using SIFT features; (b) Using AKAZE features; (c) Using ORB features.

was observed for the KITTI00 dataset. In both feature types, an improvement is seen when switching from the Um to the R-Um method, and the proposed EM-R method obtains equal or slightly better results than the R-UM. This dataset introduces the

Features	**Method**	**Relative Error**				**Absolute Error**	
		Rot [rad]		**Trans [m]**		**Rot [rad]**	**Trans [m]**
		μ	σ	μ	σ	μ	σ
SIFT	EM-R	0.0065	0.0081	0.099	0.14	**0.12**	**6.85**
	R-Um	**0.0032**	**0.0034**	**0.069**	**0.090**	0.19	12.89
	Um	0.0059	0.0054	0.15	0.19	0.29	22.15
AKAZE	EM-R	0.028	0.038	0.45	0.78	0.26	22.61
	R-Um	**0.0060**	**0.0063**	**0.14**	**0.20**	**0.21**	**16.03**
	Um	0.0096	0.0078	0.24	0.29	0.47	43.76

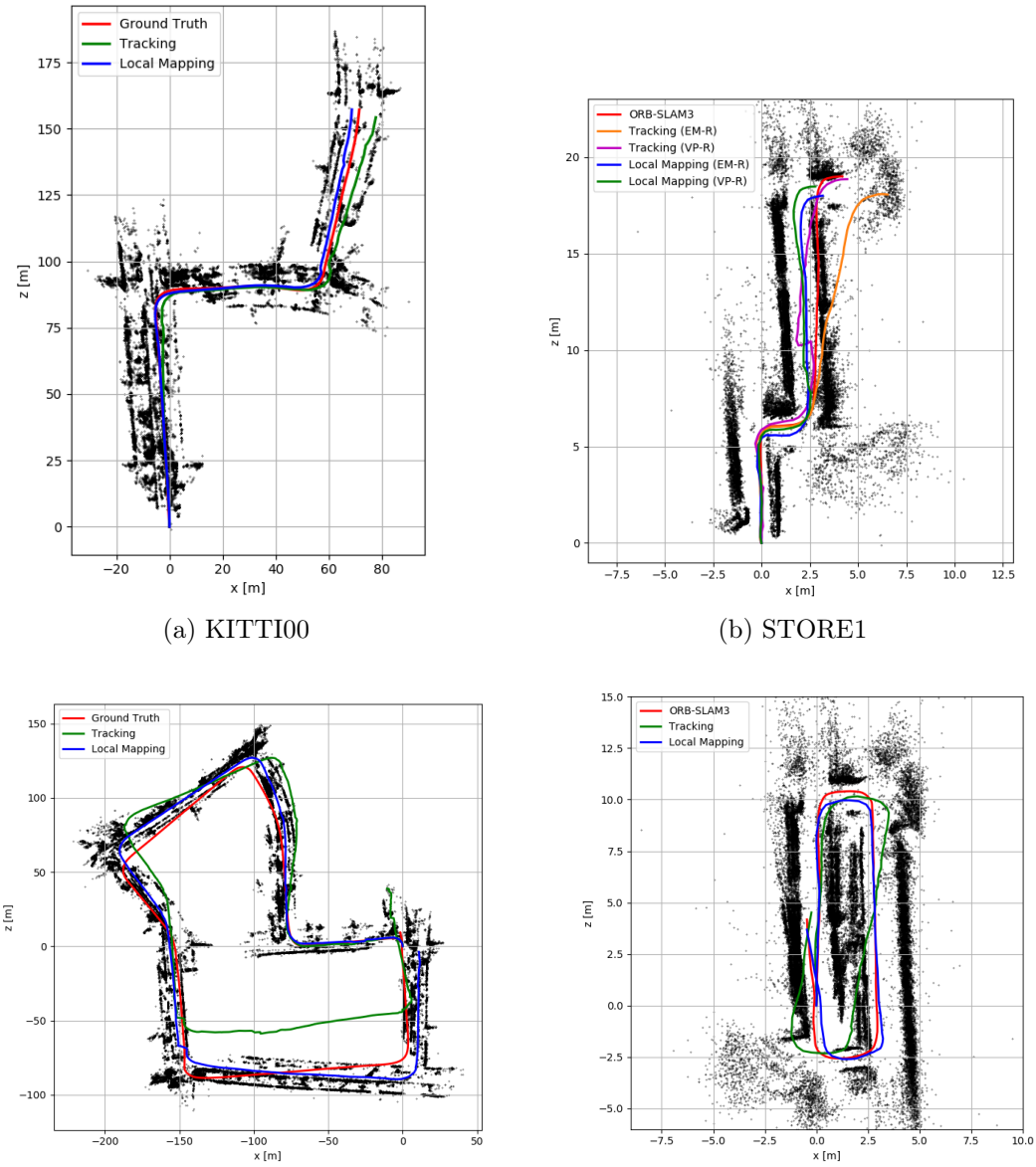
Table 5.2: Rotation and translation errors obtained on the KITTI00 dataset, using only the Tracking module.

VP-R method that was proposed specifically for the structured type environment of this dataset. Figure 5.3(c) shows the detected lines and the estimated vanishing point in the direction of camera movement (the remaining two vanishing points are outside the image). The results of this method are not, however, vastly superior to that of the EM-R and R-Um methods. This can be due to two factors. The first is that the method used for vanishing points estimation is for pinhole camera images, so it is necessary to remove distortion from the fisheye images of this dataset, which introduces motion blur. Another factor that was found is that in some parts of the store environment the vanishing points were not estimated correctly for several consecutive frames, which lead to incorrect estimates. This causes errors in the trajectory, as is the case of zone A marked in Figure 5.3(a).

5.2 Local Mapping

This section will take the best results from the previous section and add the Local Mapping module. The goal is to analyze if there are improvements in the trajectory by locally optimizing the positions of the keyframes and map points, which as seen in Section 4.2, has the objective of reducing and correcting odometry estimation errors. Therefore, only SIFT features and the methods EM-R and VP-R defined in the previous section will be used.

Figures 5.5(a), 5.5(b), 5.5(c), and 5.5(d) show the results obtained only for the Tracking module and for the Tracking module followed by Local Mapping for the four



(a) KITTI00 (b) STORE1

(c) KITTI07 (d) STORE2

Figure 5.5: Local Mapping module results: (a) Dataset KITTI00; (b) Dataset STORE1; (c) Dataset KITTI07; (d) Dataset STORE2.

datasets used. In all of them, the map points refer to the trajectory obtained with the local mapping module included (for Figure 5.5(b) with Local Mapping and the EM-R method). Table 5.3 shows the results obtained for the relative and absolute error values for the KITTI00 and KITTI07 datasets, for both configurations (Tracking with

Dataset	**Setup**	**Relative Error**				**Absolute Error**	
		Rot [rad]		**Trans [m]**		**Rot [rad]**	**Trans [m]**
		μ	σ	μ	σ		
KITTI00	Tracking	0.0065	0.0081	0.099	0.14	0.12	6.85
	Local Mapping	**0.0014**	**0.00095**	**0.030**	**0.041**	**0.045**	**3.26**
KITTI07	Tracking	0.012	0.019	0.17	0.29	0.27	33.95
	Local Mapping	**0.0022**	**0.0064**	**0.067**	**0.19**	**0.21**	**18.23**

Table 5.3: Rotation and translation errors obtained on the KITTI00 and KITTI07 datasets using both Tracking and Local Mapping modules.

or without Local Mapping).

Analyzing the estimated values for the trajectory error and the trajectories obtained from the KITTI00 and KITTI07 datasets, it can be seen that the Local Mapping module considerably improves the results. Not only does the error decrease, but also the trajectory is smoother, with no abrupt variations caused by bad odometry estimates.

The results obtained for the STORE1 and STORE2 datasets are also in line with the results obtained for the other datasets. In STORE1 when using the EM-R method the trajectory is very close to the one obtained with ORB-SLAM3. When using the VP-R method, although the abrupt variation in the trajectory is corrected, it still has a considerable variation with those obtained with the EM-R and ORB-SLAM3 methods. For the STORE2 dataset, only the EM-R method was used since it is the one that produces the best results and there is a close proximity to the ORB-SLAM3 trajectory. It is thus verified that the inclusion of the Local Mapping module improved the Tracking results.

5.3 Loop Closure

The last set of experiments aims to evaluate the two implemented types of methods for error distribution along the trajectory (keyframe optimization and transformation averaging) and also make a final analysis of the evolution of the results obtained along the pipeline with the implementation of each module.

Before conducting the experiments it was necessary to build a BoW vocabulary.

Setup	Relative Error				Absolute Error	
	Rot [rad]		Trans [m]		Rot [rad]	Trans [m]
	μ	σ	μ	σ		
Tracking	0.012	0.019	0.17	0.29	0.27	33.95
Local Mapping	**0.0022**	**0.0064**	**0.067**	**0.19**	0.21	**18.23**
Loop Closure (Transformation Averaging)	0.0031	0.016	0.081	0.30	0.19	23.76
Loop Closure (Keyframe Optimization)	0.0027	0.0099	0.075	0.22	**0.069**	21.07

Table 5.4: Rotation and translation errors obtained on the KITTI07 dataset using Tracking, Local Mapping, and Loop Closure modules.

The vocabulary is used for loop detection and serves as a reference for building words of the received images. The vocabulary used in the KITTI07 dataset was built from KITT00 and the vocabulary used in the STORE2 dataset was built from the dataset itself. Although the vocabularies created were quite specific to the datasets used, after adjustment of detection-related parameters, the detections were correct for both datasets and for other similar datasets.

Figures 5.6(a) and 5.6(c) show a comparison of the results obtained for the two implemented methods of error distribution along the trajectory for the STORE2 and KITTI07 datasets, respectively, and Table 5.4 shows the error values obtained with these methods for the KITTI07 dataset.

For the STORE2 dataset result (Figure 5.6(a)) the transformation averaging method was able to close the loop correctly. This result is due to the trajectory not having sharp variations, so it is correct to assume for this case that the accumulated error is constant along the keyframes. For the keyframe optimization method, keyframes close to the loop are corrected correctly, however keyframes far from the loop are adjusted incorrectly. This may be due to optimization parameters but may also be related to point tracking and to the connection of keyframes in the co-visibility graph, where points may be lost along the frames and therefore slightly distant keyframes have few points in common.

For the KITTI07 dataset result (Figure 5.6(c)), it can be seen that neither method was able to correctly correct the loop error. In the case of the transformation averaging method, this is mainly due to the error accumulated along the trajectory not being approximately constant, which is seen in a region of the image where there is a sudden

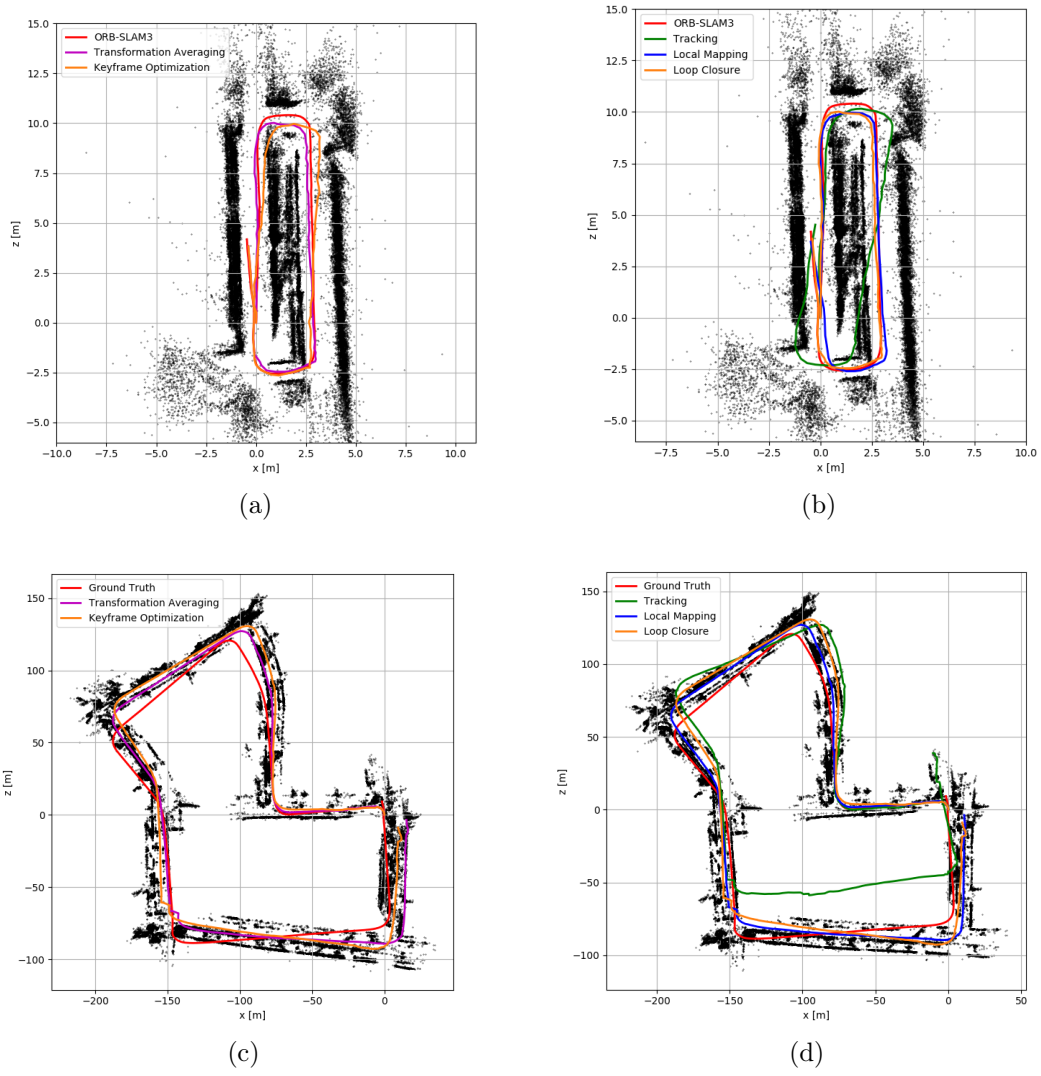


Figure 5.6: Loop Closure module results: (a) Comparison of loop error distribution methods on the STORE2 dataset; (b) Tracking, Local Mapping and Loop Closure results on the STORE2 dataset; (c) Comparison of loop error distribution methods on the KITTI07 dataset; (d) Tracking, Local Mapping and Loop Closure results on the KITTI07 dataset.

deviation in the trajectory. However, it may also be related to the estimation of the correct position of the current keyframe in the loop. For the keyframe optimization method there was also no correct loop correction. Although the absolute rotation error improved over Local Mapping and the transformation averaging method, the translation

only improved over transformation averaging and the error was higher than for Local Mapping. The keyframe optimization method is thus only able to correct the rotation of the trajectory. This may also be related to the estimation of the correct position of the current keyframe, which may be incorrect because there are too few established points between the keyframes in the loop, so the estimation does not produce a good result, or because the accumulated error is too large to be corrected in this way.

Figures 5.6(b) and 5.6(d) show a comparison of the results obtained from the Tracking module implementation only, to the Local Mapping module implementation, and finally to the Loop Closure implementation for the STORE2 and KITTI07 datasets, respectively. Table 5.4 also shows the values obtained for the Tracking and Local Mapping errors, in addition to the Loop Closure errors, for the KITTI07 dataset. A clear evolution is observed in both datasets from the results obtained with only Tracking to the Local Mapping. From Local Mapping to Loop Closure, this evolution is notable only in the STORE2 dataset, since the loop was not closed correctly in the KITTI07 dataset.

Chapter 6

Conclusions

The developed SfM system is able, based on stereo images from pinhole or fisheye cameras, to estimate the trajectory traveled by the camera while creating a map of the traveled region. In the tracking module, several feature types and odometry estimation methods were tested that can be easily chosen based on a configuration file. Hence, regarding the goal of developing a modular pipeline that allows different types of configurations, it can be concluded that the goal has been achieved. Concerning the exploration of the structure of retail stores environments, the developed method showed potential, however, because it does not work for fisheye images and because in certain regions it cannot correctly estimate vanishing points, it was not able to produce the desired results. As an alternative, another method was proposed, which, despite not exploiting the store structure, was able to produce good results. However, regarding the goal of exploring the geometrical structure of the store, the goal was incomplete. The last objective of creating an SfM pipeline based on a known SLAM system was mostly fulfilled, with some work still missing on the loop closure module since there was no correct loop closure in all the datasets tested, especially the larger ones.

Future work can be done throughout the pipeline, from improving or changing certain methods to testing parameters related to optimization, feature extraction, feature matching, and more. Regarding the Tracking module, one aspect that could improve the results is the integration of information from other sensors, such as IMU or encoders. This type of sensor gives information about the robot's motion which can be used as the initial estimate of the visual odometry estimate or as the motion value when the visual odometry does not produce a correct estimate (*e.g.* by detecting few keypoints).

Another improvement to be made involves all modules and consists of having better tracking of keypoints over several frames. Currently, this tracking is only done at the level of keyframes, which causes some detected points to be lost. By having a better tracking of the keypoints there will be more map points to be seen by several keyframes which consequently creates more connections between keyframes and improve the bundle adjustment optimization process. Overall, the whole system implemented should be optimized so that in the future it runs in real-time and a localization module should also be implemented to transition from the implemented SfM system to a SLAM system as intended for the project in which this work is included.

Bibliography

[1] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. In *European Conf. Computer Vision (ECCV)*, pages 29–42, 2010. 18

[2] D. Aiger, N. J. Mitra, and D. Cohen-Or. 4-points congruent sets for robust pairwise surface registration. In *ACM SIGGRAPH*, pages 1–10, 2008. 15

[3] P. F. Alcantarilla, A. Bartoli, and A. J. Davison. Kaze features. In *European Conf. Computer Vision (ECCV)*, pages 214–227, 2012. 12, 13

[4] P. F. Alcantarilla and T. Solutions. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, 34(7):1281–1298, 2011. 12, 13

[5] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European Conf. Computer Vision (ECCV)*, pages 404–417, 2006. 12, 13

[6] P. J. Besl and N. D. McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606, 1992. 10, 11

[7] P. Biber and W. Straßer. The normal distributions transform: A new approach to laser scan matching. In *IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, volume 3, pages 2743–2748, 2003. 10, 11

[8] M. Bosse and R. Zlot. Continuous 3d scan-matching with a spinning 2d laser. In *IEEE Int'l Conf. Robotics and Automation (ICRA)*, pages 4312–4319, 2009. 9

[9] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent

elementary features. In *European Conf. Computer Vision (ECCV)*, pages 778–792, 2010. 13

[10] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Trans. Robotics (T-RO)*, 2021. xi, 20, 21

[11] A. Chatterjee and V. Madhav Govindu. Efficient and robust large-scale rotation averaging. In *European Conf. Computer Vision (ECCV)*, pages 521–528, 2013. 19

[12] Y. Chen and G. G. Medioni. Object modeling by registration of multiple range images. *Image and Vision Computing (IVC)*, 10(3):145–155, 1992. 10, 11

[13] Y. Cho, G. Kim, and A. Kim. Deeplo: Geometry-aware deep lidar odometry. *arXiv preprint arXiv:1902.10562*, 2019. 16

[14] C. Choy, J. Park, and V. Koltun. Fully convolutional geometric features. In *IEEE Int'l Conf. Computer Vision (ICCV)*, pages 8958–8966, 2019. 16, 17

[15] I. Cvišić, J. Ćesić, I. Marković, and I. Petrović. Soft-slam: Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles. *Journal of Field Robotics*, 35(4):578–595, 2018. 9

[16] J.-E. Deschaud. Imls-slam: scan-to-model matching based on 3d data. In *IEEE Int'l Conf. Robotics and Automation (ICRA)*, pages 2480–2485, 2018. 9

[17] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 13

[18] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2012. 5, 47

[19] V. M. Govindu. Lie-algebraic averaging for globally consistent motion estimation. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–I, 2004. 19

[20] C. G. Harris, M. Stephens, et al. A combined corner and edge detector. In *Alvey Vision Conference*, volume 15, pages 10–5244, 1988. 12, 13

[21] R. Hartley, J. Trumpf, Y. Dai, and H. Li. Rotation averaging. *Int'l J. Computer Vision (IJCV)*, 103(3):267–305, 2013. 19

[22] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004. 24

[23] A. Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, pages 3946–3952, 2008. 9

[24] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, pages 2100–2106, 2013. xi, 9, 19

[25] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for rgb-d cameras. In *IEEE Int'l Conf. Robotics and Automation (ICRA)*, pages 3748–3754, 2013. 9

[26] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g 2 o: A general framework for graph optimization. In *IEEE Int'l Conf. Robotics and Automation (ICRA)*, pages 3607–3613, 2011. 29

[27] Q. Li, S. Chen, C. Wang, X. Li, C. Wen, M. Cheng, and J. Li. Lo-net: Deep real-time lidar odometry. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 8473–8482, 2019. 16

[28] Z. Li and N. Wang. Dmlo: Deep matching lidar odometry. In *IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, pages 6010–6017, 2020. 9, 16, 17

[29] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. 12

[30] X. Lu, J. Yaoy, H. Li, Y. Liu, and X. Zhang. 2-line exhaustive searching for real-time vanishing point estimation in manhattan world. In *IEEE Winter Conf. on Applications of Computer Vision (WACV)*, pages 345–353, 2017. xi, 25

[31] A. Mateus, S. Ramalingam, and P. Miraldo. Minimal solvers for 3d scan alignment with pairs of intersecting lines. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 7234–7244, 2020. 15

[32] N. Mellado, D. Aiger, and N. J. Mitra. Super 4pcs fast global pointcloud registration via smart indexing. In *Computer Graphics Forum*, volume 33, pages 205–215, 2014. 15

[33] R. Muñoz-Salinas. Dbow3. *URL: https://github. com/rmsalinas/DBow3*, 2017. 29

[34] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Trans. Robotics (T-RO)*, 31(5):1147–1163, 2015. 20

[35] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Trans. Robotics (T-RO)*, 33(5):1255–1262, 2017. iii, v, xi, 5, 9, 20, 21

[36] F. Neuhaus, T. Koß, R. Kohnen, and D. Paulus. Mc2slam: Real-time inertial lidar odometry using two-scan motion compensation. In *German Conf. on Pattern Recognition*, pages 60–72, 2018. 9, 10

[37] D. Nister. An efficient solution to the five-point relative pose problem. *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, 26(6):756–770, 2004. 24

[38] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–I, 2004. 9

[39] C. Olsson, F. Kahl, and M. Oskarsson. Branch-and-bound methods for euclidean registration problems. *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, 31(5):783–794, 2008. 10

[40] G. D. Pais, S. Ramalingam, V. M. Govindu, J. C. Nascimento, R. Chellappa, and P. Miraldo. 3dregnet: A deep neural network for 3d point registration. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 7193–7203, 2020. 16, 17

[41] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, July 2017. 17

[42] R. Raguram, J.-M. Frahm, and M. Pollefeys. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In *European Conf. Computer Vision (ECCV)*, pages 500–513, 2008. 14

[43] S. Ranade, X. Yu, S. Kakkar, P. Miraldo, and S. Ramalingam. Can generalised relative pose estimation solve sparse 3d registration? *arXiv preprint arXiv:1906.05888*, 2019. xi, 15, 16

[44] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conf. Computer Vision (ECCV)*, pages 430–443, 2006. 13

[45] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *IEEE Int'l Conf. Computer Vision (ICCV)*, pages 2564–2571, 2011. 12, 13

[46] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *IEEE Int'l Conf. Robotics and Automation (ICRA)*, pages 3212–3217, 2009. 13, 15

[47] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz. Learning informative point classes for the acquisition of object model maps. In *IEEE Int'l Conf. on Control, Automation, Robotics and Vision*, pages 643–650, 2008. 13

[48] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz. Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems (RAS)*, 56(11):927–941, 2008. 13

[49] P. H. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966. 13, 14

[50] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Robotics: Science and Systems (RSS)*, volume 2, page 435, 2009. 10, 11

[51] J. Serafin and G. Grisetti. Nicp: Dense normal based point cloud registration. In *IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, pages 742–749, 2015. 10

[52] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *International Workshop on Vision Algorithms*, pages 298–372, 1999. 18

[53] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, pages 376–380, 1991. 13, 14

[54] Y. Wang and J. M. Solomon. Deep closest point: Learning representations for point cloud registration. In *IEEE Int'l Conf. Computer Vision (ICCV)*, pages 3523–3532, 2019. 16, 17

[55] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Trans. Graphics*, 38(5):1–12, 2019. 17

[56] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald. Robust real-time visual odometry for dense rgb-d mapping. In *IEEE Int'l Conf. Robotics and Automation (ICRA)*, pages 5724–5731, 2013. 9

[57] J. Yang, H. Li, and Y. Jia. Go-icp: Solving 3d registration efficiently and globally optimally. In *IEEE Int'l Conf. Computer Vision (ICCV)*, December 2013. 10, 12

[58] J. Zhang and S. Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems (RSS)*, volume 2, 2014. xi, 8, 9

[59] J. Zhang and S. Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *IEEE Int'l Conf. Robotics and Automation (ICRA)*, pages 2174–2181, 2015. 9

[60] Q.-Y. Zhou, J. Park, and V. Koltun. Fast global registration. In *European Conf. Computer Vision (ECCV)*, pages 766–782, 2016. 14

Appendix A

Example of a retail store dataset

The datasets of the store environments were obtained using the robot whose setup was shown in Figure 5.1(a), which uses the fisheye stereo camera Intel® RealSense™ T265 shown in Figure 5.1(b). To better understand the results obtained from the STORE1 and STORE2 datasets presented in Chapter 5, Figure A.1 shows parts of the image sequence from the STORE1 dataset.

Figure A.1: Example images from STORE1 dataset.