Universidade Estadual de Campinas
Instituto de Computação

Otávio Basso Gomes

Deep Convolutional Features for Sparse and Dense Registration in RGB-D SLAM

Descritores Convolucionais Profundos para Registro Esparço e Denso em SLAM RGB-D

CAMPINAS
2021

Otávio Basso Gomes

Deep Convolutional Features for Sparse and Dense Registration in RGB-D SLAM

Descritores Convolucionais Profundos para Registro Esparço e Denso em SLAM RGB-D

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/ Orientador: Prof. Dr. Gerberth Adín Ramírez Rivera

Este exemplar corresponde à versão final da Dissertação defendida por Otávio Basso Gomes e orientada pelo Prof. Dr. Gerberth Adín Ramírez Rivera.

CAMPINAS
2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Gomes, Otávio Basso, 1986-
G585d Deep convolutional features for sparse and dense registration in RGB-D SLAM / Otávio Basso Gomes. – Campinas, SP : [s.n.], 2021.

Orientador: Gerberth Adín Ramírez Rivera.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Reconstrução tridimensional. 2. Odometria visual. 3. Redes neurais convolucionais. 4. Aprendizado de máquina. I. Ramírez Rivera, Gerberth Adín, 1986-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Descritores convolucionais profundos para registro esparço e denso em SLAM RGB-D
Palavras-chave em inglês:
Three-dimensional reconstruction
Visual odometry
Convolutional neural networks
Machine learning
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Geberth Adín Ramírez Rivera
Valdir Grassi Junior
Esther Luna Colombini
Data de defesa: 04-10-2021
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0003-2434-4839
- Currículo Lattes do autor: http://lattes.cnpq.br/5289457498213717

Universidade Estadual de Campinas
Instituto de Computação

Otávio Basso Gomes

Deep Convolutional Features for Sparse and Dense Registration in RGB-D SLAM

Descritores Convolucionais Profundos para Registro Esparço e Denso em SLAM RGB-D

Banca Examinadora:

- Prof. Dr. Gerberth Adín Ramírez Rivera
  IC/UNICAMP
- Prof. Dr. Valdir Grassi Junior
  EESC/USP
- Profa. Dra. Esther Luna Colombini
  IC/UNICAMP

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 04 de outubro de 2021

Acknowledgements

I could not start this text without pondering on how privileged I was and still am to research a topic that fascinates me in such an excellence center. Especially when living in Brazil, a country with huge social inequality in which a large percentage of our youth do not have the most basic support to flourish their talents. Yet, I was among the ones who could take the courses and study in one of our best institutions.

For that, first, I must acknowledge my parents, Rosaura Gomes and Marne Gomes, for providing all the support a person can have, from putting food on the table to emotionally and financially supporting my education. That was undoubtedly the most significant privilege in my life.

Secondly, I thank my girlfriend's parents, Marli Inês Araujo de Paula and Dorvalino Coutinho de Paula, for welcoming me into their family when mine is far from here.

Thirdly, I thank my advisor, professor Adin, who taught me how to research and write better.

I must also thank the University of Campinas for accepting me in its courses and being an important institution in developing the technology industry in our country.

Another proof of how privileged I am is the outstanding support that I have from my employer, the Eldorado Research Institute, mainly represented in the persona of Edgar Gadbem. This organization believed in my capacity and provided me with time and resources to accomplish my master's.

This dissertation is also dedicated to my girlfriend and partner in life, Larissa Araujo Coutinho de Paula, for her companionship in moments of both joy and struggle.

To my goddaughter Isabela Gomes, soon your uncle will have more time to talk to you.

Finally, I want to thank my friends, Rogério Zafalão, Adam Mendes, Augusto Valente, Bruno Mazzotti, Edgar Gadbem, Eduardo Andrade, Eduardo Fernandes, Karina Bogdan, Leonardo Domingues, Luis Pires, Marcos Martins, Matheus Alves, Melissa Carvalho, Paulo Menin, Olavo Giraldi, and Thiago Silva.

Resumo

Uma solução eficiente para corrigir desvios na estimação da trajetória da câmera em sistemas de Mapeamento e Localização Simultânea Visual (Visual Simultaneous Localization and Mapping - Visual SLAM) é usar a Otimização do Grafo de Poses (Pose Graph Optimization - PGO) com submapas da cena gerados a partir de subconjuntos dos quadros do vídeo de entrada. Embora o PGO sozinho consiga oferecer alinhamento global consistente, ainda é necessário um estágio extra para alinhar os submapas de maneira pareada usando todas as suas superfícies tridimensionais, a fim de obter uma reconstrução ainda mais fina. No entanto, ainda é desafiador construir grafos de pose com fechamentos de loops adequados e realizar os alinhamentos pareados finais quando a cena, e conseqüentemente os submapas, apresentam alterações fotométricas ao longo dos quadros do vídeo. Para tentar resolver esse problema, adicionamos descritores densos aprendidos por uma Rede Neural Convolucional (Convolutional Neural Network - CNN) aos elementos das superfícies dos submapas para melhorar a robustez na detecção de fechamento de loops e no alinhamento pareado. Recentes trabalhos apontam que os descritores aprendidos capturam contexto para além de informações estruturais ou fotométricas dos descritores clássicos anteriormente usados, e assim, beneficiando a correspondência entre imagens em situações de variação de luminosidade. Outra vantagem de aplicar descritores densos de CNNs no Visual SLAM é uma potencial integração da localização e informações semânticas em sistemas robóticos que também realizam segmentação de objetos do ambiente. Os experimentos realizados demonstram as possíveis vantagens desses descritores, produzindo resultados equivalentes às técnicas padrões para Visual SLAM. Além disso, desenvolvemos um pacote de software para Visual SLAM para sensores RGB-D que inclui os descritores aprendidos de imagens como quaisquer outras propriedades das superfícies, como suas cores ou normais.

Abstract

An efficient solution to correct the tracking-drift problem in camera trajectory estimation in Visual Simultaneous Localization and Mapping (Visual SLAM) systems is to use Pose Graph Optimization (PGO) with generated scene submaps from subsets of the input video frames. While PGO alone can provide consistent global alignment, an extra stage is still needed to align submaps evenly using all of their three-dimensional surfaces for an even finer reconstruction. However, it is still challenging to construct pose graphs with proper loop closures and perform the final paired alignments when the scene, and hence the submaps, exhibit photometric changes across video frames. To try to solve this problem, we added dense descriptors learned by a Convolutional Neural Network (CNN) to submap surface elements to improve the robustness in detecting loop closure and paired alignment. Recent works show that learned descriptors capture context beyond structural or photometric information of the previosly used classical descriptors, thus improving the correspondence between images in situations of varying brightness Another advantage of applying dense CNN descriptors in Visual SLAM is a potential integration of location and semantic information in robotic systems that also perform segmentation of objects in the environment. The experiments carried out demonstrate the possible advantages of these descriptors, producing results equivalent to the standard techniques for Visual SLAM. In addition, we have developed a Visual SLAM software package for RGB-D sensors that includes the learned image descriptors as any other surface properties, such as their colors or normals.

List of Figures

1.1 Method overview . . . 13
1.2 A Synthetic Scene for Illustrative Purposes . . . 15
1.3 Introduction to RGB-D SLAM . . . 16
1.4 Geometric Representations of the Map . . . 17
1.5 Translation, Rotation and Scale Invariance . . . 18

2.1 Rotation Representation by a $SO(3)$ Matrix . . . 25
2.2 Camera Pose Representation by a $SE(3)$ Matrix . . . 27
2.3 The Pinhole Camera Model . . . 34
2.4 Huber Estimator . . . 39
2.5 Geman-McClure Estimator . . . 40
2.6 A Naive SLAM pipeline . . . 46
2.7 The Track Drifting Problem . . . 47
2.8 Keyframes from a Trajectory . . . 48
2.9 Sample Submaps . . . 49
2.10 The Bundle Adjustment's Reprojection Error . . . 55
2.11 Pose Graph Optimization . . . 57
2.12 Choi et al. [17] pipeline . . . 60
2.13 Kähler et al. [45] pipeline . . . 61
2.14 A Convolutional Filter . . . 64
2.15 The Contrastive Loss . . . 67

3.1 Deep Surfel SLAM Architecture . . . 70
3.2 Bilateral Filtering in 1D . . . 72
3.3 Pose Graph Visualization on our System . . . 82
3.4 The Residual ResNet Convolution Block . . . 87

4.1 Two Submaps with Different Lighting Conditions . . . 92
4.2 PGO's stage in the pipeline . . . 95
4.3 PGO's stage in the pipeline . . . 96
4.4 Map of the scene Lobby from IL-RGBD . . . 97
4.5 Map of the scene Bedroom from IL-RGBD . . . 98
4.6 Map of the scene 021 from SceneNN . . . 99
4.7 Map of the scene 045 from SceneNN . . . 100
4.8 Map of the scene 081 from SceneNN . . . 101
4.9 Map of the scene 087 from SceneNN . . . 102

List of Tables

3.1 The Dilated ResNet32 Architecture 86

4.1 Alignment Metrics for Controlled Submap Registration 92
4.2 Precision and Recall of Loop Closure Detection 93
4.3 Alignment Metrics of the Edges found in Loop Closure 94
4.4 Alignment Metrics of Trajectory after Pose Graph Optimization 94
4.5 Alignment Metrics after Pairwise Submap Registration 97

Contents

1 Introduction **12**
- 1.1 Visual Simultaneous Localization and Mapping . . . 14
- 1.2 Descriptor Learning . . . 18
- 1.3 Contributions . . . 19
- 1.4 Dissertation Structure . . . 20

2 Background **22**
- 2.1 Math Notations and Definitions . . . 23
- 2.2 Motion Representation . . . 25
  - 2.2.1 Lie Groups and Algebras for SLAM . . . 27
- 2.3 Pinhole Camera Model . . . 33
- 2.4 Weighted Non-linear Least Squares Problems . . . 35
  - 2.4.1 Robust Estimation . . . 38
- 2.5 Point Cloud Registration . . . 40
  - 2.5.1 The Iterative Closest Points Algorithms . . . 41
  - 2.5.2 Sparse methods . . . 43
  - 2.5.3 Fast Global Registration . . . 45
- 2.6 Visual SLAM . . . 46
  - 2.6.1 Visual Odometry . . . 50
  - 2.6.2 Uncertainty Estimation . . . 51
  - 2.6.3 Bundle Adjustment Back-end . . . 54
  - 2.6.4 Pose Graph Optimization Back-end . . . 55
  - 2.6.5 Mapping on RGB-D SLAM . . . 58
  - 2.6.6 RGB-D SLAM . . . 59
- 2.7 Descriptor Learning . . . 62
  - 2.7.1 Handcraft Descriptors . . . 62
  - 2.7.2 Convolutional Neural Networks . . . 64
  - 2.7.3 Training . . . 67
  - 2.7.4 Datasets . . . 69

3 Methods **70**
- 3.1 Frame Preprocessing . . . 71
  - 3.1.1 Depth Image Filtering . . . 72
  - 3.1.2 Normals Computation . . . 73
- 3.2 Visual Odometry . . . 74
- 3.3 Surfel-based Mapping . . . 77
  - 3.3.1 Implementation Details . . . 81
- 3.4 Loop Closure Detection . . . 81

3.5 Pose Graph Optimization . . . 82
3.6 Pairwise Submap Registration . . . 82
3.7 Descriptor Learning . . . 85
3.7.1 CNN Architecture . . . 85
3.7.2 Self-supervised Dataset for Descriptor Learning . . . 86
3.7.3 Training . . . 88

4 Experiments **90**
4.1 Evaluation Metrics . . . 91
4.2 Training Settings . . . 91
4.3 Submap Registration under Controlled Scenario . . . 92
4.4 SLAM . . . 93
4.4.1 Loop Closure Detection . . . 93
4.4.2 Evaluation of Submap Registration on Complete SLAM . . . 95
4.4.3 Qualitative Results . . . 96

5 Conclusion **103**

Chapter 1

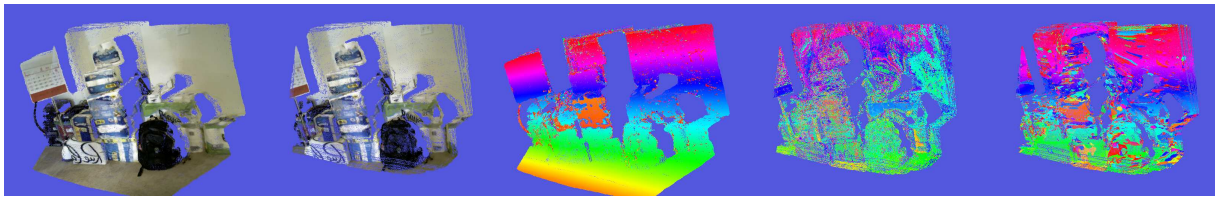
Introduction

Simultaneous Localization and Mapping (SLAM) is the problem of estimating the positions and orientations of a sensor such as a video camera or Light Detection and Ranging (LiDAR) equipment throughout a capture while also mapping the observed scene geometry. Understanding a sensor's trajectory and the map of the environment is critical to enable robots to navigate and avoid collisions or to interact with objects. Other areas that beneficiate from solving SLAM are Augmented Reality (AR) and Mixed Reality (MR), where the trajectory and mapping of a SLAM subsystem can replace the printed markers used previously to locate virtual objects in the scene in those applications.

One component across SLAM algorithms is associating different measurements of 2D pixels from the stream's frames, or even 3D points from the mapping, as representations of the same point in the environment. Associating measurements is essential because they are inputs for position and orientation estimation, conversion of 2D image coordinates into 3D ones, recognition of already seen places (or loop closure detection) and alignment of submaps of the 3D mapping being reconstructed. However, establishing those associations between environment measurements in either 2D or 3D can be prone to false or missing associations due to sensor noise, lighting conditions, or geometric transformation produced by the sensor's motion. Authors have proposed using advanced handcraft feature techniques [53, 66, 67] or learning-based methods [23, 68, 86]. Nevertheless, all those works focus on exploiting sparse feature associations, that is, working with a small set of predicted points to be good ones to track. Rather than handling sparse feature sets, we explore deep learning techniques of descriptor learning to extract dense image descriptors and incorporate them into a SLAM system. A dense image descriptor extraction is a process of obtaining a feature vector for every pixel that is more meaningful and easier to compare than its original intensity or color values from an image.

Using dense learned descriptors already provided good results for registering 3D human reconstructions [70], robot picking [29], stereo matching [85], and image alignment [69]. Because of those evidence, this dissertation's contribution is the verification of dense learned descriptors for SLAM by incorporating them as integral part of the mapping surface elements — or surfels. More specifically, we add a descriptor into every surfel of the map, and use them to improve tasks that are sensible to lighting conditions.

For experimenting with the surfel descriptors, we propose a SLAM system using two popular techniques in the SLAM community: submaps and pose graphs. Submaps refer



(a) (b) (c) (d) (e)

Figure 1.1: Our work investigates the use of dense features produced from Convolutional Neural Networks (CNNs) models for semantic segmentation for improving them since they can express context beyond photometric-based features, enabling more robust point associations In the following images, we show (a–b) two overlapping submaps built from multiple RGB-D frames fusion. (c) The first submap with its surfels assigned to a sequential RGB value. (d) Associations of the first submap's surfels w.r.t. the second one using by comparing their RGB similarity. (e) Associations as in (d) but using our model trained with $D = 16$. Notice the less amount of noise on some spots, like in the red or orange color tones. Our

to map the scene into smaller chunks that are later aligned altogether instead of mapping the whole scene into one big map. Some systems [17, 45] use this technique, especially for performance reasons, so the localization back-end does not need to consider every frame at once.

In the other part, pose graphs are an essential technique for SLAM in large-scale environments for correcting the sensor's course. It consists of treating the SLAM as a graph optimization problem. Each camera location is a node, and the edges are relative transformations between nodes that share the same view or scene geometry. By having pose nodes connection, algorithms can optimize the graph according to the confidence in the edges' measurements, so it estimates the trajectory with the highest probability of being the actual one. Hence, a fundamental step is to detect and add loop closures edges for adjusting the sequential trajectory of camera poses. As our first experiment with the deep features, we use them for detecting and estimating the transformation of loop closures between submaps with feature matching approach for finding keysurfels and aligning according to their correspondence. Once it creates the pose graph, we run a Pose Graph Optimization (PGO) routine for refining the trajectory.

Authors of similar works with submaps who also base their trajectory correction using PGO, may add a step after it to refine the alignment in a pairwise manner, since PGO itself may fail to provide tight alignment between submaps. For instance, the solution proposed by Choi et al. [17] performs fine alignment on every pair of submap along the trajectory using the Iterative Closest Point (ICP) algorithm after it runs PGO. The ICP can achieve better fine alignment because it considers all points of a submap. That is, it is a dense registration algorithm. In the algorithm of Park et al. [61], the authors include color information to improve the ICP registration. Inspired by their work, we propose ICP alignment that uses deep features for guiding the alignment consistency of two submaps after a PGO execution. The potential advantages of including deep descriptor for guiding the ICP alignment may be observed in Figure 1.1, which shows two submaps illustrating the matchings of their surfaces using RGB and deep features information. We can notice

that dense associations using deep features is more stable than with RGB information.

Our experiments show that the dense learned features can perform equivalent to the already established Fast Point Feature Histogram [67] (FPFH) and Scale-invariant Feature Transform [53] (SIFT) for loop closure detection. Whereas, for the submap registration, the dense CNN features performed better for registering submaps captured with different light conditions, where the previous method that uses the only color may fail with submaps produced during different lighting conditions.

Before continuing to our contributions, the following sections further introduce Visual SLAM and Descriptor Learning.

1.1 Visual Simultaneous Localization and Mapping

In more concrete terms, the objective of a SLAM system is, given a video stream with N frames, output N rigid transformations $P_0, P_1, \ldots, P_N$ containing the sensor's absolute position and rotation at every frame n; and a map from the scanned environment. A 3D rigid transformation is a motion that rotates and translates bodies preserving angles and distances. In other words, objects that undergo a rigid transform do not have their structure deformed. They represent the sensor poses because their translation component is interpreted as the position of the sensor in the 3D world, and their rotation part expresses the sensor's viewing orientation.

Figure 1.2 shows a high-quality synthetic 3D scene that will serve as our sample environment throughout this document in which a hypothetical sensor will move and map. Next, in Figure 1.3, it displays the steps of a SLAM output using the sample scene of Figure 1.2. Each step shows a stage of outputted sensor's pose and mapping of the environment. The process of estimating the camera poses with visual information is named Visual Odometry (VO). In some cases of robotics, autonomous navigation, or handheld device applications, the VO may be aided with other odometry sensors (e.g., wheel rotation measurements, inertial measurements) to complement the estimation.

The other part of the SLAM output is the mapping of the environment, and as show in Figure 1.3, every new frame adds more information to mapping. One of the first architectural considerations of any SLAM system is choosing how to represent the environment map which varies according to the application specification and available sensors. The map may be represented by points [57], surfels [46, 71, 83], or volumetric [17, 58, 59] structures. A sample visualization of the representation kinds is shown in Figure 1.4. Points or point-cloud represent the surface with 3D points with no explicit associations between them. Each point can optionally have a color value and a normal vector from its corresponding surface. Like point clouds, we can represent the scene with surfels represented by flat disks oriented according to the normal vector of the surface at that point. Every surfel has a radius indicating the surface's portion that it's representing. The volumetric representations are the most popular in the literature, consisting of placing the scene inside a discrete volume where the cells (or voxels) contain quantity describing the surface. The quantity can be a boolean value for occupied or not or a number value from a Truncated Signed Distance Function [20] (TSDF) which informs the distance from the

(a)

(b)

Figure 1.2: A high-quality synthetic 3D model of a real-world scene. In the next figures, we use this scenario to demonstrate SLAM concepts. (a) A top-down view of the input scene. (b) A closer look at the same scene.

cell to the nearest scanned surface. In this work, we adopt the surfel to model our maps and submaps. Since, it can represent surface parts at different detail levels, according to the sensor's distance to the surface while capturing, which we can dynamically add more surfels when the sensor is closer to the surface. Hence, this characteristic is vital to our project since it avoids mixing pixel-deep features from different image resolutions, in which descriptor values may vary.

To situate this work's applicability, we write a brief classification of SLAM systems. Most of the systems are related by three characteristics: 1) input type, 2) mapping output density, and 3) running mode.

1. ***Monocular, Stereo, RGB-D, or LiDAR***: Methods that work with one image per frame are said to be monocular systems. Stereo SLAM or RGB-D inputs can refer to methods that estimate a depth image from stereo pair per frame or with a depth-sensing device like Microsoft Kinect or structured light sensors. A depth image is a kind of image where pixels contain the distance from its point in the camera plane to the captured surface in the real world. Having depth images offers the advantage that every frame can contribute to the scene map with a dense 3D point cloud. On the other side, monocular SLAM requires triangulating pixels of correspondences to estimate its point 3D coordinate in the map [26]. The triangulation makes it challenging to obtain a dense map since correspondences with high confidence can

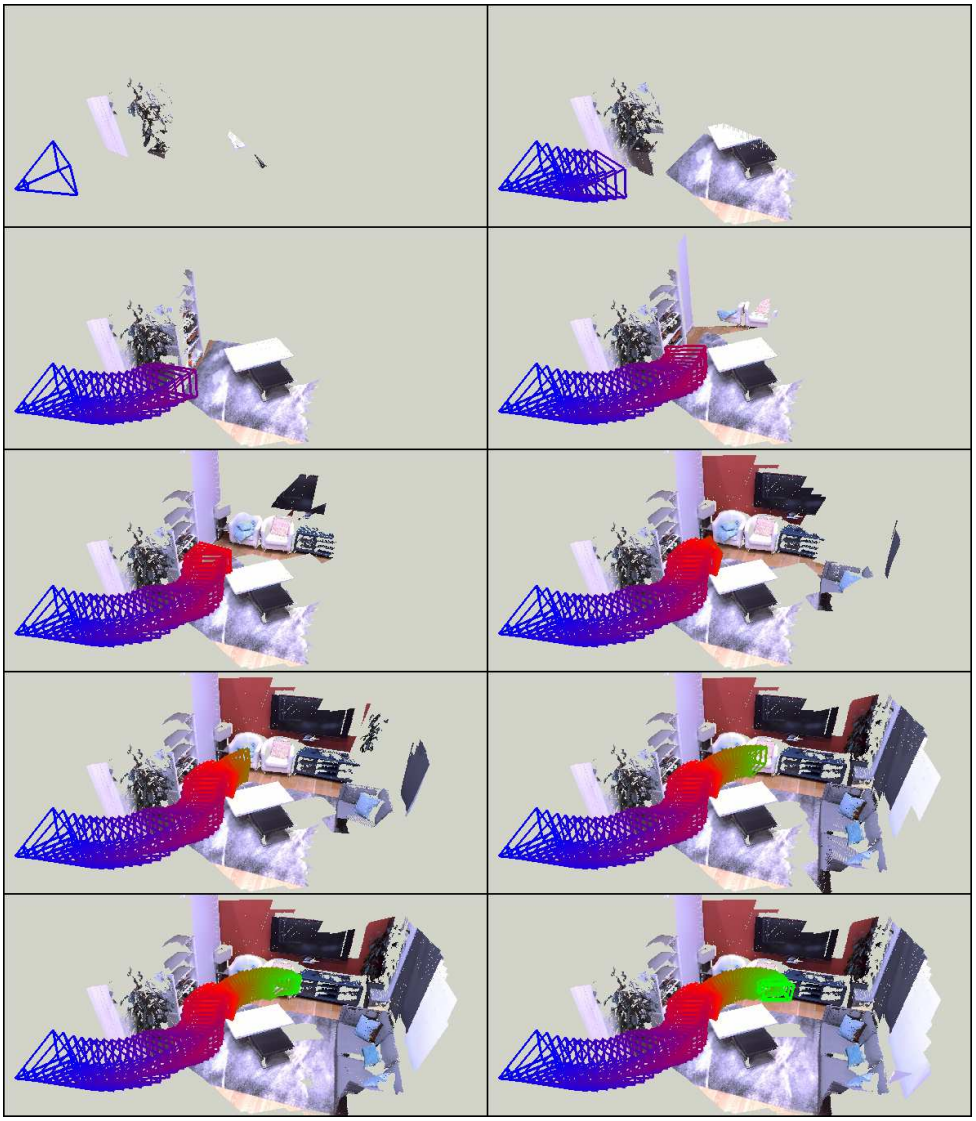


Figure 1.3: RGB-D-based SLAM mapping along time. Each image shows a moment of the camera trajectory and the produced mapping of a dense SLAM system. The pyramid-like wire drawings represent the sensor's pose from the beginning of the motion (blue) to the ending (green).

be established for few pixel pairs. Also, due to the scale ambiguity of perspective transform, the map reconstruction is limited to a scale factor unless an external sensor or reference object provides sizing information with monocular input. LiDAR sensors use the time of flight of laser signals to measure the distance from surfaces, rendering them considerably more precise than RGB-D or Stereo cameras and able to capture longer distances. Although LiDARs perceive the geometry directly as a point cloud from an almost 360° angle rather than a depth image. Some of the core techniques of SLAM for RGB-D sensors also apply to this kind of device.

2. ***Dense or Sparse***: Often correlated with non-monocular SLAM, dense methods include all valid pixels or point measurements from all frames to model the trajectory and map estimates. They will output a detailed reconstruction of the environment at the cost of requiring more computation effort and inherently add more noise from

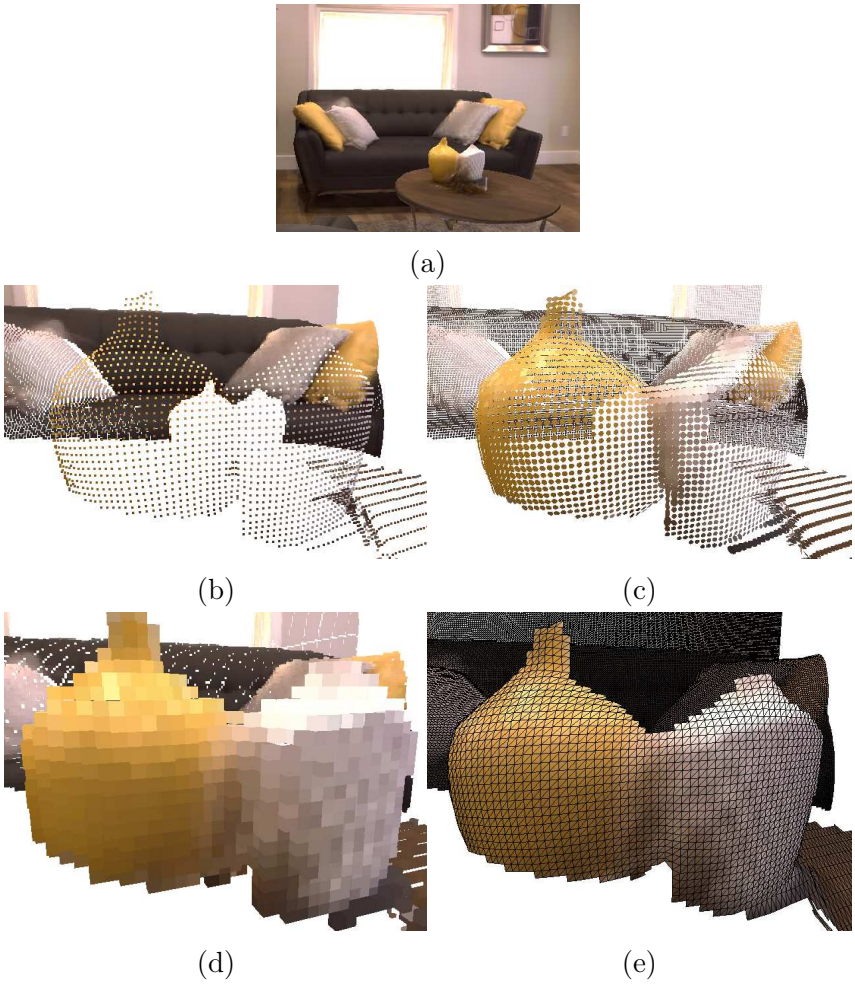


Figure 1.4: Samples of the possible map representations in the literature. (a) Input image of the frame. (b) Point cloud representation. (c) Surfels representation, each point is a disk with a variable radius. (d) Occupancy volume, the ambient is discretized into voxel cells. (d) Volumetric representation, here we show the environment with an occupancy value and a color. (e) A mesh made with triangles.

the more data used in its estimations. On the other hand, sparse methods select key points in the images as elements to the map producing less detailed geometry, but more stable estimations and less computing required. While this often relates to the input type, dense and monocular systems can be built using multiview stereo [26], and other works prefer sparse methods even when depth input is available [56].

3. ***Online or Offline***: Online methods are designed to work for real-time applications that do not require looking at previous trajectories or maps. Only the latest state is essential. Those methods are commonly applied in vehicle guidance, for example. In offline SLAM, the application is interested in inspecting the entire trajectory and map as in 3D reconstruction, which may or may not require real-time performance.

The SLAM method described here is categorized as RGB-D-dense-offline. It uses an RGB-D sensor to create a dense surfel mapping, and, as will be seen, it refines the previous trajectory to improve the current ones.

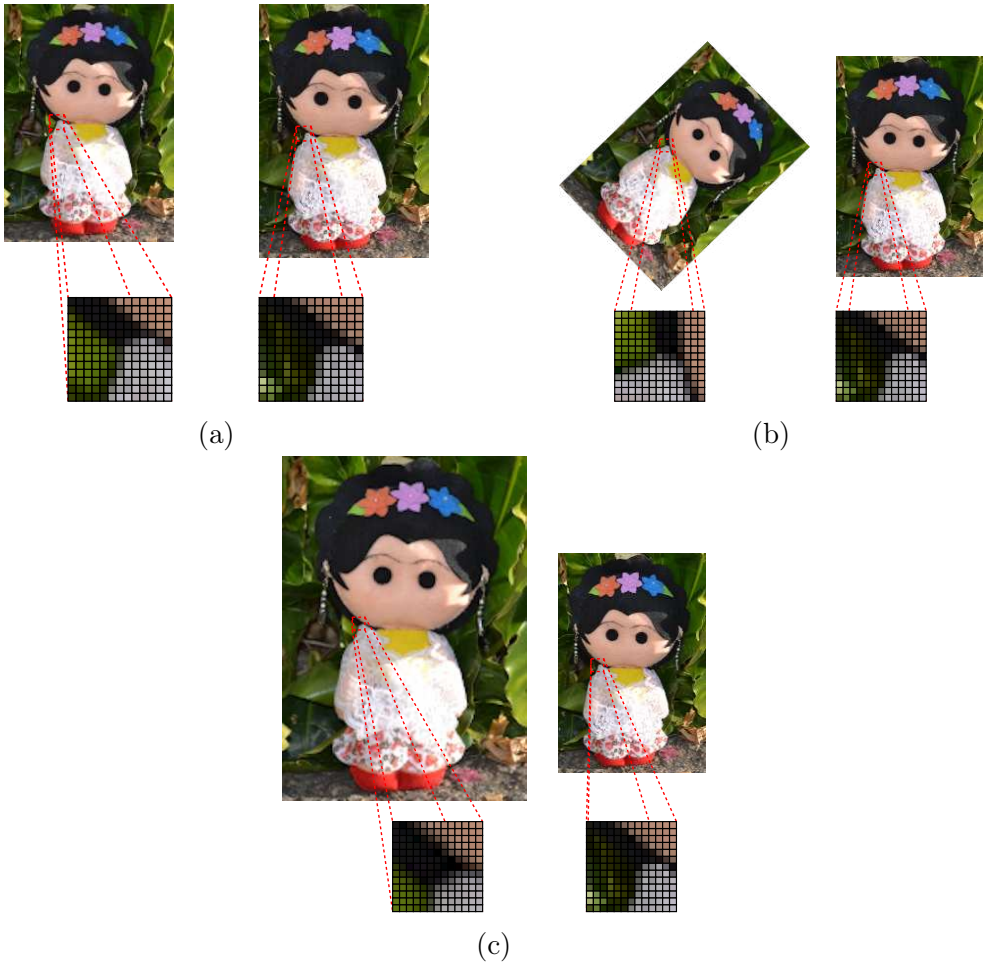


Figure 1.5: Translation, rotation, scale image transformations. In all illustrations, we show two different images. (a) Two different images with only translation motion, in this case, metrics like SSD will work regardless that both patches come from different images, their pixelwise pairs remain valid. (b) Rotation transformation. (c) Scale transformation. In the last two, metrics like SSD will not work well since both patches have different correspondence pairs along with their pixelwise 2D positions.

The choice of the RGB-D modality is a consequence of using dense descriptors in our research because its frames are convertible into dense point clouds to which we can attach a pixel' descriptor to each of its points.

1.2 Descriptor Learning

Associating pixels or points from different images as representations of the same instance in the environment is not only an essential step for SLAM. It is also at the core of other computer vision tasks such as photo stitching, stereo matching, and image retrieval.

One of the first correspondence matching methods is finding image patch pairs with the nearest difference according to their pixel values. We refer to patch any rectangular image crop centered around the target pixel. It is more reliable to compare regions rather than individual pixels since neighborhoods contain regions' structures. Directly comparing patches is particularly viable in situations like stereo matching, when the images are

transformed by a translation-only motion, shown in Figure 1.5a. In such configurations, the similarity of two patches may be calculated from a metric that aggregates the pairwise pixel-to-pixel differences, as the Sum of Squared Distances (SSD), for instance. However, under scale, rotation, or perspective transformations, the trivial pixel-to-pixel positions are changed, invalidating the differences computed by SSD. For instance, Figures 1.5b and 1.5c show how a scale or a rotation transformation changes the trivial pixelwise associations. To overcome those two transformations, established handcraft descriptors [53, 66] try to encode the shape and orientation of the region into a feature vector — enabling many vision tasks in their time — but there can be outlier sensitive, having SLAM authors [21, 57] to add filtering heuristics.

As an alternative to handcraft descriptors, the present work follows machine learning methods to produce descriptors from image priors to improve the robustness to outliers. CNN models were proven to learn image filtering operations for ground-breaking image classification, recognition, and detection in recent years. In the descriptor learning task, the model outputs feature vectors from image regions using the learned priors from a dataset of good matchings samples. Hence, learning descriptors may offer a solution to situations where the target images undergo significant geometric and illumination transformations, being challenging for handcraft descriptors to account for so many distortions. In some works [19, 29], the authors demonstrate the possibility of performing semantic comparisons with the prior information from a dataset to encode common attributes of objects. For example, Florence et al. [29] learn descriptors of household objects (shoes or toys) that identify corresponding regions of different object instances that belong to the same category, like the strings of different shoe models.

As a sample experiment using the learned feature model in SLAM, this dissertation uses the CNN architectures based on existing works [19, 29, 70] for dense descriptor extraction.

1.3 Contributions

In summary, we build an RGB-D SLAM system that integrates deep learned features into every surfel during its mapping and use those features for finding loop closures and fine alignment of submaps. From this framework, we contribute with the following:

- We build a mapping framework for SLAM that incorporates descriptors as any other property of its mapping surfaces. This allows other applications like reusing the descriptors for scene relocalization in AR or refining semantic segmentation in a similar way as suggested by McCormac et al. [55].

- We use learned features in the submaps for detecting loop closures. Our results show that learned descriptors perform equivalently to SIFT and the FPFH features for detecting and estimating the loop closure in submap-based SLAM.

- We propose a novel ICP method for aligning submaps with dense descriptors. Our results show that incorporating dense features have the potential to improve alignment tasks. However, as can be seen in Chapter 4, it depends on the CNN's model

generalization capability to perform better than color information on never seem scenarios. We compare it with the state of art method, ColoredICP [61] and RGB-based ICP, which shows that deep features may not be better, but it suggests more stable results than the other algorithms.

- Besides building our work with know architecture and loss for descriptor learning, we propose a dataset generation strategy by building a connection graph of frames from RGB-D datasets. So, each pair of sample training descriptors can have a wider variety of viewing angles. This contribution is described in Section 3.7.2.

Although our results do not present marginally better results, exploring features from CNNs may be an alternative in SLAM that already uses them for semantically information mapping. As those systems already process the image in CNNs, using them as features instead of adding another descriptor extraction mechanism may save processing time.

Besides those results, from the engineering perspective, we have also contributed with two Python libraries that allowed us to quickly prototype and test ideas on our SLAM system:

- **TensorViz** [1]: An OpenGL interface for rendering geometric models directly from the GPU memory of PyTorch's [62] tensors, avoiding unnecessary data transfers or copies between GPU or CPU. That was critical for us for inspecting our system's states.

- **SLAM Toolbox** [2]: Python module with PyTorch integration containing various structures and algorithms for building SLAM or 3D visualizations. This library is similar to the Open3D library [88] but is more geared towards GPU and surfel handling.

Their initial necessity was to build a surfel mapping mechanism to handle multiple size descriptors. However, it had grown to attend to our visualization, data loading, evaluation, VO, and registration demands.

1.4 Dissertation Structure

The dissertation structure is the following:

- Chapter 2 starts with reviewing the math prerequisites related to rigid transformation and solving non-linear least-squares problems. Next, it presents registration methods to align 3D geometry, which is later used to understand SLAM techniques and our submap alignment method. Then it further reviews the overall SLAM techniques, covering techniques not only for the RGB-D-dense-offline method but briefly states other methodologies. Moreover, its second part reviews the technique for descriptor learning using CNN from the perspective of handcraft descriptors.

[1] TensorViz - https://gitlab.com/mipl/3d-reconstruction/tensorviz
[2] SLAM Toolbox - https://gitlab.com/mipl/3d-reconstruction/slam-toolbox

- Chapter 3, we present our SLAM solution consisting of a preprocessing module to filter the incoming RGB-D frames; a visual odometry module for RGB-D sensors; loop closure detection module; submap alignment; deep dense feature extraction CNN architecture and training.
- In Chapter 4 shows the evaluation methodology and the results for performing loop closure detection and submap alignment.
- We review and draw our conclusions about this dissertation in Chapter 5.

Chapter 2

Background

The dissertation unites two main computer vision topics of high relevance: SLAM and Machine Learning (ML). This section provides an overview of Visual SLAM and ML from the point of view of RGB-D data and visual descriptor learning.

By the nature of the SLAM problem, we must start with the mathematical tools for working with trajectories and rotations. Firstly, we define the math notations and prerequisites used throughout this dissertation in Section 2.1. Next, we review matrix representation of motion transformations as special mathematical sets expressed using Group and Lie theories (Section 2.2), which differentiate rotations and give compacter representations rather than matrices. As our work is visually based, Section 2.3 defines the pinhole camera projection model to project 2D points and back project them into 3D.

SLAM can be thought of as a series of registration problems at different levels and kinds of data, so it is important to understand those algorithms before entering SLAM itself. The foremost step to understanding registration methods is to cover how to solve weighted non-linear least-squares problems. Those problems are our interest because registration is often described by minimizing an energy function that measures the alignment of two inputs given the optimization's target transformation parameters. Not only that, solving non-linear least squares will grow as an essential operation for any SLAM since we constantly estimate the sensor's most likely state given its previous one and current measurements as energy minimization. Therefore, Section 2.4 presents a framework to minimize weighted non-linear least-squares problems with Gauss-Newton (GN) routine in an abstract problem. Then, this framework is adopted during the rest of the text to describe concrete problems as the ICP algorithm, VO, and our deep-feature submap registration algorithm in Chapter 3. The GN procedure is chosen because of its simplicity, and the non-linearity property of rotations requires iterative numerical approaches.

Section 2.5 reviews the basic techniques for registering point cloud representable 3D geometry with the ICP algorithm using the GN framework previously mentioned. The ICP is a base routine for implementing VO in RGB-D SLAM, since the frames can be turned into point-clouds, which we can register together with it and accumulate the transformation to obtain the initial trajectory. Another reason for focusing on ICP is that our submap registration algorithm is an ICP algorithm adapted to measure the alignment energy from deep features. However, ICP methods usually fail badly when the 3D points are not already close to an alignment. Therefore, we also present two other algorithms

based on correspondence matching of sparse points. When point correspondences from the two geometries, it is possible to register at any initial condition but without the same precision. This family of sparse algorithms is handy in SLAM for running initial estimation guesses and to detect loop closures as done by Choi et al. [17] and we.

Section 2.6 explains how to create a naive SLAM system and why simple approaches are insufficient to create a functional system due to a problem known as tracking drift. Then we review how successful SLAM techniques solve that by dividing the problem into front-end and back-end parts. The front-end is responsible for computing estimates directly from sensor or image measurement at the per-frame level. In contrast, the back-end computes refined answers for the trajectory and mapping by globally examining the estimations of the front-end, being essential to the high quality of recent works.

Image descriptors always had an essential role in SLAM works. So, in the second part of this chapter, we discuss the steps taken from early works — many powerful techniques even in today's standards — and how authors have been researching CNNs to improve the area. In it, we review handcraft descriptors, network architectures, datasets, and training of models to outputting dense descriptors.

2.1 Math Notations and Definitions

When working with motion estimation, camera projection, or geometric processing, linear algebra's tools becomes the workhorse of algorithms in those fields. In the following section, we summarize common mathematical definitions and tools.

In this text, vectors are written in bold font, for example: $\mathbf{v}$, $\mathbf{w}$ or $\boldsymbol{\omega}$. While scalar values are in regular math typefaces: i, t or α — even when referencing a vector component, like p_1. Matrices are uppercase letters like M, T or R.

The dot product of two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^D$ is written as $\mathbf{v}^\intercal\mathbf{w}$ and defined as: $\mathbf{v}^\intercal\mathbf{w} = v_1w_1 + v_2w_2 + \cdots + v_Dw_D$. We symbolize the angle between two vectors $\mathbf{v}$ and $\mathbf{w}$ with symbol $\angle$ which is calculated using the dot product and the inverse cosine function,

$$\mathbf{n}_u \angle \mathbf{n}_s = \cos^{-1} \frac{\mathbf{v}^T\mathbf{w}}{\|\mathbf{v}\|\|\mathbf{w}\|}. \tag{2.1.1}$$

The cross product between vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^3$ is the following operation:

$$\mathbf{v} \times \mathbf{w} = \begin{bmatrix} v_2w_z - v_zw_y \\ v_zw_x - v_xw_z \\ v_xw_y - v_yw_x \end{bmatrix}. \tag{2.1.2}$$

The cross product operation of any vector $\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^\intercal$ by another one is encoded in skew-symmetric matrices:

$$[\mathbf{v}]_\times = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}, \tag{2.1.3}$$

where the vector in the middle of the brackets followed by a subscript cross symbol $\times$ constitutes the skew-symmetric matrix version of its cross product. A matrix $\mathbb{R}^{3\times3}$ is skew-symmetric if the equality $[\mathbf{v}]_\times^\intercal = -[\mathbf{v}]_\times$ holds true. This is empirically verified in

$$\begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}^\intercal = - \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}, \tag{2.1.4a}$$

$$\begin{bmatrix} 0 & v_z & -v_z \\ -v_z & 0 & v_x \\ v_y & -v_x & 0 \end{bmatrix} = \begin{bmatrix} 0 & v_z & -v_y \\ -v_z & 0 & v_x \\ v_y & -v_x & 0 \end{bmatrix}. \tag{2.1.4b}$$

The properties of the power function with skew-symmetric are

$$[\mathbf{v}]_\times^\intercal = -[\mathbf{v}]_\times \tag{2.1.5a}$$
$$[\mathbf{v}]_\times^2 = \mathbf{v}\mathbf{v}^\intercal - \mathbf{v}^\intercal\mathbf{v}I \tag{2.1.5b}$$
$$[\mathbf{v}]_\times^3 = -[\mathbf{v}]_\times \tag{2.1.5c}$$
$$[\mathbf{v}]_\times^4 = -[\mathbf{v}]_\times^2 \tag{2.1.5d}$$
$$[\mathbf{v}]_\times^5 = [\mathbf{v}]_\times \tag{2.1.5e}$$
$$\dots$$

Those properties are helpful during Section 2.2.1 to convert rotation in the form of an axis and rotation angle into matrices (or Rodrigue's rotation formula).

Another property of square matrices used by us is that the transpose of the product of two matrices A and B equals to the inverse multiplication order transposed individually, that is

$$(AB)^\intercal = B^\intercal A^\intercal \tag{2.1.6}$$

In the text, we only refer to 3-dimensional tensors, and their sizes are stated as $W \times H \times C$, with W, H, and C being its width, height, and depth. For instance, RGB images are three-dimensional tensors with three channels ($C = 3$).

We use the symbol $\mathcal{N}(x; \boldsymbol{\mu}, \Sigma)$ to represent a normal distribution parameterized by the mean $\boldsymbol{\mu}$ and the covariance matrix Σ. The symbol $\mathbb{E}[\cdot]$ defines the expected value of a random variable. Another miscellaneous identity used during the uncertainty estimation (Section 2.6.2) is that the covariance of expressions such as $A\mathbf{x} + \boldsymbol{\epsilon}$, where A and $\boldsymbol{\epsilon}$ are fixed variables, is $\operatorname{cov}(A\mathbf{x} + \boldsymbol{\epsilon}) = A \operatorname{cov}(\mathbf{x}) A^\intercal$, because

$$\begin{aligned}
\operatorname{cov}(A\mathbf{x} + \boldsymbol{\epsilon}) &= \mathbb{E}[(A\mathbf{x} + \boldsymbol{\epsilon} - \mathbb{E}[A\mathbf{x} + \boldsymbol{\epsilon}])(A\mathbf{x} + \boldsymbol{\epsilon} - \mathbb{E}[A\mathbf{x} + \boldsymbol{\epsilon}])^\intercal], && (2.1.7a)\\
&= \mathbb{E}[(A\mathbf{x} + \boldsymbol{\epsilon} - \mathbb{E}[A\mathbf{x}] - \boldsymbol{\epsilon})(A\mathbf{x} + \boldsymbol{\epsilon} - \mathbb{E}[A\mathbf{x}] - \boldsymbol{\epsilon})^\intercal], && (2.1.7b)\\
&= \mathbb{E}[(A\mathbf{x} - A\mathbb{E}[\mathbf{x}])(A\mathbf{x} - A\mathbb{E}[\mathbf{x}])^\intercal], && (2.1.7c)\\
&= \mathbb{E}[A(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^\intercal A^\intercal], && (2.1.7d)\\
&= A\mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^\intercal] A^\intercal, && (2.1.7e)\\
&= A \operatorname{cov}(\mathbf{x}) A^\intercal. && (2.1.7f)
\end{aligned}$$

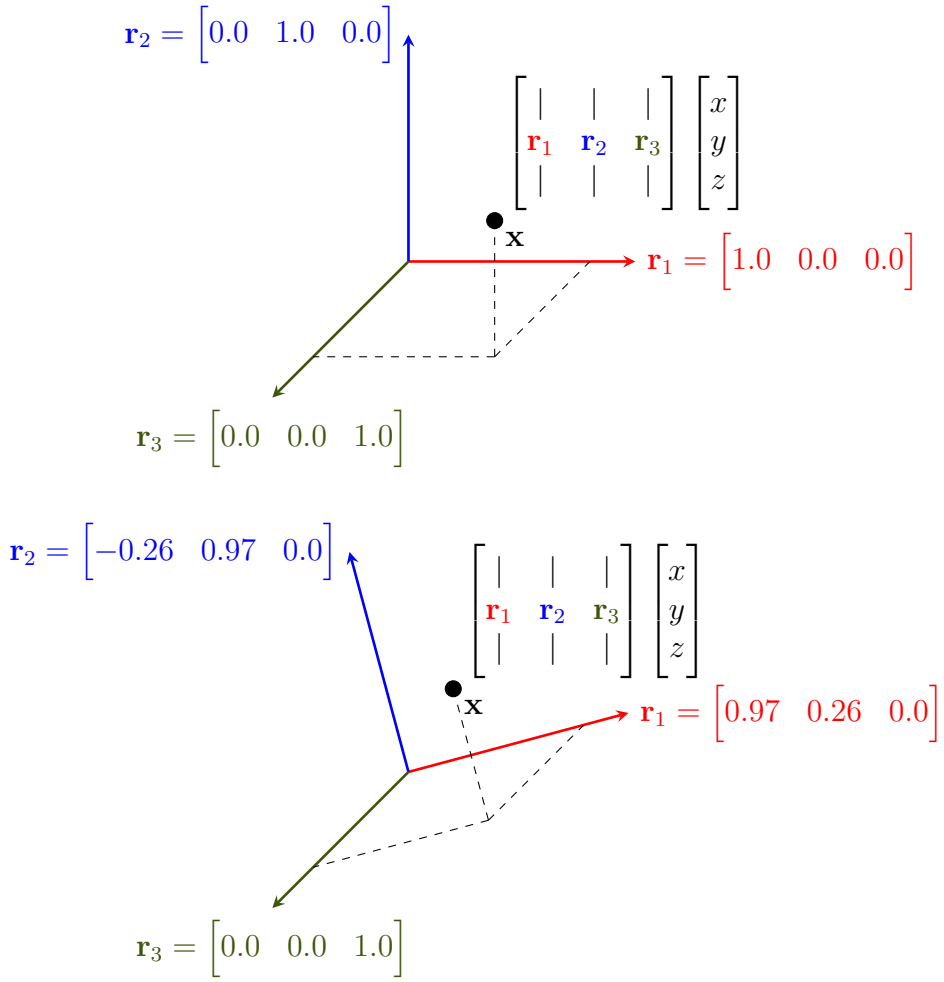


Figure 2.1: The column vectors of a $SO(3)$ rotation matrix are the axis of a coordinate system. Top: the coordinate system spanned by the identity matrix, any point that we multiply by it keeps the same. Bottom: the column vectors of a rotation matrix corresponding to its reorientation of the space. The act of multiplying the matrix by a point $\mathbf{x}$ reprojects its x, y, and z components according to a new vector basis, thus, rotating it.

2.2 Motion Representation

Orthonormal matrices $R \in \mathbb{R}^{3\times3}$ with determinant equal to one ($\det R = 1$) represents rotation around the origin of the 3D Euclidean space. Orthonormality is when a set of unit vectors are linearly independent, hence perpendicular one to another. In the case of rotation matrices, their three column vectors $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$ composing a matrix $R = \begin{bmatrix}\mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3\end{bmatrix}$ are orthonormal since a rotation must span a 3D cartesian coordinate space without scale changes, see Figure 2.1 for an illustration. Formally, linear independent vectors have 0 scalar products between them, $\mathbf{e}_1{}^\intercal\mathbf{e}_2 = \mathbf{e}_1{}^\intercal\mathbf{e}_3 = \mathbf{e}_2{}^\intercal\mathbf{e}_3 = 0$. Matrices with a reflection operation are not rotations because their determinants are negative. Therefore, those properties restrict rotation matrices to have only 3 Degrees of Freedom (DoF) besides having 9 values. In some situations, as in motion estimation, over-parameterization is not desirable, but will be seen in the following sections, it is

possible to convert them into compacter representations.

As shown in Figure 2.1, the rotation action of a matrix is applied when multiplying a point (e.g., one from a point cloud) or a vector $\mathbf{x}$ at its right side:

$$\mathbf{x}' = R\mathbf{x} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{e}_1{}^\intercal \mathbf{x} \\ \mathbf{e}_2{}^\intercal \mathbf{x} \\ \mathbf{e}_3{}^\intercal \mathbf{x} \end{bmatrix}. \tag{2.2.1}$$

The composition of multiple rotations is done by multiplying their matrices, consistently producing another valid rotation matrix. For example, $R_3 = R_2 R_1$ generates R_3 that has the same effect of multiplying a point $\mathbf{x}$ by R_1 first and then by R_2, that is

$$(R_2 R_1)\mathbf{x} = R_2(R_1\mathbf{x}). \tag{2.2.2}$$

Such properties and structure of rotation matrices made them part of it is known as the special orthogonal rotation group $SO(3)$. The field of Group Theory studies how certain mathematical objects share actions and structure. Characterizing groups is vital for many applications because it allows us to project algorithms to manipulate mathematical elements with the certainty that if the structure is kept, then its actions and properties will remain valid. Group Theory identifies a set of elements $\mathcal{G}$ as a group if the following properties are met:

- It exists a binary action, written here with a $\circ$, that always maps two elements to be inside the group's set, $\mathcal{G} \times \mathcal{G} \mapsto \mathcal{G}$. Example: $\mathcal{X} \circ \mathcal{Y} = \mathcal{Z}$ for any $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \mathcal{G}$;
- The action operation is associative: $(\mathcal{X} \circ \mathcal{Y}) \circ \mathcal{Z} = \mathcal{X} \circ (\mathcal{Y} \circ \mathcal{Z})$ for any $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \mathcal{G}$.
- The group has an identity element $I_\mathcal{G} \in \mathcal{G}$ such $\mathcal{X} \circ I_\mathcal{G} = \mathcal{X}$ and $I_\mathcal{G} \circ \mathcal{X} = \mathcal{X}$ for any element $\mathcal{X} \in \mathcal{G}$;
- The group elements are invertible: $\mathcal{X}^{-1} \circ \mathcal{X} = I_\mathcal{G}$ and $\mathcal{X} \circ \mathcal{X}^{-1} = I_\mathcal{G}$ for any $\mathcal{X} \in \mathcal{G}$;

The $SO(3)$ matrices have all those requirements. Its action operation is the matrix multiplication, also being associative. Its identity element is the 3×3 identity matrix $I^{3\times3}$. Moreover, the matrices are invertible: $R^{-1}R = I^{3\times3}$ and $RR^{-1} = I^{3\times3}$. Related to inverse, one special property of orthonormal matrix is having the transpose operation being equivalent to the inverse: $R^\intercal = R^{-1}$.

To also represent translations, the $SO(3)$ elements can be accompanied by a vector $\mathbf{t} \in \mathbb{R}^3$ for turning the action over a point $\mathbf{x} \in \mathbb{R}^3$ into rotation followed by a translation: $R\mathbf{x} + \mathbf{t}$. Those action are encompassed in 4×4 matrices having the following structure

$$\begin{bmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{2.2.3}$$

forming the special Euclidean group, denoted $SE(3)$. The matrices are 4×4 and not 3×4 dimensions to keep the associative property of the matrix multiplication, which is

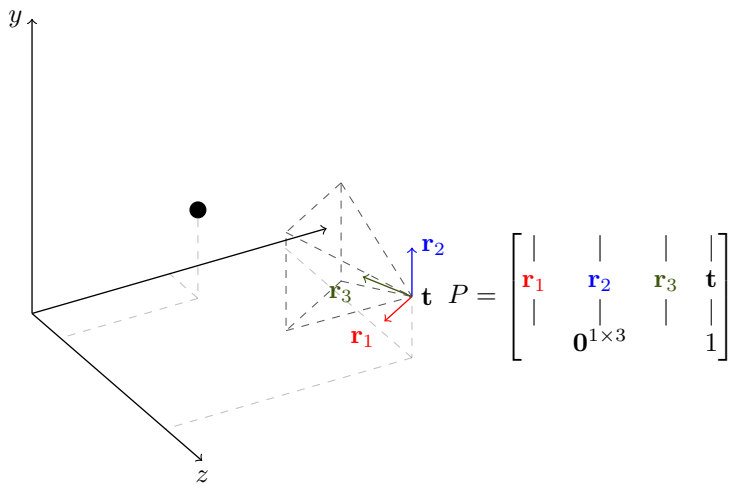


Figure 2.2: Camera pose represented by a $SE(3)$ matrix P. The dashed wireframe pyramid represents the sensor's pose. It also draws the three orthonormal columns vectors $\mathbf{r}_1$, $\mathbf{r}_2, \mathbf{r}_3 \in \mathbb{R}^3$ of the rotational part of the pose. The positional part is show in vector $\mathbf{t} \in R^3$. The last row of P is read as the vector $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$.

undefined between two 3×4 matrices. The 4×4 dimensions also impose that 3D vectors should be expressed in homogeneous coordinates (e.g., $\begin{bmatrix} x & y & z & 1 \end{bmatrix}^\intercal$). However, to unclutter the calculations, we will multiply homogenous matrices with non-homogeneous points or matrices at most occurrences in this dissertation.

When composing two transformations $T_3 = T_2 T_1$ in which T_1 is applied first, its matrix multiplication is equivalent to

$$\begin{bmatrix} R_2 & \mathbf{t}_2 \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_1 & \mathbf{t}_1 \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} R_2 R_1 & R_2 \mathbf{t}_1 + \mathbf{t}_2 \\ \mathbf{0} & 1 \end{bmatrix}. \tag{2.2.4}$$

Another helpful property is that the rotation matrix transpose can be used to aid the inverse of T:

$$\begin{bmatrix} R^\intercal & -R^\intercal \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.2.5}$$

The primary usefulness of the groups $SO(3)$ and $SE(3)$ for SLAM is being convenient representations for the sensor position and orientation at a specific time. For example, as illustrated in Figure 2.2, a $SE(3)$ matrix can represent the direction and orientation of a sensor's pose by using its rotation and translation parts.

2.2.1 Lie Groups and Algebras for SLAM

Formulated from the work of Marius Sophus Lie in the XIX century, Lie Theory relates group theory and differential geometry on manifolds. Most used in physics, Lie Theory offers a mathematical framework for SLAM and robotics engineering to compose, interpolate and differentiate geometric transformation as any other algebraic expression. In a very brief definition, a set of mathematical objects forms a Lie group if they are a group

with the additional properties:

- The elements are embedded in a differentiable (or smooth) manifold. In informal terms, elements are on a curved surface without edges or spikes, and the surface is inside a larger dimensional space.
- The group's action operation ($\circ$) and element inverse are differentiable.
- Formally, a Lie group must have a Lie bracket operator [42]. But since this is not required to use Lie Groups in our context, we do not discuss it.

This section is given a very shallow and non-rigorous overview of Lie Theory applied into the computation of registration of 3D geometry used in VO and map alignment. For a deeper discussion in our same context, the reader may refer to Bloesch et al. [6], Eade [24], or Sola et al. [74].

The $SO(3)$ group is a Lie Group because its matrices are embedded on a smooth manifold inside the set of all $\mathbb{R}^{3\times3}$ matrices. This manifold can be interpreted as the surface of a unit hypersphere since $SO(3)$ matrices do not change space volume, i.e., $\det R = 1$.

As $SO(3)$ lies on a smooth manifold, then given any trajectory defined by a function $R(t)$ over time t on its surface, it is possible to derive a vector space tangent to the surface at any element $R(t)$ with its derivative $\frac{\partial R(t)}{\partial t}$. By being a smooth surface, its results that the tangent space has the same structure at any given point of the $SO(3)$ manifold. To emerge the tangent space structure of $SO(3)$, we can differentiate the property that $R^\intercal R = I$ w.r.t. t

$$\frac{\partial R(t)^\intercal R(t) - I}{\partial t}, \tag{2.2.6}$$

apply the product and constant rules

$$R(t)^\intercal \frac{\partial R(t)}{\partial t} + \frac{\partial R(t)^\intercal}{\partial t} R(t), \tag{2.2.7}$$

and rearrange with transpose to make skew-symmetrically evident

$$R(t)^\intercal \tfrac{\partial R(t)}{\partial t} = -\tfrac{\partial R(t)^\intercal}{\partial t} R(t), \tag{2.2.8a}$$

$$= -\left(R(t)^\intercal \tfrac{\partial R(t)}{\partial t}\right)^\intercal. \tag{2.2.8b}$$

The fact that $R(t)^\intercal \frac{\partial R(t)}{\partial t}$ is equal to the negative of its transpose, $-\left(R(t)^\intercal \frac{\partial R(t)}{\partial t}\right)^\intercal$, matches the skew-symmetric structure demonstrated in Equation 2.1.4. We can further manipulate Equation 2.2.8b by replacing the right side with a skew-symmetric matrix $[\boldsymbol{\omega}]_\times$ to finally arrive at Ordinary Differential Equation (ODE) over the derivative of $\frac{\partial R(t)}{\partial t}$

$$R(t)^\intercal \tfrac{\partial R(t)}{\partial t} = [\boldsymbol{\omega}]_\times, \tag{2.2.9a}$$

$$R(t) R(t)^\intercal \tfrac{\partial R(t)}{\partial t} = R(t)[\boldsymbol{\omega}]_\times, \tag{2.2.9b}$$

$$\tfrac{\partial R(t)}{\partial t} = R(t)[\boldsymbol{\omega}]_\times. \tag{2.2.9c}$$

And when $R(t) = I$ then the result is just:

$$\frac{\partial R(t)}{\partial t} = [\boldsymbol{\omega}]_\times. \tag{2.2.10}$$

From the previous equation (Equation 2.2.10), we arrive that derivative or tangents over the $SO(3)$ manifold are skew matrices. Consequently, they can be decomposed from the linear combination of three basis skew-matrices G_1, G_2, and G_3

$$G_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \tag{2.2.11a}$$

$$G_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \tag{2.2.11b}$$

$$G_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{2.2.11c}$$

such that

$$[\boldsymbol{\omega}]_\times = \omega_1 G_1 + \omega_2 G_2 + \omega_3 G_3. \tag{2.2.12}$$

The space span by the generator matrices G_1, G_2, and G_3 is the Lie algebra of $SO(3)$, or $\mathfrak{so}(3)$ [74]. As we found the skew matrix-structure from Equation 2.2.10 when $R(t) = I$ its lie algebra is in relation to the identity matrix. Moreover, as linear combinations from the generators, the $\mathfrak{so}(3)$ members are also expressed by any vector $\boldsymbol{\omega} \in \mathbb{R}^3 = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix}$. This compact representation is one of the benefits of the Lie Theory. It assures that the members of a lie algebra are always represented by vectors having the exact dimensions of its Lie group's DoF [74], which is advantageous for numerical state estimation algorithms because they will have fewer parameters to search for optimal values. During the dissertation, we will refer to any member of $\mathfrak{so}(3)$ from a $\boldsymbol{\omega}$ vector represents with a cross product, e.g., $[\boldsymbol{\omega}]_\times$.

To go from a $\mathfrak{so}(3)$ matrix or vector to a rotation matrix in $SO(3)$, we can solve the ODE in Equation 2.2.10, which the answer is the exponential function

$$R(t) = R(0)\exp([\mathbf{w}]_\times t), \tag{2.2.13}$$

where $[\mathbf{w}]_\times t = [\boldsymbol{\omega}]_\times$ for decomposing $\boldsymbol{\omega}$ into a unitary vector $\mathbf{w}$ and its magnitude t. By setting the start condition of Equation 2.2.13 to $R(0) = I$, thus matching the $\mathfrak{so}(3)$ members, we expand the exponential in Equation 2.2.13 with the power series to obtain

a closed-form equation:

$$\exp\left([\mathbf{w}]_\times t\right) = \sum_n^\infty \frac{t^n}{n!}[\mathbf{w}]_\times^n, \tag{2.2.14a}$$

$$= I + t[\mathbf{w}]_\times + \frac{t^2}{2!}[\mathbf{w}]_\times^2 + \frac{t^3}{3!}[\mathbf{w}]_\times^3 + \frac{t^4}{4!}[\mathbf{w}]_\times^4 \ldots, \tag{2.2.14b}$$

and then rearranging the series to match Maclaurin series for sine and cosine,

$$\exp\left([\mathbf{w}]_\times t\right) = I + [\mathbf{w}]_\times \left(t - \frac{t^3}{3!} + \frac{t^5}{5!} \ldots\right) + [\mathbf{w}]_\times^2 \left(\frac{t^2}{2!} - \frac{t^4}{4!} + \frac{t^6}{6!} \ldots\right), \tag{2.2.15}$$

where the Maclaurin series for sine is

$$\sin t = \sum_n^\infty \frac{(-1)^n}{(2n+1)!} t^{2n+1} = t - \frac{t^3}{3!} + \frac{t^5}{5!} - \frac{t^7}{7!} + \ldots, \tag{2.2.16}$$

and cosine

$$\cos t = \sum_n^\infty \frac{(-1)^n}{2n!} t^{2n} = 1 - \frac{t^2}{2!} + \frac{t^4}{4!} - \frac{t^6}{6!} + \ldots. \tag{2.2.17}$$

Replacing the terms in Equation 2.2.15 with Equations 2.2.16 and 2.2.17 we obtain a closed-form for the $\mathfrak{so}(3)$ exponential map, or a.k.a., the Rodrigue's formula for exponential 3D rotation:

$$\exp([\boldsymbol{\omega}]_\times) = \exp([\mathbf{wt}]_\times) = I + [\mathbf{w}]_\times \sin\theta + [\mathbf{w}]_\times (1 - \cos\theta). \tag{2.2.18}$$

Following the definitions from Sola et al. [74], we let the uppercase exponential function being an alias for the vector representation of $\mathfrak{so}(3)$

$$\operatorname{Exp}(\boldsymbol{\omega}) \triangleq \exp([\boldsymbol{\omega}]_\times). \tag{2.2.19}$$

When working with numerical optimization methods for finding the best alignment between 3D geometric, we will often require computing the derivative of an expression

$$\frac{\partial \operatorname{Exp}(\boldsymbol{\omega}) R\mathbf{x}}{\partial \boldsymbol{\omega}}, \tag{2.2.20}$$

where $\mathbf{x} \in \mathbb{R}^3$ is a given 3D point. This derivative will be used on alignment estimation routines to state how much the change in the parameters $\boldsymbol{\omega}$ change the rotation of a point $\mathbf{x}$ that we rotate by at least the $R \in \mathbb{R}^{3\times3}$ matrix. For finding the derivative of a rotation over its parameters $\boldsymbol{\omega}$, one could use the compact and closed formulas from Gallego and Yezzi [32]. However, as our usage will target minor parameter updates in $\boldsymbol{\omega}$, we can use a linearized version often applied for numerical optimization methods [38] considering $\boldsymbol{\omega} \approx \boldsymbol{\omega}_0$, where $\boldsymbol{\omega}_0 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}$. Our derivative approximation becomes

$$\frac{\partial \operatorname{Exp}(\boldsymbol{\omega}) R\mathbf{x}}{\partial \boldsymbol{\omega}} \approx \frac{\partial \operatorname{Exp}(\boldsymbol{\omega}_0) R\mathbf{x}}{\partial \boldsymbol{\omega}_0}. \tag{2.2.21}$$

To find the approximate derivative, we first must note that the zero 3-dimensional vector $\boldsymbol{\omega_0}$ results the following decomposition in $\mathfrak{so}(3)$:

$$[\boldsymbol{\omega_0}]_\times = 0\begin{bmatrix}0&0&0\\0&0&-1\\0&1&0\end{bmatrix} + 0\begin{bmatrix}0&0&1\\0&0&0\\-1&0&0\end{bmatrix} + 0\begin{bmatrix}0&-1&0\\1&0&0\\0&0&0\end{bmatrix}, \tag{2.2.22}$$

that is, each component has a matrix that specifies its rate of change. For example, a slight movement on w_1 will result in a reciprocal scaling in the generator G_1 being its tangential direction. In other words, it is the derivative of $Exp(\boldsymbol{\omega}_0)$ in respect to w_1. Furthermore, we can compute using the same concept the partial derivatives w.r.t. $\boldsymbol{\omega}_0$ for the other components

$$\frac{\partial \operatorname{Exp}(\boldsymbol{\omega}_0)}{\partial \omega_1} = G_1, \tag{2.2.23a}$$
$$\frac{\partial \operatorname{Exp}(\boldsymbol{\omega}_0)}{\partial \omega_2} = G_2, \tag{2.2.23b}$$
$$\frac{\partial \operatorname{Exp}(\boldsymbol{\omega}_0)}{\partial \omega_3} = G_3. \tag{2.2.23c}$$

Therefore, returning to derivative from Equation 2.2.20, we can then arrange its approximation

$$\frac{\partial \operatorname{Exp}(\boldsymbol{\omega})R\mathbf{x}}{\partial \boldsymbol{\omega}} \approx \frac{\partial \operatorname{Exp}(\boldsymbol{\omega}_0)}{\partial \boldsymbol{\omega}_0} R\mathbf{x}, \tag{2.2.24a}$$
$$\approx \begin{bmatrix} G_1R\mathbf{x} & G_2R\mathbf{x} & G_3R\mathbf{x}\end{bmatrix}, \tag{2.2.24b}$$
$$\approx -[R\mathbf{x}]_\times, \tag{2.2.24c}$$

resulting in derivative approximation for use in our 3D registration algorithms.

Multiplying a rotation R with an update represented by $\operatorname{Exp}(\boldsymbol{\omega})$ as we did in Equation 2.2.20 is quite frequently in numerical methods that evolves the $SO(3)$ or $\mathfrak{so}(3)$ groups because the plus operator is not valid on those manifolds, e.g., the following expression do not produce a valid rotation

$$R + \operatorname{Exp}(\boldsymbol{\omega}) \notin SO(3). \tag{2.2.25}$$

Thus, for the algorithms to express an update over a current estimative R with parameters $\boldsymbol{\omega}$, they must use the multiplication $\operatorname{Exp}(\boldsymbol{\omega})R$. Moreover, to reuse the numerical methods formula patterns that are defined with plus $(+)$ operator and avoiding repeating expressions at the same time, we let the boxplus operator $(\boxplus)$ be

$$R \boxplus \boldsymbol{\omega} \triangleq \operatorname{Exp}(\boldsymbol{\omega})R. \tag{2.2.26}$$

The inverse order of the expression is to salient that the rightmost matrix's action is applied first[1]. The boxplus operator will be highly used in the Chapter 3 for minimizing rotation-based energy functions with the GN method.

[1]Authors do not have a canonical definition about the boxplus operator order, some may define as $R \boxplus \boldsymbol{\omega} \triangleq R\operatorname{Exp}(\boldsymbol{\omega})$. [74]

Like $SO(3)$, $SE(3)$ is a Lie group with its Lie algebra $\mathfrak{se}(3)$ that is compactly represented by vectors $\boldsymbol{\xi} \in \mathbb{R}^6$ such that

$$\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\omega} \end{bmatrix}, \tag{2.2.27}$$

with $\boldsymbol{\rho} \in \mathbb{R}^3$ and $\boldsymbol{\omega} \in \mathbb{R}^3$ being its $\mathfrak{so}(3)$ part. The $\mathfrak{se}(3)$ matrices are generated from linear combinations of a set of generators $\{G_1, G_2, G_3, G_4, G_5, G_6\}$ of its tangent space:

$$\boldsymbol{\xi} = \rho_1 G_1 + \rho_2 G_2 + \rho_3 G_3 + \omega_1 G_4 + \omega_2 G_5 + \omega_3 G_6 \tag{2.2.28}$$

where:

$$\begin{aligned} G_1 &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, G_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \\ G_4 &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, G_6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{2.2.29}$$

The generators G_4, G_5, G_6 are the same from $\mathfrak{so}(3)$, and generators G_1, G_2, G_3 corresponds to the translational components.

The exponential operator to convert from $\mathfrak{se}(3)$ into $SE(3)$ is

$$\exp\left(\begin{bmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{\rho} \\ \mathbf{0} & 0 \end{bmatrix}\right) = \sum_n^\infty \frac{1}{n!} \begin{bmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{\rho} \\ \mathbf{0} & 0 \end{bmatrix}^n, \tag{2.2.30a}$$

$$= I + \begin{bmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{\rho} \\ \mathbf{0} & 0 \end{bmatrix} + \frac{1}{2!} \begin{bmatrix} [\boldsymbol{\omega}]_\times^2 & [\boldsymbol{\omega}]_\times \boldsymbol{\rho} \\ \mathbf{0} & 0 \end{bmatrix} + \frac{1}{3!} \begin{bmatrix} [\boldsymbol{\omega}]_\times^3 & [\boldsymbol{\omega}]_\times^2 \boldsymbol{\rho} \\ \mathbf{0} & 0 \end{bmatrix} + \dots. \tag{2.2.30b}$$

The rotation part $[\boldsymbol{\omega}]_\times$ of the power series is the same of Equation 2.2.18. The translational part's closed-form is

$$V = I + \frac{1}{2!}[\boldsymbol{\omega}]_\times + \frac{1}{3!}[\boldsymbol{\omega}]_\times^2 + \frac{1}{4!}[\boldsymbol{\omega}]_\times^3 + \dots, \tag{2.2.31a}$$

$$= I + [\mathbf{w}]_\times (1 - \cos t) + [\mathbf{w}]_\times^2 (t - \sin t), \tag{2.2.31b}$$

then

$$\exp\left(\begin{bmatrix} [\boldsymbol{\omega}]_\times & \mathbf{t} \\ \mathbf{0} & 0 \end{bmatrix}\right) = \sum_n^\infty \frac{1}{n!} \begin{bmatrix} [\boldsymbol{\omega}]_\times & V\mathbf{t} \\ \mathbf{0} & 0 \end{bmatrix}. \tag{2.2.32}$$

As in the $\mathfrak{so}(3)$, the differential of Exp w.r.t. to a vector $\boldsymbol{\rho}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ is:

$$\begin{matrix} \frac{\partial \operatorname{Exp}(\boldsymbol{\xi}_0)}{\partial \rho_1} = G_1, \frac{\partial \operatorname{Exp}(\boldsymbol{\xi}_0)}{\partial \rho_2} = G_2, \frac{\partial \operatorname{Exp}(\boldsymbol{\xi}_0)}{\partial \rho_3} = G_3, \\ \frac{\partial \operatorname{Exp}(\boldsymbol{\xi}_0)}{\partial \omega_1} = G_4, \frac{\partial \operatorname{Exp}(\boldsymbol{\xi}_0)}{\partial \omega_2} = G_5, \frac{\partial \operatorname{Exp}(\boldsymbol{\xi}_0)}{\partial \omega_3} = G_6 \end{matrix}. \tag{2.2.33}$$

We can then write the approximate derivative of an expression $\operatorname{Exp}(\boldsymbol{\xi})T\mathbf{x}$ where $T = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \in SE(3)$ and a homogeneous point $\mathbf{x}' \in \mathbb{R}^4$ with

$$\begin{aligned} \frac{\partial Exp(\boldsymbol{\xi})T\mathbf{x}}{\partial \boldsymbol{\xi}} &\approx \frac{\partial \operatorname{Exp}(\boldsymbol{\rho}_0)}{\partial \boldsymbol{\xi}_0} T\mathbf{x}', && \text{(2.2.34a)} \\ &\approx \begin{bmatrix} G_1 T\mathbf{x}' & G_2 T\mathbf{x}' & G_3 T\mathbf{x}' & G_4 T\mathbf{x}' & G_5 T\mathbf{x}' & G_6 T\mathbf{x}' \end{bmatrix}, && \text{(2.2.34b)} \\ &\approx \begin{bmatrix} I_{3\times 4} & [T\mathbf{x}']_\times \\ \mathbf{0}_{1\times 4} & \mathbf{0}_{1\times 4} \end{bmatrix}. && \text{(2.2.34c)} \end{aligned}$$

Removing the unnecessary values from the homogeneous matrix and vector, we can rewrite the derivative for the point $\mathbf{x} \in \mathbb{R}^3$ with

$$\frac{\partial Exp(\boldsymbol{\xi})T\mathbf{x}}{\partial \boldsymbol{\xi}} \approx \begin{bmatrix} I_{3\times 3} & [R\mathbf{x} + \mathbf{t}]_\times \end{bmatrix}. \tag{2.2.35}$$

Remembering that the approximation works for small values of $\boldsymbol{\xi}$.

As we did for the $SO(3)$ group, we also define the boxplus operator for the $SE(3)$ group in the same manner

$$T \boxplus \boldsymbol{\xi} \triangleq \operatorname{Exp}(\boldsymbol{\xi})T. \tag{2.2.36}$$

2.3 Pinhole Camera Model

We follow the pinhole camera modeling of projecting 3D points from world space into a 2D image plane. The modeling replicates the physics of most photographic cameras, where rays of reflected light from the world converge into a small pinhole or the camera's center of projection. Those rays that pass through the pinhole will hit the projection plane with the color of its reflected point's surface, forming an image.

In this modeling, a 3D point $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^\intercal$ is transformed into 2D coordinates by dividing its x and y with z and then multiplying them by the distance of the pinhole's center to the projection plane, or namely, the camera's focal length f. The formula for projecting the point $\mathbf{x}$ into 2D with a camera with a focal length f is

$$\begin{bmatrix} u \\ v \end{bmatrix} = \pi(x, y, z)f, \tag{2.3.1}$$

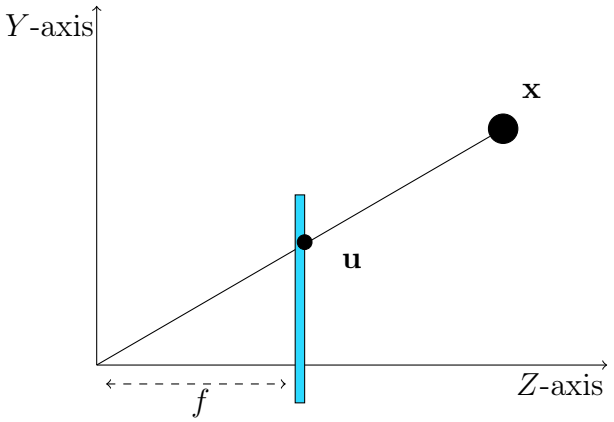


Figure 2.3: The pinhole camera model being viewed from the Y-Z axis of a 3D point $\mathbf{x}$ projected on the 2D point $\mathbf{u}$ on the image plane (blue rectangle). The $\mathbf{u}$ projection is the intersection of the ray from the $\mathbf{x}$ to the camera's projection center $\mathbf{o}$ with the image plane. The focal length f is the distance of the image plane to the projection center. In a real photographic camera, the image plane on the figure would be its lens, and the actual image plane would be before the center of the projection. Nevertheless, as this is a virtual model, we can simplify and treat the image plane in the lens' position.

in which the function π does the division by z,

$$\pi(x, y, z) = \begin{bmatrix} x/z \\ y/z \end{bmatrix}. \tag{2.3.2}$$

Figure 2.3 displays an illustration of the camera projection when looking from the Y-Z-axes.

However, for modeling the projection of real sensors, we must account that their actual pixel sizes are rectangular and not square. Also, the actual center pixel position on the image is not exactly the center of the digital image. For those reasons, computer vision algorithms model those differences between the pinhole model and real sensors with an intrinsic camera matrix K as

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.3.3}$$

where f_x and f_y are the camera's focal length multiplied by, respectively, the sensor's horizontal or vertical pixel sizes. The c_x and c_y are the camera's projection center on the image plane. We assume that the intrinsic camera parameters K of any sensor are always known during this text. That is so because the intrinsic matrix is often supplied by the sensor's manufacturer or found with camera calibration patterns. Using the intrinsic matrix, then the 2D pixel of a 3D point is

$$\begin{bmatrix} u \\ v \end{bmatrix} = \pi(K\mathbf{x}). \tag{2.3.4}$$

The point projection formula can be yet augmented by including the sensor pose matrix $T = \begin{bmatrix} R & \mathbf{t} \end{bmatrix}$ described in Section 2.2. In this way, for converting a point from its world space into camera space and then into 2D image space is given by

$$\begin{bmatrix} u \\ v \end{bmatrix} = \pi(K(R^\intercal \begin{bmatrix} x & y & z \end{bmatrix}^\intercal - R^\intercal \mathbf{t})). \tag{2.3.5}$$

In the previous equation we used the inverse matrix of T, Equation 2.2.5, to highlight that poses are in world space. Hence, its inverses map points into the camera's local space.

When depth information is available, the back projection of any pixel coordinates $\mathbf{u} = \begin{bmatrix} u & v & d \end{bmatrix}^\intercal$, with d being its sensed depth, to camera space is

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \pi^{-1}(K^{-1}\mathbf{u}) = \begin{bmatrix} d(u - c_x)/f_x \\ d(v - c_y)/f_x \\ d \end{bmatrix}, \tag{2.3.6}$$

with the function π^{-1} being the inverse of the z division

$$\pi^{-1}(u, v, d) = \begin{bmatrix} ud & vd & d \end{bmatrix}. \tag{2.3.7}$$

As the backprojection is a common operation for this dissertation, we express it by the function $\Pi(K, \mathbf{u})$, such that it maps $\mathbf{u}$ coordinates into 3D points:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \Pi(K, \mathbf{u}) \tag{2.3.8}$$

.

2.4 Weighted Non-linear Least Squares Problems

When working with problems from areas such as SLAM, robotics, or 3D reconstruction is usual to have to estimate a d-dimensional vector $\mathbf{x}^*$ for minimizing an energy function formed by N sums of weighted squared residual error terms $\mathbf{r_i}$, exemplified bellow

$$E(\mathbf{x}) = \sum_i^N \mathbf{r_i}(\mathbf{x})^\intercal w_i \mathbf{r_i}(\mathbf{x}), \tag{2.4.1}$$

where the scalar w_i is the confidence weight term for a particular residual. In this definition, we are assuming a vector-valued residual function $\mathbf{r_i}$ mapping $\mathbb{R}^d \mapsto \mathbb{R}^z$, for any other size z, but it is also common to use scalar-valued $\mathbb{R}^d \mapsto \mathbb{R}$ without loss of generality on the equations that we will develop next.

A set of residuals can refer to calculations from measurements used to estimate the alignment level during 3D geometry registration, e.g., the difference between points from point clouds, as seen in Section 2.5. However, right now, we refer to them as abstract

measurements and focus on presenting a general framework for solving such weighted non-linear least squares problems.

Finding the parameters $\mathbf{x}^*$, like the ones from Equation 2.4.1, means equalizing the first derivative of E to zero, $\frac{\partial E}{\partial \mathbf{x}} = 0$, solving it to find the extrema points, and testing if they are minima or maxima of the function. Nevertheless, a closed-form solution for finding the extrema points are challenging to achieve in many SLAM routines. Mainly because the residual functions are non-linear due to rotation transformations. A more effective approach is to use iterative numerical methods for finding a minimum starting from an initial guess.

In the areas mentioned earlier, the numerical approaches are often based on second-order optimization algorithms for fast convergence when the answer's dimension d is small. As we estimate only six parameters during this dissertation, we base our routines on the GN second-order optimization algorithm, widely adopted in SLAM [9, 17, 21, 26, 45, 46, 58, 59, 71, 83].

The GN procedure starts with a linearized version of the residual function found with a Taylor Expansion at any initial point $\hat{\mathbf{x}}$:

$$\mathbf{r_i}(\hat{\mathbf{x}} + \Delta\mathbf{x}) \approx \mathbf{r_i}(\hat{\mathbf{x}}) + J_i\Delta\mathbf{x}, \tag{2.4.2}$$

where J_i is a $N \times d$ jacobian matrix of the derivative $\frac{\partial \mathbf{r}_i}{\partial \mathbf{x}}$, or

$$J_i = \begin{bmatrix} \frac{\partial r_{i1}}{\partial x_1} & \frac{\partial r_{i1}}{\partial x_2} & \cdots & \frac{\partial r_{i1}}{\partial x_d} \\ \frac{\partial r_{i2}}{\partial x_1} & \frac{\partial r_{i2}}{\partial x_2} & \cdots & \frac{\partial r_{i2}}{\partial x_d} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_{iz}}{\partial x_1} & \frac{\partial r_{iz}}{\partial x_2} & \cdots & \frac{\partial r_{iz}}{\partial x_d} \end{bmatrix}. \tag{2.4.3}$$

For scaled-valued residuals the jacobian is just a vector $J_i = \begin{bmatrix} \frac{\partial r_i}{\partial x_1} & \frac{\partial r_i}{\partial x_2} & \cdots & \frac{\partial r_i}{\partial x_d} \end{bmatrix}$.

Replacing Equation 2.4.1 with the linearized version of the residuals from Equation 2.4.2 yields:

$$E(\mathbf{x}) \approx \sum_{i}^{N} (\mathbf{r_i}(\hat{\mathbf{x}}) + J_i\Delta\mathbf{x})^\intercal w_i (\mathbf{r_i}(\hat{\mathbf{x}}) + J_i\Delta\mathbf{x}). \tag{2.4.4}$$

Equation 2.4.4 approximates the original $E(\mathbf{x})$ at a point $\hat{\mathbf{x}}$ with a quadratic function, which the reader can visualize as a parabola-like or saddle-like shaped surface. Then the intuition of GN is that such quadratic functions have only one extremum point being also its minimum point. Following that fact, the GN algorithm runs iterations of finding the extrema point $\Delta\mathbf{x}^*$ of the energy function's approximation at $\hat{\mathbf{x}}$, and then update $\hat{\mathbf{x}}$ with the found result

$$\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} + \Delta\mathbf{x}^*, \tag{2.4.5}$$

with each iteration putting $\hat{\mathbf{x}}$ closer to a minimum of the true function. It keeps running this update until a max number of iterations or a minimum mean residual value is met.

To calculate the minimum $\Delta\mathbf{x}^*$ of the approximation in Equation 2.4.4, we equal to

zero its first-order derivative:

$$\frac{\partial \sum_i^N (\mathbf{r_i}(\hat{\mathbf{x}}) + J_i \Delta\mathbf{x}) w_i (\mathbf{r_i}(\hat{\mathbf{x}}) + J_i \Delta\mathbf{x})}{\partial \Delta\mathbf{x}} = 0 \tag{2.4.6}$$

By the sum-rule of derivation, we can expand the errors terms separately to unclutter the derivation:

$$(\mathbf{r_i}(\hat{\mathbf{x}}) + J_i \Delta\mathbf{x}) w_i (\mathbf{r_i}(\hat{\mathbf{x}}) + J_i \Delta\mathbf{x}), \tag{2.4.7a}$$
$$\mathbf{r_i}(\hat{\mathbf{x}}) w_i \mathbf{r_i}(\hat{\mathbf{x}}) + 2\mathbf{r_i}(\hat{\mathbf{x}}) w_i J_i \Delta\mathbf{x} + (J_i \Delta\mathbf{x})^\intercal w_i J_i \Delta\mathbf{x}, \tag{2.4.7b}$$
$$\mathbf{r_i}(\hat{\mathbf{x}}) w_i \mathbf{r_i}(\hat{\mathbf{x}}) + 2\mathbf{r_i}(\hat{\mathbf{x}}) w_i J_i \Delta\mathbf{x} + \Delta\mathbf{x}^\intercal J_i^\intercal w_i J_i \Delta\mathbf{x}, \tag{2.4.7c}$$

rewriting it with the variables H_i, $\mathbf{b}_i$, and c_i:

$$H_i = J_i^\intercal w_i J_i, \tag{2.4.8a}$$
$$\mathbf{b}_i = 2\mathbf{r}_i(\hat{\mathbf{x}}) w_i J_i, \tag{2.4.8b}$$
$$c_i = \mathbf{r}_i(\hat{\mathbf{x}}) w_i \mathbf{r}_i(\hat{\mathbf{x}}), \tag{2.4.8c}$$
$$c_i + 2\mathbf{b}_i \Delta\mathbf{x} + \Delta\mathbf{x}^\intercal H_i \Delta\mathbf{x}, \tag{2.4.8d}$$

with $H_i \in \mathbb{R}^{d \times d}, \mathbf{b}_i \in \mathbb{R}^d, c_i \in \mathbb{R}$. Using the sum rule again, Equation 2.4.6 is rewritten to

$$H = \sum_i^N H_i, \tag{2.4.9a}$$
$$\mathbf{b} = \sum_i^N \mathbf{b}_i, \tag{2.4.9b}$$
$$c = \sum_i^N c_i, \tag{2.4.9c}$$
$$\frac{\partial c + 2\mathbf{b}\Delta\mathbf{x} + \Delta\mathbf{x}^\intercal H \Delta\mathbf{x}}{\partial \Delta\mathbf{x}}, \tag{2.4.9d}$$

for which the solution is

$$\frac{\partial c + 2\mathbf{b}\Delta\mathbf{x} + \Delta\mathbf{x}^\intercal H \Delta\mathbf{x}}{\partial \Delta\mathbf{x}} = 2\mathbf{b} + (H + H^\intercal)\Delta\mathbf{x}. \tag{2.4.10}$$

Since the matrix H is symmetric because Equation 2.4.8a obtains it from a matrix product with its transpose. Then, the term $(H + H^\intercal)$ of Equation 2.4.10 can be replaced with $2H$. Simplifying, it results in:

$$H\Delta\mathbf{x} = -\mathbf{b}, \tag{2.4.11}$$

which we can efficiently solve Equation 2.4.11 with the Cholesky decomposition algorithm for obtaining the update $\Delta\mathbf{x}^*$ [37].

In the Sections 2.4.1, 3.2 and 3.6, we will employ the GN algorithm by replacing the matrix H, vector $\mathbf{b}$, and scalar w_i according to actual residual functions without changing the routine.

Unlike Stochastic Gradient Descent (SGD) used by ML training, the GN algorithm requires fewer iterations to converge at the cost of more memory required by the hessian matrices. Although the latter is not a problem since SLAM optimizes a small set of parameters compared to ML methods, or in some large dimensional problems such as Bundle Adjustment or PGO, the algorithm implementation may take advantage of know sparsity constraints on H matrices. Unfortunately, the main caveat of GN is that its quadratic function approximations are sensitive to initialization at plateau regions, making the approximation into a hyperplane and getting stuck. A manner to improve the ability of getting out of plateau regions is replacing the GN with the Levenberg-Marquardt (LM) algorithm which incorporates the GN and Gradient Descent (GD) according to the approximations of the residual function.

2.4.1 Robust Estimation

Frequently, a subset of residual terms from energy function such as Equation 2.4.1 corresponds to outlier measurements, like false point correspondences introduced by a moving object during capture of point clouds. When such outlier cases happen, even with a large portion of the current estimation producing the correct measurements, it is likely to introduce a sizeable residual error value, deviating the optimization from the actual minimum. To reduce the impact of those circumstances in SLAM, M-estimators have been found [1, 8, 38] an effective way to modulate the effect of high residual terms. In essence, a M-estimator $\mathcal{L}$ controls the cost of high residuals. One example is the Huber loss,

$$\mathcal{L}_\delta(r) = \begin{cases} \frac{r}{2} & \text{if } r \leq \delta \\ 2\sqrt{\delta r} - \delta & \text{otherwise} \end{cases}, \tag{2.4.12}$$

with the δ being a given threshold value. Note that r should be the squared residual, so $\mathcal{L}_\delta(r)$ does not return a negative value.

To use a M-estimator in an energy function, we simply evaluate it for every residual. For instance, we transform an expression like

$$E(\mathbf{x}) = \sum_i^N \mathbf{r}_i(\mathbf{x})^\intercal w_i \mathbf{r}_i(\mathbf{x}) \tag{2.4.13}$$

into

$$E_\mathcal{L}(\mathbf{x}) = \sum_i^N \mathcal{L}\left(\mathbf{r}_i(\mathbf{x})^\intercal w_i \mathbf{r}_i(\mathbf{x})\right). \tag{2.4.14}$$

To illustrate the attenuation of the Huber function in relation to the raw residual value, we show a comparison in Figure 2.4 with a squared residual.

To use M-estimator function with GN optimization algorithm, we can calculate the new energy function derivative

$$\frac{\partial \mathcal{L}(\mathbf{r}_i(\mathbf{x})^\intercal w_i \mathbf{r}_i(\mathbf{x}))}{\partial \mathbf{x}}, \tag{2.4.15}$$

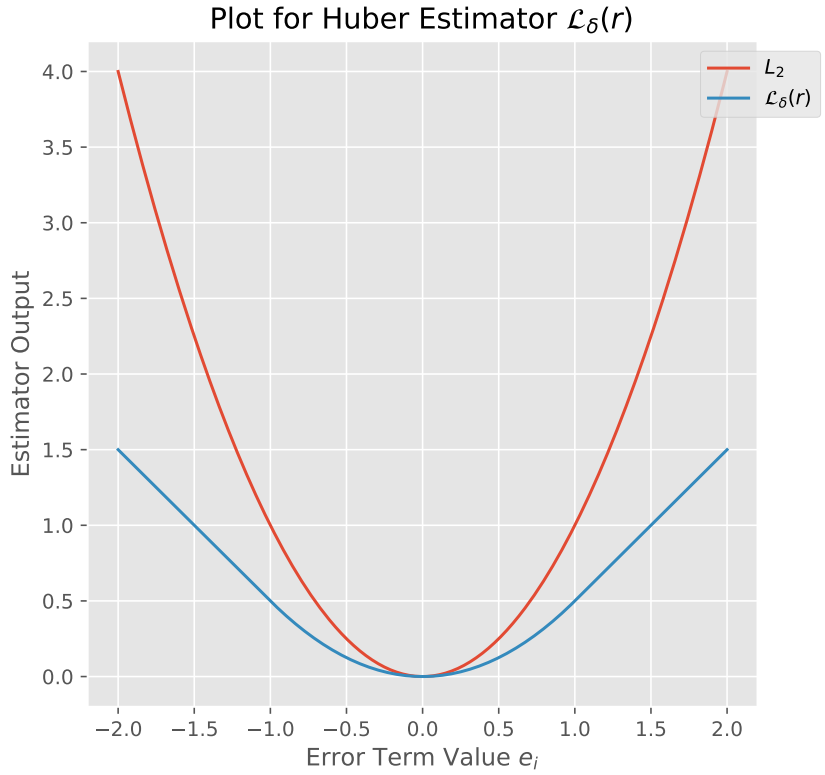


Figure 2.4: The Huber estimator (blue line) and the actual output of the error term (red). The Huber loss diminishes larger errors caused by outliers.

which is expanded with the chain rule

$$\frac{\partial \mathcal{L}(\mathbf{r}_i(\mathbf{x})^\intercal w_i \mathbf{r}_i(\mathbf{x}))}{\partial \mathbf{r}_i(\mathbf{x})^\intercal w_i \mathbf{r}_i(\mathbf{x})} \frac{\partial \mathbf{r}_i(\mathbf{x})^\intercal w_i \mathbf{r}_i(\mathbf{x})}{\partial \mathbf{x}}, \tag{2.4.16}$$

resulting that the first partial derivative term is just a scalar value that gets multiplied with the same derivatives from the original GN's quadratic residual. As it is a simple multiplier, we can mix the derivative's value with the existing weight factors,

$$w_i \leftarrow w_i \frac{\partial \mathcal{L}(\mathbf{r}_i(\mathbf{x})^\intercal \mathbf{r}_i(\mathbf{x}))}{\partial \mathbf{x}}, \tag{2.4.17}$$

and continue the routine exactly as in Section 2.4 without changing the rest of the equations.

During our experiments on dense registration from Chapter 3, we use the Huber loss of Equation 2.4.12, having the following derivative

$$\frac{\partial \mathcal{L}_\delta(r)}{\partial r} = \min\left(1, \sqrt{\frac{\delta}{r}}\right). \tag{2.4.18}$$

While for estimating the registration of loop closures with the algorithm proposed by

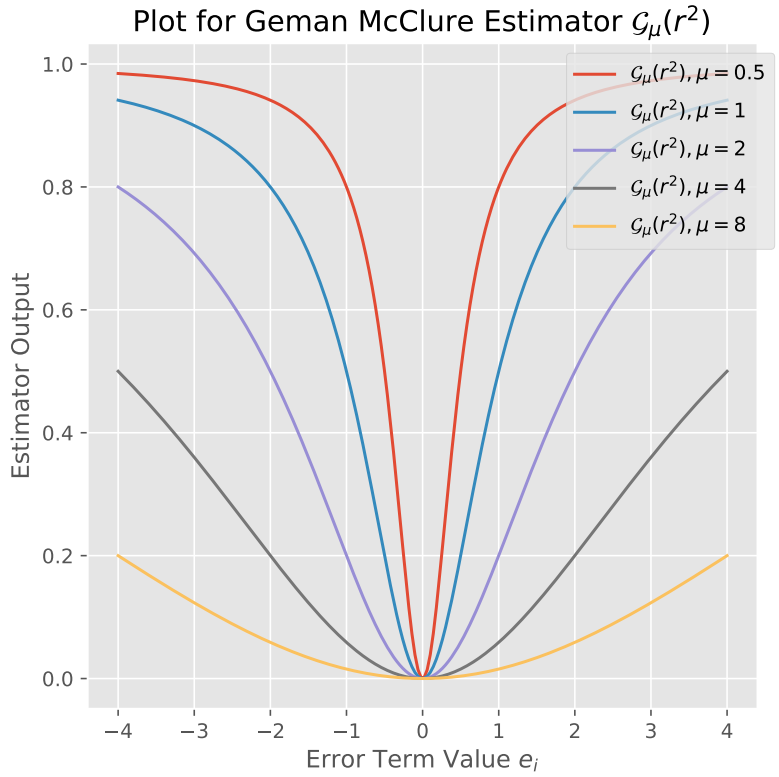


Figure 2.5: The Geman-McClure robust estimator with different settings of the parameter μ.

Zhou et al. [87], their procedure uses the Geman-McClure estimator

$$\mathcal{G}_\mu(r) = \mu \left(1 - \frac{1}{1 + r/\mu}\right), \tag{2.4.19}$$

where μ controls the effect of residuals on the object, see Figure 2.5 for its attenuation factor with different values of μ.

2.5 Point Cloud Registration

Registration of 3D geometry is a common operation on SLAM. This section starts reviewing the ICP algorithm for aligning dense and already close point clouds. After, we review techniques based on feature matching: the Kabsch-Umeyama [80] (Kabsch-Umeyama) and Fast Global Registration [87] (FGR) algorithms that can register severe miss-aligned point clouds. Although the algorithms are defined with point cloud representation, they might be applied regardless of whether 3D data was obtained as depth sensors, 3D reconstruction, or 3D Computer-Aided Design (CAD) models.

Later on Chapter 3, we apply ICP for computing RGB-D odometry and submap alignment, and the FGR for detecting loop closures.

2.5.1 The Iterative Closest Points Algorithms

The most popular form of dense alignment is the family of ICP algorithms for geometry consistency between points clouds. The ICP algorithms work in iterations consisted of two steps. According to a current transformation estimative T_{st}, the first step finds the closest point $\mathbf{q}_i$ in the target set for every $i$nth point $\mathbf{p}_i$ in the source set. The second step updates the current transformation estimative according to a cost function to move closer to their actual neighbors in the next iteration. Among the cost function to guide ICP, the first one is the distance between points $\mathbf{p}_i$ and $\mathbf{q}_i$, known as point-to-point cost metric [4]. Shortly after, Chen and Medioni [16] propose the point-to-plane cost metric for faster convergence. It offers a smoother error residual since the source points "slide" over the target surface. The point-to-plane cost is

$$E_{\text{icp}}(T) = \sum_{i}^{N} r_i(T) w_k r_i(T), \tag{2.5.1a}$$

$$r_i(T) = \mathbf{n}_i{}^{\intercal}(T\mathbf{p}_i - \mathbf{q}_i), \tag{2.5.1b}$$

meaning that conforming to the candidate T_{st}, the term r_i is the distance of the source point $\mathbf{p}_i$ from the target's surface that its corresponding closest point $\mathbf{q}_i$ belongs. The vectors $\mathbf{n}_i$ are the normal of the surface associated with the target points $\mathbf{q}_i$. The normal vectors can be obtained from depth images, point clouds, or mesh geometries. The scalar w_i is a weighting factor of the matching's confidence. To find the closest points quickly, we can query them from a KD-Tree. Another way, when the point clouds originated from RGB-D images, is to project and back project the points to find the nearest ones immediately. This later technique is described in Section 3.2.

In order to improve the original point-to-plane cost function in Equation 2.5.1 against outlier associations, we can complement it with a robust error score, like the Huber loss $\mathcal{L}_\delta$

$$E_{\text{icp}}(T) = \sum_{i}^{N} \mathcal{L}_\delta(r_i(T) w_k r_i(T)), \tag{2.5.2a}$$

$$r_i(T) = \mathbf{n}_i{}^{\intercal}(T\mathbf{p}_i - \mathbf{q}_i), \tag{2.5.2b}$$

Next, we show how to minimize the non-linear least-squares problem of Equation 2.5.1 with the GN optimization framework described in Section 2.4. GN is a popular choice for ICP, but other authors may choose to use LM or even first-order gradient descent.

We start the actual minimization of Equation 2.5.2 by establishing its quadratic approximation:

$$E(T \boxplus \boldsymbol{\xi}) \approx \sum_{i}^{N} (r_i(T \boxplus \boldsymbol{\xi}) + J_i\boldsymbol{\xi}) w_i (r_i(T \boxplus \boldsymbol{\xi}) + J_i\boldsymbol{\xi}), \tag{2.5.3}$$

with $T = \begin{bmatrix} R \mid \mathbf{t} \end{bmatrix}$. The approximation in Equation 2.5.3 replaces the plus (+) in Equation 2.4.2 with the box plus ($\boxplus$) operator. The change is necessary because we are opti-

mizing over the $SE(3)$ manifold, so the state update operation must keep a valid member of the manifold. Consequently, this also results that we can parameterize the state update with $\boldsymbol{\xi} \in \mathfrak{se}(3)$, being beneficial because we only need to estimate 6 parameters instead of 12 of the full matrix. To compute the Jacobian $J_i \in \mathbb{R}^6$ of the point-to-plane distance r_i

$$J_i = \frac{\partial \mathbf{n}_i{}^\intercal \left(T \boxplus \boldsymbol{\xi} \mathbf{p}_i - \mathbf{q}_i\right)}{\partial \boldsymbol{\xi}}, \tag{2.5.4a}$$

$$= \frac{\partial \mathbf{n}_i{}^\intercal T \boxplus \boldsymbol{\xi} \mathbf{p}_i}{\partial \boldsymbol{\xi}} - \frac{\partial \mathbf{n}_i{}^\intercal \mathbf{p}_i}{\partial \boldsymbol{\xi}}, \tag{2.5.4b}$$

$$= \frac{\partial \mathbf{n}_i{}^\intercal T \boxplus \boldsymbol{\xi} \mathbf{p}_i}{\partial \boldsymbol{\xi}}. \tag{2.5.4c}$$

Using the chain rule, we obtain

$$J_i = \frac{\partial \mathbf{n}_i{}^\intercal T \boxplus \boldsymbol{\xi} \mathbf{p}_i}{\partial T \boxplus \boldsymbol{\xi} \mathbf{p}_i} \frac{\partial T \boxplus \boldsymbol{\xi} \mathbf{p}_i}{\partial \boldsymbol{\xi}}. \tag{2.5.5}$$

For readability, let $\mathbf{x} = T \boxplus \boldsymbol{\xi} \mathbf{p}_i$, then the first term of the jacobian is

$$\frac{\partial \mathbf{n}_i{}^\intercal T \boxplus \boldsymbol{\xi} \mathbf{p}_i}{\partial T \boxplus \boldsymbol{\xi} \mathbf{p}_i} = \frac{\partial \mathbf{n}_i{}^\intercal \mathbf{x}}{\partial \mathbf{x}} = \mathbf{n}_i{}^\intercal. \tag{2.5.6}$$

Equation 2.2.35 gives the analytical answer for the second term

$$\frac{\partial T \boxplus \boldsymbol{\xi} \mathbf{p}_i}{\partial \boldsymbol{\xi}} = \frac{\partial \operatorname{Exp}(\boldsymbol{\xi}) T \mathbf{p}_i}{\partial \boldsymbol{\xi}} = \begin{bmatrix} I_{3\times 3} & -[R\mathbf{p}_i + \mathbf{t}]_\times \end{bmatrix} \tag{2.5.7}$$

Replacing the terms of Equation 2.5.5 with Equation 2.5.6 and Equation 2.5.7:

$$J_i = \mathbf{n}_i{}^\intercal \begin{bmatrix} I_{3\times 3} & -[R\mathbf{p}_i + \mathbf{t}]_\times \end{bmatrix} \tag{2.5.8}$$

Let $\mathbf{v} = -\mathbf{n}_i \times (R\mathbf{p}_i + \mathbf{t})$ and $\mathbf{n}_i = \begin{bmatrix} n_1 & n_2 & n_3 \end{bmatrix}$, we can rewrite it

$$J_i = \begin{bmatrix} n_1 & n_2 & n_3 & v_1 & v_2 & v_3 \end{bmatrix}. \tag{2.5.9}$$

Using the analytical jacobian in Equation 2.5.9, we can then plug J_i into the terms H_{geom} and $\mathbf{b}_{\text{geom}}$

$$H_{\text{geom}} = \sum_i^N J_i^\intercal w_i J_i, \tag{2.5.10a}$$

$$\mathbf{b}_{\text{geom}} = \sum_i^N r_i(T) w_i J_i, \tag{2.5.10b}$$

and we set w_i to value of the Huber loss derivative

$$w_i = \mathcal{L}'_\delta(r_i(T)^2) \tag{2.5.11}$$

The optimal $\boldsymbol{\xi}^*$ for only the geometric term can be found using the Cholesky decom-

Algorithm 1 Point-to-plane ICP

Input: Source points set P; target points set Q; initial guess transformation matrix $\hat{T} \in SE(3)$.

$T \leftarrow \hat{T}$
while maximum number of iterations **do**
  let $R, \mathbf{t} = T$
  for every point $\mathbf{p} \in P$ **do**
    $\mathbf{q}_i = \text{find\_nearest\_neighbor}(Q, R\mathbf{p} + \mathbf{t})$ ▷ Finds the nearest point in the set Q
    $\mathbf{n}_i =$ normal associated with $\mathbf{q}_i$
    $J_i \leftarrow \begin{bmatrix} n_1 & n_2 & n_3 & -\mathbf{n}_i \times (R\mathbf{p}_i + \mathbf{t}) \end{bmatrix}$
    $r_i \leftarrow (\mathbf{n}_i^\intercal (R\mathbf{p}_i + \mathbf{t} - \mathbf{q})i))^2$
    $w_i \leftarrow \mathcal{L}_\delta(r_i)$
  end for

  $H \leftarrow \sum_i J_i^\intercal w_i J_i$
  $\mathbf{b} \leftarrow \sum_i r_i w_i J_i$

  $\boldsymbol{\xi}^* \leftarrow \text{Cholesky}(H, \mathbf{b})$
  $T \leftarrow T \boxplus \boldsymbol{\xi}^*$
end while
output T

position and integrated on the current estimative T with the boxplus operator:

$$T \leftarrow T \boxplus \boldsymbol{\xi}^* \tag{2.5.12}$$

After an update, the algorithm starts another iteration of approximation and update until a max number of iterations is met. The ICP for only the geometric term is presented in Algorithm 1.

It is important to note that the GN is naturally sensitive to poor initial parameters. Its estimation can not reach a good solution if any unrelated geometry regions have similar properties that pull the optimization to a poor local minimum. To avoid local minima, authors often try to use the result of sparse registration as an initial guess [21, 30] or the use of gaussian pyramids [27, 28, 45, 58, 71, 83, 88].

2.5.2 Sparse methods

This section explains the Kabsch-Umeyama and FGR, which work by computing the registration from sparse point matches, rendering them invariant to initial transformation and usable for loop closure detection. That is, similarly to ICP, the goal of those algorithms is to estimate a rigid transformation $\begin{bmatrix} R \mid \mathbf{t} \end{bmatrix}$ which best aligns two sets of paired points $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\}$ and $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N\}$ where their index are ordered by correspondence. The point correspondence mechanism may be carried by matching 3D points according to the local similarity of descriptors associated with each one. Such local features may be extracted from methods like Oriented FAST and rotated BRIEF [66] (ORB), SIFT, or

FPFH.

Kabsch-Umeyama

The Kabsch-Umeyama is one of the simplest methods. It first estimates the rotation matrix R by finding the rotation matrix that maximizes the trace of the points' covariance matrix, which minimizes the covariance values that are not in the diagonal. Meaning that, given the point covariance matrix

$$\text{cov}(\mathbf{p}, \mathbf{q}) = \sum_{i}^{N} \mathbf{p}_i^\intercal \mathbf{q}_i, \tag{2.5.13}$$

any matrix such that must have highest covariances for $\text{cov}(x, x)$, $\text{cov}(y, y)$ and $\text{cov}(z, z)$ coordinates than for others like $\text{cov}(x, y)$ or $\text{cov}(z, y)$ for representing an optimal rotation alignment. Alternatively, the Kabsch-Umeyama can be proof by minimizing the average distance error

$$\arg\min_{R} \sum_{i}^{N} (Rp_i - q_i)^T (Rp_i - q_i) \tag{2.5.14}$$

Computing its first derivative

$$\begin{gathered} \frac{\partial \sum_i^N (Rp_i - q_i)^T (Rp_i - q_i)}{\partial R}, \\ \frac{\partial \sum_i^N p_i^T R^T R p_i^T - p_i^T R^T q_i - q_i^T R p_i + q_i^T q_i}{\partial R}, \\ \sum_{i}^{N} p_i^T R^T q_i - q_i^T R p_i = \sum_{i}^{N} -2 p_i^T R^T q_i. \end{gathered} \tag{2.5.15}$$

Minimizing Equation 2.5.15 is equivalent to maximizing

$$\arg\max_{R} \sum_{i}^{N} p_i^T R^T q_i. \tag{2.5.16}$$

We can find the optimal matrix for maximizing Equation 2.5.16 using the SVD decomposition $\sum_i^N p_i^T q_i = USV^\intercal$, resulting that the optimal rotation matrix is $R = VU^\intercal$ [25]. Furthermore, once we have the rotation R, we find the vector $\mathbf{t}$ by translating the mass centers:

$$\mathbf{t} = R^* \bar{p} - \bar{q}, \tag{2.5.17}$$

where $\bar{p}$ and $\bar{q}$ are the center of masses or simple their mean points:

$$\bar{p} = \frac{1}{N} \sum_{i}^{N} p_i \qquad \bar{q} = \frac{1}{N} \sum_{i}^{N} q_i \tag{2.5.18}$$

For making more robust to outlier associations, the Kabsch-Umeyama is often run with Random Consensus Sampling (RANSAC), so it can detect and remove outliers by running the Kabsch-Umeyama multiple times with different correspondences subsets.

2.5.3 Fast Global Registration

For robustness to outlier proposes, the Kabsch-Umeyama algorithm is run with the RANSAC framework. However, besides introducing indeterminacy to the estimation, the RANSAC's random process is slower since multiple executions are necessary to improve the answer.

As a faster alternative algorithm, Zhou et al. [87] proposed the FGR algorithm, which filters outliers by minimizing the following cost function with the robust Geman-McClure estimator:

$$E\left(\left[R \mid \mathbf{t}\right]\right) = \sum_{i}^{N} \mathcal{G}_{\mu}\left(\|R\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\|^2\right) \tag{2.5.19}$$

where $\mathcal{G}_{\mu}$ is the Geman-McClure estimator parameterized with a given μ.

Nevertheless, rather than directly estimating Equation 2.5.19, the authors noted that is more comprehensive and faster to execute when we formulate it with line processes. Line processes refers, in our context, in including a scalar set $L = \{l_i\}$ expressing the belief of every $i$nth correspondence between points $\mathbf{p}_i$ and $\mathbf{q}_i$ being true. Each l_i goes from 0–1 indicating either less or more emphasis on the distance between the $R\mathbf{p}_i + \mathbf{t}$ and $\mathbf{q}_i$. In that way, under line processes, we estimate not only the optimal pose $\left[R \mid \mathbf{t}\right]$, but also the optimal set L

$$E\left(\left[R \mid \mathbf{t}\right], L\right) = \sum_{i}^{N} l_i \|R\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\|^2 + \sum_{i}^{N} \Psi(l_i). \tag{2.5.20}$$

To avoid just assigning 0 to every l_i, hence obtaining 0 energy during minimization, Zhou et al. [87] express the function $\Psi(l_i)$ to penalize lower values of l_i as the following

$$\psi(l_i) = \mu(\sqrt{l_i} - 1)^2, \tag{2.5.21}$$

where μ is the same parameter of the Geman-McClure estimator. The function Ψ represents a curve in such way that it outputs higher values when the l_i inputs are closer zero, therefore penalizing the marking the correspondence as outlier, and lower ones when l_i are closer to one. Optimizing Equation 2.5.20 is efficiently implemented using the GN method, in which we can at each step separately estimate first the pose $\left[R \mid \mathbf{t}\right]$ and then all l_i, which have the following derivative:

$$\frac{\partial E}{\partial l_i} = \|R\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\|^2 + \mu\frac{\sqrt{l_i} - 1}{\sqrt{l_i}}, \tag{2.5.22}$$

then setting the first derivative to zero and solving for l_i

$$l_i = \left(\frac{\mu}{\mu + \|R\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\|}\right)^2. \tag{2.5.23}$$

Where the method for updating $\left[R \mid \mathbf{t}\right]$ during one GN step is similar to the ICP in Section 2.5.

We may observe what is known as the Black-Rangarajan [5] duality between robust estimation and line process by substituting the result of l_i from Equation 2.5.23 into Equation 2.5.20 which then becomes Equation 2.5.19, demonstrating the equivalence of the optimization of both energy functions

In the methodology Chapter 3, we use the FGR along with a heuristic for determining alignment success for detecting loop closures.

2.6 Visual SLAM

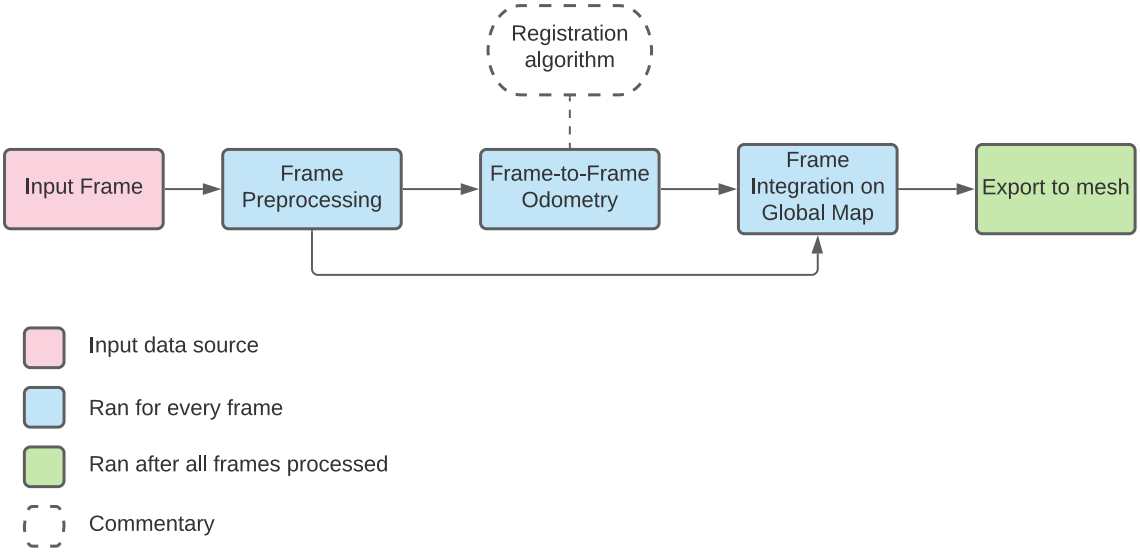
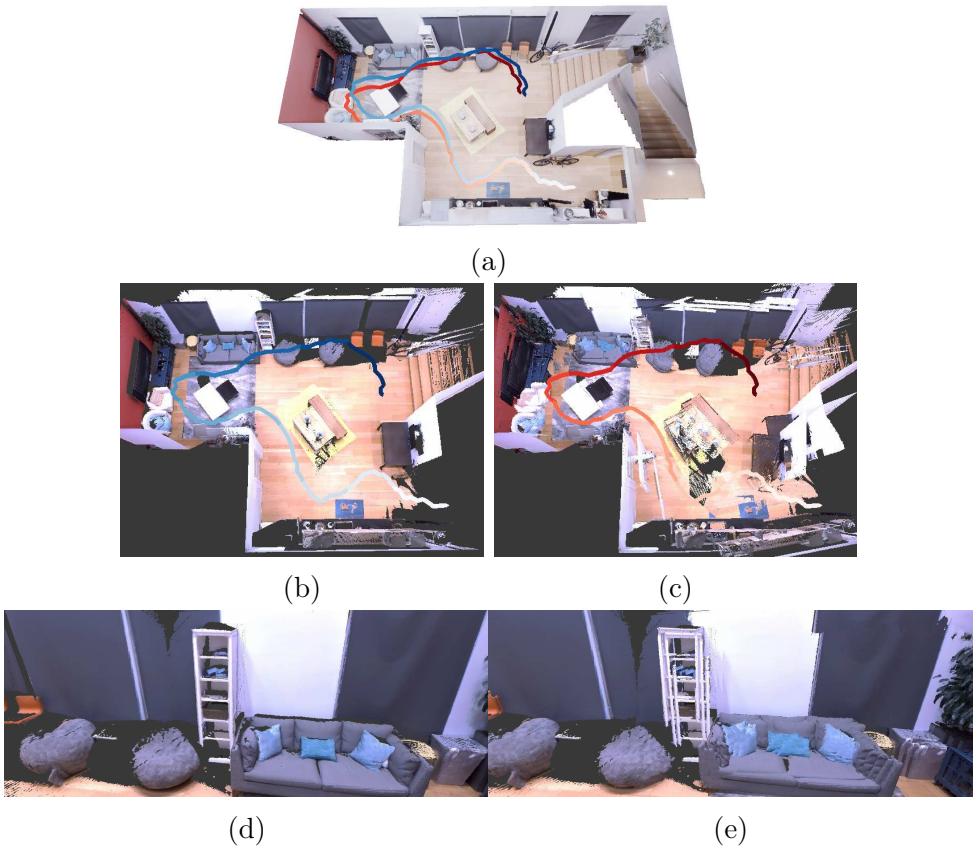


Figure 2.6: A Naive Visual SLAM pipeline. An input source feeds the pipeline with newly acquired frames. The frames can undergo a preprocessing stage to alleviate noise. After, the pipeline uses frame-to-frame odometry to integrate relative pose estimation into a trajectory, but as we will exam, this suffers from severe drift, requiring global alignment routines. Once the frame pose is known, any 3D geometry may be integrated into a global map. This operation combines previous 3D information with the newly received ones to refine the global map. In the end, once all frames are processed, the user may export the final map into a mesh.

As stated in the introduction, the goal of SLAM is to find the camera trajectory and the scene's mapping, as show in Figure 1.3. First, let examine the first part of the problem, the VO, or finding the camera's poses $\{P_1, P_2, \ldots, P_N\}$ throughout the N frames of the stream. Any pose P_i of the camera at the frame i can be decomposed into the accumulation of relative transformations of previous frames: $P_i = IT_{21} \ldots T_{(i-1)(i-2)}T_{i(i-1)}$. That is, each $T_{i(i-1)}$ is a relative $SE(3)$ transformation from a pose P_i to its previous one P_{i-1}. A pose P_i can then be stated recursively

$$P_i = \begin{cases} P_{i-1}T_{i(i-1)} & i \geq 0 \\ I & i = 0 \end{cases}. \tag{2.6.1}$$

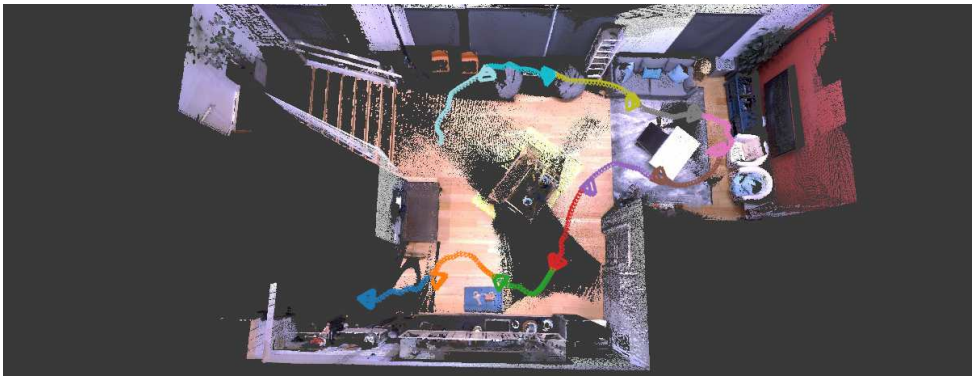
The geometric interpretation is that the points from the first frame are oriented at the origin of the coordinate system, or the identity matrix I. In essence, a VO algorithm can align the frames in respect to their previous one using a registration algorithm, and hence finding the transformations $T_{i(i-1)}$. And by accumulating them, the VO algorithm estimate the absolute poses P_i. For the second part, constructing the mapping, in the case of monocular inputs, it is possible to estimate the map's elements by triangulating the intersection of rays originating from pixel correspondences across the frames. However, for our case of RGB-D input, the system can back project the depth images into point clouds (or other representations like surfels or volumes), position them in the world space according to the cameras' poses, and fuse them all into a single map.



(a) (b) (c) (d) (e)

Figure 2.7: Tracking drift is the problem of algorithms predicting sensor poses that gradually deviate from the actual trajectory. (a) Input scene overlaid with the ground truth and a drifted trajectory. Whiter colors mean near the start of the trajectory, and darker colors are closer to the motion's end. (b) The resulting scene map using the correct trajectory. (c) The resulting scene map using the ill-estimated trajectory. Note the various miss alignments. (d) Correct alignment. (e) Misalignment of geometry caused by track drifting.

Given this overall function of a naive SLAM, with its parts shown in Figure 2.6, aligning pairs of adjacent frames and accumulating their relative transformations as in Equation 2.6.1 would be a straightforward but ineffective solution due to a problem known as tracking drift. Algorithms for registering two frames like the ones of Section 2.6.1 will inevitably add small errors at every estimation due to sensor noise, miss-associations



(a)

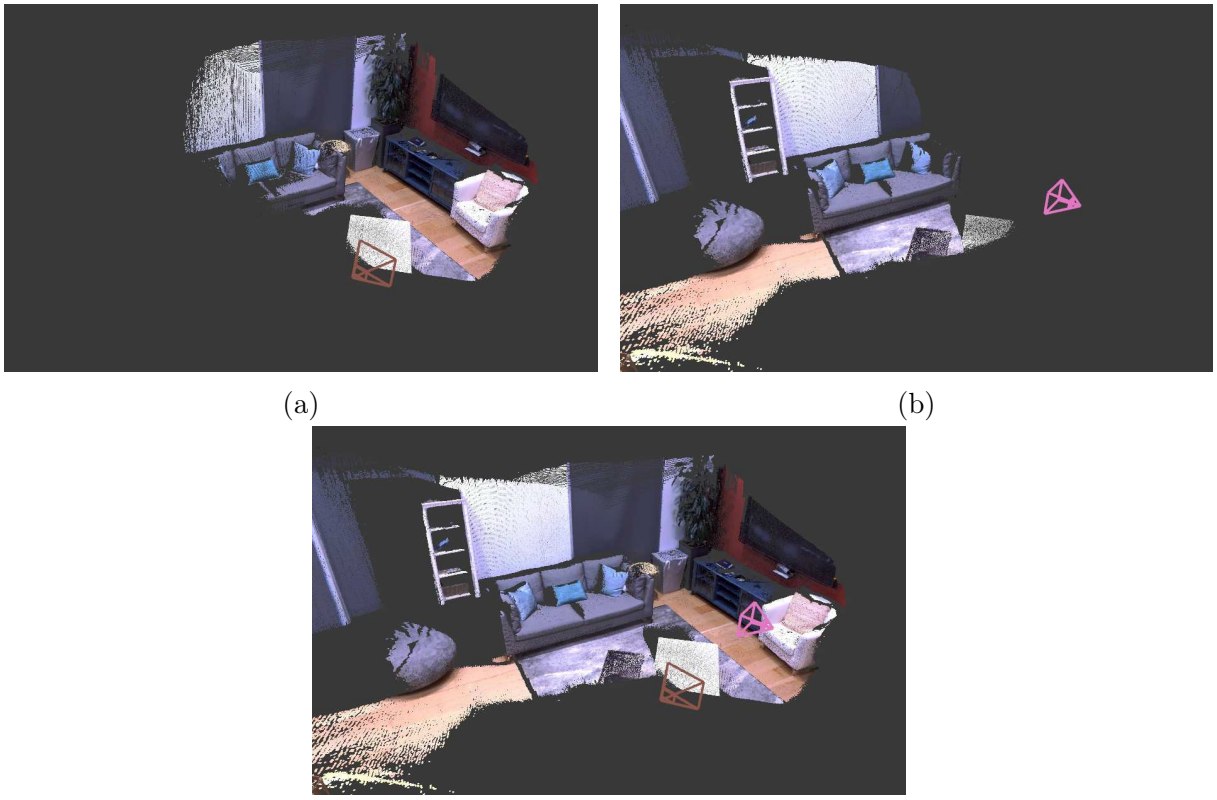
(b)

Figure 2.8: (a) A camera trajectory and its frames and keyframes (larger wireframe drawings). We mark the frames with the same colors as their parent keyframes. (b) Closer view.

of features, or just lack of information in the scene (or the aperture problem). Those small errors will accumulate at every estimation $T_{i(i-1)}$, and in few dozens of frames, the trajectory will likely deviate from the actual one. Figure 2.7 illustrate how tracking drift affects the mapping.

One of the first strategies to mitigate the tracking drift problem is submapping. The camera tracking drift is unnoticeable for small frame intervals even when using accumulation-based frame-to-frame registration algorithms [17]. With that fact in mind, various authors [15, 17, 41, 45, 54] proposed methods where the trajectory is subdivided into keyframes, with each interval between them contributes with a submap. Figure 2.8 shows a trajectory with keyframes along the frames' poses in their intervals. Figure 2.9 shows two submaps that we produce from the frames in the interval between their keyframes. Later the submaps are aligned, and being the fusion of multiple frames, they tend to present less noise and completer geometry, which an additional registration may improve the one found during VO.

Submapping alone only reduce the error for small trajectories and can be better applicable with dense mapping. Although, to overcome the tracking drift, we require optimization routines to align multiple frames, keyframes or submaps globally. One example of such method is Bundle Adjustment (BA), often used in structure from motion. It optimizes the poses and mapping by minimizing the error between 2D reprojection of



(a) (b)

(c)

Figure 2.9: (a) A submap of the scene, its keyframe's pose is shown in Brown color. (b) Another neighbor submap. Its starting camera pose is shown in Pink. (c) Both submaps aligned together

estimated 3D points and their original 2D keypoints coordinates from the frames. Another example is the PGO, which optimizes the poses by maximizing the consistencies between odometry and the found loop closure transformations. Usually, those global optimization methods are not invoked to estimate every frame's parameters; otherwise, the processing would be computationally expensive for bigger streams. Instead, besides of submaps-based input for global optimization [17, 45], other authors [21, 56, 57] select a few keyframes to optimize the poses and map globally. Consequently, the non-keyframes in the intervals are modified according to the global result.

However, the weak estimation from VO algorithms remain important since they will serve as initial state for the global optimization algorithms start from, that without it would render a massive parameter space to search. This framework of using estimations from VO and then globally optimizing them reflected that SLAM system architectures are typically divided into front-end and back-end. The front-end is responsible for performing initial guesses for the trajectory and the map using the sensors' measurements directly. It is also responsible for establishing other measurements, such as detecting loop closures, landmarks, key points, and correspondences. The back-end is responsible for the global optimization routine using the inputs from the front-end to update the poses and map into the state with the most probability. We describe the workings of two classic back-end procedures: BA and PGO, in respectively, Sections 2.6.3 and 2.6.4.

2.6.1 Visual Odometry

The central part of the front-end is the VO estimation. The methods that accumulate relative transformation in the same fashion of Equation 2.6.1 from pairs of consecutive frames are referred to as frame-to-frame tracking. Another modality is frame-to-model tracking [58, 83]: the front-end register the sensor's input w.r.t. to the current reconstructed map representation. Frame-to-model tracking is more advantageous because it leverages less noise and more complete geometry of the mapping than what single frame information may offer.

Engel et al. [27] categorize the VO methods regarding if they consider dense or sparse set of pixels; and if they use pixels' color or intensity values or prefer indirect descriptors to associate points in images. The two more relevant category permutations are indirect-sparse and dense-direct.

Indirect-sparse are methods that infer the relative $SE(3)$ transformation of two cameras using sparse keypoints correspondences pairs found by image descriptor matchings [53, 66]. Often, the keypoint correspondences serve as inputs for estimation of the essential matrix $\mathbf{E}$ (calibrated case) [60] or of the fundamental matrix $\mathbf{F}$ (uncalibrated case) [52] that relates the epipolar geometry of the two camera poses. With one of those matrices, the rotation matrix R and the translation vector $\mathbf{t}$ composing a relative transformation can be extracted. If the 3D positions of the key points are known, then the transformation can alternatively be estimated by minimizing least-squares differences of the keypoint correspondences [25]. Nevertheless, all the algorithms require only a few correspondences that are often selected in conjunction with RANSAC to remove outliers [56, 57].

Direct-dense tracking refers to methods that use all valid pixels to estimate the tracking without converting the pixels into an intermediate descriptor. For instance, Steinbrücker et al. [75] proposed to jointly optimize the photometric alignment cost in addition to the ICP's geometric cost, applied on many frame-to-frame or frame-to-model systems [17, 21, 45, 46, 71, 83]. The photometric consistency obtained by a candidate estimative $T_{st} = \begin{bmatrix} R \mid \mathbf{t} \end{bmatrix}$ between the pixel intensities at coordinates $\mathbf{u}$ of a source image I^s w.r.t. to their corresponding projection in a target image I^t can be modelled by us as

$$e_{\mathbf{u}} = I^s(\mathbf{u}) - I^t(\pi(KT_{s,t}\Pi(K, \mathbf{u}))), \tag{2.6.2a}$$

$$\arg\min_{T_{st}} \sum_{\mathbf{u}}^{I^s_\Omega} e_{\mathbf{u}} w_{\mathbf{u}} e_{\mathbf{u}}. \tag{2.6.2b}$$

The term $e_{\mathbf{u}}$ is the residual photometric error of the estimative T_{st}. It projects a pixel coordinate $\mathbf{u}$ from the source image into the target with the expression $\pi(KT_{st}\Pi(K, \mathbf{u}))$. That is, it first back-projects $\mathbf{u}$ into its camera 3D space, then transforms it with the estimative $T_{s,t}$, and reprojects on the target's image. The back-projection requires that $\mathbf{u}$ also have a depth value d associated with it, requiring a depth image. The scalar $w_{\mathbf{u}}$ is a confidence weight that the mapped pixels on both images are representations of the same point, which may be affected by occlusions or wrong transformation estimation. For instance, its value may be a difference measure of the intensities of $\mathbf{u}$ at the source image

and its projection on the target.

A monocular system may use this kind of dense photometric consistency like in LSD-SLAM [26]. After initial alignments from a sparse indirect algorithm, LSD-SLAM infers the pixels' d coordinate with the stereo match of images from different frames.

2.6.2 Uncertainty Estimation

Estimating the uncertainty of estimations or measurements is a fundamental feature for SLAM's front-ends because various methods for tuning the localization on back-ends rely on modifying the poses according to the degree to which their parameters can be assumed to be true. Good examples are the Kalman filtering or the PGO techniques, the latter being used on our system. They use the uncertainties of relative transformations between estimated sensor poses either acquired from odometry or loop closures to infer the most likely trajectory. In our case, the target is to estimate the uncertainty from the results of the 3D geometry alignment algorithms such as ICP, being, as will be seen, a different process than with physical sensor of wheel metering. The uncertainty of an estimated rigid motion parameters $\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\rho} & \boldsymbol{\omega} \end{bmatrix}$ with $\boldsymbol{\rho} \in \mathbb{R}^3, \boldsymbol{\omega} \in \mathbb{R}^3$, either from a physical sensor or an algorithm is usually expressed in the form of covariance matrices,

$$\operatorname{cov}(\boldsymbol{\xi}) = \begin{bmatrix} C_{\boldsymbol{\rho}\boldsymbol{\rho}} & C_{\boldsymbol{\rho}\boldsymbol{\omega}} \\ C_{\boldsymbol{\omega}\boldsymbol{\rho}} & C_{\boldsymbol{\omega}\boldsymbol{\omega}} \end{bmatrix}. \tag{2.6.3}$$

That is, submatrices $C_{\boldsymbol{\rho}\boldsymbol{\rho}}, C_{\boldsymbol{\omega}\boldsymbol{\omega}} \in \mathbb{R}^{3\times3}$ are, respectively, the covariance for translation and rotation parameters, and submatrices $C_{\boldsymbol{\rho}\boldsymbol{\omega}}$ and $C_{\boldsymbol{\omega}\boldsymbol{\rho}}$ are their cross-covariances.

To compute the covariance, we need an unbiased estimator of $\boldsymbol{\xi}$, which we denoted by its expected value $\mathbb{E}[\boldsymbol{\xi}]$, then

$$\operatorname{cov}(\boldsymbol{\xi}) = \mathbb{E}[(\boldsymbol{\xi} - \mathbb{E}[\boldsymbol{\xi}])(\boldsymbol{\xi} - \mathbb{E}[\boldsymbol{\xi}])^\intercal]. \tag{2.6.4}$$

For physical odometry devices like wheel sensors or Inertial Measurement Units (IMUs), a covariance matrix can be calibrated once using ground truth trajectories or from other forms of high precision sensing that provide the true parameters $\bar{\boldsymbol{\xi}}$ for the expected value $\mathbb{E}[\boldsymbol{\xi}] = \bar{\boldsymbol{\xi}}$ [11, 35, 49].

However, contrary from physical sensors, the result of algorithms dramatically depends on the inputted measurements. For example, wheel sensors will likely suffer slight variations according to the terrain. Whereas visual inputs can easily change. Let consider small lobby and corridor scenes. In those situations, the uncertainty on those ICP's answers will be different because the corridor scene will provide more information from the walls since they are closer to the sensor. For that reason, we must estimate the covariance for each call taking in account the inputs. One class of methods for estimating such type of covariance scenario where a registration algorithm aligns a source point cloud into a target one is to sample multiple executions of it with randomly generated arguments. On this estimation technique, the random arguments are usually initial transformation perturbations or generated noisy versions of the target point cloud. Often with the aid of

Monte-Carlo simulation, they obtain a numeric approximation of the true expected value $\mathbb{E}[\boldsymbol{\xi}]$ for plugging it on Equation 2.6.4.

Although providing good results, techniques based on numeric simulation are inefficient for real-time applications, making authors [2, 10, 63] look for closed-form solutions of an algorithm's covariance. Since being the most used method for point cloud alignment in robotics, the closed-form covariance estimation techniques are mostly based on the ICP algorithm. Whatsoever, they can be used to estimate the answer of other algorithm if they are also destined for point cloud pair alignment.

The first closed-form covariance estimation technique investigate by us is the Hessian method initially purposed by Bengtsson and Baerveldt [3] for point-to-point error. It has this name because it is based on hessian matrix of the linearized ICP's cost function. Although, Bengtsson and Baerveldt [3] uses the point-to-point cost, we derive here the one based on point-to-plane in Bonnabel et al. [7]. The rationale of this method is to treat the ICP's output as linear least squares problem for which is possible to obtain a covariance. First, it is necessary to linearize point-to-plane cost function of Equation 2.5.1 that measures the alignment of N correspondent point pairs $\{\mathbf{p}_i\}$ into $\{\mathbf{q}_i\}$, replicated next:

$$f(\boldsymbol{\xi}) = \sum_i \left\| \left(\mathbf{q}_i - \mathrm{Exp}(\boldsymbol{\xi})\mathbf{p}_i\right) \cdot \mathbf{n}_i \right\|^2. \tag{2.6.5}$$

For small angles, the rotation component $\mathrm{Exp}(\boldsymbol{\omega})$ from $\mathrm{Exp}(\boldsymbol{\xi})$ can be approximated by

$$\mathrm{Exp}(\boldsymbol{\omega}) \approx I + [\boldsymbol{\omega}]_\times, \tag{2.6.6}$$

then we approximate the transformation of the source point $\mathbf{p}_i$ to

$$\mathrm{Exp}(\boldsymbol{\xi})\mathbf{p}_i \approx (I + [\boldsymbol{\omega}]_\times)\mathbf{p}_i + \boldsymbol{\rho}. \tag{2.6.7}$$

Hence, it is possible to linearize Equation 2.6.5 into

$$f(\boldsymbol{\xi}) \approx \sum_i \left\| (\mathbf{p}_i + [\boldsymbol{\omega}]_\times \mathbf{p}_i + \boldsymbol{\rho} - \mathbf{q}_i) \cdot \mathbf{n}_i \right\|^2 \tag{2.6.8}$$

To transform Equation 2.6.8 into a linear least squares problem we rearrange it using the row vectors $\mathbf{j}_i \in \mathbb{R}^{1\times 6}$ and scalars $r_i \in \mathbb{R}$, defined as

$$\begin{aligned} \mathbf{j}_i &= \begin{bmatrix} -(\mathbf{p}_i \times \mathbf{n}_i)^\intercal & -\mathbf{n}_i^\intercal \end{bmatrix}, \\ r_i &= \mathbf{n}_i^T(\mathbf{p}_i - \mathbf{q}_i) \end{aligned} \tag{2.6.9}$$

hence, obtaining

$$f(\boldsymbol{\xi}) \approx \sum_i (r_i - \mathbf{j}_i\boldsymbol{\xi})^2. \tag{2.6.10}$$

Using the normal equation, the solution for $\boldsymbol{\xi}$ in Equation 2.6.10 is

$$\boldsymbol{\xi} = \left[\sum_i \mathbf{j}_i^\intercal \mathbf{j}_i\right]^{-1} \sum_i \mathbf{j}_i^\intercal r_i. \tag{2.6.11}$$

Let

$$H = \sum_i \mathbf{j}_i^\intercal \mathbf{j}_i \tag{2.6.12}$$

we can now write the covariance of $\boldsymbol{\xi}$, aided by the property of Equation 2.1.7, as

$$\operatorname{cov}(\boldsymbol{\xi}) \approx \operatorname{cov}\left(H^{-1} \sum_i \mathbf{j}_i^\intercal r_i \right), \tag{2.6.13a}$$

$$\approx H^{-1} \operatorname{cov}\left(\sum_i \mathbf{j}_i^\intercal r_i \right) \left[H^{-1}\right]^\intercal, \tag{2.6.13b}$$

$$\approx H^{-1} \sum_i \mathbf{j}_i^\intercal \operatorname{var}(r_i) \sum_i \mathbf{j}_i \left[H^{-1}\right]^\intercal. \tag{2.6.13c}$$

We can assume the residual error is normally distributed,

$$r_i = \mathcal{N}(0, \sigma^2), \tag{2.6.14}$$

with a given standard deviation σ^2 obtained from sensor measurements. Following, equation 2.6.13 is then simplified to

$$\operatorname{cov}(\boldsymbol{\xi}) \approx H^{-1} \sum_i \mathbf{j}_i^\intercal \operatorname{var}(r_i) \sum_i \mathbf{j}_i \left[H^{-1}\right]^\intercal, \tag{2.6.15a}$$

$$\approx H^{-1} \sigma^2 \sum_i \mathbf{j}_i^\intercal \sum_i \mathbf{j}_i \left[H^{-1}\right]^\intercal, \tag{2.6.15b}$$

$$\approx \sigma^2 H^{-1} H \left[H^{-1}\right]^\intercal. \tag{2.6.15c}$$

As H is a symmetric matrix, we may omit the transpose, leaving the result to just

$$\operatorname{cov}(\boldsymbol{\xi}) \approx \sigma^2 H^{-1}. \tag{2.6.16}$$

In order to be accurate as the numeric techniques, the covariance estimation using Equation 2.6.16 must account for precise values of residual's variance σ^2, which according to Bonnabel et al. [7] will be optimistic if based on identically distributed gaussian noise. However, many authors [17, 45, 83] found acceptable the following approximation

$$\operatorname{cov}(\boldsymbol{\xi}) \approx H^{-1}. \tag{2.6.17}$$

The method is named Hessian because the resulting matrix matches the second derivative approximation $\sum_i \mathbf{j}_i^\intercal \mathbf{j}_i$ of the cost function f. Once having the covariance matrix, we may analyze its eigenvectors and eigenvalues for finding the maximum axis of variation, for besides using in global alignment methods, verifying the tracking errors [83].

A more realistic approach was proposed by Censi [10]. He treats the ICP procedure as a function

$$\boldsymbol{\xi} = \operatorname{icp}(\mathbf{z}), \tag{2.6.18}$$

where $\mathbf{z}$ represents the input points $\mathbf{p}_i$ and $\mathbf{q}_i$ correspondences generated by the transfor-

mation $\boldsymbol{\xi}$. With this assumption, we can take its Taylor series for approximation

$$\text{icp}(\mathbf{z}) \approx \text{icp}(\mathbf{z}_0) + \frac{\partial \text{icp}}{\partial \mathbf{z}_0}(\mathbf{z} - \mathbf{z}_0), \tag{2.6.19a}$$

$$\approx \text{icp}(\mathbf{z}_0) + \frac{\partial \text{icp}}{\partial \mathbf{z}_0}\mathbf{z} - \frac{\partial \text{icp}}{\partial \mathbf{z}_0}\mathbf{z}_0. \tag{2.6.19b}$$

This formulation allows us to use the Equation 2.1.7 to extract its covariance noting that $\text{icp}(\mathbf{z}_0)$ and $\frac{\partial \text{icp}}{\partial \mathbf{z}_0}$ are fixed values

$$\text{cov}(\boldsymbol{\xi}) \approx \frac{\partial f}{\partial \mathbf{z}_0} \text{cov}(\mathbf{z}) \left[\frac{\partial f}{\partial \mathbf{z}_0}\right]^\intercal. \tag{2.6.20}$$

To compute the required derivative $\frac{\partial f}{\partial \mathbf{z}_0}$, Censi [10] applies the central limit theorem which states that

$$\frac{\partial f}{\partial \mathbf{z}_0} = \left[\frac{\partial^2 f}{\partial \boldsymbol{\xi}^2}\right]^{-1} \frac{\partial^2 f}{\partial \mathbf{z} \partial \boldsymbol{\xi}}, \tag{2.6.21a}$$

$$= H^{-1} \frac{\partial^2 f}{\partial \mathbf{z} \partial \boldsymbol{\xi}}. \tag{2.6.21b}$$

Resulting that the covariance can be expressed as

$$\text{cov}(\boldsymbol{\xi}) \approx H^{-1} \left(\frac{\partial^2 f}{\partial \mathbf{z} \partial \boldsymbol{\xi}}\right) \text{cov}(\mathbf{z}) \left(\frac{\partial^2 f}{\partial \mathbf{z} \partial \boldsymbol{\xi}}\right)^\intercal H^{-1}, \tag{2.6.22}$$

noting again the hessian H transpose can be ignored because of its symmetry. The Equation 2.6.22 provides more realistic estimations of the covariance because the term $\frac{\partial^2 f}{\partial \mathbf{z} \partial \boldsymbol{\xi}}$ accounts for changes in the shape of the cost function f. However, its matrix shape is $6 \times 6N$ which must be numerically calculated, rendering considerable computation requirements for real-time use [63].

Albeit covariance estimation, other form of uncertainty is assessing whether the tracking or registration was successful or not, which can be evaluated with heuristic as the ratio of inlier keypoints [17, 57], or the ratio of the residual error from energy functions like in Equations 2.5.1 and 2.6.2 [83]. Kähler et al. [45] goes as far as creating Support Vector Machine (SVD) classifier to determine whether a registration was successful or not, which inputs the hessian covariance approximation among other parameters.

2.6.3 Bundle Adjustment Back-end

Although oriented popular for photogrammetry, BA is also used as back-end in many [9, 21, 56] SLAM systems. We do not use on our system, but provide now a quick overview of it for sake of completeness.

In BA a set of initial 3D points estimates $\{\mathbf{x}_j\}$ are indexed along a set of is 2D keypoints appearance $\{\mathbf{u}_{i,j}\}$ along the N frames. Moreover, it is also given a set of initial guesses for the camera poses $\{P_i\}$. Both sets $\{\mathbf{x}_j\}$ and $\{P_i\}$ are initial guesses from the front-end, and therefore, it is likely that many reprojections of $\mathbf{x}_j$ on their appearances frames will not precisely match one to another. Let the vectors $\mathbf{e}_{i,j} = \mathbf{u}_{ij} - \pi(KP_i\mathbf{x}_j)$ contain the reprojection error of a point $\mathbf{x}_j$ with respect to its original keypoints $\mathbf{u}_{ij}$,

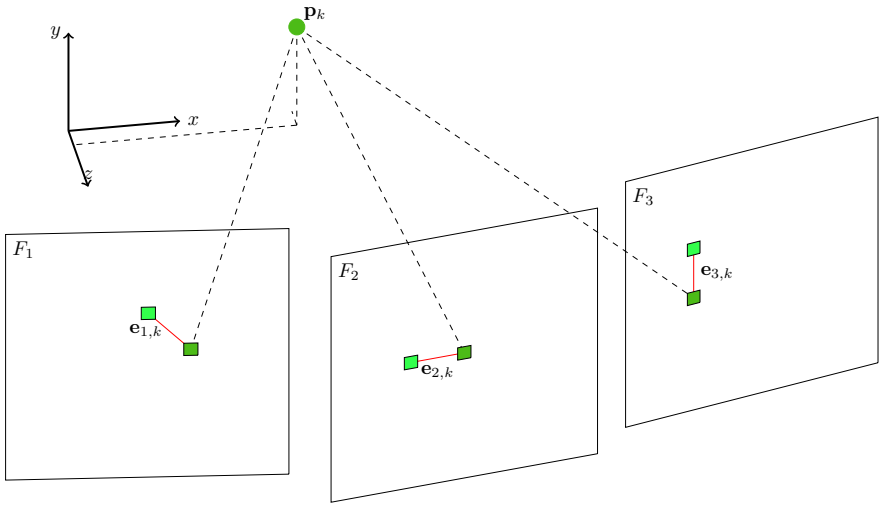


Figure 2.10: The reprojection errors in Bundle Adjustment. The errors $\mathbf{e}_{i,k}$ are the difference between initially found 2D features (lighter green) in the images F_1, F_2, and F_3 and the reprojection of the estimated 3D point $\mathbf{p}_k$ on the images (darker green). The goal of bundle adjustment is to correct the poses and map's 3D point to diminish those reprojection errors.

which is illustrated on Figure 2.10. The goal of BA is to optimize the positions in $\{\mathbf{x}_j\}$ and poses $\{P_i\}$ by minimizing the squared reprojection errors $\mathbf{e}_{ij}$ of all $\mathbf{x}_j$ to be the close as possible to their respective source keypoints $\mathbf{u}_{ij}$. In other words, we can formulate the BA optimization with the following equation

$$E_{\text{ba}}(\{P_i\}, \{\mathbf{x}_j\}, \{\mathbf{u}_{ij}\}) = \underset{P_{1:N}, \mathbf{x}_{1:J}}{\arg\min} \sum_{i,j} \mathbf{e}_{i,j}^{\intercal} \Omega_{i,j} \mathbf{e}_{i,j}. \tag{2.6.23}$$

Besides the keypoint coordinates $\mathbf{u}_{i,j}$, it is possible to supply the information matrices $\Omega_{i,j} \in \mathbb{R}^{2\times 2}$ which weights the confidence of the i, j keypoint $\mathbf{u}_{i,j}$ along the X-axis and Y-axis [57].

BA is extensively used throughout different kinds of SLAM being either monocular, stereo, sparse, or dense. As examples, Maier et al. [54] and Mur-Artal and Tardós [56] uses BA to align odometry and keyframes. Cao et al. [9] aligns submap with BA.

2.6.4 Pose Graph Optimization Back-end

PGO optimizes the camera poses into a maximum likely configuration according to the level of uncertainty estimated on the odometry and loop closure measurements. Other works may also include landmark detection measurements, but this is not discussed in this dissertation for simplicity. The poses and the measurements are described in a probabilistic graphical model for finding the most likely configuration. More specifically, a pose graph is a directed graph where the camera poses are the nodes $\mathcal{V} = \{P_1, P_2, \ldots, P_{N-1}, P_N\}$, and the relative transformation between poses, either acquired by VO or loop closure detection, are the edges $\mathcal{E} = \{(T_{12}, \Omega_{12}), \ldots, (T_{st}, \Omega_{st})\}$. Expressly, the edge set is the union of the sets $\mathcal{O}$ and $\mathcal{C}$ of, respectively, odometry and

loop closures estimations,

$$\mathcal{O} = \{(T_{12}, \Omega_{12}), (T_{23}, \Omega_{23}), \ldots, (T_{(N-1)N}, \Omega_{(N-1)N})\}, \tag{2.6.24a}$$

$$\mathcal{C} = \{(T_{st}, \Omega_{st}) \mid \forall \text{ loop closures from a source pose } s \text{ to a target pose } t\}, \tag{2.6.24b}$$

$$\mathcal{E} = \mathcal{O} \cup \mathcal{C}. \tag{2.6.24c}$$

The edges are composed of a relative transformation T_{st} of the source pose s to the target pose t, and the uncertainty of T_{st} in the form of an information matrix Ω_{st} estimated by the front-end with techniques such the ones on Section 2.6.2 — information matrices are the inverse of covariances matrix, $\Omega_{st} = \text{cov}(T_{st})^{-1}$. As the edges have an associated uncertainty, we can formulate the probability of the whole pose graph arrangement by joining how much uncertainty each node in $\mathcal{V}$ has

$$p(\mathcal{V} \mid \mathcal{E}) = \prod_{i}^{N} p(P_{i+1} \mid P_i, T_{i(i+1)}) \cdot \prod_{s,t \in \mathcal{C}} p(P_t \mid P_s, T_{st}), \tag{2.6.25a}$$

$$= \prod_{s,t \in \mathcal{E}} p(P_t \mid P_s, T_{st}). \tag{2.6.25b}$$

We may explicitly refer to the probability of a pose graph as two separated products, one for odometry and another for loop closures (Equation 2.6.25a), but we can group them as the same (Equation 2.6.25b). For the conditional probabilities given an edge with source node P_s and relative transformation T_{st}, we may assume a normal distribution centered around the nodes relative transformation and spread with the uncertainty of the transformation's covariance matrix

$$p(P_t \mid P_s, T_{st}) = \mathcal{N}(T_{st}; P_t^{-1} P_s, \text{cov}(T_{st})), \tag{2.6.26}$$

that can be expanded as

$$p(P_t \mid P_s, T_{st}) = \eta \exp(-\frac{1}{2} \mathbf{r}_{st} \Omega_{st} \mathbf{r}_{st}), \tag{2.6.27a}$$

$$\mathbf{r}_{st} = T_{st} - P_t^{-1} P_s. \tag{2.6.27b}$$

Since the minus operation is not part of the $SE(3)$ group, making the numerical optimization search on non-valid elements of the parameters' manifold, we can rewrite the residual term replacing the minus with the inverse T_{st}^{-1}:

$$\mathbf{r}_{st} = T_{st}^{-1} P_t^{-1} P_s. \tag{2.6.28}$$

The expression $P_t^{-1} P_s$ computes the relative transformation from node s to t, meaning that our objective is to adjust the poses to match the transformation T_{st};

Therefore, the goal of PGO is to maximize the probability in Equation 2.6.24a, which

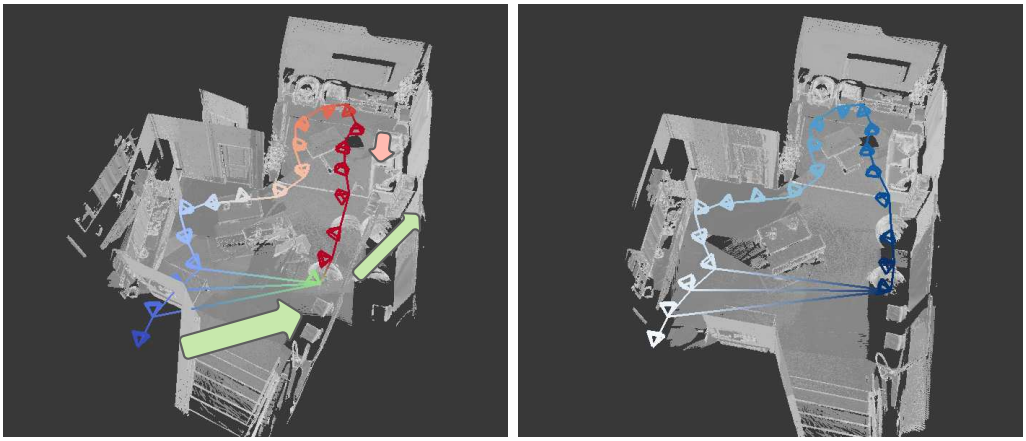


Figure 2.11: (a) Sample initial pose graph. The blue-to-red lines are odometry edges. Redder specifies the accumulated uncertainty over the odometry measurements. The green-to-blue lines are loop closure edges. In this illustration, the loop closure edges are well aligned and have low uncertainty, thus making PGO adjust the poses represented by the green arrows. The red arrow symbolizes the pose from the odometry nodes with higher uncertainty. (b) Result of the PGO algorithm. It adjusts the high accumulated uncertainties on the odometry edges using loop closure measurements.

is the same as minimizing its log-likelihood sum

$$\arg\max_{\mathcal{V}} p(V \mid \mathcal{E}) = \arg\min -\log\left(\sum_{s,t\in\mathcal{E}} p(P_t \mid P_s, T_{st})\right), \tag{2.6.29a}$$

$$= \arg\min \sum_{s,t\in\mathcal{E}} \|\mathbf{r}_{st}\Omega_{st}\mathbf{r}_{st}\|. \tag{2.6.29b}$$

A visual intuition of PGO is that edges act like springs pulling the pose nodes. For example, suppose a loop close's spring has more compression rate than a sequence of odometry ones. In that case, they will be unwound according to the accumulated sum of their compression rates until they can balance the one with the loop closure. In this example, the compression rate is the level of uncertainty of their information matrices. With submaps, PGO acts like in Figure 2.11, the loop closure edge will bend the odometry ones to adjust the trajectory.

Following Equation 2.6.29, we again arrive on a least square problem that is solvable using numerical algorithms such as GN or LM. Given the residual formulation of Equation 2.6.28, we can linearize the Equation 2.6.29 as we did for the other energy functions

$$\mathbf{r}_{st}(\hat{P}_s \boxplus \boldsymbol{\xi}_s, \hat{P}_t \boxplus \boldsymbol{\xi}_t) \approx \mathbf{r}_{st}(T_{st}\hat{P}_s, \hat{P}_t) + J_{st}\begin{bmatrix}\cdots & \boldsymbol{\xi}_s & \cdots & \boldsymbol{\xi}_t & \cdots\end{bmatrix}, \tag{2.6.30}$$

and fit using second order algorithm such as the GN or the LM. We use $\boldsymbol{\xi}_s \in \mathbb{R}^6, \boldsymbol{\xi}_t \in \mathbb{R}^6$ to define, respectively, the increment over the parameters of the pose of nodes s and t. The regions with the dots $(\cdots)$ symbolizes zeroed values, i.e., elements with no contribution to the derivative for the poses at nodes s and t. For a graph with V nodes, we must define

jacobian matrices $J_{st} \in \mathbb{R}^{6\times 6V}$ for every term expressing an edge s to t

$$J_{st} = \frac{\partial T_{st}^{-1}(P_t \boxplus \boldsymbol{\xi}_t)^{-1} P_s \boxplus \boldsymbol{\xi}_s}{\partial \begin{bmatrix} \cdots & \boldsymbol{\xi}_s & \cdots & \boldsymbol{\xi}_t & \cdots \end{bmatrix}} = \begin{bmatrix} \cdots & A_{st} & \cdots & B_{st} & \cdots \end{bmatrix}, \tag{2.6.31}$$

in which the matrices $A \in \mathbb{R}^{6\times 6}$ and $B \in \mathbb{R}^{6\times 6}$ are the jacobians

$$A_{st} = \frac{\partial T_{st}^{-1}(P_t \boxplus \boldsymbol{\xi}_t)^{-1} P_s \boxplus \boldsymbol{\xi}_s}{\partial \boldsymbol{\xi_s}}, \tag{2.6.32a}$$

$$B_{st} = \frac{\partial T_{st}^{-1}(P_t \boxplus \boldsymbol{\xi}_t)^{-1} P_s \boxplus \boldsymbol{\xi}_s}{\partial \boldsymbol{\xi_t}}. \tag{2.6.32b}$$

In PGO version purposed by Choi et al. [17], they add a line process set $L = \{l_{st}\}$ over the loop closure edges, thus making their optimization function be

$$\arg\min_{\mathcal{V},L} \sum_{s,t\in\mathcal{E}} \|\mathbf{r}_{st}\Omega_{st}\mathbf{r}_{st}\| \, l_{st} + \Psi(l_{st}) \tag{2.6.33}$$

where l_{st} for odometry's edges are fixed and always equal to 1. Its penalty for marking an edge as an outlier is

$$\Psi(l_{st}) = \mu(\sqrt{l_i} - 1)^2. \tag{2.6.34}$$

We can then proceed the optimization in the same sense of the FGR algorithm, by updating the estimation of L and the poses separately.

Implementations like the one [88] used by us take advantage of the sparse structure of the jacobians at Equation 2.6.31 for calculating the hessian matrix approximation $H = J_{st}^{\intercal} J_{st} \in \mathbb{R}^{6V\times 6V}$. Instead of computing the product of jacobians directly, the implementation may compute the H matrices in subparts [37]. Once knowing how to compute the H term, then the implementations can carry numerical optimization method with it.

PGO is applied in various SLAM modalities, including aligning submaps [17, 45] or keyframes [21, 56, 57]. For instance, InfiniTAM [45] creates TSDF volumes submaps with a fixed number of frames. The submaps are globally aligned by setting them on a PGO with loop-closures detected via Ferns descriptor coding [34]. Choi et al. [17] build a pose graph made of submaps like InfiniTAM but introduces line-process to mitigate the effect of outlier loop closure edges.

Our backend uses the PGO algorithm and implementation of Choi et al. [17], later integrated into the Open3D library [88] by Intel Corporation.

2.6.5 Mapping on RGB-D SLAM

In RGB-D systems, the frame's point cloud already provides a dense geometry to update the map once a camera pose is estimated. In those systems, the integration of a frame into the map finds which points from the frame's point cloud should be added as a new element or fused with an already existing map element.

Most of major works [17, 21, 45, 58, 59] on RGB-D SLAM represent their scene's map with a TSDF volume. The elements of TSDF volume contain a positive or negative measurement of the distance from it to its nearest scene's surface point. A positive

TSDF value corresponds to an element in front of the surface, while a negative one places the element in the surface's back. Consequently, placing the actual surface at the zero-crossings elements can be efficiently converted into a mesh using the marching cubes algorithm [58]. Initial TSDF volume implementations were based on allocating fixed size volume, consuming much memory, that later Nießner et al. [59] improved by offering a hash-based sparse data structure to store it, rendering significant reduction in memory requirement.

Another popular map representations are point clouds or surfels, which can offer less memory usage and more efficient deformation than volumes. However, they cannot be transformed in a mesh with the marching cubes algorithm. Although, recently, Schöps et al. [72] has proposed an algorithm that delivers such conversion from surfels.

In either volumetric or point-based representations, the algorithm for integrating an RGB-D image casts a ray from every point and finds where it hits the current model's surface. Those incoming rays with no intersection in the map are added as new elements to the latter, having their color and other properties set according to their pixels. Otherwise, those rays' points that does intersect with the map are fused with their respective map elements. The fusion process averages the attributes of the map element with the incoming pixel, hence improving the element with more information about its position and color, for example. Differently from volumetric representations, the point representations have more complex integration algorithm because it requires fast point search structures such as a KD-tree or rendering the scene into an index map [46] for query the ray intersections with the map. Whereas in volumetric representations, the nearest neighbors query can be realized in constant complexity.

Another challenge in mapping is cope with noise that produce outlier 3D points on the reconstruction. To remove wrong points, Klingensmith et al. [48] suggest using space carving on TSDF volumes, meaning that it removes voxels with lower confidence in front of new geometry. A similar strategy is implemented by Keller et al. [46]. On the other hand, recently machine learning methods [12, 64, 77] based on TSDF were proposed to train models that smooth noise based on priors and even to complete missing parts of the map that were not observed. For instance, if only the front side of a chair was captured, the model could predict the non-captured backside.

Regarding to the inclusion of feature into mapping elements, as far we could research, the closest work to ours is from McCormac et al. [55], where they use the descriptors to segment the scene, but do not apply them into the SLAM process.

2.6.6 RGB-D SLAM

So far, we have have covered works of different SLAM modalities, in this section we will focus on RGB-D or stereo SLAM works that reassemble our system. Using consumer affordable sensors, like RGB-D or stereo cameras, is a desirable target in the literature since they are less expensive than LiDAR equipments or can be turned into portable dense reconstruction sensors.

As already stated in this dissertation, video streams offer an overwhelming quantity of data, therefore most of the works can be classified (among other categories) in those that

break the video input into submaps [17, 45, 54, 88], keyframes [21, 56, 71] for applying global optimizations.

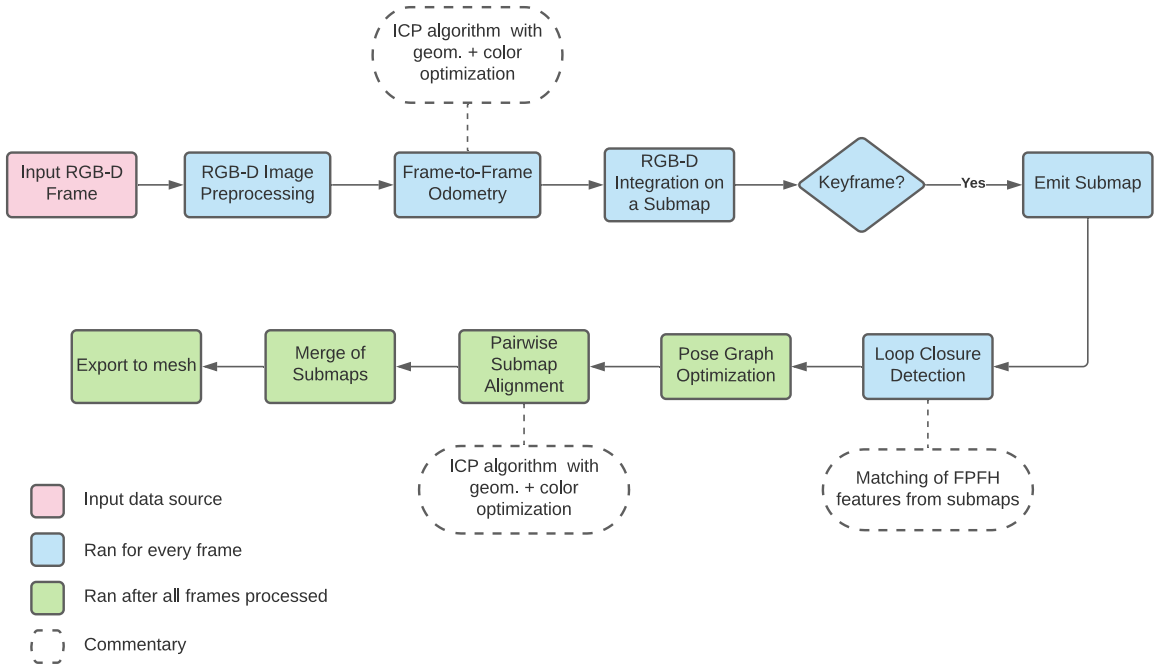


Figure 2.12: Pipeline for 3D reconstruction suggested by Choi et al. [17] (Redwood). The authors divide the scene into submaps, where loop closure is detected, and PGO with line-process performs global alignment. The authors include ICP registration on the submaps after PGO as a manner to increase the final map alignment.

As already mentioned, we base our pipeline on the Redwood [17] offline reconstruction system, which originated the Open3D project [88]. The Redwood system works by generating submaps with frame-to-frame odometry; detecting loop closure between the submaps using FPFH; global aligning them with PGO and finally applying pairwise registration of submaps for further refining the alignments. Its pipeline is illustrate in Figure 2.12. Our work first differs from Redwood in the choice of map representation by the use surfels while theirs is based on TSDF volume. Although, the main difference resides in the loop closure and the pairwise map alignment steps where we replace their methods with ones powered by the deep descriptors of the surfels.

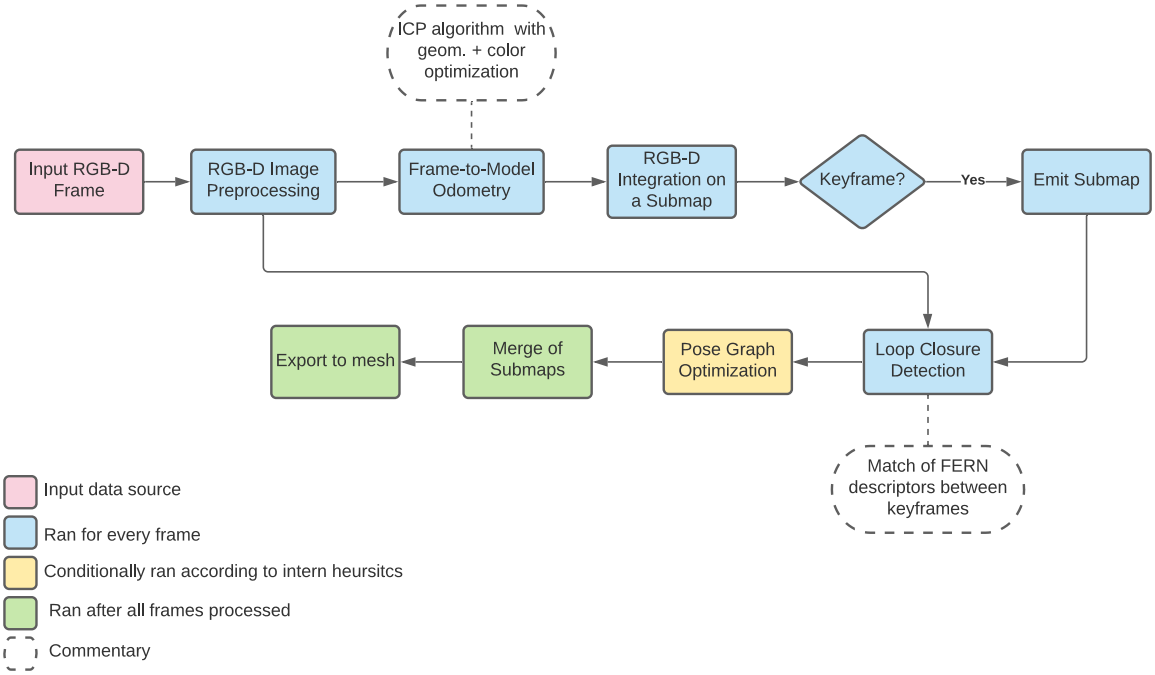


Figure 2.13: This Figure shows a simplification of the pipeline for RGB-D SLAM done by Kähler et al. [45]. We highlight that their pipeline uses submaps and finds loop closures using depth images from the frames. The yellow coloring of the pose graph optimization stage represents that their system runs PGO continuously during the execution. For more detail, see its original article.

Another important submap-based SLAM system is InfiniTAM [45], which also uses a pose-graph within the submap level. Unlike Redwood's pipeline, theirs is planned to execute PGO continuously during the execution inside the subset of active submaps that the system maintains. Figure 2.13 depicts InfiniTAM's pipeline.

Nevertheless, the third category of works, besides submap or keyframe, can be drawn from those [21, 83] that continuously deform the global map. Rather than having explicit submaps or keyframes, this architecture mode changes the map on the spots when a loop closure is found.

One state-of-the-art method to highlight is BundleFusion [21], which applies global optimization at the keyframe level and the frames between keyframe intervals in a hierarchical fashion of resolving the camera poses. The BundleFusion's global optimization is based on sparse and then dense registration. Its sparse method uses SIFT matching to find the keypoints correspondences. And its dense optimization part uses photometric and geometric consistency for aligning the frames. The system runs in real-time, continuously adjusting the map according to the new refinements in the previous camera poses in the process of re-integrating the past frames. Since adjusting overlapping keyframes by sparse features matching during global optimization and correcting them during re-integration renders BundleFusion an implicit loop closure correction mechanism.

ElasticFusion [83] is another work that performs loop closure correction without requiring a pose graph. The method is based on surfel map representation and uses frame-

to-model odometry while tracking active and inactive map portions. When a loop closure is detected by the analysis of ferns [34] features between the active and inactive map portions, the system registers the portions while deforming the map's surfels to match their alignment. All its registrations use geometric and photometric consistencies.

Besides BundleFusion and ElasticFusion not being directly submap or PGO based, in our belief, those methods could potentially gain from the research of (learned) dense features to improve their sparse or dense registration pipelines.

2.7 Descriptor Learning

Image descriptor learning aims to obtain a function g that outputs high discriminative feature vectors for images. Furthermore, as in previous handcraft feature methods, the comparison mechanism between descriptors is measured via a metric function, such as the ℓ_2 distance or cosine similarity. The motivation behind learning descriptors is to try to improve the invariance to noise, lighting, and geometric transformations over handcraft ones through prior information, which can also account for semantic invariance. Semantic invariance refers to matching regions of the same functionality on different objects, such as the wing parts of two different airplane models.

In our case, we obtain a descriptor for every pixel that is later integrated on the surfels of submaps. Therefore, the input to our trained model will be a $W \times H \times 3$ image tensor and then it outputs a $W \times H \times D$ descriptors tensor. In this modality, some recent good results on using CNNs to model such function from previous works [18, 19, 23, 29, 70] hence motivating our interest on evaluate their potential use for SLAM.

The training of descriptor learning CNN models can be divided into three main parts: network architecture (Section 2.7.2), training procedure (Section 2.7.3), and dataset generation (Section 2.7.4). Nevertheless, to understand CNN architectures for descriptor learning and how their literature evolved, it is necessary first to review the concepts taken from handcraft descriptors.

2.7.1 Handcraft Descriptors

Authors of handcraft descriptors tried to produce image patch descriptors that output close distance feature vector of patches representing the same point independently of the distortions that it suffers from one image to another. Some of the most common distortions are geometric transformations, including translation, scale and rotation, lighting changes, and noise.

For dealing with the previously mentioned actions, methods mainly relied on capturing edges and corners since they are distinguishable features on images. The pixel representation of those structures is characterized by sudden intensity change in pixel neighborhoods. Its direction and magnitude can be approximated by convolving the image with the Sobel filters. By convolving the image I at a coordinate $\begin{bmatrix} u & v \end{bmatrix}$ with the

sobel filters G_x and G_y we obtain the gradient vector $\nabla(u, v)$ at its position

$$\nabla(u, v) = \begin{bmatrix} I(u, v) * G_x \\ I(u, v) * G_y \end{bmatrix}, \tag{2.7.1}$$

where

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{2.7.2}$$

The $*$ operator is the discrete image convolution operation defined by

$$(I * K)(u, v) = \sum_{i}^{K_w} \sum_{j}^{K_h} I(i, j) K(u - i, v - j), \tag{2.7.3}$$

where K is a kernel such as G_x or G_y. The values K_w and K_h are the kernel's width and height.

The Sobel kernels can be understood as applying the finite differences method for computing numerical derivatives of a discrete signal function $f(u, v)$ which inputs u and v are pixel coordinates, and its output is one of the color's channel at that point.

One of the first descriptors to offer better light-invariance by the use of gradient vectors was the Histogram of Oriented Gradients [22] (HoG) descriptor for image patches. It is a histogram of the gradient's magnitudes according to their orientation angle. In other words, each histogram's bin refers to a rotation angle interval from 0 to 360 degrees (or unsigned bins from 0 to 180). After a final step of normalizing the histogram, it results that a HoG descriptor is less sensitive to illumination changes since the captured gradient structures are independent of the pixels' brightness level. To also account for scale-invariance the HoGs may be computed at multiple scales [65].

The SIFT method is composed of two stages: finding keypoints that remain stable at different scales and then building the descriptor around the keypoint's region. The keypoint detector tests for scale invariance of pixels by verifying if they lie in saddle-points of the scale-space [53] representation of the image. Then in the descriptor building for a keypoint, it extracts the descriptor from the image at the scale that found the best response during the scale-space test, providing scale-invariance to the representation. A descriptor is computed from 16×16 pixels patches centered around the keypoint, which are further subdivided into 8×8 cells, resulting in 4 HoG descriptors. The bins of 4 HoGs are rotated to their most predominant orientation for providing a high level of rotation invariance. And finally, the 4 HoGs are concatenated into one feature vector. The deep CNN-based Universal Correspondence Network [19] (UCN) revisits the concept of estimating a good scale and orientation using a layer to learn not only those transformations but the parameters of an affine transformation. As histograms based, both HoG and SIFT descriptors are comparable with ℓ_2 distance to find the most similar matches. One of the few dense handcraft features is DAISY [79] (DAISY) which computes similar descriptor values of SIFT but using improved performance constructs for the dense extraction.

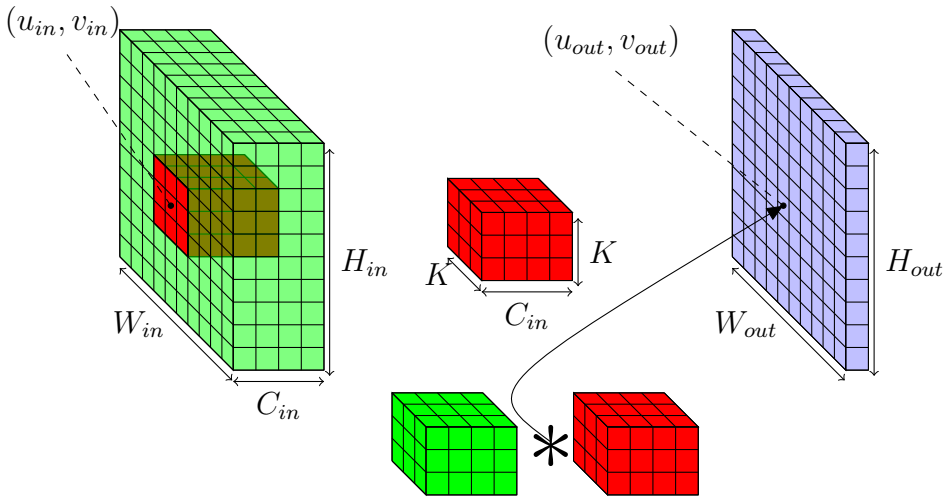


Figure 2.14: Convolution of an input tensor (green) with a single filter (red) at the point $(u_{\text{in}}, v_{\text{in}})$. The operation will convolve each channel of the input tensor with its corresponding channel of the filter, represented in the bottom. The operations will be sum resulting in a single value that is placed in a channel of the output tensor (purple) at the position $(u_{\text{out}}, v_{\text{out}})$. In the figure, we show the slice of output tensor corresponding to the index of the filter.

2.7.2 Convolutional Neural Networks

A type of neural network, CNNs has established a new paradigm for state-of-the-art visual recognition tasks. Besides much-existing literature explaining their operation being rooted in the biology of mammals' vision, we opt for a more straightforward interpretation analogous to the Sobel filters, that is closer to the concepts of image descriptors studied by this dissertation. If the Sobel filters can capture corners and edges, then other kernels with the appropriate configurations (size, stride, and padding) and values can respond to richer geometric structures in images. The layers of a CNN are in essence those kernels, which its values are denominated as weights — in the same sense of a Multilayer Percepton (MLP) network. In a process designated as training in the ML literature, optimization algorithms such as the standard SGD or Adam [47] finds the kernels' values in such a way that minimize a loss function of a learning task.

A basic 2D neural convolutional layer is defined to accept a 3-dimensional input tensor F_{in}, let its size be $W_{in} \times H_{in} \times C_{in}$, and outputs a new one, F_{out} of size $W_{out} \times H_{out} \times C_{out}$. We often refer to the $W \times H$ extent of a tensor as its spatial axis and the C extent as its channel, depth, or descriptor axis. While W_{out} and H_{out} will depend on the input tensor's spatial dimensions, the axis C_{out} is set independently from them, and it specifies how many filters should be learned by the layer. A single filter is a sequence of two-dimensional kernels in the same number of input channels C_{in}. Each filter convolves a different input channel. Then, given spatial coordinate (u_{in}, v_{in}) in the input tensor, the results of the C_{in} convolutions are all sum, and put at its corresponding coordinate $(u_{out}, v_{out}, k_{out})$ in the output tensor, where k_{out} is the current filter's index. Such 2D convolutional operation is

illustrated Figure 2.14 and can be defined in the following formula,

$$F_{out}(u_{out}, v_{out}, k_{out}) = B(k_{out}) + \sum_{k}^{C_{in}} F_{in}(u_{in}, v_{in}, k) * K_{out,k} \tag{2.7.4}$$

where $K_{out,k}$ is the $k$nth convolution kernel of the filter C_{out}. Commonly, the CNN architecture also includes a bias value to the filters, represented by the function $B(\cdot)$ in Equation 2.7.4, making them into affine transformations.

With an effective training process, the multiple filters of one convolutional layer can output new images that highlight multiple kinds of significant patterns for the learning task. Because the output tensor of a convolutional layer exhibits response to patterns, they are referred to as feature maps. Furthermore, combining the feature maps with elementwise activation functions such as Sigmoid, ReLU, PReLu yet better salient the patterns on those tensors. Nevertheless, typical CNN architectures are sequences of convolutional layers in such a way that they learn hierarchical relations between low-level structures, such as edges, at the layers closer to the input, and high-level features, such as objects or human contours, it the layers in the end.

An illustrative example of CNN architectures for descriptor learning are the works from Schmidt et al. [70] and Florence et al. [29], which we also use in our experiments. Their proposed architectures are based on Dilated ResNet [84] (DRN), which is yet an extension of the ResNet [40]. ResNet is an architecture initially designed for classifying entire image content, e.g., whether an image displays a cat or a dog. Its difference for other works is the inclusion of residual connections between blocks of convolution layers. Its authors noted that summing the output of a convolutional layer block with its input allows deeper architectures without suffering the vanishing gradient during training that happens when stacking too many network layers. More information about residual connections and their effect on training may found in its original article [40].

However, like many architectures for classification tasks, ResNet applies spatial resolution reduction throughout the feature maps from early convolutional blocks to the last one. That is done to enlarge the later layers' receptive fields. The receptive field is a term from the biological foundation of CNNs to designate the spatial size of the input image that end-up contributing to a layer's output. For instance, a filter that operates on a down sampled feature map has a greater receptive field than the ones with the same kernel size but operates directly in the input tensor. This behavior happens because the elements of the down sampled feature map will represent a larger portion of the image.

The rationale for having larger receptive fields from gradually down sampling is to improve scale invariance. It will unite local information to disambiguate global objects, which is beneficial for the classification task, but it has the cost of losing local information.

The reduction of the spatial size of a feature map may happen by three means. The first one is when the layer has no border padding configuration. For instance, a 5×5 convolution will convolve a 25×25 feature map into 21×21 if there is no $2-$element padding to the borders. The second mean are layers configured with stride size β higher than one for traversing the input feature map. The routine convolves the kernel at a coordinate (u, v) and steps to the coordinates $(u + \beta, v + \beta)$ where convolution is calculated again. For

example, a convolution with $\beta = 2$ has the effect of reducing the spatial resolution by 2 of the outputted feature maps. The other manner is to include spatial pooling layers on the architecture. Like the convolutional ones, those layers are applied over a sliding window of fixed size over the input feature map, but they aggregate the window values into one conforming to a metric. Usually, it is the maximum or the average value. Pooling layers also accept a stride parameter for downsampling input tensors. One important use of the max-pooling layer is to improve the translation invariance for small shifts, which is the consequence of passing forward the maximum filter responses [36].

However, too much downsampling is not desirable on descriptor and segmentation tasks because both tasks are sensitive to small details. Consider thin objects like a camera tripod or a pen; such objects are likely to disappear from the final feature maps, which can arrive to be small spatial sizes as 7×7 in state of the art CNNs for image classification like ResNet. Hence, to adapt ResNet for those learning tasks, where such reduction would remove detail, the authors of DRN proposed the use of convolution with dilated kernels (or atrous convolutions) for not losing receptive field size while keeping larger spatial size at the final feature maps. Unlike a usual convolution in which the kernel window is multiplied with consecutive elements, a dilated convolution selects the input tensor elements within a step size between them. Its convolution formula is like Equation 2.7.3 but adding a step, or dilatation size d

$$(I * K)(u, v) = \sum_{m}^{K_w} \sum_{n}^{K_h} I(md, nd)K(i - m, j - n). \tag{2.7.5}$$

Note that the standard convolution layer can be thought as having $d = 1$. When dilated convolution is put at the end layers, its kernels can capture larger receptive fields than possible without down sampling the feature maps. Therefore, the network architecture may be projected not to reduce the spatial resolution at the end and thus to have more spatial descriptors while keeping sensitive to context.

Another important decision of CNN architectures present in works for descriptor [19, 23, 29, 70] and segmentation [14, 51, 84] tasks are being fully-convolutional, in other words, its network only has convolutional or feature map transformation layers. Being fully convolutional allow them to adapt to any resolution at the input, but on the other hand, the architecture cannot have any fully connected neural network layer that is often used to classify the feature maps. Fortunately, the fully connected mechanism can be reproduced along the channel axes of the feature map with pointwise or 1×1 convolutions. A pointwise layer has 1×1 kernel size. Since its computation is reduced to just

$$F_{out}(u_{out}, v_{out}, k_{out}) = B(k_{out}) + \sum_{k}^{C_{in}} F_{in}(u_{in}, v_{in}, k) * K_{k,out} \tag{2.7.6}$$

The pointwise convolution layers act like a linear regression over the C_{in} channel elements in the input tensor to learn a mapping to a $C_{out}-$size descriptor. For example, in segmentation tasks, those element-wise descriptors can form the probabilities for each pixel's class after an activation function.

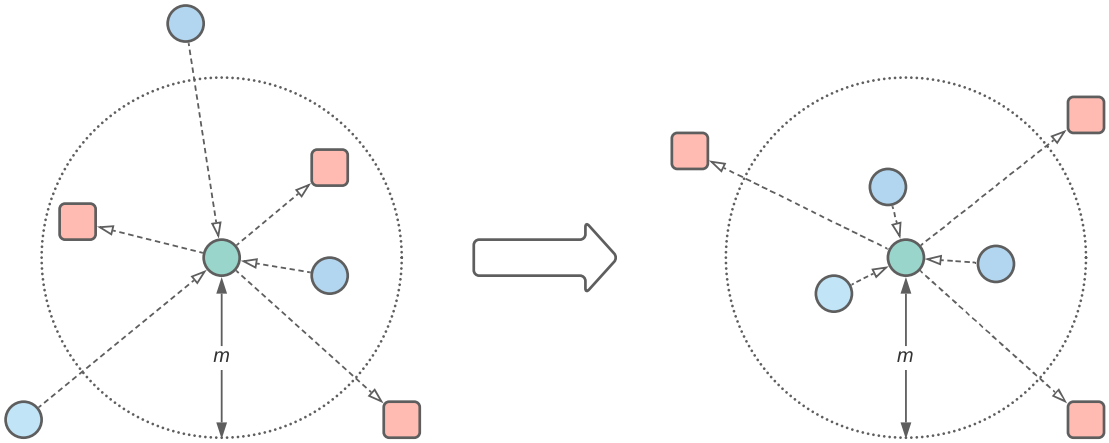


Figure 2.15: The intent of the contrastive loss for one training descriptor, symbolized by the green circle in the middle, is to attract those other descriptors from samples that are correlated with it (blue circles), and at the same time to repeal those that are from different samples to be far at least the radius m. So if the training starts with the left image, it should transform to the one in the right.

One interesting improvement on CNN specifically for descriptor learning was made in UCN, which has a layer to explicitly learn the parameters of a 2D affine transformation for reorienting the receptive fields in a similar fashion of SIFT for providing scale and rotation invariance. Such layers are beneficial to keep the architecture lightweight since other forms to coop with scale and rotation invariance are adding more filters to recognize rotated and scaled patterns [18].

Still, specifically for descriptor learning, some authors [18, 19, 23] found it beneficial for training convergence to include ℓ_2 normalization layer before the final pointwise descriptor layer, so it mitigates the training algorithm from calculating weights with overflowing values.

For a final note in the review of works, some authors explore other types of convolutions according to the input data, e.g., 3D convolutions [86] for volumetric representation, and sparse convolutions [18] for point clouds or similar representations.

2.7.3 Training

For training CNNs of dense descriptor models, most works [18, 19, 23, 29, 70] rely on a version of the contrastive loss [39] for objective function that is minimized using one of the standard optimization methods such SGD or Adam [47]. The basic contrastive loss formula from Hadsell et al. [39] drives small values of a distance function d when a positive pair of descriptors is input and drives the distance to be at least far by a margin m from one another with negative pairs. In other words, the contrastive loss pulls positive pairs to be closer and pushes away unrelated descriptors, as illustrated in Figure 2.15.

Let the sets $\mathcal{P}$ and $\mathcal{N}$ be, respectively, the positive and negative pairs of pixels descriptors produced by a model from two training images, with them we formulate the

contrastive loss [39]:

$$\mathcal{L}(\mathcal{P},\mathcal{N}) = \frac{1}{\|\mathcal{P}\|}\sum_{\mathbf{a},\mathbf{b}}^{\mathcal{P}^+} d(\mathbf{a},\mathbf{b})^2 + \frac{1}{\|\mathcal{N}\|}\sum_{\mathbf{a}^-,\mathbf{b}^-}^{\mathcal{N}} \max(m - d(\mathbf{a}^-,\mathbf{b}^-),0)^2 \quad , \tag{2.7.7}$$

for positive pairs, the loss is the distance function itself since the goal is to be small for correspondent samples. On the other hand, for the negative pair of descriptors, the $\max(\cdot, 0)$ function ensures that only distances lower than the margin are penalized. Any distance result larger than the margin m will evaluate to a negative number and max out by zero. Without a margin m, during training, the model may bias towards favoring the recall of true-negative and losing the precision for true-positives because the set of negative pairs tends to be much larger and more diverse than the positive ones. Lastly, the two terms are normalized according to their set sizes.

To improve contrastive loss train effectiveness, authors have added some weighting or mining hard-negatives pairs [18, 19, 29, 70]. The negative pairs are the ones that remain inside the margin according to the model at the current state, i.e. samples with $d(\mathbf{a}^-,\mathbf{b}^-) < m$. Those patches will contribute more to the model's discrimination learning because they will not evaluate to zero in the negative term of Equation 2.7.7. Florence et al. [29] normalizes the negative term by the hard negative count, which by their empirical test improves training. The only change is the loss of Equation 2.7.7 is to replace the total negative set size by the count of hard negatives, which we store in the scalar h,

$$\mathcal{L}(\mathcal{P},\mathcal{N}) = \frac{1}{\|\mathcal{P}\|}\sum_{\mathbf{a},\mathbf{b}}^{\mathcal{P}} d(\mathbf{a},\mathbf{b})^2 + \frac{1}{h}\sum_{\mathbf{a}^-,\mathbf{b}^-}^{\mathcal{N}} \max(m - d(\mathbf{a}^-,\mathbf{b}^-),0)^2 \quad . \tag{2.7.8}$$

During training, the network model is forward two times, one for each image, and its weights update the backpropagation of the loss value according to the optimizer algorithm.

Although, the more effective approach is to actively search for hard negatives [18, 19], either using brute force search or query with KD-Tree. Choy et al. [18] search hard-negative pairs in respect for the two positive descriptor, resulting in two sets of hard-negatives, $\mathcal{N}_\mathbf{a}$ and $\mathcal{N}_\mathbf{b}$. Following Lin et al. [50], they also add a positive margin value

m_p, resulting in

$$\begin{aligned}\mathcal{L}(\mathcal{P}, \mathcal{N}_{\mathbf{a}}, \mathcal{N}_{\mathbf{b}}) = \frac{1}{\|\mathcal{P}\|} \sum_{\mathbf{a},\mathbf{b}}^{\mathcal{P}} \max[d(\mathbf{a}, \mathbf{b}) - m_p, 0]^2 \\ + \frac{1}{\|\mathcal{N}_{\mathbf{a}}\|} \sum_{\mathbf{a},\mathbf{a}^-}^{\mathcal{N}_{\mathbf{a}}} \max[m - d(\mathbf{a}, \mathbf{a}^-), 0]^2 \ . \\ + \frac{1}{\|\mathcal{N}_{\mathbf{b}}\|} \sum_{\mathbf{b},\mathbf{b}^-}^{\mathcal{N}_{\mathbf{b}}} \max[m - d(\mathbf{b}, \mathbf{b}^-), 0]^2\end{aligned} \tag{2.7.9}$$

In our work we use the less aggressive method of reweighing the loss function of Equation 2.7.8.

2.7.4 Datasets

Unlike other tasks based on contrastive learning such as face identification, the datasets applied for descriptor learning are not instantly available with the image pairs. Especially for dense descriptor learning, the works rely on extracting the pairs from datasets of RGB-D videos annotated with camera poses. It is possible to back project pixel coordinates from the depth image and map to its respective representation in other images because of that. Meanwhile, the negative pairs can be collected from random images without necessarily having overlapping views. However, it is often more efficient to collect them from the same image pairs to obtaining hard negatives. This self-supervision strategy was extensively exploited [17, 29, 31, 70, 85, 86] using base datasets such KITTI [33] and SceneNN [43]. In Section 3.7.2, we show details how we generate those samples using self-supervision from a RGB-D dataset. A second strategy is to generate computer graphics synthetic images, which DeTone et al. [23] uses to initialize the training of its keypoint detection model.

Chapter 3

Methods

In this chapter, we discuss our SLAM solution using deep features at the stages of loop closure detection and alignment of submaps. Figure 3.1 illustrates our pipeline.

The core concept of this work is to extract deep dense descriptors from every incoming

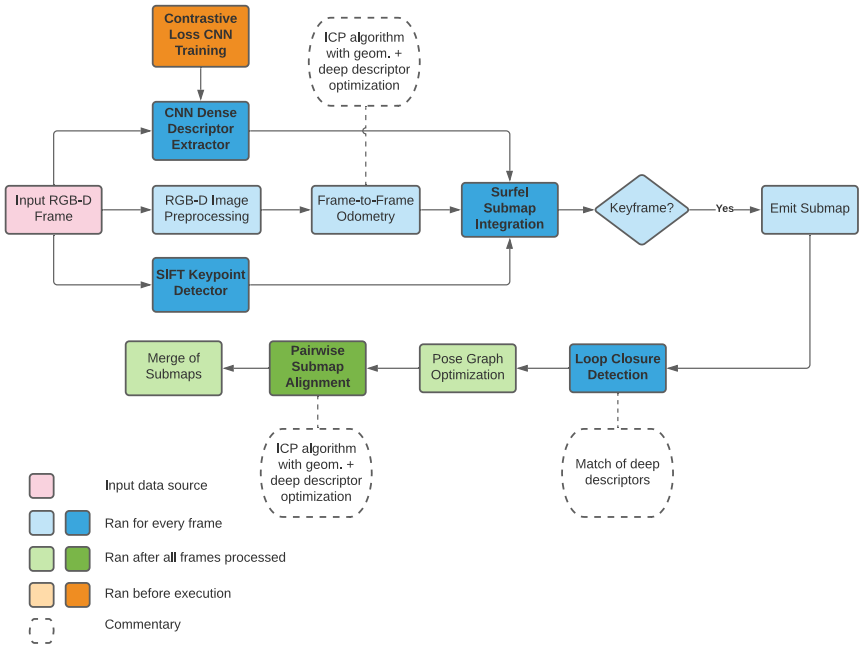


Figure 3.1: The architecture of our SLAM with deep features integrated during surfel mapping. The pipeline first processes an input RGB-D frame (red box) to reduce noise. Then the system extracts dense features from a CNN model and detects key pixels using SIFT that are later used for selecting key surfels in the submap for loop closure detection. With the RGB-D frame and CNN features, the system fuses those pieces of information on the current submap. When the system emits a new submap, it finds loop closures between all the previous submaps. After processing all submaps, we run our PGO back-end to correct global drift in the trajectory using the found loop closures. The final steps run the pairwise submap alignment for tight alignment and merge all submaps into one point cloud, creating the final map. Our work's main contributions and differences from other authors are highlighted with bold fonts and darker colors, namely, the CNN feature submaps and their use for loop closure detection and pairwise submap alignment.

frame and to incorporate them as a property of surfels in a submap-based SLAM. The deep features are then used at loop closure and submaps registration tasks for improving the final trajectory estimation.

The dense descriptors are produced by a CNN trained by us for extracting comparable descriptors. Its architecture is the DRN, a ResNet version designed for image segmentation having larger receptive fields, so its final layers produce descriptors with larger spatial sizes. Its training process uses a contrastive loss function that relates the ℓ_2 distances of the descriptors according to the representativeness of their pixels being the same point in the scene. To obtain data for such training, we employ a strategy of finding every pair of frames with view overlaps from a series of datasets of RGB-D sequences. From those pairs, we extract positive sample correspondence of points, i.e., pixel coordinates from different images that are representations of the same point. Section 3.7 contains the details about our descriptor learning architecture and training.

Aside from the integration of deep features, our SLAM architecture follows the front-end and back-end division. The front-end tracks the frame-to-frame VO with an ICP algorithm that estimates the pose from geometric and photometric consistency. The front-end is also responsible for generating the submaps and executing the loop closure routine, hence, building the pose graph with the initial estimations. The initial pose of a submap's node is the odometry of its first camera. Moreover, the edge connecting with the next submap's node is the relative transformation of its last camera pose between the first camera pose of the next one. The frame intervals that build the submaps are simply chosen by emitting a new keyframe at every 50 frames. We choose this interval size because it provided on our experiments a good tradeoff between track drifting and subdivision of the input. The data structures and algorithms to create the surfel submaps and incorporate the learned descriptor into them are in Section 3.3. We describe the method for detecting and creating loop closure edges in Section 3.4. The back-end part accepts the front-end's pose graph and uses the PGO algorithm of Choi et al. [17] to global optimize the submaps poses.

Finally, in Section 3.6, we present the last part of our pipeline, the submap registration algorithm, an ICP adapted to minimize the alignment energy of the deep features fused in every submap.

Nevertheless, before all those sub modules, the first step of our pipeline is the frame preprocessing stage for reducing noise and computing the depth image normals. Our system carries those transformation throughout the rest of the SLAM pipeline.

3.1 Frame Preprocessing

In the frame preprocessing stage, the system filters the frame's depth image and computes per point surface's normal vectors in this beginning stage.

Filtering the depth image is crucial to obtain reliable results from VO because it is typical for consumer RGB-D sensors to produce high levels of noise since they deal with signals in the range of 16-bit integers. Without depth filtering, the VO, for instance, is likely to get stuck in its optimization objective.

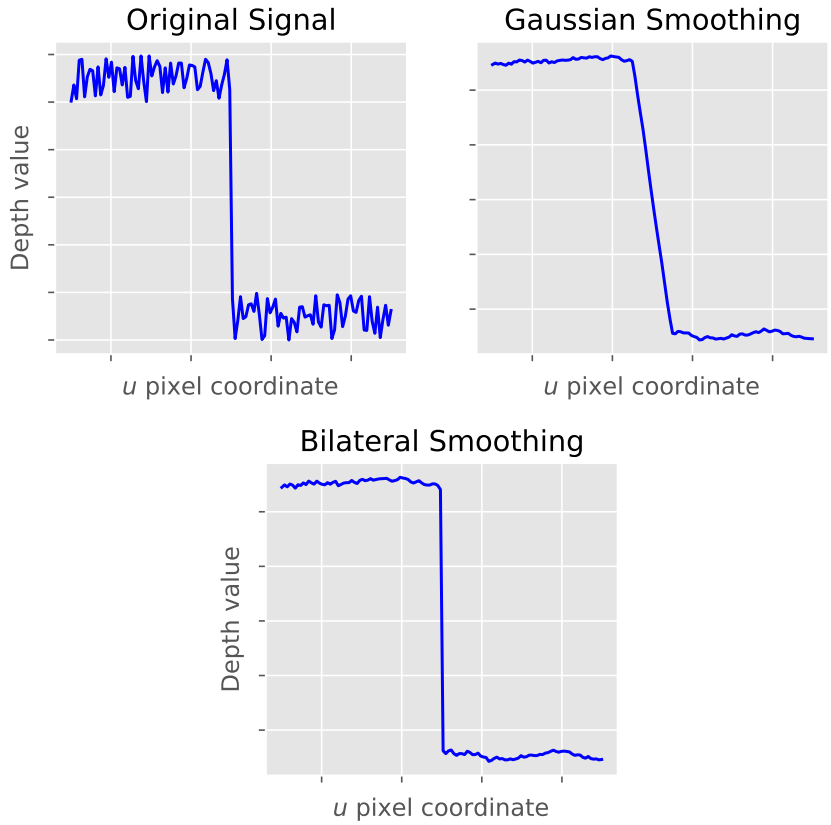


Figure 3.2: A sample depth image scanline (top left). The same scanline after being processed with a gaussian filter (top right). The same scanline after being processed with a bilateral filter (bottom middle).

The normal vectors computed in this stage are used many times throughout the system. First, they are inputs to the ICP's point-to-plane distance during VO and for the fine alignment of submaps. Another example is on the surfel mapping, the angle of the normal vectors from a point in the incoming frame and surfel serves as criteria for merging the information. Furthermore, the normal guides the 3D direction in which the mapping viewer draws a surfel's circle.

3.1.1 Depth Image Filtering

For reducing the noise of depth images, we, as other authors [45, 46, 58, 71, 83], smooth the depth image using the bilateral filter algorithm for preserving edges. Edge preserving operations are essential properties of any processing on depth image since altering depth pixels can significantly change the map's geometry. For example, in the 1D signal corresponding to a horizontal line of a depth image shown in Figure 3.2, a standard Gaussian filter replaces depth discontinuities with a soft decay. In contrast, the bilateral filter smooths the signal and preserves hard discontinuities at the same time.

To understand the bilateral filter's formula, we must first analyze the Gaussian filter

$$\hat{I}_d(\mathbf{u}) = \frac{1}{\sigma_s\sqrt{2\pi}} \sum_{i,j\in\Omega_\mathbf{u}} I_d(i,j) \exp\left(-\frac{1}{2} \frac{\left\| \mathbf{u} - \begin{bmatrix} i & j \end{bmatrix}^\intercal \right\|^2}{2\sigma_s^2}) \right), \tag{3.1.1}$$

where I_d is the input depth image and $\hat{I}_d$ is the output. The point $\mathbf{u}$ is the current pixel's coordinates that the filtering is passing. We let the set $\Omega_\mathbf{u}$ denote the pixel neighborhood coordinates around the filtering window. The exponential term is a Gaussian function on the spatial domain with σ_s being its standard deviation. The bilateral filter differs by adding a second statement that measures the depth discontinuities according to a second normal curve parameterized with the standard deviation σ_d

$$\hat{I}_d(\mathbf{u}) = \frac{1}{\sum_{i,j\in\Omega_\mathbf{u}} \mathcal{W}_{i,j}} \sum_{i,j\in\Omega_\mathbf{u}} I_d(i,j)\mathcal{W}_{i,j}, \tag{3.1.2a}$$

$$\mathcal{W}_{i,j} = \exp\left(-\frac{1}{2} \left(\frac{\left\| \mathbf{u} - \begin{bmatrix} i & j \end{bmatrix}^\intercal \right\|^2}{2\sigma_s^2} + \frac{(I_d(\mathbf{u}) - I_d(i,j))^2}{2\sigma_d^2} \right) \right). \tag{3.1.2b}$$

This extra gaussian function on the signal space results that elements with small depth discontinuities, it approaches 0, and the value of $\mathcal{W}_{i,j}$ becomes closer to the gaussian function on the spatial domain. Otherwise, elements with significant discontinuities will provide a large negative number to the exponential function, hence gradually eliminating a neighbor's contribution to the filter at the coordinate $\mathbf{u}$. Contrary to the standard Gaussian filter of Equation 3.1.1, the bilateral filter algorithm must recalculate the normalization factor to account for the contributions of the depth values. A naive implementation can make a bilateral filter computationally expensive, but fortunately, it can significantly be accelerated by implementing it with a bilateral grid data structure [13].

3.1.2 Normals Computation

To compute the normal vectors of a captured surface, first consider a 3D point originated from the pixel coordinate $\mathbf{u} = \begin{bmatrix} u & v & d \end{bmatrix}^\intercal$ of a depth image. Let its back projection into the 3D local space of its camera be

$$\mathbf{p}_{uv} = \Pi(K, \mathbf{u}). \tag{3.1.3}$$

For computing the normal vector associated with $\mathbf{p}_{uv}$, we approximate the surface where it lies by creating a plane considering its direct pixel neighborhood on 3D space. The plane is built from two coplanar vectors $\mathbf{v}_u \in \mathbb{R}^3$ and $\mathbf{v}_v \in \mathbb{R}^3$ that we calculate from the back projection of the pixel neighbors of $\mathbf{p}_{u,v}$ in the horizontal and vertical pixel directions.

Then the plane's normal vector can be obtained from their cross products:

$$\mathbf{n}_{u,v} = \frac{1}{\|\mathbf{v}_u \times \mathbf{v}_v\|} \mathbf{v}_u \times \mathbf{v}_v \tag{3.1.4}$$

The division ensures that the resulting normal vector is unit length.

However, to calculate the $\mathbf{v}_u$ and $\mathbf{v}_v$ vectors, it is necessary to select the right neighbors. Creating the vectors from fixed neighbors of $\mathbf{p}_{u+1,v}$ like $\mathbf{v}_u = \mathbf{p}_{u+1,v} - \mathbf{p}_{u,v}$ and $\mathbf{v}_v = \mathbf{p}_{u,v+1} - \mathbf{p}_{u,v-1}$ may result in wrong normal products if any of the points are from depth discontinuities, invalid depth values, or even being to close. To avoid those cases, we follow the work of Schops et al.'s [71] and select the neighbor pairs that are within an ideal length between $[\epsilon_1, \epsilon_2]$. Otherwise, it chooses the smaller vector that connects with the center:

$$\mathbf{v}_u = \begin{cases} \mathbf{p}_{u+1,v} - \mathbf{p}_{u-1,v} & \text{if } \epsilon_1 \leq \left\| \mathbf{p}_{u+1,v} - \mathbf{p}_{u-1,v} \right\| \leq \epsilon_2 \\ \mathbf{p}_{u,v} - \mathbf{p}_{u-1,v} & \text{if } \left\| \mathbf{p}_{u,v} - \mathbf{p}_{u-1,v} \right\| \leq \left\| \mathbf{p}_{u+1,v} - \mathbf{p}_{u,v} \right\| \\ \mathbf{p}_{u+1,v} - \mathbf{p}_{u,v} & \text{otherwise} \end{cases}, \tag{3.1.5a}$$

$$\mathbf{v}_v = \begin{cases} \mathbf{p}_{u,v-1} - \mathbf{p}_{u,v+1} & \text{if } \epsilon_1 \leq \left\| \mathbf{p}_{u,v-1} - \mathbf{p}_{u,v+1} \right\| \leq \epsilon_2 \\ \mathbf{p}_{u,v} - \mathbf{p}_{u,v+1} & \text{if } \left\| \mathbf{p}_{u,v} - \mathbf{p}_{u,v+1} \right\| \leq \left\| \mathbf{p}_{u,v-1} - \mathbf{p}_{u,v} \right\| \\ \mathbf{p}_{u,v-1} - \mathbf{p}_{u,v} & \text{otherwise} \end{cases} \tag{3.1.5b}$$

3.2 Visual Odometry

Our SLAM solution computes the VO in a frame-to-frame manner. We estimate 3×4 rigid transformation matrices $T = \begin{bmatrix} R & \mathbf{t} \end{bmatrix}$ for aligning every incoming RGB-D image into the space of its previous one. The final camera poses matrices are the accumulated composition of the pairwise registrations as calculated in Equation 2.6.1.

For computing the transformation T, as other authors [17, 45, 71, 75, 83], we estimate the combined geometric and photometric dense consistency residuals for aligning the frames

For the photometric cost term E_{photo}, we use the same one from Equation 2.6.2

$$\begin{aligned} E_{\text{photo}}(T) &= \sum_{\mathbf{u} \in I^s} \mathcal{L}_\delta \left(r_{\mathbf{u}}(T) w_{\mathbf{u}} r_{\mathbf{u}}(T) \right), \\ r_{\mathbf{u}}(T) &= I^t(\pi(KT\Pi(K, \mathbf{u}))) - I^s(\mathbf{u}), \end{aligned} \tag{3.2.1}$$

its goal to measure the alignment of the transformation T as the difference of the pixelwise intensities of the target image I^t and the source one I^s. The differences are computed by back-project every valid pixel coordinate $\mathbf{u}$ of the source image, transforming them by T, and projecting again on the target image to retrieve its correspondent intensity value. A scalar $w_{\mathbf{u}}$ is again added for discarding bad matches, it is set to either $\mathcal{L}_\delta(r_{\mathbf{u}} r_{\mathbf{u}})$ or 0 according to a threshold the corresponding pixels' intensity difference.

As we did with the geometric term in Section 2.5.1, we apply the GN steps (Section 2.4). The approximation of the photometric residual is

$$r_{\mathbf{u}}(T \boxplus \boldsymbol{\xi}) = I^t(\pi(KT \boxplus \boldsymbol{\xi}\Pi(K, \mathbf{u}))) - I^s(\mathbf{u}), \tag{3.2.2a}$$

$$\approx r_{\mathbf{u}}(T) + J_{\mathbf{u}}\boldsymbol{\xi}. \tag{3.2.2b}$$

A semi-analytical form of the Jacobian $J_{\mathbf{u}}$ can be found by first applying the chain rule. For convenience, let $\mathbf{p}_{\mathbf{u}} = \Pi(K, \mathbf{u})$

$$J_{\mathbf{u}} = \frac{\partial I^t(\pi(KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}})) - I^s(\mathbf{u})}{\partial \boldsymbol{\xi}} = \frac{\partial I^t(\pi(KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}))}{\partial \boldsymbol{\xi}}, \tag{3.2.3}$$

applying the chain rule

$$\frac{\partial I^t(\pi(KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}))}{\partial \pi(KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}})} \frac{\partial \pi(KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}})}{\partial KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}} \frac{\partial KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}}{\partial T \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}} \frac{\partial T \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}}{\partial \boldsymbol{\xi}}. \tag{3.2.4}$$

The jacobian of the image function $I^t(\cdot, \cdot)$ is computed numerically with the finite differences method as follows:

$$\begin{bmatrix} \frac{\partial I^t}{\partial u_x} & \frac{\partial I^t}{\partial u_y} \end{bmatrix} = \begin{bmatrix} \frac{I^t(u+h,v) - I^t(u-h,v)}{2h} & \frac{I^t(u,v+h) - I^t(u,v-h)}{2h} \end{bmatrix}, \tag{3.2.5}$$

where h is a sufficiently small value, and the image function $I^t(\cdot, \cdot)$ returns bilinearly interpolated intensity values for input coordinates with decimal values.

For expanding the projection function jacobian, let the expression $KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}$ be assigned to the variable $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^\mathsf{T}$:

$$\frac{\partial \pi(KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}})}{\partial KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}} = \frac{\partial \pi(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial \begin{bmatrix} x/z & y/z \end{bmatrix}}{\partial \mathbf{x}}, \tag{3.2.6a}$$

$$= \begin{bmatrix} \frac{\partial x/z}{\partial x} & \frac{\partial x/z}{\partial y} & \frac{\partial x/z}{\partial z} \\ \frac{\partial y/z}{\partial x} & \frac{\partial y/z}{\partial y} & \frac{\partial y/z}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{1}{z} & 0 & -\frac{x}{z^2} \\ 0 & \frac{1}{z} & -\frac{y}{z^2} \end{bmatrix}. \tag{3.2.6b}$$

To compute the next jacobian, let $T \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}$ be assigned to M:

$$\frac{\partial KT \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}}{\partial T \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}} = \frac{\partial KM}{\partial M} = K \tag{3.2.7}$$

The last jacobian matches the $\boxplus$ derivative in Equation 2.2.35:

$$\frac{\partial T \boxplus \boldsymbol{\xi}\mathbf{p}_{\mathbf{u}}}{\partial \boldsymbol{\xi}} = \frac{\partial \operatorname{Exp}(\boldsymbol{\xi})T\mathbf{p}_{\mathbf{u}}}{\partial \boldsymbol{\xi}} = \begin{bmatrix} I_{3\times 3} & -[\hat{R}\mathbf{p}_{\mathbf{u}} + \hat{\mathbf{t}}]_\times \end{bmatrix} \tag{3.2.8}$$

Updating Equation 3.2.4 with Equations 3.2.5 to 3.2.8 and the K matrix definition from

Equation 2.3.3, results in:

$$J_{\mathbf{u}} = \begin{bmatrix} \frac{\partial I^t}{\partial u_x} & \frac{\partial I^t}{\partial u_y} \end{bmatrix} \begin{bmatrix} \frac{1}{z} & 0 & -\frac{x}{z^2} \\ 0 & \frac{1}{z} & -\frac{y}{z^2} \end{bmatrix} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I_{3\times 3} & -[R\mathbf{p}_{\mathbf{u}} + \mathbf{t}]_\times \end{bmatrix}. \tag{3.2.9}$$

We can evaluate the expression in the middle to

$$\begin{bmatrix} \frac{1}{z} & 0 & -\frac{x}{z^2} \\ 0 & \frac{1}{z} & -\frac{y}{z^2} \end{bmatrix} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{f_x}{z} & 0 & \frac{c_x}{z} - \frac{f_x x + c_x z}{z^2} \\ 0 & \frac{f_y}{z} & \frac{c_y}{z} - \frac{f_y y + c_y z}{z^2} \end{bmatrix}. \tag{3.2.10}$$

Turning Equation 3.2.9 into

$$J_{\mathbf{u}} = \begin{bmatrix} \frac{\partial I^t}{\partial u_x} & \frac{\partial I^t}{\partial u_y} \end{bmatrix} \begin{bmatrix} \frac{f_x}{z} & 0 & \frac{c_x}{z} - \frac{f_x x + c_x z}{z^2} \\ 0 & \frac{f_y}{z} & \frac{c_y}{z} - \frac{f_y y + c_y z}{z^2} \end{bmatrix} \begin{bmatrix} I_{3\times 3} & -[R\mathbf{p}_{\mathbf{u}} + \mathbf{t}]_\times \end{bmatrix}. \tag{3.2.11}$$

Let $\mathbf{r} \in \mathbb{R}^3$ be

$$\mathbf{r} = \begin{bmatrix} \frac{\partial I^t}{\partial u_x} & \frac{\partial I^t}{\partial u_y} \end{bmatrix} \begin{bmatrix} \frac{f_x}{z} & 0 & \frac{c_x}{z} - \frac{f_x x + c_x z}{z^2} \\ 0 & \frac{f_y}{z} & \frac{c_y}{z} - \frac{f_y y + c_y z}{z^2} \end{bmatrix}, \tag{3.2.12}$$

and let $\mathbf{v} \in \mathbb{R}^3$ be

$$\mathbf{v} = -\mathbf{r} \times (R\mathbf{p}_{\mathbf{u}} + \mathbf{t}), \tag{3.2.13}$$

then resolving Equation 3.2.11, the final values of $J_{\mathbf{u}}$ are

$$J_{\mathbf{u}} = \begin{bmatrix} r_1 & r_2 & r_3 & v_1 & v_2 & v_3 \end{bmatrix} \tag{3.2.14}$$

With the jacobian for $J_{\mathbf{u}}$, we now compute H_{photo}, $\mathbf{b}_{\text{photo}}$

$$H^{\text{photo}} = \sum_{\mathbf{u}}^{I^s} J_{\mathbf{u}} w_{\mathbf{u}} J_{\mathbf{u}}, \tag{3.2.15a}$$

$$\mathbf{b}^{\text{photo}} = \sum_{\mathbf{u}}^{I^s} r_{\mathbf{u}}(T) w_{\mathbf{u}} J_{\mathbf{u}}. \tag{3.2.15b}$$

To combine both the geometric and photometric terms, we change Algorithm 1 to sum H^{geom} with $H^{\text{photo}}\alpha^{\text{photo}}$, and $\mathbf{b}^{\text{geom}}$ with $\mathbf{b}^{\text{photo}}\alpha_{\text{photo}}$. The final VO algorithm is listed on Algorithm 2, where we added a common [45, 83] heuristic for discarding potential bad associations, if the normals of two corresponding points $\mathbf{p}_i$ and $\mathbf{q}_i$ have normals with not more than 45 degrees apart.

While computing the VO, we ran the Algorithm 2 with a 3-scale Gaussian pyramid for a 640×480 input image. Executing the algorithm first at smaller resolution scales improves the convergence because the energy function will be smoother since high-frequency details are diminished in those scales. By having a better initial guess obtained from a lower-scale run, the algorithm will less likely attract its parameter estimation to a local minimum.

In the final step, our SLAM system also returns the inverse of the hessian of the

Algorithm 2 RGB-D ICP

Input: Source intensity, depth, and normal images $I^s, D^s \in \mathbb{R}^{W\times H}, N^s \in \mathbb{R}^{W\times H\times 3}$; target intensity, depth and normal images $I^t, D^t \in \mathbb{R}^{W\times H}, N^t \in \mathbb{R}^{W\times H\times 3}$; intrinsic matrix $K \in \mathbb{R}^{3\times 3}$; initial guess transformation matrix $\hat{T} \in SE(3)$.

$T \leftarrow \hat{T}$
while maximum number of iterations **do**
  let $R, \mathbf{t} = T$
  for Every pixel coordinate $\mathbf{u} = \begin{bmatrix} u & v & d \end{bmatrix}$ in the depth image D^s **do**
    let $\mathbf{p}_i = \Pi(K, \mathbf{u})$,
      $\mathbf{v} = D^t[\pi(R\mathbf{p}_i + \mathbf{t})]$,
      $\mathbf{q}_i = \Pi(K, \mathbf{v})$,
      $\mathbf{n}_i = N^t[\mathbf{u}]$

    $J_i^g \leftarrow \begin{bmatrix} n_1 & n_2 & n_3 & -\mathbf{n}_i \times (R\mathbf{p}_i + \mathbf{t}) \end{bmatrix}$
    $r_i^g \leftarrow (\mathbf{n}_i^\intercal (R\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i))^2$
    $w_i^g \leftarrow \begin{cases} \mathcal{L}_\delta(r_i^g) & \mathbf{n}_i \angle N^s[\mathbf{v}] \leq 70^\circ \\ 0 & \text{otherwise} \end{cases}$

    let $\mathbf{r} = \begin{bmatrix} \frac{\partial I^t}{\partial u_x}\frac{f_x}{z} & \frac{\partial I^t}{\partial u_y}\frac{f_y}{z} & \frac{\partial I^t}{\partial u_x}(\frac{c_x}{z} - \frac{f_x x + c_x z}{z^2}) + \frac{\partial I^t}{\partial u_y}\frac{f_y}{z}(\frac{c_y}{z} - \frac{f_y y + c_y z}{z^2}) \end{bmatrix}$
    $J_i^p \leftarrow \begin{bmatrix} r_1 & r_2 & r_3 & -\mathbf{r} \times (R\mathbf{p}_i + \mathbf{t}) \end{bmatrix}$
    $r_i^p \leftarrow (I^t(\mathbf{v}) - I^s(\mathbf{u}))^2$
    $w_i^p \leftarrow \begin{cases} \mathcal{L}_\delta(r_i^p) & r_i^p \leq \epsilon_I \\ 0 & \text{otherwise} \end{cases}$
  end for

  $H \leftarrow \sum_i J_i^{g\intercal} w_i^g J_i^g + \sum_i J_i^{p\intercal} w_i^p J_i^p \alpha_p$
  $\mathbf{b} \leftarrow \sum_i r_i^g w_i^g J_i^g + \sum_i r_i^p w_i^p J_i^p \alpha_p$

  $\boldsymbol{\xi}^* \leftarrow \text{Cholesky}(H, \mathbf{b})$
  $T \leftarrow T \boxplus \boldsymbol{\xi}^*$
end while
output T, H^{-1} ▷ It also outputs the inverse hessian for uncertainty approximation.

geometric term, H^{-1}_{geom}, as the found transformation uncertainty. This matrix later will be the information matrix for the edges that unites two odometry-connected submaps.

3.3 Surfel-based Mapping

Our surfel mapping is based on previous works [46, 82, 83] on surfel SLAM. In essence, every surfel of a submap at an index s have the following properties:

- A position $\mathbf{p}_s \in \mathbb{R}^3$;
- A normal vector $\mathbf{n}_s \in \mathbb{R}^3$ for, among other uses, orienting the surfel's disk;

- An RGB color value $\mathbf{c}_s \in \mathbb{R}^3$ from their pixel assessments;
- A radius value $r_s \in \mathbb{R}$ of its circle;
- A descriptor $\mathbf{g}_s \in \mathbb{R}^D$ obtained from the inference of our CNN feature model with the frames' images;
- A confidence value $w_s \in \mathbb{R}$ indicating its stability. A surfel is considered stable enough when it reaches a certain threshold. Our SLAM system increases it when new incoming frame's pixel is found to correspond to its surfel.
- A timestamp $t_s \in \mathbb{N}$ of its last update time. The timestamp will be used to remove surfels that remain unstable for a certain period.
- Keypoint count k_s. The number of times that a pixel measurement of this surfel was detected to be a good feature to track by the SIFT detector.
- For comparing with the deep features for loop closure detection, we extract an optional SIFT descriptor $\mathbf{s}_s \in \mathbb{R}^{128}$. The descriptor is only allocated if the surfel was detected to be a keypoint on one of its image measurements.

The surfel mapping submodule is called at every incoming RGB-D frame to integrate its pixels into the current submap. The integration can either add the RGB-D pixels as new surfels or update existing ones with new measurements on those that are likely to be the same point on the scene. By updating an existing surfel with a depth pixel, the mapping submodule refines the surfel's attributes by merging their values. Hence, the update operation also increases the stability w_s value since the surfel incorporates more measurements from the scene. When a surfel reaches the stability threshold, it is declared part of the submap and cannot be removed. After integrating a frame, a second stage finds close stable surfels and merges them to remove redundant surfels from the submap. Before exiting its per frame call, the mapping subsystem deletes surfels that remained unstable for a certain amount of time.

As stated in Section 1.1, the surfel mapping can adapt to the detail given by the capture at different distance moments from the surfaces. This detail capture is possible because, at the integration step, when the algorithm finds a surfel merge with a pixel representing a smaller surfel, the algorithm will reduce the surfel radius, so it represents a lesser portion of its surface. This is later show on Equation 3.3.3e.

The system stops reconstructing the current submap when it emits a new keyframe. It saves the submap to the memory inside a new pose graph node. Upon stopping, the system also reset the mapping subsystem for preparing to generate the next submap.

Now we look in more detail at the integration of an incoming RGB-D frame. Given a frame, we iterate on its valid depth coordinates $\mathbf{u} = \begin{bmatrix} u & v & d \end{bmatrix}^\intercal$ and find the surfel closest to ray formed from the camera's center $\mathbf{t}$ to its back projection $\mathbf{x}$. The camera's center $\mathbf{t}$ is just the translation vector of the current camera pose $\begin{bmatrix} R \mid \mathbf{t} \end{bmatrix}$ obtained from the VO. Let $\mathbf{x} \in \mathbb{R}^3$ be the back projection for some $\mathbf{u}$

$$\mathbf{x} = \Pi(K, \mathbf{u}) \tag{3.3.1}$$

Then we calculate the distance from the ray from $\mathbf{t}$ passing through $\mathbf{x}$ and any surfel candidate $\mathbf{p}_s$ using the ray to point distance

$$d(\mathbf{x}, \mathbf{p}_s) = \frac{\left\|(\mathbf{p}_s - \mathbf{t}) \times (R\mathbf{x})\right\|}{\|R\mathbf{x}\|}. \tag{3.3.2}$$

Since we need to transform $\mathbf{x}$ into world space by multiplying it with the camera pose $\begin{bmatrix} R \mid \mathbf{t} \end{bmatrix}$, instead of explicitly calculating the ray with $R\mathbf{x} + \mathbf{t} - \mathbf{t}$, we write only with the rotation matrix since both translations cancel each out.

Nevertheless, for being considered for merging with an incoming ray, a surfel candidate must be no farther than a small threshold distance from it. Also, the surfel's normal and the normal associated with image coordinate $\mathbf{u}$ — calculated during preprocessing (Section 3.1.2) — must be not more than 45° degrees apart, i.e., $\mathbf{n}_s \angle \mathbf{n}_\mathbf{u} < 45°$.

When a surfel and an incoming ray satisfy the distance and normal requirements, then the mapping submodule merges the point $\mathbf{x}$ properties into the surfel by the weighted average according to the surfel's confidence w_s and an estimated confidence $w_\mathbf{u}$ of the incoming point:

$$\mathbf{p}_s \leftarrow \frac{w_s \mathbf{p}_s + w_\mathbf{u}(R\mathbf{x} + \mathbf{t})}{w_s + w_\mathbf{u}}, \tag{3.3.3a}$$

$$\mathbf{n}_s \leftarrow \frac{w_s \mathbf{n}_s + w_\mathbf{u}(R^{-1})^\intercal \mathbf{n}_\mathbf{u}}{w_s + w_\mathbf{u}}, \tag{3.3.3b}$$

$$\mathbf{c}_s \leftarrow \frac{w_s \mathbf{c}_s + w_\mathbf{u} \mathbf{c}_\mathbf{u}}{w_s + w_\mathbf{u}}, \tag{3.3.3c}$$

$$\mathbf{g}_s \leftarrow \frac{w_s \mathbf{g}_s + w_\mathbf{u} \mathbf{g}_\mathbf{u}}{w_s + w_\mathbf{u}}, \tag{3.3.3d}$$

$$r_s \leftarrow \min(r_s, r_\mathbf{u}), \tag{3.3.3e}$$

$$k_s \leftarrow \begin{cases} 1 & \text{if the pixel is a keypoint,} \\ 0 & \text{otherwise} \end{cases} \tag{3.3.3f}$$

$$t_s \leftarrow [\text{current timestamp}], \tag{3.3.3g}$$

$$w_s \leftarrow w_s + w_\mathbf{u}. \tag{3.3.3h}$$

The $w_\mathbf{u}$ confidence level of the incoming point measurement is based on a Gaussian function with the mean being the pixel's distance to the image's center

$$w_\mathbf{u} = \exp\left(-\frac{\left\| \begin{bmatrix} u & v \end{bmatrix}^\intercal - \begin{bmatrix} K_{13} & K_{23} \end{bmatrix}^\intercal \right\|}{2\alpha^2} \right), \tag{3.3.4}$$

where α is an empirically found [46] ratio set to $\alpha = 0.6$. This weighting originated from the work of Keller et al. [46]. They have observed that depth values near to image's center are freer from noisy measurements. In Equation 3.3.3b, the incoming normal vector $\mathbf{n}_\mathbf{u}$ is oriented according to the world space using the transposed inverse of the rotation $(R^{-1})^\intercal$ to remove any potential scaling. We set the fusion to estimate a conservative small radius

value $r_{\mathbf{u}}$ for the incoming element. Its formula uses the normal's z coordinate for weighting the size according to the degree that it is facing the camera

$$r_{\mathbf{u}} = \frac{1}{\sqrt{2}} \frac{d/f}{[\mathbf{n}_{\mathbf{u}}]_z} \tag{3.3.5}$$

Equation 3.3.3e merges the radius with the lesser one, so it increases the mapping resolution. Equation 3.3.3f increment the surfel's counter if SIFT detects the source pixel as a keypoint. When running for evaluating the SIFT descriptors, we also append it to the optional feature field $\mathbf{f}_s$

$$\mathbf{f}_s \leftarrow \mathbf{f}_s + [\text{Incoming SIFT descriptor}]. \tag{3.3.6}$$

If the integration algorithm does not find any compatible surfel to match the current ray, then it adds a new surfel with its properties being

$$\begin{aligned}
\mathbf{p}_{S+1} &\leftarrow R\mathbf{x} + \mathbf{t}, \\
\mathbf{n}_{S+1} &\leftarrow (R^{-1})^{\intercal}\mathbf{n}_{\mathbf{u}}, \\
\mathbf{c}_{S+1} &\leftarrow I(\mathbf{u}), \\
r_{S+1} &\leftarrow r_{\mathbf{u}}, \\
\mathbf{g}_{S+1} &\leftarrow G(\mathbf{u}), \\
t_{S+1} &\leftarrow [\text{current timestamp}], \\
w_{S+1} &\leftarrow w_{\mathbf{u}} \\
k_{S+1} &\leftarrow \begin{cases} 1 & \text{if the pixel is a keypoint,} \\ 0 & \text{otherwise} \end{cases}.
\end{aligned} \tag{3.3.7}$$

Again, during the SIFT descriptor evaluation mode we set the descriptor:

$$\mathbf{f}_s \leftarrow \text{incoming SIFT descriptor}, \tag{3.3.8}$$

if the incoming point is keypoint.

For querying surfels that are close to the casted rays in an efficient manner, we employ the technique [46] of generating an index map by rendering the submap's surfels from the perspective of the current pose. However, rather than drawing any visual properties of the surfels, we write the surfels' indices s using the graphics hardware pipeline to a 32-bit integer framebuffer. Later, the mapping submodule converts it into an image for fast look-up by the integration algorithm. We render the index map into an image four times bigger than the input frame, so it can account for overlapping surfel's points. With the index map, the integration algorithm can efficiently lookup the index of the surfels already in the submap that are potential neighbors of any incoming ray's origin pixel $\mathbf{u}$.

After updating the submap with an RGB-D frame, we do further processing to merge stable surfels that end up being near after sequential updates. The merge algorithm renders an indexmap again and uses it to locate pairs of stable surfels that are normal vector compatible and have overlapping radii. Those pairs are then merged similarly to

the pixels of Equation 3.3.3.

The last step is to remove unstable surfels. We remove every surfel in the submap with a confidence value w_s less than a minimum threshold $\mathcal{W}$, and an elapsed time greater than $\mathcal{T}$ since its last update. In other words, all surfels s with $w_s < \mathcal{W}$ and $t_c - [\text{current timestamp}] > \mathcal{T}$ are removed. The removal operation will delete most spurious surfels and those with insufficient measurements to have a reliable position.

3.3.1 Implementation Details

Our system stores the submap surfels in GPU memory. They are accessible by both OpenGL for rendering and by our CUDA kernels for manipulating the surfels directly during an update, merge, or removal operations. Those functionalities are inside a custom Python package TensorViz [1].

For rendering and maintaining a submap, we allocate a large memory chuck in the GPU for holding a maximum possible number of surfels. Along with the other surfels' properties, we add an extra flag property to indicate if they are allocated or not. Therefore, the surfel adding and delete operations merely change this flag. Those surfel entries with the flag marked are passed to a geometry shader that generates the surfel's disk during rendering.

3.4 Loop Closure Detection

The loop closure detection submodule runs after the system outputs a new submap. It iterates throughout all previous submaps and adds a loop closure between two of them if the FGR keypoint-based registration algorithm finds an alignment result that ensures a low ratio of correspondences outliers.

As saw in Section 2.5.3, FGR is based on sparse keypoint comparison, for selecting them from our dense submaps, we select the $\mathbf{p}_s$ surfels that have keypoint counter k_s greater than or equal to 2. Those keypoints are then matched together using their deep features $\mathbf{g}_s$ on their surfels to find the closest pairs. With those 3D point correspondences from surfels, we proceed in finding the alignment using the FGR implemented by the Open3D [88] project.

After executing the fast registration for a pair, the system adds a loop closure edge between them to the current pose graph if the answer has a ratio of outliers to inliers above 5%. The added edge's covariance is the simple hessian approximation of Equation 2.6.17 described in Section 2.6.2. As the approximation is based on the ICP, we calculate the hessian of the linearized point-to-plane distance of the estimated transformation using all surfel's points from a pair of submaps.

[1] https://gitlab.com/mipl/3d-reconstruction/tensorviz

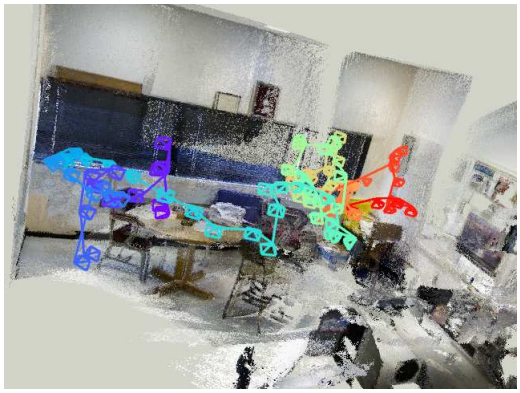


Figure 3.3: Visualization of a pose graph along with its submaps.

3.5 Pose Graph Optimization

From the VO with RGB-D images (Section 3.2), we find the odometry edges while also constructing the submaps' geometries. We also have defined a loop closure method (Section 3.4) method to find edges between non-adjacent submaps. Now with those nodes and edges, and along with their uncertainties in the form of covariance matrices (Section 2.6.2), we input them into the PGO algorithm for globally estimating the submaps poses.

For performing the PGO, we use the algorithm proposed by Choi et al. [17] from the Open3D library [88] implementation. Their version of the PGO algorithm is the line-process for filtering outlier loop closures, which we described in Section 2.6.4.

The system does not use the submaps during the PGO, but we keep them in the nodes to facilitate the implementation of visualization, as in Figure 3.3, and other registration algorithms that require them. An edge, either estimated from odometry or by loop closure, contains the transformation T_{st} being the relative one from the pose of the node s to the t.

3.6 Pairwise Submap Registration

Our solution runs pairwise registration for refining the alignment between consecutive submaps after PGO execution. This registration algorithm uses the same methodology of GN optimization that we employed for the VO module. Although, its alignment energy function calculates the squared differences of features from the nearest target surfels to the source ones when both surfel sets are aligned with a transformation T

$$E_{\text{feat}}(T) = \sum_{s}^{S} \mathcal{L}_{\delta}(\mathbf{r}_s(T)^{\intercal} \omega_s \mathbf{r}_s(T)), \tag{3.6.1a}$$

$$\mathbf{r}_s(T) = G(T\mathbf{p}_s) - \mathbf{g}_s^s, \tag{3.6.1b}$$

where S is the total number of surfels in the source set. The value ω_s is the surfel association confidence score based on their difference [2]. The vectors $\mathbf{p}_s \in \mathbb{R}^3$ and $\mathbf{g}_s^s \in R^D$ are the $i$nth position and feature of the source surfel presented in Section 3.3. The function G calculates the weighted-sum of the features of the k-nearest surfels on the target submap around a 3D coordinate $\mathbf{p} \in \mathbb{R}^3$

$$G(\mathbf{p}) = \frac{1}{\sum_{k \in \text{knn}(\mathbf{x})} \text{sim}(\mathbf{q}_k, \mathbf{p})} \sum_{k \in \text{knn}(\mathbf{x})} \mathbf{g}_k^t \, \text{sim}(\mathbf{q}_k, \mathbf{p}), \tag{3.6.2}$$

where $\mathbf{q}_k$ is the k-closest neighbor surfel to $\mathbf{x}$ in the target submap's surfels, and $\mathbf{g}_k^t$ is its feature descriptor. The function $\text{sim}(\cdot, \cdot)$ returns the similarity of points $\mathbf{q}_k$ and $\mathbf{p}$ by their inverse distance:

$$\text{sim}(\mathbf{q}_k, \mathbf{p}) = \frac{1}{(1 + \|\mathbf{q}_k - \mathbf{p}\|)}. \tag{3.6.3}$$

For querying the closest surfels of the expression $knn(\mathbf{x})$, the implementation uses a KD-Tree for fast searching over the target surfel's 3D positions. Continuing the preparation for using GN, we linearize the residual term $\mathbf{r}_s$ from Equation 3.6.1 with a $\boldsymbol{\xi} \in \mathfrak{se}(3)$ increment on the current transformation matrix T

$$\mathbf{r}_s(T \boxplus \boldsymbol{\xi}) \approx \mathbf{r}_s(T) + J_s \boldsymbol{\xi}. \tag{3.6.4}$$

Decomposing the jacobian with the chain rule

$$J_s = \frac{\partial G(T \boxplus \boldsymbol{\xi} \mathbf{p}_s) - \mathbf{g}_s^s}{\partial \boldsymbol{\xi}}, \tag{3.6.5a}$$

$$= \frac{\partial G(T \boxplus \boldsymbol{\xi} \mathbf{p}_s)}{\partial \boldsymbol{\xi}}, \tag{3.6.5b}$$

$$= \frac{\partial G(T \boxplus \boldsymbol{\xi} T \mathbf{p}_s)}{\partial T \boxplus \boldsymbol{\xi} T \mathbf{p}_s} \frac{\partial T \boxplus \boldsymbol{\xi} T \mathbf{p}_s}{\partial \boldsymbol{\xi}}. \tag{3.6.5c}$$

The second term of the jacobian was analytically derived in Equation 2.2.35, expanding into

$$\frac{\partial T \boxplus \boldsymbol{\xi} \mathbf{p}_s}{\partial \boldsymbol{\xi}} = \frac{\partial \, \text{Exp}(\boldsymbol{\xi}) T \mathbf{p}_s}{\partial \boldsymbol{\xi}} = \begin{bmatrix} I_{3\times 3} & -[R\mathbf{p}_s + \mathbf{t}]_\times \end{bmatrix}. \tag{3.6.6}$$

However, in the case of the first jacobian, its value must be numerically computed. To visualize the terms, consider their structure:

$$\frac{\partial G(T \boxplus \boldsymbol{\xi} \mathbf{p}_s)}{\partial T \boxplus \boldsymbol{\xi} \mathbf{p}_s} = \frac{\partial G(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial G_1}{\partial x} & \frac{\partial G_1}{\partial y} & \frac{\partial G_1}{\partial z} \\ \frac{\partial G_2}{\partial x} & \frac{\partial G_2}{\partial y} & \frac{\partial G_2}{\partial z} \\ \vdots & \dots & \vdots \\ \frac{\partial G_D}{\partial x} & \frac{\partial G_D}{\partial y} & \frac{\partial G_D}{\partial z} \end{bmatrix}, \tag{3.6.7}$$

where we let the vector $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^\intercal$ be $T \boxplus \boldsymbol{\xi} \mathbf{p}_s$ for writing convenience. The c subscript in G_c refers to the channel index of the returned feature. To compute the numerical partial derivatives $\frac{\partial G_c}{\partial x}, \frac{\partial G_c}{\partial y}, \frac{\partial G_c}{\partial z}$ for all c channels of the descriptor, we use a limited version of

[2] In our previous equations we designated the letter w for the weight factors, but in this section we change the symbol to an ω to not mix with the surfel weight w_s.

the finite differences. First, let the vectors $\mathbf{h}_x$, $\mathbf{h}_y$ and $\mathbf{h}_z$ hold a small increment h over, respectively, the x, y and z-direction like below

$$\mathbf{h}_x = \begin{bmatrix} h, 0, 0 \end{bmatrix}^\intercal, \mathbf{h}_y = \begin{bmatrix} 0, h, 0 \end{bmatrix}^\intercal, \mathbf{h}_z = \begin{bmatrix} 0, 0, h \end{bmatrix}^\intercal. \tag{3.6.8}$$

When computing the completer finite differences for ∂G_c, we should perturb the input's coordinates with by h value on x, y and z directly on G, as following

$$\frac{\partial G_c}{\partial x} = \frac{G_c(\mathbf{x} + \mathbf{h}_x) - G_c(\mathbf{x} - \mathbf{h}_x)}{2h}, \tag{3.6.9a}$$

$$\frac{\partial G_c}{\partial y} = \frac{G_c(\mathbf{x} + \mathbf{h}_y) - G_c(\mathbf{x} - \mathbf{h}_y)}{2h}, \tag{3.6.9b}$$

$$\frac{\partial G_c}{\partial z} = \frac{G_c(\mathbf{x} + \mathbf{h}_z) - G_c(\mathbf{x} - \mathbf{h}_z)}{2h}. \tag{3.6.9c}$$

Nevertheless, the function G_c would require two KNN searches per coordinate, six in total for the derivatives w.r.t. to x, y, and z. To mitigate sensitively performance impact, we adopt an approximation to reuse the same KNN query by assuming that small increments do not change the query's neighborhood, but just their weighted similarity suffers modifications. In that way, we expand the definition of Equation 3.6.1 and add the small increments of finite differences on the similarity function

$$\frac{\partial G_c(\mathbf{x})}{\partial x} \approx \frac{\sum_k^{\text{knn}(\mathbf{x})} (g_{kc}^t \, \text{sim}(\mathbf{p}_k^t, \mathbf{x} + \mathbf{h}_x) - g_{kc}^t \, \text{sim}(\mathbf{p}_k^t, \mathbf{x} - \mathbf{h}_x))}{2h}, \tag{3.6.10a}$$

$$\frac{\partial G_c(\mathbf{x})}{\partial y} \approx \frac{\sum_k^{\text{knn}(\mathbf{x})} (g_{kc}^t \, \text{sim}(\mathbf{p}_k^t, \mathbf{x} + \mathbf{h}_y) - g_{kc}^t \, \text{sim}(\mathbf{p}_k^t, \mathbf{x} - \mathbf{h}_y))}{2h}, \tag{3.6.10b}$$

$$\frac{\partial G_c(\mathbf{x})}{\partial z} \approx \frac{\sum_k^{\text{knn}(\mathbf{x})} (g_{kc}^t \, \text{sim}(\mathbf{p}_k^t, \mathbf{x} + \mathbf{h}_z) - g_{kc}^t \, \text{sim}(\mathbf{p}_k^t, \mathbf{x} - \mathbf{h}_z))}{2h}, \tag{3.6.10c}$$

where g_{kc} is the value on the c channel of the feature in the $k$nth surfel of the target submap. Finally, joining the results of Equations 3.6.6 and 3.6.10 into one jacobian product

$$J_s = \begin{bmatrix} \frac{\partial G_1}{\partial x} & \frac{\partial G_1}{\partial y} & \frac{\partial G_1}{\partial z} \\ \frac{\partial G_2}{\partial x} & \frac{\partial G_2}{\partial y} & \frac{\partial G_2}{\partial z} \\ & \cdots & \\ \frac{\partial G_D}{\partial x} & \frac{\partial G_D}{\partial y} & \frac{\partial G_D}{\partial z} \end{bmatrix} \begin{bmatrix} I_{3\times3} & -[R\mathbf{p}_s + \mathbf{t}]_\times \end{bmatrix}, \tag{3.6.11}$$

we can then compute its H and $\mathbf{b}$ values to use with GN

$$H_{\text{feat}} = \sum_s J_k^\intercal \omega_s J_s, \tag{3.6.12}$$

$$\mathbf{b}_{\text{feat}} = \sum_s \mathbf{r}_s(T) \omega_s J_s. \tag{3.6.13}$$

where we adjust $\omega_s = \mathcal{L}'_\delta(r_s(T)^2)$.

Besides feature term, we also compute geometric terms with the point-to-plane cost. The submap registration final algorithm like Algorithm 2, only replacing the projection-

based mechanism for finding the point pair correspondences with a KD-Tree.

From our research, our algorithm is the first designed to register point clouds that contain dense and high-dimensional features on non-grid structures using our numerical approximation of the nearest neighbor differentiation. The closest methods include color information from RGB-D images, having an intrinsic grid structure for numerical differentiation (i.e., Equation 3.2.5), or the state-of-art ColoredICP [61] that computes intensity gradients from point clouds by projecting their points onto virtual cameras.

As in VO, it is also beneficial to execute the submap registration with multiple scales for gradually converging the estimation into a global minimum. We use the same gaussian pyramid mode of downsampling the submaps by powers of 2. To downsample loose point sets as the surfel submaps, we put the surfels into a volume of a given voxel size at every step. During this process, we average the properties of the surfels that end up on the same voxels. Although we use a volume for this process, it returns a new lower resolution submap.

3.7 Descriptor Learning

So far, we have been examining how this dissertation's RGB-D SLAM system works; this next section examines how to train the deep descriptor CNN model to extract the pixelwise descriptors that we integrate on the submap and use for the loop closure and pairwise registration tasks.

3.7.1 CNN Architecture

We use the DRN architecture from previous works [29, 70] that had good results on dense descriptor learning. To describe the architecture, we will refer to a convolutional block as an arrange of few convolutional, batch normalization, and activation layers. Those three layers are frequently connected on our architectures. The ResNet authors add the batch normalizations after every convolutional layer throughout the architectures to improve training convergence speed [44]. The DRN model family consists of an initial convolutional block, a ResNet-32 architecture with dilated convolutions, and final convolution and upscale layers to generate the per-pixel descriptors. In the following paragraphs, a description of those blocks is given.

The first convolutional block consists of a convolution layer, batch normalization, and activation layer. It convolves the entire image with a 7×7 kernel size filter, which shows effectiveness in various tasks to capture local patterns in imagining tasks.

The second part is basically the ResNet-32 architecture being a set of groups of a convolutional block know as residual convolutional blocks. A residual convolutional block is a sequence of 2 convolutions, each one followed by a batch normalization and activation layers. Together with them, there is a skip connection summing the input feature map with the output one, following the residual mechanism to mitigate the gradient vanishing problem [40]. Figure 3.4 has a diagram of a residual convolutional block. When specifying a residual block, the convolutional layers have the same parameters, except for stride size for avoiding over downsampling. In the skip connection, ResNet-32 has a third convolution

Operation Layer/Block	Number of Filters	Kernel Size	Stride	Activa -tion	Dilata -tion	Size
Input				-	-	$640 \times 480 \times 3$
Convolution Block	64	7×7	2×2	ReLU	-	$320 \times 240 \times 64$
MaxPool	-	3×3	2×2	-	-	$160 \times 120 \times 64$
ResNet Group (3)	64	3×3	1×1	ReLU	-	$160 \times 120 \times 64$
ResNet Group (4)	128	3×3	2×2	ReLU	-	$80 \times 60 \times 128$
ResNet Group (6)	256	3×3	1×1	ReLU	2×2	$80 \times 60 \times 256$
ResNet Group (3)	512	3×3	1×1	ReLU	4×4	$80 \times 60 \times 512$
Pointwise Convolution	D	1×1	1×1	-	-	$80 \times 60 \times D$
Upscale	-	-	-	-	-	$640 \times 480 \times D$

Table 3.1: The Dilated ResNet32 architecture. The number inside the parenthesis on the ResNet Groups refers to the number of ResNet's basic blocks are inside the group. The number D refers to the dense descriptor size.

layer for downsampling when the first one downscales (e.g., stride size greater than 1) the spatial size of the input, so the element-wise sum with the output can be valid. The ResNet-based architectures are divided into groups because they can be modified to increase or reduce the number of parameters according to the task by just modifying the number of residual blocks. In the ResNet-32 DRN configuration, the network has 4 ResNet Groups, with respectively, 3, 4, 6, and 3 consecutive residual blocks show in table 3.1, where the number of residual blocks inside a group is indicated between parenthesis, like ResNet Group (3) to indicate 3 ResNet blocks. Also, in the architecture, the two later blocks include the dilated convolution so that the features at the end include larger receptive fields to compute the descriptor.

The last layers learn a linear regression to map the 512-channel size into our target descriptor size. At that point, the spatial size of the layer 80×60, to have a per-pixel descriptor, is upscaled to match the input resolution size.

With this architecture we input a 640×480 RGB images and obtain a descriptor of each pixel, either being $D = 16$ or $D = 32$.

3.7.2 Self-supervised Dataset for Descriptor Learning

To generate the training samples, we build an approach of selecting two random overlapping image pairs from the RGB-D sequence dataset and finding correspondent pixel pairs with the aid of the depth images and annotated camera poses.

One of the critical challenges of such approaches is how to find image pairs with diverse view angles in an efficient manner. For instance, randomly choosing frame pairs

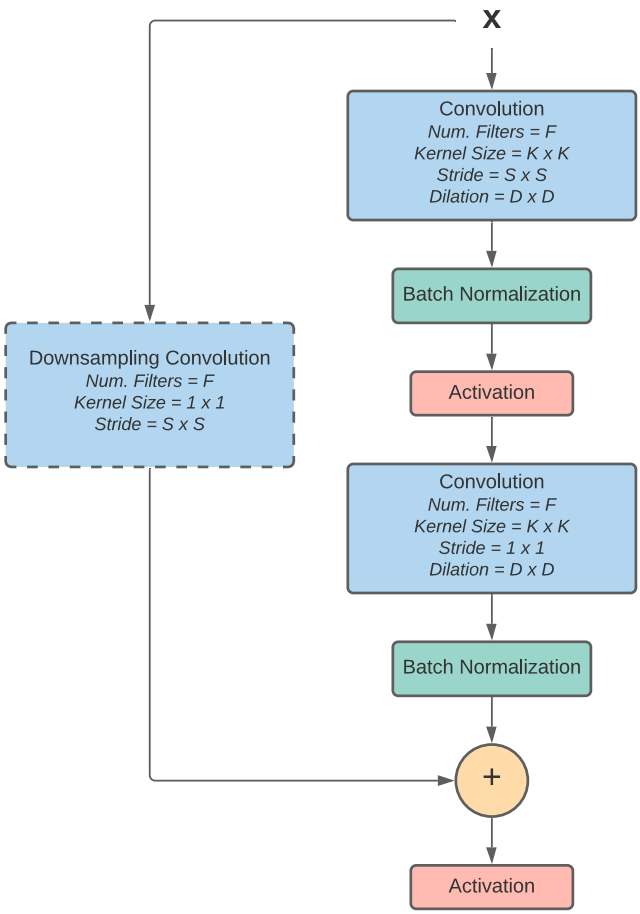


Figure 3.4: Diagram of the ResNet's residual convolution block. The input feature map **x** first passes through a convolution layer that may downsample it and then to a second convolution for learning more convolutional filters. After, the feature map **x** is added to the result to mitigate the vanish gradient problem. If the first layer downsamples the input, then the connection also has one for matching the tensor sizes. The batch normalization layers act for improving training convergence.

may take too many trials to find overlapping ones. A way to avoid too many trials from random processes is limiting the sampling to the ones near in time, but it will degrade the viewing angle variety. To have the best of both ideas, before training, we build an undirected graph connecting every frame pair permutation with a view intersection. Our graph building routine considers a view overlap as any frame pair with at least 50% shared viewing and additionally their viewing angles are at a maximum of 75° apart. To calculate the shared viewing area, we compute [3] the polygon resulting from the intersection of the two cameras' frustum geometry. A camera frustum is a closed pyramid polygon structure build from pose and intrinsic parameters annotation that represents the volume which a camera observes. The angle restriction is for avoiding cases when the sensors' frustums overlap but they are fronting each other, hence, not observing the same scene.

[3] We use polygon boolean operations from the Computational Geometry Algorithms Library [78] (CGAL)

Our self-supervision mechanism was key for training our network with larger RGB-D scenes since it could generate training samples with diverse viewing angles. As far we could research, this was the first work to use such a scheme of computing frustum overlaps, with other works based on random frame sampling from small sequences [29, 70] or extensively sampling over all frames [86] offline.

After preprocessing all RGB-D sequences into graphs, we wrap them into one dataset. During training, the sample batching procedure selects two random frame nodes from an RGB-D sequence's graph and selects around 4000 samples of correspondence point pairs. Theoretically, it is possible to use a massive number of pairs since the images have around 640×480 resolution, but that would demand too much memory. A pixel correspondence sample is found by selecting random coordinates $\mathbf{u}$ in one of the images, then back projecting and transforming in world space. Once in world space, we look for the other position of pixel $\mathbf{u}$ on the second frame by transforming it into its camera space and projecting it into its depth image. In summary, this process has the following steps, let $\mathbf{p}$ be the depth pixel $\mathbf{u}$ of the first image in world space

$$\mathbf{p} = P_1 \Pi(K, \mathbf{u}), \tag{3.7.1}$$

where $P_1 \in SE(3)$ is the first camera's pose transform the back-projected point into world space. The matrix K is the ground-truth intrinsic matrix of the sensor. Then to project into the second image, it is transformed into its camera space:

$$\mathbf{v} = \pi(K P_2^{-1} \mathbf{p}), \tag{3.7.2}$$

where P_2 is the pose of the second camera, it is inverse to back project it on the image. Before emitting the pair, we verify if $\mathbf{v} \in R^2$ was not occluded by testing if its back-projection this time matches the same world position of point $\mathbf{p}$, discarding any pairs with a distance greater than a small threshold.

After obtaining the true point correspondences on each image, then the system proceeds to generate the negative ones by either selecting few random pixels for each positive point $\mathbf{u}$ and $\mathbf{v}$, or by perturbing them with small increments to provide hard-negative candidates.

We experiment with sequences from the Indoor Lidar-RGBD Scan Dataset [61] (IL-RGBD) and SceneNN [43] (SceneNN) RGB-D datasets. Each dataset contains a series of RGB-D sequences with trajectory ground-truth.

3.7.3 Training

For training, we use the contrastive pairwise loss from Choy et al. [18], which we reviewed in Section 2.7.3. But in this time, we rewrite its formula adapting for dense descriptor extraction. Let the set $\mathcal{P} = \{(\mathbf{a}_i, \mathbf{b}_i)\}$ be positive pairs of descriptors $\mathbf{a} \in \mathbb{R}^D$ and $\mathbf{b} \in \mathbb{R}^D$ that the model outputs. Each descriptor $\mathbf{a}_i$ and $\mathbf{b}_i$ are extracted from the channel axes at positive matching coordinates $\mathbf{u}$ and $\mathbf{v}$ from overlapping image pairs that our dataset

generated, that is, we create $\mathcal{P}$ in the following manner

$$\mathcal{P} = \{(\mathcal{G}_1(\mathbf{u}), \mathcal{G}_2(\mathbf{v})) \mid \forall \mathbf{u}, \mathbf{v} \in \mathcal{B}\}, \tag{3.7.3}$$

where $\mathcal{G}_1$ and $\mathcal{G}_2$ represent the slicing of the feature map from the DRN's last layer after the forward with the first and second overlapping images from the dataset. We say slicing because both $\mathcal{G}_1$ and $\mathcal{G}_2$ returns the descriptor at the input coordinates $\mathbf{u}$ and $\mathbf{v}$. Also, in the Equation 3.7.3, the variable $\mathcal{B}$ indicates the set of pixel coordinate pairs from the current batch. In similar fashion, we also build the sets of negative descriptor pairs $\mathcal{N}_\mathbf{a} = \{(\mathbf{a}_i, \mathbf{a}_i^-)\}$ and $\mathcal{N}_\mathbf{b} = \{(\mathbf{b}_i, \mathbf{b}_i^-)\}$ with non-correspondent descriptors from two images using their feature maps $\mathcal{G}_1$ and $\mathcal{G}_2$.

With those inputs, the training routine computes the loss as the following

$$\begin{aligned}\mathcal{L}(\mathcal{P}, \mathcal{N}_\mathbf{a}, \mathcal{N}_\mathbf{b}) = &\frac{1}{|\mathcal{P}|} \sum_{\mathbf{a},\mathbf{b}}^{\mathcal{P}} \max[\|\mathbf{a} - \mathbf{b}\| - m_p, 0]^2 \\ &+ \frac{1}{|\mathcal{N}_\mathbf{a}|} \sum_{\mathbf{a},\mathbf{a}^-}^{\mathcal{N}_\mathbf{a}} \max[m - \left\|\mathbf{a} - \mathbf{a}^-\right\|, 0]^2 \,, \\ &+ \frac{1}{|\mathcal{N}_\mathbf{b}|} \sum_{\mathbf{b},\mathbf{b}^-}^{\mathcal{N}_\mathbf{b}} \max[m - \left\|\mathbf{b}^- - \mathbf{b}\right\|, 0]^2\end{aligned} \tag{3.7.4}$$

with m_p being the positive margin value in the loss function as suggested by Choy et al. [18]. The optimization algorithm used to train using the loss is the Adam optimizer. The batch size is set to one because one image already produces many positive samples, rendering multiple images per batch unnecessary.

Chapter 4

Experiments

We experiment with our proposed algorithms for loop closure detection and dense registration of submaps with DRN models that output pixelwise descriptors with dimension sizes $D = 16$ and $D = 32$.

The first analyzed experiment in this section is the quality of the dense submap registration algorithm separated from the complete SLAM system. By separated, we mean comparing algorithms with submaps that we create with ground-truth odometry. So, the submaps have less noise, and the surfels always contain fusions of the same corresponding deep descriptors. Hence, this testing provides a controlled scenario from odometry errors caused by sensor noise or VO's algorithm's lack of robustness. We compare our ICP with both training, namely, Feature-ICP ($D = 16$) and Feature-ICP ($D = 32$), and with state of the art in point cloud submap registration, ColoredICP [61]. In addition, we add our same ICP algorithm for RGB color, namely RGB-ICP, and using only geometry information, Geom-ICP, for testing the effect of descriptor learning.

Sequentially, we evaluate the effectiveness of the learned features for loop closure and submap registration in our SLAM system — using the VO module developed in Section 3.2 — for correctly estimating trajectories. As previously stated, our goal is to explore the usage of CNN features for SLAM. In that sense, we do not evaluate different SLAM methods, but we use our proposed pipeline changing the loop closure features and submap registration part with well-established methods. For the loop closure detection, we compare the deep features with the FPFH and SIFT descriptors. For the pairwise submap registration in the SLAM, we compare with the methods: Feature-ICP ($D = 16$), Feature-ICP ($D = 32$), RGB-ICP, Geom-ICP, and Colored-ICP [61]. In the end, we display the resulting mappings for qualitative comparisons.

The RGB-D sequences during our experiments are the "Bedroom", "Boardroom", "Loft", and "Lobby" from the IL-RGBD; and "021", "030", "045," "081," and "087" from the SceneNN dataset. Both datasets contain ground truth trajectories generated by state-of-the-art localization and reconstruction methods [17, 21].

4.1 Evaluation Metrics

We compare the difference in translation and rotation components to evaluate the registration transformations between two geometries. The evaluation metrics compares every pair i of a resulting relative rigid transformation $T_i \in \mathbb{R}^{4\times4}$ and a ground truth $G_i \in \mathbb{R}^{4\times4}$. For each ith pair, we compute the root mean squared Relative Translational Error (RTE)

$$\text{RTE} = \sqrt{\frac{\sum_i^N \left\|\text{trans}(G_i^{-1}T_i)\right\|^2}{N}}, \tag{4.1.1}$$

and the root mean squared Relative Rotation Error (RRE)

$$\text{RRE} = \sqrt{\frac{\sum_i^N \cos^{-1}\left(\text{clip}\left(\frac{\text{tr}\left(\text{rot}\left(G_i^{-1}T_i\right)\right)-1}{2}\right)\right)}{N}}, \tag{4.1.2}$$

where trans$(\cdot)$ is the translational part of a 4×4 rigid motion matrix, rot$(\cdot)$ specifies its 3×3 rotation matrix, on RRE, the residual rotation angle is measured using the matrix's trace, tr$(\cdot)$ [76], and the clipping function is

$$\text{clip}(x) = \max\left(-1, \min\left(x, 1\right)\right). \tag{4.1.3}$$

On the comparisons of the complete SLAM, instead of the RTE, we compute the root mean squared Absolute Trajectory Error [76] (ATE) for comparing the entire trajectory of poses P_i with their corresponding ground truth Y_i

$$\text{ATE} = \sqrt{\frac{\sum_i^N \left\|\text{trans}(Y_i^{-1}AP_i)\right\|^2}{N}}, \tag{4.1.4}$$

where A is a matrix that aligns the predicted trajectory P_i with the ground truth Y_i. We compute the matrix A using Kabsch-Umeyama algorithm for alignment from point correspondences. This alignment is precise since each point trans(Y_i) corresponds to trans(P_i).

As a final note, the reason for using root mean squared errors in Equations 4.1.1, 4.1.2 and 4.1.4 is to accentuate the impact of estimation errors [76].

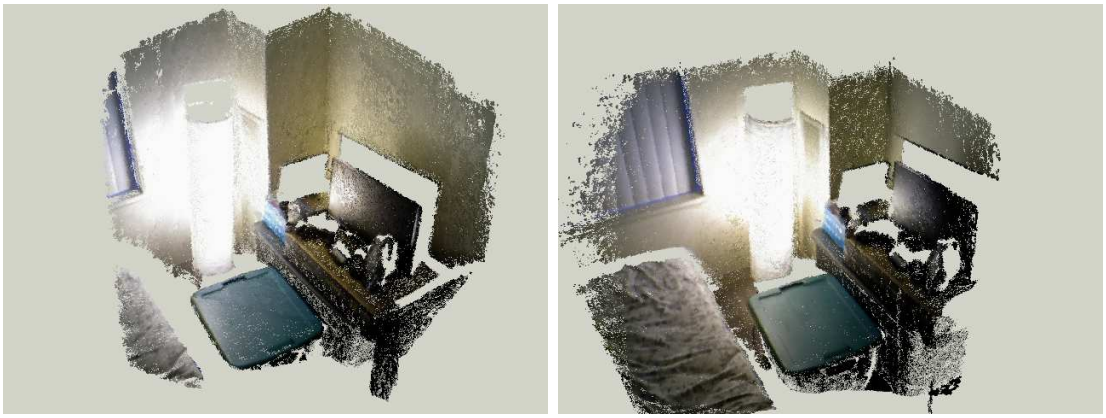
4.2 Training Settings

We train for 50 epochs on all experiments using Adam Optimizer, dropping every ten epochs' learning rate. In most experiments, we set the learning to vary between 5×10^{-4} to 5×10^{-1}. The training scenes are SceneNN's 016, 021, 030, and 032. The complete training and evaluation system is available on our Gitlab.[1]

[1] https://gitlab.com/mipl/3d-reconstruction/slam-feature

Table 4.1: Alignment Metrics for Controlled Submap Registration (RRE values are in degrees).

Algorithm	ColoredICP [88]		Geom-ICP		RGB-ICP		Feature-ICP ($D = 16$)		Feature-ICP ($D = 32$)	
Scene	RTE	RRE	RTE	RRE	RTE	RRE	RTE	RRE	RTE	RRE
Bedroom	0.0991	3.7162	0.1297	4.6003	0.2276	12.2094	0.1390	4.1657	0.1459	4.3618
Loft	0.1847	8.5957	0.2425	10.3742	0.2035	8.8559	0.2434	10.3663	0.2428	10.3381
Boardroom	0.4549	9.9260	0.7042	24.3149	0.4824	15.6287	0.4391	20.1797	0.4412	20.9569
021	0.2200	5.4541	0.0494	1.4103	0.1963	5.7274	0.0496	1.4489	0.0495	1.4174
030	0.3063	7.4158	0.0857	1.4597	0.1629	4.2100	0.0938	2.6667	0.0849	1.4814
045	0.2391	6.1203	0.0553	1.4831	0.2655	6.2827	0.0547	1.4366	0.0546	1.4356
081	0.0346	1.0184	0.0281	0.6059	0.0878	4.5969	0.0268	0.6052	0.0268	0.5736
087	0.0330	0.8818	0.0275	0.7029	0.0548	2.0559	0.0274	0.7060	0.0276	0.7033



(a) (b)

Figure 4.1: Two Submaps with Different Lighting Conditions. Note the more brightness on wall regions and the reflexes on the green chest from (a) to (b).

4.3 Submap Registration under Controlled Scenario

After generating submaps using the ground truth odometry, we select pairs of overlapping submaps with at least 30% of surface overlap to create our test set for the registration algorithm. From the ground truth odometry, we also extract their relative ground truth transformations $G_i \in \mathbb{R}^{4\times4}$. We then disturb the submaps' orientations with random rotations up to 5 degrees and random translations up to 0.1 units during evaluation. This initialization is enough to create challenging alignment but not enough to make most ICP convergence infeasible.

All algorithms use the same coarse-to-fine alignment settings of 3 pyramid levels with downsampling voxel sizes 0.03 and 0.02, and the last level is the original submaps. For the Feature-ICP and RGB-ICP, we run the experiments with the alignment of the deep or RGB features only at the final (or denser) level, running the two first layers only with the geometric term. We do this after noticed during the experiments that the deep features are more sensitive to initial alignment.

Table 4.1 contains our obtained results for our selected scenes. We observe that CNN features obtained better results in the scenes from SceneNN, where the training data may share certain scenes objects (like tables or chairs). For SceneNN's "045" and "081," ColoredICP obtained lower scores because, for some submaps, the algorithm outputted

Table 4.2: Precision and recall of the features for detecting loop closures.

Feature	FPFH [67]		SIFT [53]		DRN($D = 16$)		DRN($D = 32$)	
Scene	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
Loft	0.43	0.98	0.20	0.99	0.31	0.99	0.32	0.98
Lobby	0.79	0.99	0.59	0.99	0.83	0.99	0.82	0.99
Boardroom	0.95	0.15	0.99	0.19	1.00	0.15	1.00	0.15
Bedroom	1.00	0.21	0.99	0.31	1.00	0.16	1.00	0.14
021	1.00	0.34	0.97	0.42	0.99	0.31	0.99	0.31
045	0.44	0.98	0.34	0.98	0.55	0.98	0.55	0.98
081	0.47	0.99	0.30	0.99	0.50	0.99	0.51	0.99
087	0.41	0.98	0.24	0.98	0.43	0.98	0.45	0.98

severe misalignment. We hypothesize that our experimental evaluation using surfels includes more noise than the TSDF submap proposed by its authors [61]. As for the overall better results of the Geom-ICP for most cases compared to the other color or deep feature, this is likely due to the submaps having photometric differences like in Figure 4.1. Results which can be noted from the poor results of RGB-ICP.

Furthermore, we have found for various cases that the residual values when the submaps are aligned with the ground truth transformation are greater than the ones from the GN iterations. Therefore, our learned features do not ideally map their differences minima to the transformation ones. However, they perform better than RGB, indicating some level of generalization for scenes not contained in the training dataset.

4.4 SLAM

This section shows comparisons of the features on our complete SLAM pipeline. We first analyze the effectiveness of the learned features for loop closure detection and its effect on the PGO as indicated in Figure 4.2. Lastly, as in Figure 4.3, we compare the Feature-ICP, RGB-ICP, Geom-ICP, and ColoredICP on the task of registering submaps. We ran all scenes to the maximum of 4500 frames due to time constraints of processing times.

4.4.1 Loop Closure Detection

For evaluating the deep features on loop closure detection, we select the same models with descriptor sizes of $D = 16$ and $D = 32$, namely, DRN($D = 16$) and DRN($D = 32$), and compare them with the geometric FPFH features, and the image-based SIFT features. As being point-cloud based, the FPFH features are directly extracted from the submap surfels. In the SIFT-based loop closure detection case, we replace the deep features with the SIFT ones, and also using the keysurfels selection as explained in Sections 3.3 and 3.4.

We again create our test set with submaps that share 30% surface overlap, producing a ground truth pose graph. From this ground truth graph, we extract the precision and recall of the edges. The precision provides the rate of true edges that our system detects. While the recall provides the percentage of the edges that it detects. Table 4.2 shows their results on each feature. We observe that all methods have similar precision and

Table 4.3: Evaluation metrics of the loop closure found transformations. The RRE values are displayed in degrees.

Feature	FPFH [67]		SIFT [53]		DRN($D = 16$)		DRN($D = 32$)	
Scene	RTE	RRE	RTE	RRE	RTE	RRE	RTE	RRE
Loft	0.334	7.492	0.393	15.480	0.370	16.786	0.322	7.560
Lobby	0.571	3.489	0.574	3.219	0.556	3.487	0.561	3.569
Boardroom	0.619	24.663	0.062	2.484	0.030	1.902	0.040	2.229
Bedroom	0.045	2.331	0.060	3.605	0.061	3.532	0.057	3.382
021	0.068	2.645	0.075	3.442	0.074	2.734	0.083	3.547
045	0.479	15.331	0.488	14.298	0.484	14.680	0.480	14.732
081	0.280	25.505	0.271	25.989	0.263	27.130	0.267	26.105
087	0.525	9.349	0.519	9.478	0.518	9.558	0.521	9.729

Table 4.4: Evaluation metrics of the trajectories after the pose graph optimization. The RRE are displayed in degrees.

Feature	Odometry Ref.		FPFH [67]		SIFT [53]		DRN($D = 16$)		DRN($D = 32$)	
Scene	ATE	RRE	ATE	RRE	ATE	RRE	ATE	RRE	ATE	RRE
Loft	0.548	0.044	4.400	0.380	2.652	0.582	0.998	0.076	0.585	0.065
Lobby	0.261	0.044	0.399	0.085	0.685	0.114	1.352	0.266	0.209	0.054
Boardroom	0.277	0.044	3.962	0.651	2.293	0.262	0.290	0.037	0.330	0.032
Bedroom	0.172	0.057	0.086	0.035	0.073	0.040	0.142	0.052	0.103	0.041
021	0.266	0.086	0.137	0.066	0.229	0.059	0.243	0.062	0.107	0.053
045	0.173	0.033	0.387	0.178	0.076	0.036	0.091	0.037	0.115	0.042
081	0.208	0.032	0.090	0.031	0.186	0.028	0.477	0.092	1.410	0.207
087	0.245	0.057	0.318	0.036	0.209	0.029	0.317	0.070	0.239	0.038

recall, highlighting that the deep features are smaller in dimensional size, SIFT being 128-dimensional and FPFH being 33-dimensional.

Next, Table 4.3 shows the result for estimated loop closures transformation between the submaps. Some values of the rotational error are up to 26°, but this is expected as this alignment must deal with more challenging initial conditions than registering submaps that already have an initial alignment. From that table, it is possible to conclude that FPFH has provided overall more tight alignments than the other features. However, the DRN($D = 32$) has better stability since it demonstrated a rotational error larger than 10° degrees in two scenes in which all methods also had a similar RRE.

Our final evaluation on sparse registration is how the features impacts the result of the PGO algorithm, show in Table 4.4, in the sense that better features offer better loop closures, and hence improve the result of the PGO procedure. In this case, we looking at the result of the pipeline after the PGO stage, as highlighted in Figure 4.2. Instead of comparing the RTE metric, this evaluation uses the ATE metric because we are only interested in the final pose trajectory for now. The table also displays the results considering only the VO trajectory to aid the analysis. For the scenes Loft and Lobby from IL-RGBD, the result of PGO using any feature was worse than just using the VO trajectory. Those results that were worse than the VO are drawn with red colored font on the table. Although, it is possible to notice that the methods have similar results, with the loop closures with DRN($D = 32$) producing better results in the most of the cases.

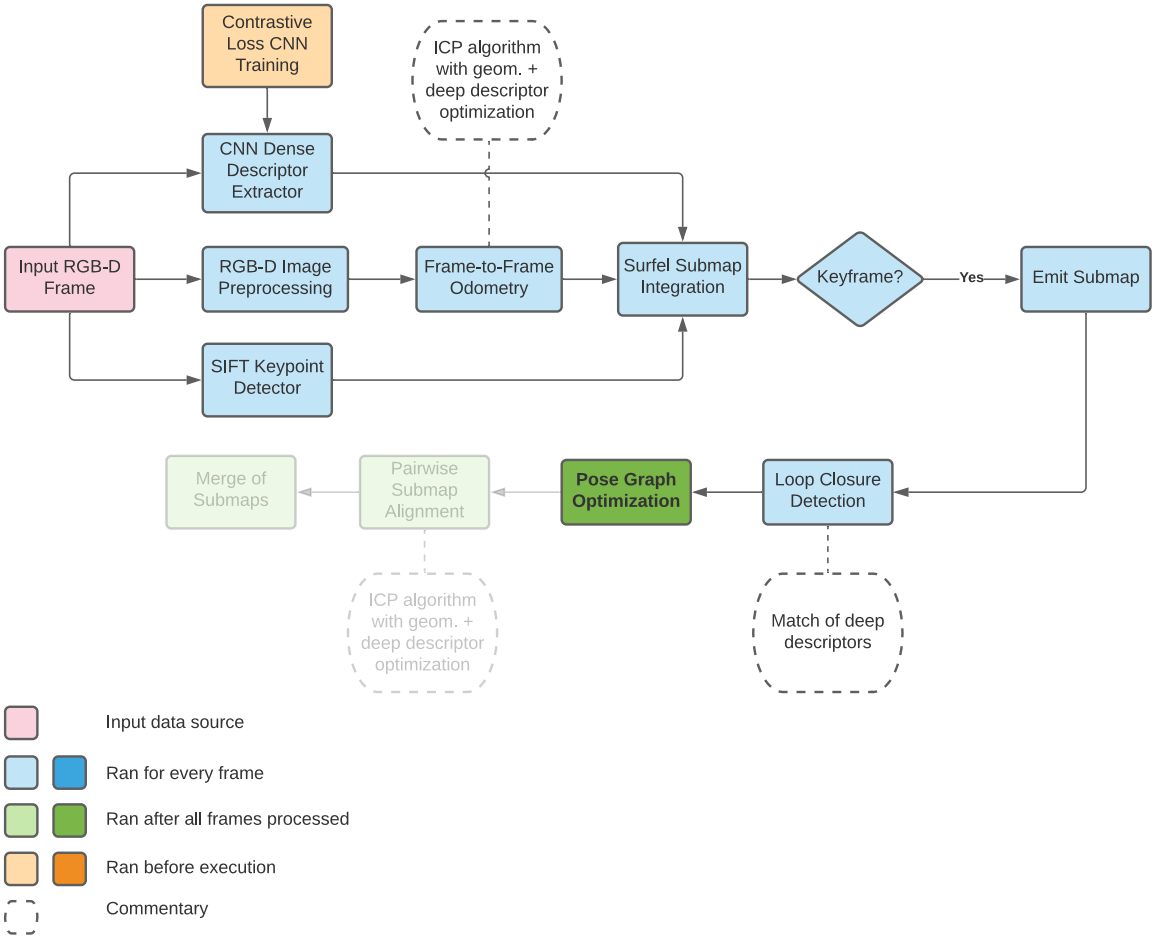


Figure 4.2: We evaluate the final quality of the sparse features for loop closure detection by comparing their effect on the trajectory produced after the PGO stage on the pipeline, highlighted in the dark green on the diagram.

Moreover, one possible improvement is uniting the features to increase robustness since some feature kinds were good on some scenes while others were poor.

4.4.2 Evaluation of Submap Registration on Complete SLAM

Table 4.5 compares the final pairwise submap pose alignment step — which the system calls after PGO. That is, we are evaluating the pipeline after its final alignment stage, as show in Figure 4.3. In Table 4.5, we again mark the result worse than the pure visual odometry in red colored font.

As in the controlled evaluation scenario described in Section 4.3, the results shows that the deep based feature did not provide a considerable gain on average about only using geometric alignment of submaps, reinforcing the same arguments about the minima representativeness of the feature in relation to optimal transformations.

Although, surprisingly, the pairwise registration algorithms could correct the PGO estimation for scenes “Loft” and “Lobby” in most of the algorithms. And for the scenes “081” and “087”, Feature-ICP($D = 32$) obtained better score than the other methods.

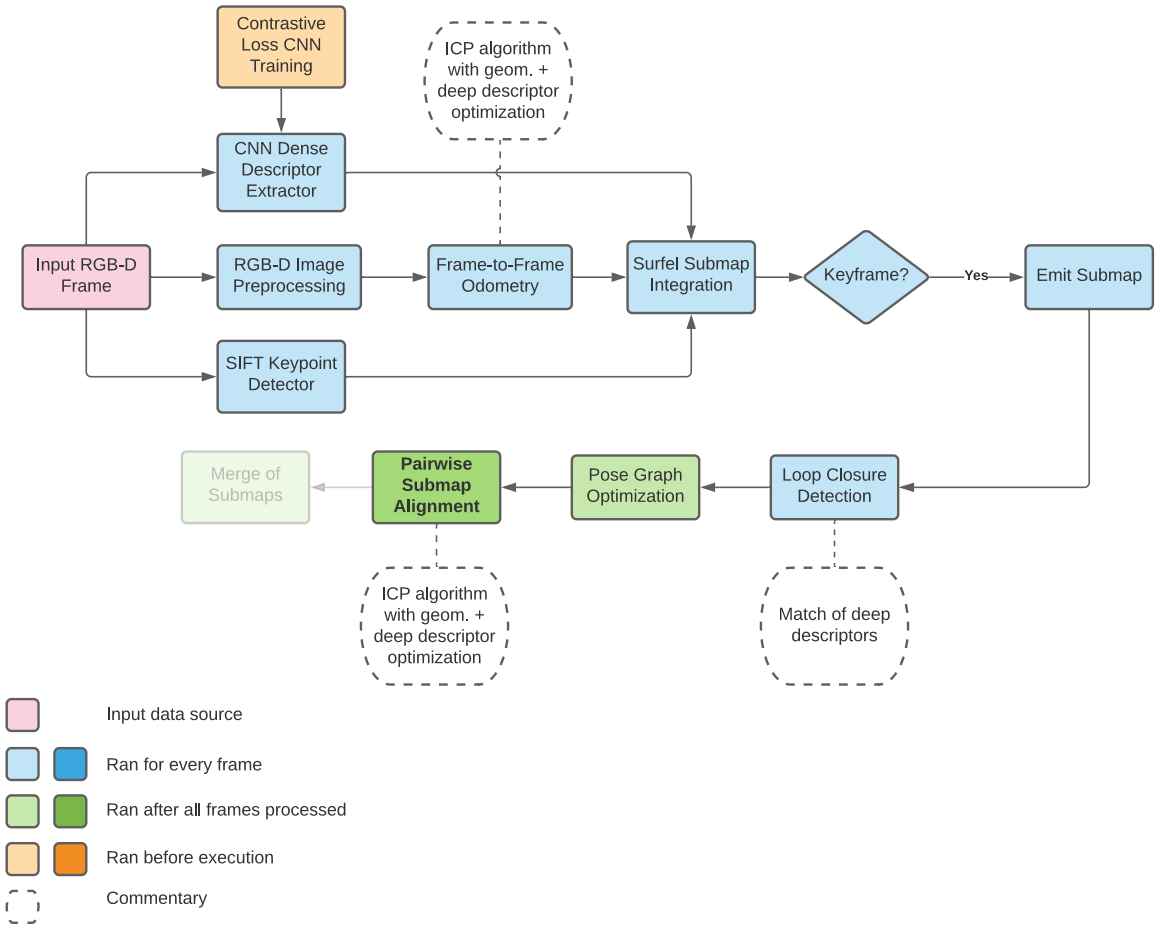


Figure 4.3: As our final evaluation of the pipeline, we compare the results from the pairwise submap alignment stage (dark green), which produces the final trajectory of our SLAM.

4.4.3 Qualitative Results

In the subsequent Figures, our obtained maps for the scenes are shown. Those results do not include the scene “Boardroom” because no method achieved sensible results.

Table 4.5: Evaluation of SLAM after Dense registration

Algorithm	Odometry Ref.		FPFH+ColoredICP[53, 67]		SIFT+RGB-ICP		SIFT+Geom-ICP		Feature-ICP(D=16)		Feature-ICP(D=32)	
Scene	ATE	RRE	ATE	RRE	ATE	RRE	ATE	RRE	ATE	RRE	ATE	RRE
Loft	0.548	0.044	1.811	0.342	0.125	0.0227	2.748	0.584	0.0774	0.024	0.0765	0.023
Lobby	0.261	0.044	0.106	0.039	0.486	0.043	0.457	0.045	0.158	0.031	0.231	0.032
Boardroom	0.277	0.044	0.328	0.145	0.295	0.042	0.207	0.040	0.210	0.038	0.216	0.037
Bedroom	0.172	0.057	0.101	0.045	0.105	0.027	0.105	0.026	0.070	0.025	0.063	0.022
021	0.266	0.086	0.717	0.145	0.168	0.045	0.166	0.044	0.199	0.049	0.197	0.048
045	0.173	0.033	0.611	0.179	0.089	0.037	0.090	0.036	0.096	0.038	0.098	0.038
081	0.208	0.032	0.705	0.148	0.931	0.123	0.257	0.031	0.257	0.030	0.236	0.026
087	0.245	0.057	0.353	0.111	0.273	0.036	0.264	0.036	0.293	0.035	0.269	0.039

(a) FPFH+ColoredICP (b) SIFT+RGB-ICP

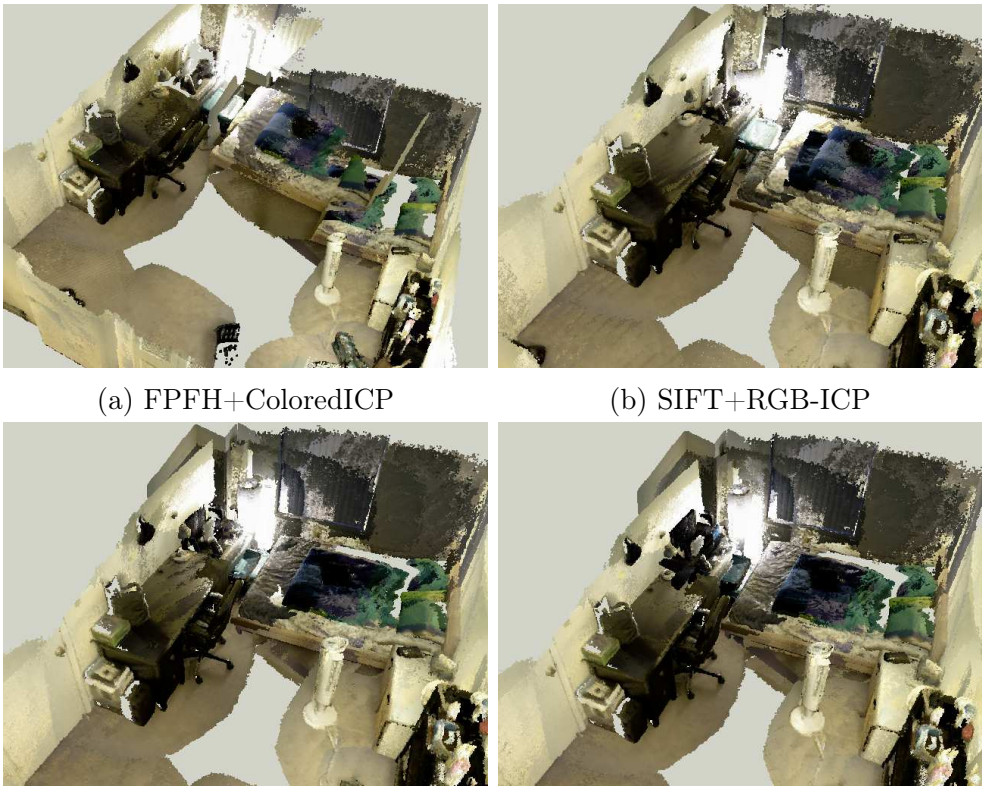
(c) DRN($D = 16$) (d) DRN($D = 32$)

Figure 4.4: Map results of the scene "Lobby" from IL-RGBD. (a), (c), and (d) Note that the left part of the scene is correctly aligned with floor. (b) Part of the submap not aligned.

(a) FPFH+ColoredICP (b) SIFT+RGB-ICP

(c) DRN($D = 16$) (d) DRN($D = 32$)

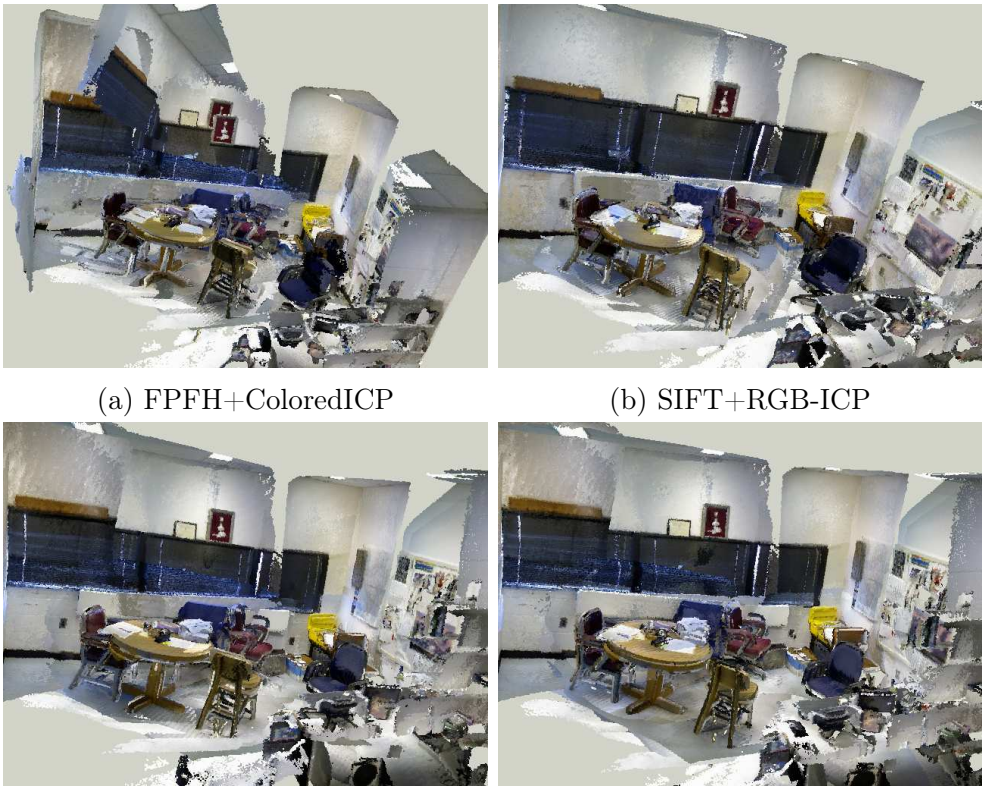
Figure 4.5: Map results of the scene "Bedroom" from SceneNN. (a) Best alignment of the submaps over (b), (c), and (d).



(a) FPFH+ColoredICP (b) SIFT+RGB-ICP

(c) DRN($D = 16$) (d) DRN($D = 32$)

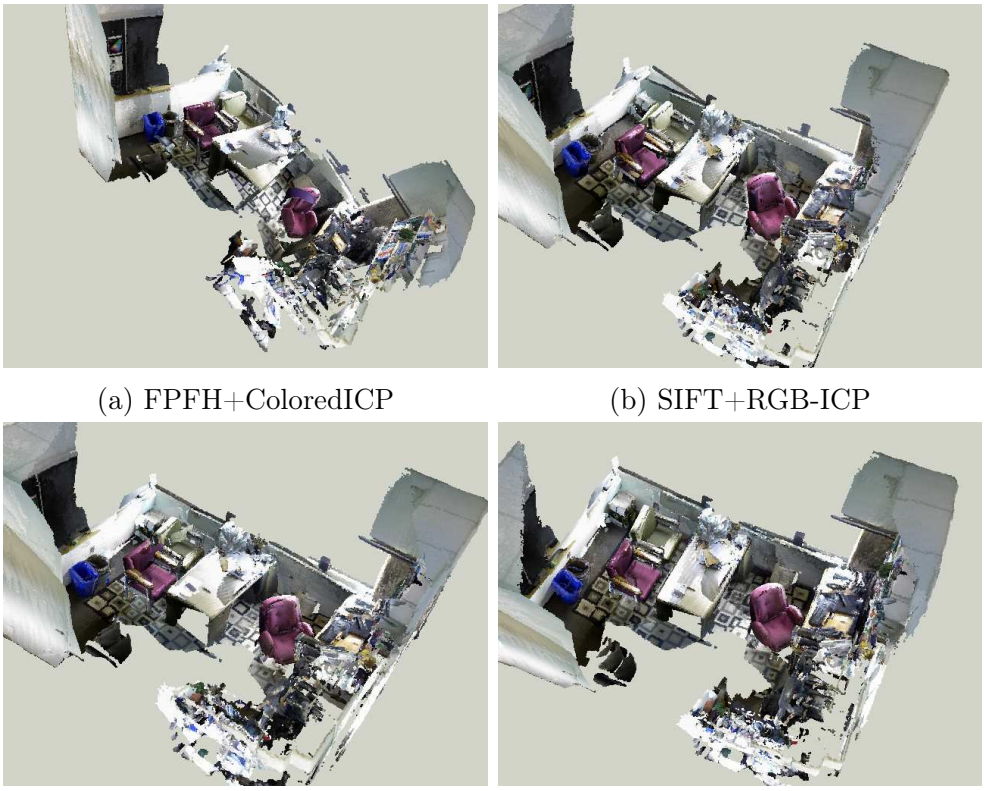
Figure 4.6: Map results of the scene "021" from SceneNN. The deep-feature based methods usefully found aligned the bed submaps



(a) FPFH+ColoredICP (b) SIFT+RGB-ICP

(c) DRN($D = 16$) (d) DRN($D = 32$)

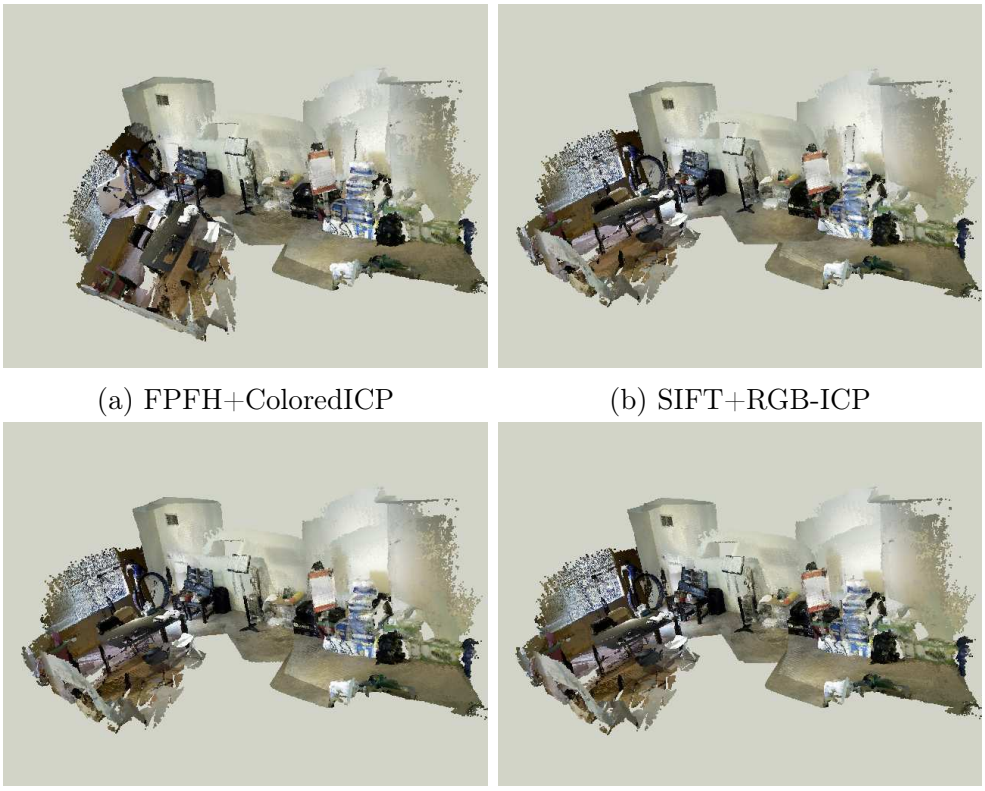
Figure 4.7: Map results of the scene "045" from SceneNN. (a) The method has a poorly aligned submap floating. (b) Result with the tightest alignment. (c) and (d) have some misaligned submaps.



(a) FPFH+ColoredICP (b) SIFT+RGB-ICP

(c) DRN($D = 32$) ICP (d) DRN($D = 32$) ICP

Figure 4.8: Map results of the scene "081" from SceneNN. (a) Misaligned result, then floor bends. (b) Misaligned submaps on the border. (c) (d) Both contains misaligned submaps.



(a) FPFH+ColoredICP (b) SIFT+RGB-ICP

(c) DRN($D = 16$) (d) DRN($D = 32$)

Figure 4.9: Map results of the scene "087" from SceneNN. (a) The floor bends in one. (b), (c) and (d) Better alignments.

Chapter 5

Conclusion

This dissertation presented an RGB-D-based SLAM system that repurposed the features from a CNN architecture designed initially for semantic segmentation for two SLAM tasks: loop closure detection and submap registration.

We first highlight as a contribution our simple and yet efficient frustum overlap frame graph to match pixel samples from large RGB-D sequences for the generation of self-supervised data. That mechanism was vital to create samples with various angles for the successful training of descriptor models. From our research, we did not find any similar algorithm available, rather than choosing random frames.

The next contribution that we review is our mapping framework that integrates dense features into its surface elements, which we use for detecting loop closures and registration of submaps. Future works may use features in the map to develop frame-to-model registration and relocalization modules for aligning the current stream to a previously perceived map, with applications in AR and MR. Another potential gain of the features in the map is to aid semantic reasoning tasks or reuse the later processings on SLAM tasks, as we did in this work.

Our third contribution is the application of the deep features integrated on the submaps for performing loop closure detection providing similar results in PGO compared to SIFT or FPFH.

The final contribution that we highlight is our joint optimization of the point-to-plane cost and feature residual of submaps within the ICP framework. This method differs from existing ICP algorithms [61, 73] by enabling numerical optimization with high dimensional features that occupy a non-grid structure.

However, we cannot conclude from the presented results that learning techniques may surpass handcraft features for SLAM tasks, especially in environments not seen during model training. Still, our loop closure and PGO have shown comparable results with advanced methods, even in unseen scenes from the training dataset. That suggests future research on new architectures, like in recent publications [69, 81] that show promising results for deep learning-based image registration.

From the software engineering perspective, we contribute with new libraries for 3D visualization (TensorViz [1]) and prototyping RGB-D SLAM systems (SLAM-Toolbox [2]).

[1] TensorViz - https://gitlab.com/mipl/3d-reconstruction/tensorviz
[2] SLAM Toolbox - https://gitlab.com/mipl/3d-reconstruction/slam-toolbox

Besides our exploration with SLAM features, we also presented a detailed overview of SLAM methods' main components and techniques, including mathematical background, energy optimization algorithms, main SLAM front-ends and back-ends, and mapping representation. We also presented a review on descriptor learning using self-supervised learning of CNNs models.

Bibliography

[1] Philippe Babin, Philippe Giguere, and François Pomerleau. Analysis of robust functions for registration algorithms. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[2] Martin Barczyk, Silvere Bonnabel, and François Goulette. Observability, covariance and uncertainty of ICP scan matching, 2014.

[3] Ola Bengtsson and Albert-Jan Baerveldt. Robot localization based on scan-matching—estimating the covariance matrix for the IDC algorithm. *Robotics and Autonomous Systems*, 44(1):29–40, 2003.

[4] Paul J Besl and Neil D McKay. Method for registration of 3-D shapes. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 586–606, 1992.

[5] Michael J Black and Anand Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *International Journal on Computer Vision*, 19(1):57–91, 1996.

[6] Michael Bloesch, Hannes Sommer, Tristan Laidlow, Michael Burri, Gabriel Nuetzi, Péter Fankhauser, Dario Bellicoso, Christian Gehring, Stefan Leutenegger, Marco Hutter, et al. A primer on the differential calculus of 3D orientations. *arXiv preprint arXiv:1606.05285*, 2016.

[7] Silvere Bonnabel, Martin Barczyk, and François Goulette. On the covariance of ICP-based scan-matching techniques. In *2016 American Control Conference (ACC)*, pages 5498–5503. IEEE, 2016.

[8] Michael Bosse, Gabriel Agamennoni, Igor Gilitschenski, et al. *Robust estimation and applications in robotics*. Now Publishers, 2016.

[9] Yan-Pei Cao, Leif Kobbelt, and Shi-Min Hu. Real-time high-accuracy three-dimensional reconstruction with consumer RGB-D cameras. *ACM Transactions on Graphics*, 37(5):1–16, 2018.

[10] Andrea Censi. An accurate closed-form estimate of ICP's covariance. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3167–3172, 2007.

[11] Simone Ceriani, Giulio Fontana, Alessandro Giusti, Daniele Marzorati, Matteo Matteucci, Davide Migliore, Davide Rizzi, Domenico G Sorrenti, and Pierluigi Taddei. Rawseeds ground truth collection systems for indoor self-localization and mapping. *Autonomous Robots*, 27(4):353–371, 2009.

[12] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local SDF priors for detailed 3D reconstruction. In *European Conference on Computer Vision (ECCV)*, pages 608–625, 2020.

[13] Jiawen Chen, Sylvain Paris, and Frédo Durand. Real-time Edge-Aware image processing with the bilateral grid. In *ACM Conference on Special Interest Group on Graphics and Interactive Techniques (SIGGRAPH)*, 2007.

[14] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFS. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2017.

[15] Weinan Chen, Lei Zhu, Yisheng Guan, C Ronald Kube, and Hong Zhang. Submap-based pose-graph visual SLAM: A robust visual exploration and localization system. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6851–6856, 2018.

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

[17] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5556–5565, 2015.

[18] Christopher Choy, Jaesik Park, and Vladlen Koltun. Fully Convolutional Geometric Features. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8958–8966, 2019.

[19] Christopher B Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2414–2422, 2016.

[20] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *ACM Conference on Special Interest Group on Graphics and Interactive Techniques (SIGGRAPH)*, pages 303–312, 1996.

[21] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface reintegration. *ACM Transactions on Graphics*, 36(4):1, 2017.

[22] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893, 2005.

[23] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 224–236, 2018.

[24] Ethan Eade. Lie groups for 2d and 3d transformations. *URL http://ethaneade. com/lie. pdf, revised Dec*, 117:118, 2013.

[25] David W Eggert, Adele Lorusso, and Robert B Fisher. Estimating 3-D rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5):272–290, 1997.

[26] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, pages 834–849, 2014.

[27] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, 2017.

[28] Andrew W Fitzgibbon. Robust registration of 2D and 3D point sets. *Image and Vision Computing*, 21(13-14):1145–1153, 2003.

[29] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *Conference on Robot Learning (CoRL)*, pages 373–385, 2018.

[30] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, 2014.

[31] David Gadot and Lior Wolf. PatchBatch: A batch augmented loss for optical flow. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4236–4245, 2016.

[32] Guillermo Gallego and Anthony Yezzi. A compact formula for the derivative of a 3-D rotation in exponential coordinates. *Journal of Mathematical Imaging and Vision*, 51(3):378–384, 2015.

[33] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[34] Ben Glocker, Jamie Shotton, Antonio Criminisi, and Shahram Izadi. Real-time RGB-D camera relocalization via randomized ferns for keyframe encoding. *IEEE Transactions on Visualization and Computer Graphics*, 21(5):571–583, 2014.

[35] P Goel, SI Roumeliotis, and GS Sukhatme. Robot localization using relative and absolute position estimates in proc ieee int. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1999.

[36] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep Learning*, volume 1. MIT press Cambridge, 2016.

[37] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based SLAM. *IEEE Transactions on Intelligent Transportation Systems*, 2(4):31–43, 2010.

[38] Giorgio Grisetti, Tiziano Guadagnino, Irvin Aloise, Mirco Colosi, Bartolomeo Della Corte, and Dominik Schlegel. Least squares optimization: From theory to practice. *Robotics*, 9(3):51, 2020.

[39] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 1735–1742, 2006.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[41] Bing-Jui Ho, Paloma Sodhi, Pedro Teixeira, Ming Hsiao, Tushar Kusnur, and Michael Kaess. Virtual occupancy grid map for submap-based pose graph SLAM and planning in 3D environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2175–2182, 2018.

[42] Roger Howe. Very basic lie theory. *The American Mathematical Monthly*, 90(9): 600–623, 1983.

[43] Binh-Son Hua, Quang-Hieu Pham, Duc Thanh Nguyen, Minh-Khoi Tran, Lap-Fai Yu, and Sai-Kit Yeung. SceneNN: A scene meshes dataset with annotations. In *IEEE International Conference on 3D Vision (3DV)*, pages 92–101. IEEE, 2016.

[44] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[45] Olaf Kähler, Victor A Prisacariu, and David W Murray. Real-time large-scale dense 3D reconstruction with loop closure. In *European Conference on Computer Vision (ECCV)*, pages 500–516, 2016.

[46] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3D reconstruction in dynamic scenes using point-based fusion. In *IEEE International Conference on 3D Vision (3DV)*, pages 1–8, 2013.

[47] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[48] Matthew Klingensmith, Ivan Dryanovski, Siddhartha S Srinivasa, and Jizhong Xiao. Chisel: Real time large scale 3D reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Robotics: Science and Systems (RSS)*, volume 4, page 1, 2015.

[49] Kooktae Lee, Woojin Chung, and Kwanghyun Yoo. Kinematic parameter calibration of a car-like mobile robot to improve odometry accuracy. *Mechatronics*, 20(5):582–595, 2010.

[50] Jie Lin, Olivier Morere, Vijay Chandrasekhar, Antoine Veillard, and Hanlin Goh. Deephash: Getting regularization, depth and fine-tuning right. *arXiv preprint arXiv:1501.04711*, 2015.

[51] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[52] H Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133–135, 1981.

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

[54] Robert Maier, Jürgen Sturm, and Daniel Cremers. Submap-based bundle adjustment for 3D reconstruction from RGB-D data. In *German Conference on Pattern Recognition (GCPR)*, pages 54–65, 2014.

[55] John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. Semanticfusion: Dense 3D semantic mapping with convolutional neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4628–4635, 2017.

[56] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5): 1255–1262, 2017.

[57] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31 (5):1147–1163, 2015.

[58] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[59] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics*, pages 1–11, 2013.

[60] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

[61] Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Colored point cloud registration revisited. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 143–152, 2017.

[62] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[63] Sai Manoj Prakhya, Liu Bingbing, Yan Rui, and Weisi Lin. A closed-form estimate of 3D ICP covariance. In *2015 14th IAPR International Conference on Machine Vision Applications (MVA)*, pages 526–529. IEEE, 2015.

[64] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3D representations at high resolutions. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3577–3586, 2017.

[65] Grégory Rogez, Jonathan Rihan, Carlos Orrite-Urunuela, and Philip HS Torr. Fast human pose detection using randomized hierarchical cascades of rejectors. *International Journal on Computer Vision*, 99(1):25–52, 2012.

[66] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2564–2571, 2011.

[67] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (FPFH) for 3D registration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3212–3217, 2009.

[68] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4938–4947, 2020.

[69] Paul-Edouard Sarlin, Ajaykumar Unagar, Mans Larsson, Hugo Germain, Carl Toft, Viktor Larsson, Marc Pollefeys, Vincent Lepetit, Lars Hammarstrand, Fredrik Kahl, et al. Back to the feature: Learning robust camera localization from pixels to pose. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3247–3257, 2021.

[70] Tanner Schmidt, Richard Newcombe, and Dieter Fox. Self-supervised visual descriptor learning for dense correspondence. *IEEE Robotics and Automation Letters*, 2(2): 420–427, 2016.

[71] Thomas Schops, Torsten Sattler, and Marc Pollefeys. Bad SLAM: Bundle adjusted direct RGB-D SLAM. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 134–144, 2019.

[72] Thomas Schöps, Torsten Sattler, and Marc Pollefeys. Surfelmeshing: Online surfel-based mesh reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[73] James Servos and Steven L Waslander. Multi-channel generalized-ICP: A robust framework for multi-channel scan registration. *Robotics and Autonomous Systems*, 87:247–257, 2017.

[74] Joan Sola, Jeremie Deray, and Dinesh Atchuthan. A micro lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*, 2018.

[75] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Real-time visual odometry from dense RGB-D images. In *IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 719–722, 2011.

[76] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[77] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2088–2096, 2017.

[78] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.2 edition, 2020.

[79] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):815–830, 2009.

[80] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Computer Architecture Letters*, 13(04):376–380, 1991.

[81] Lukas Von Stumberg, Patrick Wenzel, Nan Yang, and Daniel Cremers. Lm-reloc: Levenberg-marquardt based direct visual relocalization. In *IEEE International Conference on 3D Vision (3DV)*, pages 968–977, 2020.

[82] Thibaut Weise, Thomas Wismer, Bastian Leibe, and Luc Van Gool. In-hand scanning with online loop closure. In *IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 1630–1637, 2009.

[83] Thomas Whelan, Stefan Leutenegger, R Salas-Moreno, Ben Glocker, and Andrew Davison. ElasticFusion: Dense SLAM without a pose graph. In *Robotics: Science and Systems (RSS)*, 2015.

[84] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 472–480, 2017.

[85] Jure Žbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *The Journal of Machine Learning Research*, 17 (1):2287–2318, 2016.

[86] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3DMatch: Learning local geometric descriptors from RGB-D reconstructions. In *IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1802–1811, 2017.

[87] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In *European Conference on Computer Vision (ECCV)*, pages 766–782, 2016.

[88] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv preprint arXiv:1801.09847*, 2018.