Fabian Schenk

Edge-based Simultaneous Localization and Mapping

DOCTORAL THESIS

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Friedrich Fraundorfer
Institute for Computer Graphics and Vision

Assoc. Prof. Dr. Torsten Sattler
Chalmers University of Technology

Graz, Austria, March 2020

Abstract

Simultaneous Localization and Mapping (SLAM) and Visual Odometry (VO) are two of the longest-standing problems in robotics and computer vision and still heavily researched today. Both are of great importance for autonomous transportation, UAV navigation, augmented and virtual reality and 3D reconstruction since all these applications ask two basic questions: *"Where is the robot?"* and *"How does its environment look like?"*. SLAM tries to answer both of these questions by simultaneously estimating the robot's position while mapping the environment around it. VO can be considered a building block for SLAM and focuses on estimating the robot's motion within a local area, while SLAM operates on a global map.

The release of light-weight and cheap RGBD sensors, which can record a scene's geometry and texture in real-time, has opened up many new possibilities for SLAM and VO. While the classical indirect approaches that extract features, establish correspondences and optimize a reprojection error still exist, new direct methods that maximize the photo-consistency between two images have emerged in recent years. They do not rely on features but work directly on intensity values, which makes them susceptible to illumination changes and dependent on an initialization close to the minimum. Other direct methods that align the dense point clouds from the sensor directly are typically constrained to small areas and require a distinct geometric structure.

In this thesis, we propose a novel direct method for RGBD sensors that aligns edges to estimate the camera motion. Edges have many favorable qualities since they are more robust than intensity values, are fast to compute and greatly reduce the image information due to their sparse representation. We first show how to formulate the edge-based camera motion estimation problem and how to solve it efficiently. To improve the robustness in indoor scenes with many texture-less surfaces, e.g. white walls, floors or ceilings, we then propose to jointly align edges and the dense point clouds. Finally, we present the first edge-based RGBD SLAM system, which runs a local mapper to refine camera poses, the depth of the edges and the camera intrinsics and a global mapper to perform loop closure and relocalization.

Since edge detection is a crucial part of our edge-based methods, we present an extensive comparison between traditional and machine-learned edge detectors, where the latter show a stable performance even under severe motion blur and in scenes with low illumination. Then we demonstrate the capabilities of our VO and SLAM methods on various benchmark datasets covering a wide variety of camera motions and scenes. Our edge-based VO methods outperform state-of-the-art approaches on most of the datasets and also our SLAM system performs in the range of other popular systems, many of which are computationally far more expensive.

Keywords: computer vision, robotics, simultaneous localization and mapping, SLAM, visual SLAM, visual odometry, VO, edge-based, direct method

Kurzfassung

Simultanes Lokalisieren und Kartieren (SLAM) und Visuelle Odometrie (VO) sind zwei der ältesten Probleme der Robotik und Bildverarbeitung und werden auch heute noch intensiv erforscht. Beide sind von enormem Interesse für das autonome Transportwesen, die UAV Navigation, sowie augmentierte bzw. virtuelle Realität und 3D Rekonstruktion, da alle diese Anwendungen zwei grundlegende Fragen aufwerfen: *"Wo ist der Roboter?"* und *"Wie sieht dessen Umgebung aus?"*. SLAM versucht beide dieser Fragen zu beantworten, indem es die Position eines Roboters bestimmt und gleichzeitig dessen Umgebung kartiert. VO kann als ein Baustein für SLAM angesehen werden und beschäftigt sich mit der Schätzung der Roboterposition in einer lokalen Umgebung, während SLAM auf einer globalen Karte arbeitet.

Das Erscheinen von leichten und kostengünstigen RGBD Sensors, welche die Geometrie und Textur einer Szene in Echtzeit aufnehmen können, hat zahlreiche neue Möglichkeiten für SLAM und VO eröffnet. Während klassische indirekte Ansätze, die interessante Punkte finden, Korrespondenzen herstellen und anschließend den Rückprojektionsfehler optimieren, immer noch verwendet werden, wurden in den letzten Jahren direkte Methoden basierend auf der Photokonsistenz entwickelt. Diese Methoden sind nicht auf interessante Punkte angewiesen, sondern arbeiten direkt auf den Intensitätswerten des Bildes, was sie aber anfällig auf Beleuchtungsunterschiede macht und eine Initialisierung nahe dem Minimum erfordert. Eine weitere Art von direkten Methoden ist, die dichten Punktwolken vom RGBD Sensor direkt aufeinander zu registrieren, was aber typischerweise nur für kleine Bereiche funktioniert und ausgeprägte geometrische Strukturen erfordert.

In dieser Arbeit, präsentieren wir eine neuartige direkte Methode für RGBD Sensoren, die Kanten aufeinander registriert um die Kamerabewegung zu bestimmen. Kanten haben die Vorteile, dass sie deutlich robuster als Intensitätswerte sind, schnell berechnet werden können und zusätzlich die Bildinformation reduzieren. Zuerst zeigen wir, wie man die Kamerapositionsbestimmung für unsere kanten-basierte Methode formulieren und ef-

fizient lösen kann. Um die Robustheit in Innenräumen mit zahlreichen beinahe texturlosen Flächen, z.B. weiße Wände, Böden oder Decken, zu erhöhen, präsentieren wir danach eine gemeinsame Registrierung von Kanten und dichten Punktwolken. Zum Abschluss, stellen wir das erste kanten-basierte RGBD SLAM System vor, welches eine lokale Optimierung enthält, um Kamerapositionen, die Tiefe der einzelnen Kanten und die Kameraparameter zu verfeinern, aber zusätzlich auch eine globale Karte nutzt um Schleifenschließungen und Relokalisierung zu ermöglichen.

Da Kantendetektion eine wesentliche Rolle für unsere kanten-basierten Methoden spielt, präsentieren wir einen umfangreichen Vergleich zwischen traditionellen und gelernten Kantendetektoren, wobei letztere deutliche bessere Ergebnisse bei Bewegungsunschärfe und geringer Beleuchtung aufweisen. Anschließend zeigen wir die Fähigkeiten unserer VO und SLAM Methoden anhand verschiedenster Benchmarkdatensätze, welche eine Vielzahl von Bewegungen und Szenen umfassen. Unsere kanten-basierte VO Methode übertrifft andere moderne Methoden auf beinahe allen Datensätzen und auch unser SLAM System liegt im Bereich anderer populärer Systeme, die oftmals deutlich mehr Rechenleistung erfordern.

Schlüsselwörter: Bildverarbeitung, Robotik, simultanes Lokalisieren und Kartieren, SLAM, visuelle Odometrie, VO, kanten-basiert, direkte Methode

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

Place, Date

Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort, Datum

Unterschrift

Acknowledgments

Pursuing my PhD at the ICG was the most interesting but also the most challenging time of my life. I am so grateful for the exciting years at the ICG, the many incredible people I have met and the numerous conferences and seminars I attended.

I would like to pay my special regards to my supervisor Friedrich Fraundorfer for the trust he put in me, the many fruitful research meetings and paper discussions, his guidance and support throughout the past years. I will always remember our exciting trips to our partners in France, where we tried our first snails. Further, I would also like to thank Torsten Sattler for being my second supervisor.

This PhD thesis would not have been possible without my colleagues at the ICG, who helped me through difficult times with their input, new insights or sometimes just with an after-work beer. I want to specifically thank the active and former members of the 3D Vision Group and Robot Vision Group: Michael Maurer, Jesùs Pestana Puerta, Rafael Weilharter, Christian Sormann, Christian Mostegel, Thomas Holzmann, Markus Bergen, Ludwig Mohr, Gernot Riegler, David Ferstl and Christian Reinbacher. I also would like to thank Alexander Tscharf, Alexander Bornik and Johannes Höller for the joint publications and their great collaboration on various challenging projects. A special thanks to Martin Urschler, who encouraged me to pursue a PhD after my Master's thesis, and to Horst Bischof and Matthias Rüther for giving me the opportunity to start at the ICG.

Finally, I wish to express my deepest gratitude to my friends, and family members for their support, love, and encouragement. I am deeply indebted to my mother, who is always there for me and without her, I would not be where I am today. In particular, I would like to thank my girlfriend Marie for being at my side for the past years and especially for her patience and love even during the many work-intensive days and nights.

Contents

List of Figures

List of Tables

1 Introduction

Contents

The technological advances of the last decades have paved the way for autonomous robots that can perform many elaborate tasks without the need for human supervision or interference. In households, robots assist humans in their everyday life by vacuuming or wiping floors, cleaning swimming pools or mowing the lawn. They can also perform exploratory tasks in areas that are difficult to reach such as air, underground, underwater or even hazardous environments, e.g. outer space, nuclear power plants or mines. Another huge area of application is the inspection of critical infrastructure such as bridges, roads, and power lines. Finally, autonomous robots in the form of self-driving cars, buses, trucks, and trains will completely change the transportation of goods and people. The many practical applications are discussed in greater detail in Section 1.2.

Even after decades of research, robots only work autonomously in constrained environments, e.g. in the garden, and true autonomy in complex environments such as urban areas is still a topic of intensive research. One of the key requirements for true autonomy is the robot's ability to perceive and interact with an unknown environment, which introduces two major questions: (1) *"Where is the robot?"*, i.e. what is the current position of the robot in the map, and (2) *"How does the environment look like?"*, i.e. generate a map of the environment. When looking at these questions individually, (1) can be solved by localizing the robot against a known map, while (2) boils down to generating a map from the measurements at known robot positions (see Fig. 1.1). However, in nearly all practical applications neither the map nor the positions are known beforehand, which leads to a joint problem of generating a map of an unknown environment, while simultaneously lo-

calizing within it. What makes this so difficult is that it is in its essence a chicken-and-egg problem since localization, i.e. the estimation of the robot's position, requires a map, while the creation of the map requires the robot's position (see Fig.1.1). In the literature, this problem is known as Simultaneous Localization and Mapping or Structure and Mapping (SaM) and is considered one of the major challenges in robotics.

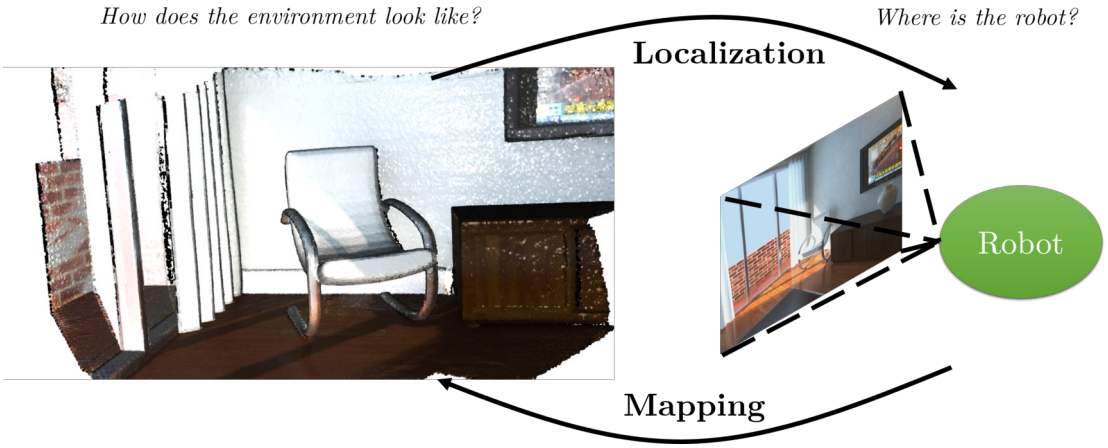


Figure 1.1: Simultaneous Localization and Mapping (SLAM) is a classical chicken-and-egg problem posing two questions (1) *"Where is the robot?"* and (2) *"How does the environment look like?"*. To solve this problem, it is necessary to have an accurate map to localize against, i.e. estimate the robot's current pose, but at the same time the robot's pose is required to generate an accurate map.

The goal of *SLAM* [15, 31] is to estimate a global and consistent path of the robot, which is often referred to as *trajectory* (see Fig. 1.2). When the robot explores the environment, *SLAM* obtains the *trajectory* incrementally since it estimates the current position with respect to one or several previous positions. With each new estimate, an error, albeit small, is introduced into the *trajectory*, which accumulate and cause *drift*, a difference between the real and the estimated trajectory. During continuous exploration, it is only possible to reduce *drift* through, e.g. optimizing over the last N positions in a *sliding window* or *windowed* Bundle Adjustment (BA). In order to correct the *drift*, *SLAM* has to keep track of a global map to detect when places are revisited, even when the map itself is not required for the robot's task. The global map is typically represented as a sparse point cloud (see Fig. 1.2), as a pose graph (see Fig. 4.25) with or without landmarks, or as dense model. Returning to a previously visited place corresponds to a *loop* in the *trajectory*, which makes it possible to measure the global *drift* as the difference between the current position and the position at the previous visit (see Fig. 1.3 and 4.24). With the information gained from the detected *loop*, the global *drift* in the *trajectory* and the map can be greatly reduced via *loop closure*. Detecting already visited scene parts and how to efficiently and accurately integrate the corresponding *loop* constraint are two key challenges in *SLAM*.

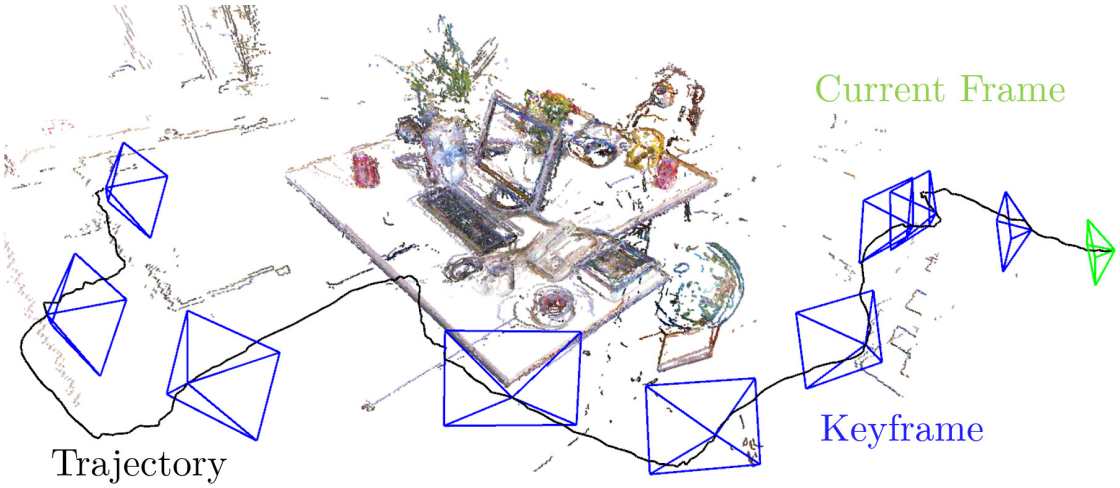


Figure 1.2: Map of the environment as sparse point cloud.

Even though this thesis focuses on *SLAM*, it is important to discuss the distinction to Visual Odometry (VO) [120, 131], which essentially addresses the same problem. Scaramuzza and Fraundorfer [51, 131] describe the difference in the sense that *VO* cares about *local* consistency, while *SLAM* focuses on *global* consistency. *Local* consistency in this context means that *VO* tries to create a locally accurate *trajectory* and map but does not have any way of detecting already visited places. In contrast, *SLAM* aims to reach a *globally* consistent *trajectory* and map, i.e. the same places should be at the same position after revisiting. Figure 1.3 visualizes the connection between *SLAM* and *VO*, where *VO* incrementally estimates the poses and is an inherent part of *SLAM*. In the literature [131, 149], this connection is typically defined as:

$$SLAM = \text{Visual Odometry} + \text{Global Mapper}, \tag{1.1}$$

where the **Visual Odometry** estimates the relative camera motions between frames and optimizes a local map and the **Global Mapper** maintains and refines a global map, while also handling loop closure and relocalization.

Since *VO* and *SLAM* each have their advantages and disadvantages, the choice is often not trivial and additionally depends on the application, the available computational resources. Due to the additional constraints introduced by the *loop closure*, *SLAM* is potentially more accurate but not necessarily more robust. Even one wrong *loop closure* due to outliers, e.g. mismatch in a highly redundant scene, can greatly reduce the overall accuracy of the *trajectory* and map or even lead the whole system to fail. Another issue is that the *drift* caused by incremental pose estimation varies depending on the visible parts of the scene, i.e. pose estimation is often much more accurate in highly textured areas than in scarcely textured or cluttered parts. Since *loop closure* distributes the *drift* over the

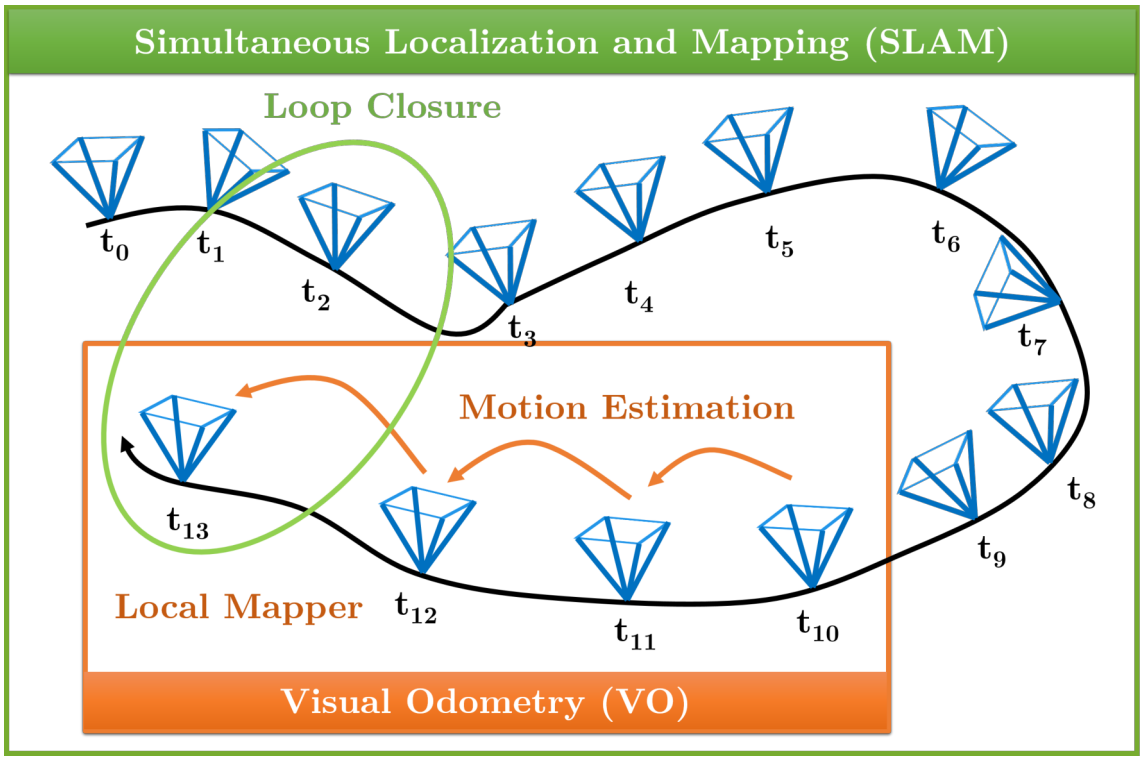


Figure 1.3: A typical *VO* system comprises motion estimation between frames and a local mapper to additionally optimize poses within a window. *VO* is an integral part of *SLAM* but *SLAM* additionally has a global mapper to handle loop closures and relocalization.

whole *trajectory*, this can decrease the quality in well-estimated areas. The computational resources are another point to consider, specifically for mobile robots. *VO* only considers a local area, which keeps the overall problem constrained and much easier to solve than optimizing over potentially thousands of frames as in *SLAM*. Thus, many systems follow the policy introduced by Klein and Murray [84] and run a *VO* system at frame-rate and a global optimization or mapping in the background. While the *VO* system keeps an accurate *trajectory* over a small area, the mapping system optimizes for *global* consistency in the background. Due to the huge amount of image data, the global mapper typically only processes specific frames, the Keyframes (KFs), and all the non-*KFs* are discarded.

Over the past decades, *SLAM* has been addressed by a wide variety of algorithms and systems equipped with different sensors. In the beginning, the systems were rather basic with just a wheeled robot moving in a planar scene and reconstructing the environment from laser range or sonar measurements. With the technological advances, vision-based sensors or cameras have become smaller, faster and more energy-efficient such that have found their way onto mobile robots and even smartphones. Cameras have opened a completely new research area for *SLAM* since they provide access to much more information than laser range finders or sonar. When solving the *SLAM* problem with vision-based sensors, the procedure is sometimes also called *visual SLAM* or *vSLAM*.

1.1 Challenges and Limitations in SLAM

Since *SLAM* has been researched for decades, many people in the robotics and vision community ask the question: *"Is SLAM solved?"* [15]. During decades of research also *SLAM* has evolved and contains so many different topics that no universal answer can be given. Some parts such as *SLAM* in small, static and constrained indoor environments are well-understood and might be considered *"solved"* but with the ever-evolving technologies, e.g. Unmanned Aerial Vehicle (UAV) or RGBD sensors, new challenges arise and state-of-the-art systems still suffer from many limitations. In this section, we will jointly discuss the limitations and remaining challenges for *SLAM* in indoor environments and focus on the robustness to outliers and poor visual input, the computational efficiency, and ease of use. For a broader discussion, we refer to the extensive survey by Cadena et al. [15] also covering limitations for outdoor, long-term and semantic *SLAM*.

Current *SLAM* systems typically rely on an iterative estimation with a robust cost function and can detect tracking losses in cases of aggressive motions or partially covered sensor. However, they are still susceptible to outliers for two reasons: (i) outlier detection and the down-weighting depends on the initial guess and (ii) even one single outlier influences future estimations by introducing a wrong linearization point, thus slowly corrupting the system. This type of slowly progressing error is extremely hard to detect until it is too late to recover from it. *Fail-safe* or at least *failure-aware SLAM* remains an open problem and is especially critical for practical applications.

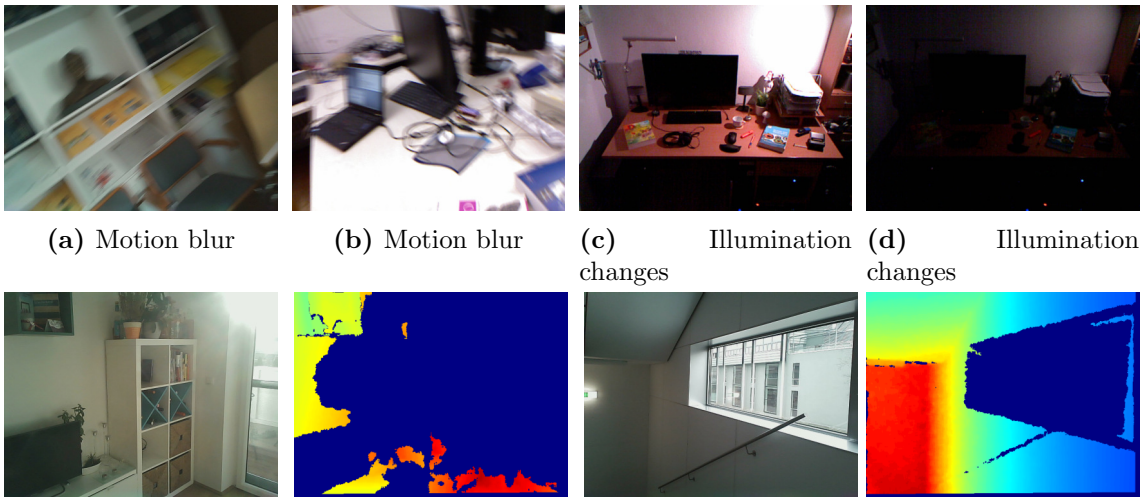
There is another set of problems arising from the sensor mounted on the robot. Monocular *SLAM* inherently suffers from various limitations since depth is not observable in one single image. Typical problems are that (i) the map and trajectory are only known up to scale, which can result in scale drift in longer trajectories, (ii) the initialization of the depth values is difficult and error-prone because it highly depends on the first few camera motions and finally (iii) camera motion estimation fails under (nearly) pure rotations since points cannot be triangulated. With the release of RGBD sensors such as the Microsoft Kinect [61, 163], the Orbbec Astra and the Intel RealSense [82], which can record a scene's texture as RGB image and its geometry as depth map (see Fig. 4.2a), all of these issues can be solved. However, there are still many challenges for *visual SLAM*:

- Motion blur is one of the most common problems in practice since it occurs even at high frame rates of around 60 fps and causes a significant loss of image information.
- Scarcely textured areas, e.g. white walls, ceiling or floors, do not contain any valuable information and make motion estimation very unreliable.
- Illumination conditions can vary considerably even within the same scene (see Fig. 1.4c and 1.4d), which leads to the problem that even successive images sometimes do not match.
- Large motions caused by fast camera movements or a low-frame rate introduce prob-

lems when estimating the relative motion since the optimization starts far away from the minimum.

RGBD sensors deliver depth maps, which might have invalid measurements due to reflective surfaces or glass (see Fig. 1.4g and 1.4h). Figures 1.4e and 1.4f depicts another common issue, where sunlight renders the infrared pattern of RGBD sensors unusable.

Due to the hardware developments in the past years, sensors today have a high resolution and frame rate, which means that a huge amount of data has to be processed in real-time. Many *SLAM* systems rely on a strong Graphics Processing Unit (GPU) to process the data, which is typically not possible in mobile robots with limited resources, e.g. a *UAV* flying with one battery and an onboard CPU. For such robots, computationally inexpensive methods running on a CPU are of great importance and heavily researched at the moment.



(a) Motion blur **(b)** Motion blur **(c)** Illumination changes **(d)** Illumination changes

(e) RGB with sunlight **(f)** Depth with sunlight **(g)** RGB Image **(h)** Depth with on glass

Figure 1.4: General challenges in visual *SLAM* are motion blur (a) and (b) and illumination changes within the same scene (c) and (d). RGBD sensors often have problems in sunlight (e) and (f) or on reflective or glass surfaces (g) and (h).

Probably the most crucial limitation of *SLAM* systems for practical applications is that they do not work in a simple plug-and-play manner but require expert knowledge. First, the intrinsic calibration of the sensor and the calibration with respect to a robot are non-trivial problems. Second, nearly all systems have a set of thresholds, which are scene- or sensor-specific and greatly influence the overall accuracy. Thus, current methods require in-depth knowledge of the pose estimation algorithm and the implementation. *SLAM* system that automatically choose the correct parameters for each sensor and arbitrary scenes are still an open topic.

To summarize, there are still many open topics in *SLAM* and similarly in *VO* and both problems are far from *"solved"*. Therefore, both remain among the most active research

areas in computer vision and robotics. With the introduction of new sensor such as event cameras, affordable Lidar sensors and the abundance of cameras in mobile devices, this is not likely to change anytime soon. In this thesis, we address some of the discussed limitations with a novel edge-based approach, show how to increase robustness in difficult scenes and present a very fast real-time capable system.

1.2 Applications for SLAM and VO

SLAM and *VO* have an abundance of practical applications and a complete overview would exceed the context of this thesis. Thus, we will focus on indoor applications, where RGBD sensors can be used, such as augmented reality, *UAVs*, 3D reconstruction but also discuss the booming field of autonomous transportation or driving.

Augmented Reality (AR) With the ever-present mobile phone and wearable devices such as the Microsoft HoloLens or Daqri SmartGlass, augmented reality is now a rapidly progressing field. At the core of each Augmented Reality (AR) application is typically either a *SLAM* or *VO* system, since the device has to localize itself within an unknown environment. While most of the wearables have an integrated depth sensor, smartphones and tablets with depth-sensing capabilities start to emerge.

Unmanned Aerial Vehicles (UAVs) In the last years, *UAVs* have gotten smaller and are frequently used in indoor environments for tasks like autonomous stocktaking in warehouses [110]. Since there is no GPS signal available in enclosed environments, the *UAVs* have to rely on vision-based systems. On such mobile platforms, RGBD sensors are a popular choice since no onboard depth map computation is necessary and they allow for easy obstacle avoidance. Real-time *SLAM* and *VO* algorithms are key ingredients for robust and accurate indoor navigation.

3D Mapping Mapping the 3D world we live in is a long-standing challenge in computer vision. An accurate and dense map of the environment is valuable for documentation, e.g. for the construction industry, for crime scenes. Nowadays, *building information modeling*, where every building has a digital twin, which is maintained throughout the whole life-cycle of the building, is one of the most promising applications in the construction industry. Another application is generating 3D models of objects to incorporate them into augmented and virtual reality visualizations. Dense models take very long to compute with image-based methods and often suffer from shortcomings in uniformly textured areas. However, with modern RGBD sensors, it is possible to generate a globally consistent 3D map in real-time.

Autonomous Transportation This is probably the area, which is going to influence our everyday life the most. Autonomous cars and buses will transport people in and

between cities and promise to reduce car accidents, traffic jams and the necessity for individual cars. To transport goods, fleets of autonomous cars and trucks have been launched in recent years and drones are starting to deliver packages to remote or difficult to reach areas. *SLAM* and *VO* are the core components to drive all those autonomous vehicles but for true autonomy, new challenges such as moving objects in a scene, changing weather and illumination conditions have to be solved. Thus, *SLAM* and *VO* will continue to be an important research field for the coming years or even decades.

1.3 Contributions

This thesis summarizes three publications [132–134] in the fields of *VO* and *SLAM*, where I am the first author. These publications can be interpreted as a step-by-step development of a novel edge-based RGBD *SLAM* system, which was the first of its kind upon its initial publication. The complete *SLAM* system is also the main outcome of this thesis.

In [133], we developed REVO, an edge-based *VO* algorithm, which significantly outperformed state-of-the-art edge-based methods. We proposed an edge-based quality measure, which is used for both, *KF* selection and tracking loss detection. Further, we evaluated modern machine-learned edge detectors and demonstrated that even though not trained for the *VO* task, they can improve robustness and accuracy. Finally, we showed the favorable qualities of edge-based methods under rapid motion, where photometric and point-cloud based methods often fail. REVO is available as open-source, runs at around 50-60 fps on a CPU and has been well received by the community.

The second publication [132], focused on the robustness of *VO* in challenging indoor scenes. A common problem especially indoors are large untextured scene parts such as white walls, ceilings or floors. Since we work with an RGBD sensor, we aim to utilize all the available data and proposed to combine our REVO [133] with a geometric cost term in an Iterative Closest Point (ICP) point-to-plane formulation. We then jointly optimize both errors, which gives increased stability in texture-less areas, since depth is still observable there. Additionally, we include a way to refine the current *KF* pose with respect to N spatially close *KFs* to close small loops and increase accuracy. We also implemented a method to enhance edge detections with detections from higher-resolution levels.

These developments culminated in RESLAM [134], the first edge-based RGBD SLAM system. REVO [133] with the enhancements presented in [132] served as basis for the edge-based relative motion estimation. Instead of optimizing only with respect to N spatially close frames, we perform a full local *BA* in a sliding window over all involved model parameters. These include the initial depth of the edges, the camera poses, and the camera intrinsics. To increase the speed and facilitate the use of more sophisticated edge detectors, we propose to only detect edges on the highest resolution and to down-scale the Distance Transform (DT) instead of detecting edges on all scale levels. This method shows similar results to the edge enhancement [132] but is significantly faster. Another integral part is the detection of already visited places, which allows for loop closure and

Publication	Motion Estimation	Local Mapper	Global Mapper	Year	Conference	Code
REVO [133]	✓	✗	✗	2017	IROS	✓
REVO+ICP[132]	✓	✓	✗	2017	BMVC	part of [133]
RESLAM [134]	✓	✓	✓	2019	ICRA	✓

Table 1.1: The contributions from each publication to the individual components of a *SLAM* system (see Fig. 1.3), the year of publication, the conference and if the code is open-source.

relocalization after a system pause or tracking loss. Like REVO, also RESLAM is available as open-source software.

Throughout the last years, we significantly contributed to the field of *VO* and *SLAM* specifically in the recent area of direct edge-based methods. To encourage further research, we released two of our systems as open-source to the community. Table 1.1 shows an overview of all the publications and their respective contributions to the various components of *SLAM*, the year of publication, the conference and if the code is available as open-source.

1.4 Thesis Outline

This thesis is structured into six chapters as follows. Chapter 2 introduces different types of *SLAM* and shows an overview of the most important publications in the fields of *SLAM* and *VO*. Chapter 3 lays out the mathematical notations and theoretical foundations to understand direct methods and the algorithms presented in this thesis. Our contributions are described in great detail in Chapter 4, which is divided according to the typical *SLAM* components *motion estimation*, *local mapper* and *global mapper*. We start with the basic edge-based *VO*, discuss how to incorporate an additional geometric term and explain the steps to make it a full *SLAM* system. Edge detection is a central point in this work and discussed in Chapter 5 alongside quantitative and qualitative evaluations. Chapter 6 contains qualitative and quantitative experiments on standard benchmark datasets as well as qualitative results on various RGBD recordings. We conclude this thesis in Chapter 7 and discuss future research directions.

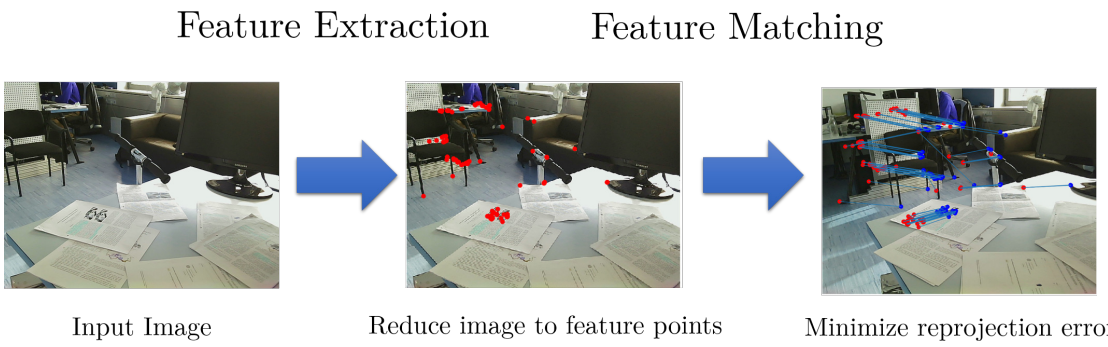
2 Related Work

Contents

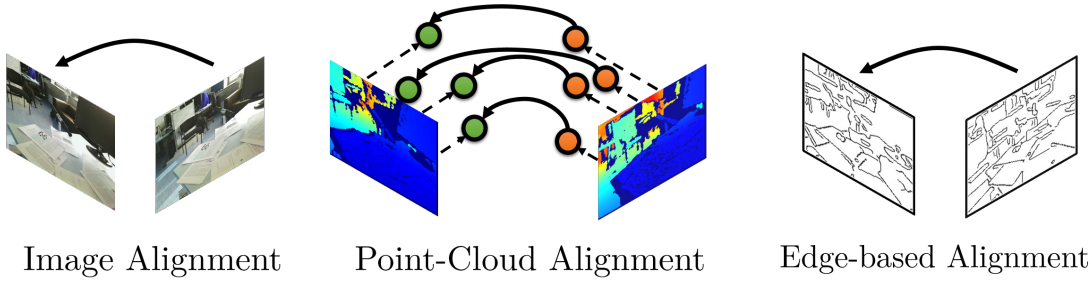
This thesis presents our contributions to the fields of Visual Odometry (VO) and Simultaneous Localization and Mapping (SLAM) and therefore we will review both. In this chapter, we jointly present the related work since they are closely intertwined with *VO* being an integral part of *SLAM* (see Fig. 1.3) Further, many state-of-the-art works follow a publication process similar to ours and start as *VO* approach and culminate in full *SLAM* system.

SLAM and *VO* have been an integral part of computer vision and robotics since the beginnings and a vast body of publications exists. Since a complete review would exceed the scope of this thesis, we only discuss seminal works, recent developments, and all the methods that are part of the experiments (see Chap. 6). For a comprehensive introduction to the history and early works in *SLAM*, we refer to the surveys by Durrant and Bailey [5, 31] and for more recent surveys to [149, 162]. In [15], Cadena et al. discuss future research directions and problems in *SLAM*. Similarly, we point to the tutorials by Scaramuzza and Fraundorfer [51, 131] for the area of *VO*.

We classify *SLAM* systems in two different ways not specific to any type of sensor or robot: (1) *map-centric* or *frame-based* and (2) *indirect*, *direct* and *semi-direct*. *Map-centric SLAM* systems align new frames against a local or global map, which is either sparse or dense (often referred to as model). While visually pleasing maps can be created this way, errors in the map are very problematic since all the new frames are aligned against the erroneous map. Feature-based [36, 116] methods and volumetric fusion systems [78, 79, 117] follow a *map-centric* paradigm. Alignment involves either reprojecting features into



(a) Classical *indirect* methods extract and match features and then minimize a reprojection error.



(b) *Direct* methods either images, point-clouds or edges directly without correspondence matching.

Figure 2.1: (a) Classical *indirect* methods extract features, thereby reducing the image to sparse interesting points, then match those features to a global map and finally estimate the pose by minimizing a reprojection error. In contrast, (b) *direct* methods do not rely on feature matching but align images, point-clouds or edges directly.

the current frame or rendering a view from the model to align against. In contrast, *frame-based* methods align with respect to a previously estimated frame(s), e.g. one or more Keyframes (KFs) [37, 38, 132, 133]. From a computational point of view, this is cheaper than aligning against a global map, since in *SLAM* there is typically a significant overlap with the last *KFs* and neither a costly rendering nor a search through an abundance of features is necessary. Another benefit is that the estimates between the frames are highly accurate and the accumulated drift in the global map can be corrected by loop closure.

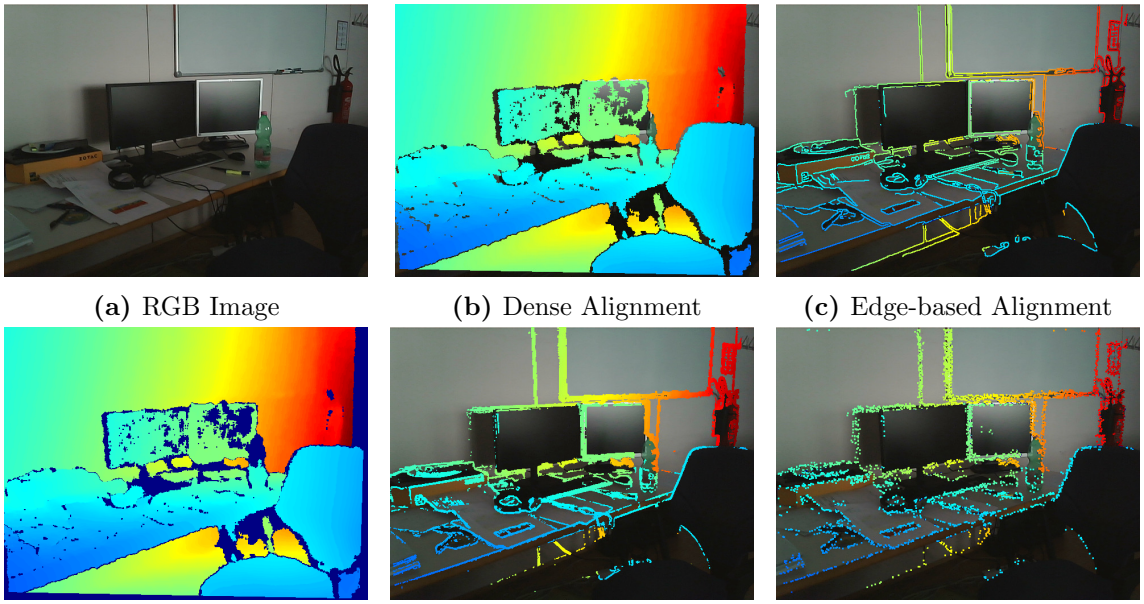
The related work review is structured into *indirect*, *direct* and *semi-direct* methods covered in Sections 2.1 - 2.3. Each section starts with a short history and overview of publications followed by a detailed description of the most important works. Figure 2.1 visualizes *indirect* and *direct* methods with their sub-categories. One interesting property to distinguish between methods is the image information involved in the pose estimation, which can be *sparse*, *dense*, *semi-dense* or *edge-based* (see Fig. 2.2). Section 2.4 summarizes most of the discussed *SLAM* and *VO* publications.

2.1 Indirect Methods

Indirect or feature-based methods have been studied since the early days of computer vision and form the basis of many *SLAM* [26, 36, 84, 116], *VO* [84, 85, 92, 120] and Structure from Motion (SfM) [1, 67, 135] systems. Figure 2.1a shows the typical processing pipeline, feature extraction and correspondence matching followed by a minimization of a reprojection error. The core principle is the reduction of the image to a few interesting keypoints (see Fig. 2.2f) with one of the many detectors such as SIFT [102], SURF [7], ORB [127] or corner detectors [63, 126]. Then each keypoint is described in terms of a descriptor to match correspondences with local or global map features (see Fig. 2.1a). One of the major pitfalls in indirect methods is the error-prone correspondence matching, which even with a robust RANSAC [47] step still often fails, especially whenever repetitive structures occur. After establishing the correspondences, the camera motion is estimated with a reprojection error, which minimizes the distance between a detected feature $\mathbf{p}_i$ and its reprojected correspondence $\mathbf{p}_j$ [65, 150]:

$$E = \sum_{\mathbf{p}_i, \mathbf{p}_j \in \mathcal{P}} d(\mathbf{p}_i, \mathbf{p}_j)^2, \tag{2.1}$$

where $\mathcal{P}$ is the set of correspondences and $d(.)$ a distance function. Note that (2.1) is a purely geometric error containing no appearance information.



(a) RGB Image **(b)** Dense Alignment **(c)** Edge-based Alignment

(d) Depth Map **(e)** Semi-Dense Alignment **(f)** Sparse Alignment

Figure 2.2: Different selected pixel regions for dense, semi-dense, sparse and edge-based alignment with color-coded selected pixels.

A known problem of feature-based approaches is that due to the sparse selection of

points (see Fig. 2.2f), the spatial distribution of the extracted features influences the pose estimation, which is a significant limitation in practice. For example, indoor scenes consist of areas, where many features are detected, e.g. posters, keyboards, and nearly feature-free parts, e.g. ceilings or walls. This often introduces a bias in the motion estimation since areas with many feature points contribute more to the reprojection error than areas with very few points.

Early monocular systems address the *SLAM* problem by filtering [21, 25, 26, 34, 112], also referred to as *Kalman filtering*. The basic idea is to update a joint probability distribution over all the involved model parameters with each new frame and to marginalize old state variables to keep the optimization bounded. The current state-of-the-art filtering-based method is MSCKF, which was originally published by Mourikis et al. [112] and then continuously improved over the next years [54, 94, 95, 113]. Strasdat et al. [146] conclude that optimization-based approaches have a higher accuracy than filtering-based ones at the same computational cost. Further, they show that to improve the accuracy of *SLAM* it is better to increase the number of features since more frames only add robustness to the system due to their highly redundant nature.

Since then, (non-linear) optimization-based approaches maintaining a sparse graph of *KFs* with their associated landmarks have become very popular [36, 84, 85, 92, 114, 116]. One of the earliest approaches, which also coined the term Visual Odometry, was the seminal work by Nistér et al. [120]. Shortly afterward, Klein and Murray proposed PTAM [84], the first system to separate tracking and mapping, which has since become the conceptual standard for most state-of-the-art systems. Leutenegger et al. [92] also include an IMU and introduce many improvements to the windowed optimization to keep it accurate and computable in real-time. Instead of keypoints, Eade and Drummond [34] rely on edge features, which are also used as an alternative or addition to keypoints in other approaches [85, 105].

The release of RGBD sensors was quickly followed by indirect *SLAM* systems such as [73] and [68], which assign the depth values from the sensor to features detected in the RGB image. While [73] applies a classical *indirect* pipeline with RANSAC and reprojection error minimization, [68] uses GICP [138] to align the images and set up a pose graph. In [36], Endres et al. presented RGBDSLAM, which increased accuracy and robustness compared to previous systems and was released as open-source implementation. One of the most popular and versatile systems is ORB-SLAM [114, 116], which runs in real-time on a CPU with a wide variety of sensors. In the following, we will describe selected works [36, 84, 85, 105, 116] in more detail.

Parallel Tracking and Mapping (PTAM) [84]: In their seminal work PTAM, Klein and Murray introduce several paradigms, which are still heavily used in many state-of-the-art systems. One of their key paradigms is to decouple the *SLAM* problem into separate *tracking* and *mapping* components, which makes it possible for both to run in separate threads and without the need to share any data association. *Tracking* assumes that the

map is fixed and estimates the relative camera pose in relation to the map. *Mapping* maintains, extends and refines the map but only processes *KFs* to reduce the highly redundant information from every single frame. PTAM uses FAST-10 [126] corners with their surrounding patches as features and defines their depth in terms of the *KF* they were first observed in. The tracking module estimates the current pose by first projecting 3D points from the map into the current frame under a motion model and then searching for corresponding matches in a two-step process. First, the initial pose is refined with 50 coarse matches and then reestimated with a total of 1000 matches. PTAM initializes the map in a semi-supervised manner with specific motion patterns in the beginning and continuously extends it by new *KFs*. A new *KF* is selected after either N frames or a certain distance between cameras. To estimate the depth of new points, PTAM performs an epipolar line search. While exploring, PTAM only refines a local portion of the map but switches to global refinement once the environment is well-explored and no new *KFs* are needed. One of the major downsides of PTAM [84, 85] is that it works only in relatively small areas such as desktop scenes since the size of the map quickly becomes too large for global Bundle Adjustment (BA) to handle. Further, it lacks loop closing capabilities and has problems with scarcely textured or repetitive surfaces present in most office scenes.

PTAM with edgelets (PTAM-e) [85]: In [85], Klein and Murray propose to use short straight lines, so-called edgelets [33, 34], in addition to keypoints in PTAM Tracking only short line segments has many favorable properties compared to the complete line, since they can deal with partially occluded and broken edges and even curves can be divided into many segments. To compute edgelets, PTAM-e extracts Canny [16] edges in each *KF* and breaks the edge chains at high curvature points to get straight edges. Through an epipolar search, PTAM then finds correspondence in a nearby *KF* for triangulation and rejects the correspondence if it cannot be verified in a third *KF*. The distance between corresponding lines is computed at two points at a distance of 5px from the edgelets projected center along the normal vector. Further, PTAM-e estimates the depth in a frame-to-frame manner by searching for correspondences along the edgelet's normal vector also taking motion blur into account [86]. Even with these improvements, PTAM-e suffers from the same problems as PTAM such as no loop closing capabilities and the restriction to small spaces.

ORB-SLAM [114, 116]: The current state-of-the-art feature-based system is ORB-SLAM2 [116], which extends the purely monocular ORB-SLAM [114] to stereo setups and RGBD sensors. Both systems [114, 116] rely on the same ORB features [127] for pose estimation, mapping and place recognition. ORB features are faster to extract than, e.g. SIFT [102], are invariant to scale and offer certain robustness against auto-gain, auto-exposure, and illumination changes. At its core, ORB-SLAM2 [116] only distinguishes between monocular and stereo camera setups. To process RGBD and stereo data identically, ORB-SLAM2 first extracts features in the RGB image, which is considered the left image, and then computes a virtual right coordinate like [145]. Similar to [123], they clas-

sify a stereo point as *close* if its depth is smaller than 40 times the stereo or RGBD baseline and as *far* otherwise. *Close* keypoints can be immediately triangulated safely, while *far* ones offer accurate rotation information and are triangulated once they are supported by multiple viewpoints. All points without valid depth from the depth map are considered a monocular keypoint and are also triangulated over multiple viewpoints. ORB-SLAM2 has a place recognition module [52] for relocalization in case of errors or to continue in an already mapped scene and loop closure. The complete pipeline comprises three separate components (1) localizer, (2) local and (3) global mapping, which run in parallel thread. For every new frame, the localizer tries to estimate the world position with respect to matched features by optimizing the reprojection error. Whenever the number of matches drop beneath a threshold, a new *KFs* is created and inserted into a covisibility graph, where nodes are *KFs* and edges in between represent shared observations. ORB-SLAM initially creates many *KFs* but removes redundant ones later. The local mapper then optimizes over a set of covisible *KFs* and their shared points. Finally, after loop closure and Pose Graph Optimization (PGO), the global mapper runs a full *BA* over all the *KFs* and points. ORB-SLAM2 is a very popular system since it is easy to run, versatile in terms of scenes and sensors, and also runs in real-time on the CPU of decent computer. However, scarcely textured areas are challenging since most feature detections come from noise and wrong correspondence matching often occurs. Highly redundant textures pose another problem since the same feature is detected multiple times and tracking often fails

RGBDSLAM [36]: At the time of its initial release, RGBDSLAM was one of the first feature-based *SLAM* systems designed for RGBD sensors. Its primary components are analogous to standard feature-based systems as they extract and match features. Since many feature-based approaches have difficulties with wrong correspondences and the resulting wrong pose estimates, RGBDSLAM penalizes non-occluding transformations via an environment measurement model. The idea is to reproject a 3D point from one depth map to the other, finding the corresponding point via projective data association and classifying the points into inliers, outliers, and occlusions. This is closely related to the Iterative Closest Point (ICP) point-to-point formulation with projective data association discussed in Section 4.1.3. RGBDSLAM searches for local loop candidates among the last N and spatially close *KFs* as well as K randomly sampled *KFs* from the pose graph with a bias towards older frames. To close large loops, RGBDSLAM maintains a set of very few distinct *KFs* from which again several *KFs* are randomly drawn.

Edge SLAM [105]: The idea of Edge SLAM is closely related to standard feature-based systems in the sense that it detects features and establishes correspondences with an optical flow tracker. The main difference to other systems [84, 116] is that it relies on edge points as features. Loop closure is based on image moment invariants [72], where every *KF* is divided into 16 quadrants and then each quadrant is matched to another *KF* based on the edge detections and intensities. The threshold to consider a *KF* a match is

given as the minimum matching computed to the five previous *KF* but it is only considered a loop candidate if three successive *KFs* are above the threshold.

2.2 Direct Methods

In recent years, *direct* methods that avoid correspondence matching have become very popular. They can be further divided into *image*-, *point cloud*- and *edge*-based alignment as depicted in Figure 2.1b. While processing the complete image information [22, 80] avoids bias to spatial distribution, the computational burden is significant and low-gradient regions do not carry valuable information for the optimization. Thus, the selection of the residuals is a critical point for direct methods. Figure 2.2 shows an overview of the different selection strategies, where *dense* selects the complete image, *semi-dense* high-gradient regions, *edge-based* points corresponding to edge detections and *sparse* only a few interesting points.

Image alignment methods are probably the most intuitive, as they simply align two images trying to minimize their intensity differences, which is equal to maximizing their photo-consistency. Thus, such methods are often referred to as *photo-consistency*-based. Comport et al. [22] proposed a very early *dense VO* method, which aligns subsequent stereo image pairs. In [40], Engel et al. present a *semi-dense VO* system for a monocular camera. This served as the foundation for the well-received LSD-SLAM [38, 153], which has been extended in several follow-up works [18, 39]. Recently, Engel et al. proposed DSO [37], which relies on a *sparse* formulation and shows better accuracy than LSD-SLAM even without loop closure.

The introduction of RGBD sensors has had a significant impact on *VO* and *SLAM* research since the depth map offers the possibility to compute a *dense* point cloud for *image* and/or *point cloud* alignment. Due to the readily available depth information, many works that generate dense models in real-time have been published. One of the most influential work for real-time *point cloud* alignment was KinectFusion by Newcombe et al. [75, 117], which aligns incoming depth maps against a dense volumetric model with a geometric error based on a point-to-plane *ICP* formulation. KinectFusion has since inspired works such as the extension Kintinuous [157, 158], a full *SLAM* system based on combined *image* and *point cloud* alignment, and systems for large-scale scenes such as the very fast InfiniTAM [78, 79] or [119]. All of these approaches require a strong Graphics Processing Unit (GPU) to process the *dense* model-to-frame alignment. Around the same time, Whelan et al. [156] presented an RGBD *VO* approach for *image* alignment, which also generates a dense volumetric reconstruction but requires a *GPU*. ElasticFusion [159] is a *SLAM* system based on surfels instead of a dense volumetric model and combines a photometric- and *ICP*-based error. Dai et al.'s BundleFusion [24] applies a coarse feature-based approach followed by a photometric and geometric alignment, thereby implicitly closing loops. Recently, BAD-SLAM [136] showed how to incorporate a full *BA* in a surfel-based model. All of those approaches require at least one strong *GPU* to be able

to run in real-time.

Since robots are mobile and often have to work on constrained hardware, other approaches that run in real-time on a CPU but do not generate a *dense* 3D model are of great interest. The first direct approaches that use the depth map to compute a *dense* 3D point cloud for *image* alignment are [4] and [143] followed by Tykkälä et al.[151], who additionally incorporate a geometric error in the motion estimation. Building upon [4, 143], Kerl et al. [81] introduced the *dense* DVO, which includes a robust sensor model and runs in real-time on a CPU. Shortly afterward, they added a global mapper and an additional geometric error in DVO-SLAM [80]. RGBDTAM [23] combines also photometric and geometric errors, computes a sparse map and performs loop closure all on a CPU.

The downside of geometric errors with an *ICP*-based formulation is that they typically only work in areas with geometric structure and often fail in indoor scenes with many flat surfaces such as walls and floors. Methods that align *images* face a similar issue whenever areas are not sufficiently textured, e.g. white walls. Further, when working with raw intensity values, illumination changes and motion blur are critical points requiring special handling. Another disadvantage of both methods is that they easily run into local minima whenever the initialization is not close to the global minimum as shown in [89, 133], which can only be partially addressed by a coarse-to-fine optimization scheme.

Instead of intensity values, many recent methods use the more robust edges to align two frames (see Fig. 2.1b). Tarrio and Pedre [76] detect Canny edges [16] and try to match them between images by searching along the normal direction, which is computationally expensive and error-prone. In their direct edge-alignment (D-EA), Kuse and Shen [89] instead pre-compute the distance to the closest edge at each pixel position with a distance transform (DT) [44] and optimize with a subgradient method. Wang et al. [154] jointly minimize edge distance and a photometric error at high-gradient pixels.

Most robust systems do not rely on just one error but a combination of either *photometric* and *ICP* [24, 80, 99, 136, 158, 159] or *photometric* and edge-based [154] or *ICP* and edge-based [132] error. A combination of multiple errors proves to be more robust but is computationally more demanding.

DVO-SLAM [80, 81]: Kerl et al. introduced DVO [81], one of the first real-time capable *VO* systems to directly minimize a dense photometric error. DVO uses the complete image information and does not reduce it to high-gradient regions, which has some positive effects on robustness but adds significant computational burden to the whole system. In their follow up work DVO-SLAM [80], they extend DVO to a full *SLAM* system and estimate motion by a combination of geometric and photometric error. DVO-SLAM proposes an entropy-based *KF* selection algorithm, which computes an entropy ratio between two relative motions: (i) from the last *KF* to the current frame and (ii) from the last *KF* to very next one. If this ratio is below a threshold, it takes the previous frame, i.e. the last good one, as a new *KF*. For loop closure, DVO-SLAM relies on metrically nearest neighbor search in a sphere centered at the most recent *KF* position, which typically cannot

detect large-scale loop closures after drift. While this method might work in small indoor environments, loop closures are more important in large-scale scenes, where DVO-SLAM can show significant drift. Another issue of DVO is that even though its implementation uses parallelized operations, the extensive computations restrict DVO to "half" resolution of 320×240, thereby sacrificing accuracy.

RGBDTam [23]: RGBDTam combines a semi-dense photometric and dense geometric error for relative pose estimation. Instead of high-gradient regions, they select pixels at Canny edges [16]. The main difference to other approaches is that RGBDTAM incorporates multi-view constraints with uncertainties. This makes it possible to refine the initial depth measurements or even estimate depth for regions with invalid depth measurements from the RGBD sensor. Similar to [116], RGBDSLAM runs a standard bag-of-words approach [52] based on ORB features for loop closure. Additionally, they aim to reduce the number of newly generated keyframes by re-activating old ones.

Elastic Fusion (EF) [159]: Whelan et al. [159], proposed Elastic Fusion, a surfel-based dense *SLAM* system. Similar to [78, 117, 156], EF performs a dense frame-to-model tracking and combines photometric and geometric errors. EF splits the model into an active part, which is used for pose estimation and data fusion, and an inactive part containing parts no longer in the field of view. In each frame, EF tries to register the associated inactive with the active part and if it is successful, a loop closure is detected. To close the loop, the model is non-rigidly deformed into the current frame and the inactive part is re-activated since it is visible again. However, this procedure only works for small-scale loops and in order to detect large-scale loops, EF applies a Fern-based bag-of-words approach [55]. If a large-scale loop is found, EF proceeds similar to the small-scale loop closure case. EF heavily relies on the GPU due to the rendering of the depth and intensity image for their frame-to-model tracking. While for desktop-scale environments the 3D models are appealing and consistent, in larger scenes, EF suffers from inaccurate camera pose estimation, which can easily cause tracking losses.

Bundle Fusion (BF) [24]: BF employs a sparse-then-dense global pose optimization, where a coarse alignment is computed from a set of sparse feature correspondences as an initialization for a dense alignment. The dense alignment comprises a photometric error evaluated on the gradient of the luminance and a geometric error based on point-to-plane *ICP*. One of the major differences to other approaches is that BF optimizes over the complete hierarchy of RGBD frames, thereby implicitly closing loop and performing relocalization. In order to process the complete hierarchy efficiently, the authors propose to first locally align short sequences of consecutive frames, followed by a global alignment in relation to each other in a second run. Similar to [78, 159], BF also creates a dense volumetric model of the scene in real-time but with the key difference that already integrated data can be updated or removed from the model. By deleting at an old pose and

integrating at an updated one, the model stays consistent after pose refinements and even after loop closure. The optimization over the whole trajectory makes BF one of the most accurate system available but it comes at the high computational cost of two GPUs.

KinectFusion [117] and Kintinuous [156]: The seminal work **KinectFusion [117]** by Newcombe et al. was one of the first to use the depth map from a Microsoft Kinect and sparked a new branch of research. KinectFusion generates a dense volumetric model directly from the depth maps and aligns incoming depth maps to the model according to a geometric point-to-plane *ICP* error. However, KinectFusion has several shortcomings such as the relatively small model size restricting it to desktop scenes and its fragile tracking system. In [156], Whelan et al. addressed many of these limitations with their **Kintinuous** system. Instead of a fixed-sized model, it relies on a rolling buffer, which only keeps the most recent part of the model, to enable tracking arbitrarily long sequences. KinectFusion also introduces an improved pose estimation based on a combination of photometric and geometric error.

BAD-SLAM [136] In contrast to previous direct *SLAM* systems, BAD-SLAM introduces a novel direct *BA* formulation. BAD-SLAM is a map-centric approach based on surfels because they are easy to update, fuse and deform after a loop closure. The surfels are parametrized by the depth of the *KF* they were first observed in and can be interpreted as landmarks similar to feature-based approaches [116]. Compared to voxel-based representations [79, 117, 119], surfels can represent details of arbitrary scale and also very thin surfaces. Similar to most state-of-the-art approaches, BAD-SLAM selects *KFs* and only processes the non-*KFs* to estimate the pose to the next *KF*. Since BAD-SLAM is a map-centric approach, instead of aligning the individual frames [23, 80, 134], it projects each surfel into each frame and minimizes a geometric point-to-plane error and a photometric error. The difference to [159] is that BAD-SLAM works with gradients of images instead of raw intensity values. On most benchmark sequences, BAD-SLAM shows impressive results in terms of accuracy due to their global *BA*. Even though the *BA* runs only on *KFs*, the problem quickly becomes too large to compute it in real-time. Thus, BAD-SLAM runs an optimization scheme, which alternates between cost optimization and surfel updates. The authors also study properties of various *SLAM* systems and demonstrate that most systems are highly sensitive to rolling shutter, and the RGBD sensor's calibration and synchronization.

InfiniTAM [78, 79] Kähler et al. [79] proposed InfiniTAM, a fast volumetric fusion framework, which builds upon the ideas of KinectFusion [117]. In addition to speed, the main contribution was a memory-efficient model, which stores volumetric data only in areas with a surface instead of simply allocating a fixed volume as in KinectFusion. In their follow-up work, Kähler et al. [78] added loop closure and relocalization capabilities with a random-fern based bag-of-words approach [55]. The key difference to earlier works [79,

117, 156] is that [78] splits the model into small locally accurate submaps of around 50 *KFs*, which remain fixed throughout the sequence. Once a loop is detected, only the relative transformations between the submaps has to be corrected instead of correcting all the frames, which greatly reduces the computational burden and does not require any changes to the model.

RKSLAM [99]: RKSLAM first performs a fast dense photometric and geometric alignment [81] to get a coarse estimate of the transformation. This is followed by an accurate sparse feature-based alignment to additionally increase the robustness. Individual features are tracked by estimating a global homography, which can also handle strong rotations and fast motions but is restricted to planar surfaces or pure rotations. For loop closure, RKSLAM extracts ORB features [127] and matches them against a database with the bag-of-words approach presented in [115]. The main contribution is an efficient incremental *BA* with several speed-ups such as only partly recomputing the Hessians and Jacobians instead of building them from scratch. RKSLAM also maintains a dense volumetric model [79] and updates it, whenever poses are corrected or updated, e.g. in the case of loop closure. Similar [24, 155], after loop closure or *BA*, the volumetric model is updated through a reintegration step.

Large-Scale Direct-SLAM (LSD-SLAM) [38]: LSD-SLAM is one of the most popular direct *SLAM* systems for monocular cameras and an extension of [40]. It is *KF*-based and estimates the relative motion by minimizing a semi-dense photometric error (see Fig. 2.2e). Once the camera moves a certain distance from the last *KF*, a new *KF* is created and its depths are initialized from the estimates of the last *KF*. Non-*KFs* are only used to refine the current *KF* in a filtering-based approach as presented in [40]. The global map consists of a pose graph, where *KFs* are nodes and relative transformations between them are edges. LSD-SLAM searches for loop closure candidates among the last 10 *KFs* as well as in a database [56].

Direct Sparse Odometry (DSO) [37]: Following up on the successful LSD-SLAM, Engel et al. [37] developed a novel *VO* system for a monocular camera called DSO. In contrast to semi-dense methods [38], DSO selects a sparse set of high-gradient points (compare Fig. 2.2f to 2.2e) and 7 pixels from their neighborhood. DSO estimates the relative motion between a *KF*, where the depths of the sparse points are known, and a current frame. It also maintains a local sliding window of *KFs*, where all involved model parameters, namely the camera intrinsics, camera poses, brightness terms and the inverse depths of points are optimized in a fully direct formulation. This direct *BA* is the main difference to hybrid approaches such as SVO [49, 50] or [91], which require an indirect method for such optimizations. Whenever the local window becomes too large, a *KF* is marginalized [92] instead of simply dropped from the system (see Sec. 4.4.2.3 for more

details). DSO shows very promising results and even models the full photometric calibration of the camera, including lens attenuation, gamma correction and known exposure times. However, its accuracy greatly suffers under rolling shutter cameras and it lacks loop closure capabilities. Both issues were addressed in follow-up papers by Gao et al. [53] and Schubert et al. [137].

Direct Edge Alignment (D-EA)[89]: D-EA is a direct *VO* method, which aligns the edge detections of two frames instead of the images. The goal is to minimize a geometric error given as the distance between the edge detections in a current frame and the reprojected edges from *KF*. Kuse and Shen [89] propose to pre-compute the distance to the closest edge in form of a Distance Transform (DT) instead of performing the costly search for the closest edge (see Sec. 4.1.1).The authors argue that, while direct methods work well in practice, their cost functions are not differentiable and convergence is not guaranteed. Thus, instead of applying the popular Gauss-Newton (GN) or Levenberg-Marquardt (LM) algorithms, they propose to apply the less efficient sub-gradient [12] method instead. D-EA builds upon DVO [81] and similarly runs on 6 pyramid levels at a maximum resolution of 320×240, which again reduces the accuracy. In the experiments, Kuse and Shen show that the convergence basin of D-EA is larger than that of DVO [81] by simply skipping every 2^{nd}, 3^{rd} or 4^{th} frame to simulate larger motions. While this work presents some interesting contributions, D-EA suffers from many problems such as slow convergence speed due to the subgradient method and the requirement to compute the *DT* for every frame, and inaccurate pose estimates mainly caused by the low resolution.

Canny-VO [165] and [164]: In [164], Zhou et al. compute a semi-dense pixel region in each frame by thresholding the gradient's norm. Then the current frame and a *KF* are aligned by reprojecting the *KF*'s curve to the current frame. The alignment is closely related to the *ICP*-based formulation in [88] and edge-based errors [89, 133, 154]. The authors argue that the problem of a standard *DT* is that the residual is always positive, thus the *GN* method is not applicable. To address this problem they propose two alternative formulations: (1) approximate Nearest Neighbor (NN) fields (ANNFs) [164] and (2) oriented nearest neighbor fields (ONNFs) [165]. Instead of storing the Euclidean distance like the *DT*, ANNFs keep the x- and y-coordinates of the *NN* and ONNFs are divided into separate distance fields according to the edge orientation. The different formulations are discussed in detail in Section 4.1.1. One of the downsides of Canny-VO is the increased computational burden when computing eight separate *DTs*.

2.3 Semi-Direct Methods

Direct methods are potentially more accurate in frame-to-frame relative motion estimation but lack the capabilities to reuse the map or to efficiently perform global optimization. Thus, combinations of direct and indirect methods have emerged in recent years [14, 49,

50, 91, 97]. We will focus on the most interesting works by Forster et al.[49, 50] and Lee and Civera [91].

Semi-Direct Visual Odometry (SVO)[49, 50]: Similar to PTAM [84], SVO also decouples mapping and pose estimation but instead of relying on features for both components, the pose estimation is based on a direct method. SVO extracts corner features [84] and in scarcely-textured areas also pixels along high-gradient regions, i.e. edges, similar to [33]. For pose estimation, SVO initializes by a sparse image alignment, which minimizes the photometric error at reprojected pixel positions of features seen in the last frame. The initialization is followed by a refinement step, which aligns with respect to the oldest possible frame instead. This is followed by another alignment step with the oldest possible frame to establish feature correspondences. These correspondences are then jointly refined in a *BA* and the depths updated with a filtering-based method. SVO is still widely used for mobile robots like Unmanned Aerial Vehicles (UAVs) due to its speed and low computational cost. Since it is in its essence only a *KF*-based *VO* system and does not perform any local or global refinement, it cannot compete with systems such as DSO [37] and ORB-SLAM [114].

Loosely-Coupled Semi-Direct SLAM [91]: Thus, in a recent publication Lee and Civera [91] proposed to combine DSO [37] and ORB-SLAM [114] in a loosely-coupled way. DSO is responsible for estimating camera poses and optimizing *KF* poses, sparse depth and camera intrinsics in a full local *BA*. Once DSO marginalizes a *KF*, the already initialized frame is passed on to ORB-SLAM. ORB-SLAM maintains a full global map and performs loop closure and relocalization. The main difference to SVO [49, 50] and other semi-direct approaches [14, 97] is that this system maintains two separate maps, one for DSO and one for ORB-SLAM. One of the most interesting points of this work is that the two methods complement each other very well. For example, DSO can still estimate relative poses in scarcely textured or highly repetitive scenes, where ORB-SLAM often fails and ORB-SLAM increases overall accuracy due to loop closures and also provides relocalization capabilities.

2.4 Conclusion

This chapter first established several classifications for *VO* and *SLAM*, then provided a short historical overview for *indirect*, *direct* and *semi-direct* methods and finally discussed seminal works and recent publications with a focus on RGBD-based systems.

There are still many challenges in state-of-the-art systems and echo of the methods suffers from its own limitations. *Indirect* methods contain an error-prone correspondence matching step and have problem in scarcely textured or repetitive scenes. In contrast, photometric *direct* formulations require an initialization close to the minimum and are

Visual Odometry Systems													
System	Photo	Selection	ICP	Edge-based	Feature	Input Data	CPU/GPU	Local Opt.	Global Opt.	Map Centric	Loop Closure	Release	Open-Source
DVO [81]	✓	D				RGBD	C					2013	✓
FOVIS [73]					✓	RGBD	C			✓		2011	✓
D-EA [89]				✓		RGBD	C					2016	✓
SVO [50]				✓		Mono	C	✓	✓	✓		2016	✓
ICPCUDA [158]				✓		Depth	C	✓		✓		2015	✓
Zhou et al. [164]				✓		RGBD	C	✓				2017	
DSO [37]	✓	S				Mono	C	✓				2017	✓
REVO [133]				✓		RGBD	C	✓				2017	✓
REVO+ICP [132]			✓	✓		RGBD	C	✓				2017	
Canny-VO [165]				✓		RGBD	C	✓				2019	
Simultaneous Localization and Mapping Systems													
PTAM [84]				✓		Mono	C	✓	✓	✓	-	2008	✓
LSD-SLAM [38]	✓	SD				Mono	C	✓	✓		[56]	2014	✓
DVO-SLAM [80]	✓	D	✓			RGBD	C	✓	✓		Own	2013	✓
Kintinuous [156]	✓	D	✓			RGBD	G	✓	✓	✓		2013	✓
RGBDSLAM [36]					✓	RGBD	C	✓	✓	Own		2014	✓
ORB-SLAM2 [116]				✓		RGBD	C	✓	✓	✓	[115]	2016	✓
BundleFusion [24]	✓	SD	✓		✓	RGBD	G	✓	✓	✓		2016	✓
Elastic Fusion [159]						RGBD	G	✓	✓	✓	[55]	2016	✓
RGBDTAM [23]	✓	E	✓			RGBD	C	✓	✓		[115]	2017	✓
InfiniTAMv3 [78]	✓	D	✓			Depth	G	✓	✓	✓	[55]	2017	✓
BAD-SLAM [136]	✓	D	✓			RGBD	G	✓	✓		[115]	2019	✓
Lee et al. [91]	✓	S			✓	Mono	C	✓	✓	✓	[115]	2019	✓
RESLAM [134]			✓			RGBD	C	✓	✓		[55]	2019	✓

Table 2.1: A list of different *VO* and *SLAM* systems with their respective properties. Our contributions are depicted in bold.

susceptible to illumination changes, while point cloud-alignment is computationally demanding due to the dense formulation and needs geometric structure. Edge-based approaches have many favorable qualities such as a larger convergence basin and a certain robustness to illumination changes but edge detection can be costly compared to using raw intensity values and the distribution of the edges can introduce a bias in the estimation.

Table 2.1 provides separate overviews of *SLAM* and *VO* systems discussed in this chapter with our contributions highlighted in bold. For each system, Table 2.1 lists the error type, the input data, whether it runs a CPU or GPU, the contained components, the release date and the availability of the code as open-source software. Note that ICPCUDA [158] is not explicitly discussed in this chapter, since it is not a publication but simply a fast implementation of geometric frame-to-model alignment similar to [79, 118, 158].

3

Direct Methods - Theory and Background

Contents

For many decades indirect methods have dominated the field and are still in use today. However, due to their favorable qualities, direct formulations have largely replaced feature-based methods in most of the Visual Odometry (VO) and Simultaneous Localization and Mapping (SLAM) systems. This chapter aims to give the reader a compact overview of the mathematical tools and formulations required to understand direct methods and the algorithms presented in this thesis. The notations and terminologies presented in this chapter are in line with most state-of-the-art literature.

Section 3.2 starts with the basic camera model, then shows how to compute, transform and reproject 3D points and then gives a concise overview of 3D rigid body motions also discussing their advantages and disadvantages. In Section 3.3, we present direct image alignment, which is the building block for many state-of-the-art direct algorithms [37, 38]. Edge-based and point cloud-based direct methods are discussed in Sections 4.1.1 and 4.1.3 in the next chapter. Finally, Section 3.4 derives various parameter estimation techniques and shows how to set up a robust optimization.

3.1 Notations and Conventions

Before describing the methods and algorithms, we introduce the mathematical notations used throughout this thesis. Additional definitions and mathematical tools such as the *vec* operator, the skew-symmetric matrix and the Kronecker product are described in

Entity	Notation
Scalar	a, c_i
Vector in 2D, e.g. position in image space	$\mathbf{p} = (x, y)^{\mathrm{T}}$
Vector in 3D	$\mathbf{P} = (X, Y, Z)^{\mathrm{T}}$
Homogeneous vector in 2D	$\tilde{\mathbf{p}} = (x, y, w)^{\mathrm{T}}$
Homogeneous vector in 3D	$\tilde{\mathbf{P}} = (X, Y, Z, w)^{\mathrm{T}}$
Matrix, Image	$\mathbf{M} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$
Vector Space	$\mathbb{R}^3$
Mapping Function	$\pi : \mathbb{R}^3 \to \mathbb{R}^2$
Sets	$\mathcal{L}, \mathcal{E}$

Table 3.1: List of notations used in this thesis.

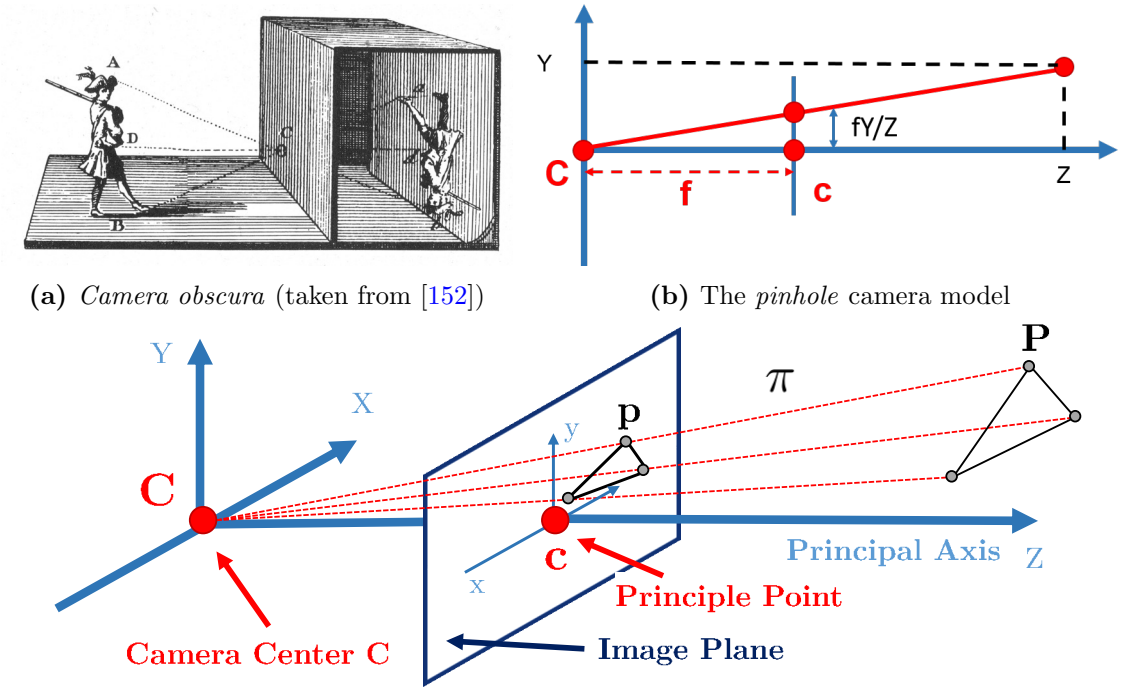
Appendix C.2. Scalars are depicted in italic fonts, e.g. x or c_i. Vectors in 2D and 3D are written as a single lowercase or uppercase bold letter, respectively, e.g. $\mathbf{v}$ or $\mathbf{V}$. Their homogeneous versions have an additional $\tilde{}$ symbol on top, e.g. $\tilde{\mathbf{v}}$ or $\tilde{\mathbf{V}}$. Matrices and images are represented as a single uppercase bold letter, e.g. $\mathbf{M}$, and indexed in column-major order $\mathbf{M}(x, y) = \mathbf{M}(\mathbf{p})$, where $\mathbf{p} = (x, y)^T$. We denote the $N \times N$ identity matrix as $\mathbf{I}_N$ and the zero matrix as $\mathbf{0}_N$. Vector spaces are given in double-lined upper case letters and the mapping functions between different spaces in small Greek letters, e.g. $\pi : \mathbb{R}^3 \to \mathbb{R}^2$. Sets are represented as calligraphic letters, e.g. $\mathcal{E}$ or $\mathcal{K}$. Table 3.1 summarizes the notations described above.

3.2 Camera Model and 3D Rigid Body Motions

In this section, we describe the image formation process and underlying camera model to map the 3D world onto the 2D image space. We rely on the *pinhole* camera model, which is widely used to model digital cameras due to its simplicity and linearity. For details about other camera models such as affine or non-central models, we refer the reader to [65].

Probably the best way to visually explain the *pinhole* camera model is the camera obscura (see Fig. 3.1a), a dark room or box with an infinitesimal small hole, the *pinhole*, in one of the walls. Objects outside the box reflect light from an external source, then light rays pass through the *pinhole* and project a top-down image of the scene onto the

opposite wall. Figure 3.1b and 3.1c show the *pinhole* camera model viewed from different angles. The distance between the *pinhole* and the image plane is called the focal length f, the position of the hole in the wall is given by the principal point c and the hole itself is the camera center **C**. The only difference to Figure 3.1a is that the image plane lies in front of the camera center and not behind it so that the image is not projected top-down. However, a real camera has a complex lens system and an aperture, which introduces several physical limitations such as *vignetting*, *chromatic aberration* and only has limited *depth of field*. For a detailed description of these limitations alongside other problems specific to digital cameras, we refer to [148]. Even though a real camera is quite different



(a) *Camera obscura* (taken from [152])

(b) The *pinhole* camera model

(c) Projection of 3D points **P** into image space **p** with projection function π.

Figure 3.1: The camera obscura (a) lets light pass through a pinhole such that it is projected onto the opposite wall. (b,c) depict the pinhole camera model viewed from different angles and (c) additionally demonstrates the projection of a 3D point **P** onto the image plane **p**.

from a *pinhole* camera, this model is used in nearly all state-of-the-art image processing pipelines and many of these problems can be addressed or at least mitigated by intrinsic camera calibration [11, 45, 65]. The intrinsic parameters of the camera, i.e. the focal lengths f_x, f_y and the principal point $[c_x, c_y]$, are typically encoded as 3×3 camera matrix

$\mathbf{K}$:

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.1}$$

with its corresponding inverse camera matrix $\mathbf{K}^{-1}$:

$$\mathbf{K}^{-1} = \begin{pmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.2}$$

From a point $\mathbf{p} = (p_x, p_y)^T$ in image space, the corresponding 3D point $\mathbf{P} = (X, Y, Z)^T$ can be computed via $\mathbf{K}^{-1}$ as:

$$\mathbf{P} = \mathbf{K}^{-1}\tilde{\mathbf{p}}Z, \tag{3.3}$$

where $\tilde{\mathbf{p}} = (p_x, p_y, 1)^T$ and Z its depth. When a depth map $\mathbf{Z}$ is available, e.g. from an RGBD sensor, Z is simply given as $Z = \mathbf{Z}(\mathbf{p})$. For ease of notation, we define an inverse projection function π^{-1}, which computes a 3D point $\mathbf{P}$ from a pixel position $\mathbf{p}$ and its corresponding depth in the respective camera frame:

$$\mathbf{P} = \pi^{-1}(\mathbf{p}, Z) = \left(\frac{p_x - c_x}{f_x} Z, \frac{p_y - c_y}{f_y} Z, Z \right)^T = (X, Y, Z)^T. \tag{3.4}$$

While $\mathbf{K}^{-1}$ computes a 3D point $\mathbf{P}$, the camera matrix $\mathbf{K}$ projects a point from 3D to homogeneous coordinates $\tilde{\mathbf{p}}$:

$$\tilde{\mathbf{p}} = (p'_x, p'_y, w)^T = \mathbf{K}\mathbf{P}. \tag{3.5}$$

A dehomogenization, i.e. dividing the first two elements by the third, transforms $\tilde{\mathbf{p}}$ back to the image plane:

$$\mathbf{p} = \left(\frac{p'_x}{w}, \frac{p'_y}{w} \right)^T. \tag{3.6}$$

Similar to (3.4), we also define a projection function π to unify (3.5) and (3.6):

$$\mathbf{p} = \pi(\mathbf{P}) = \left(\frac{Y f_x}{Z} + c_x, \frac{X f_y}{Z} + c_y \right). \tag{3.7}$$

Figure 3.2 depicts the use of π^{-1} to compute the 3D point $\mathbf{P}_i$ from $\mathbf{p}_i$ and π to project the 3D point to $\mathbf{p}'_i$. The inverse projection function π^{-1} only computes a point in the local camera coordinate system and analogously the projection function π only projects a 3D point to the image plane if it is defined in the local camera coordinate system.

However, *VO* and *SLAM* typically deal with multiple cameras with respective coordinate systems. To reference individual cameras and points in one system, its essential to describe them with respect to a common *world coordinate system* as shown in Figure 3.2 with the origin at $\mathbf{W}$. While the choice of the position of the *world coordinate system's*

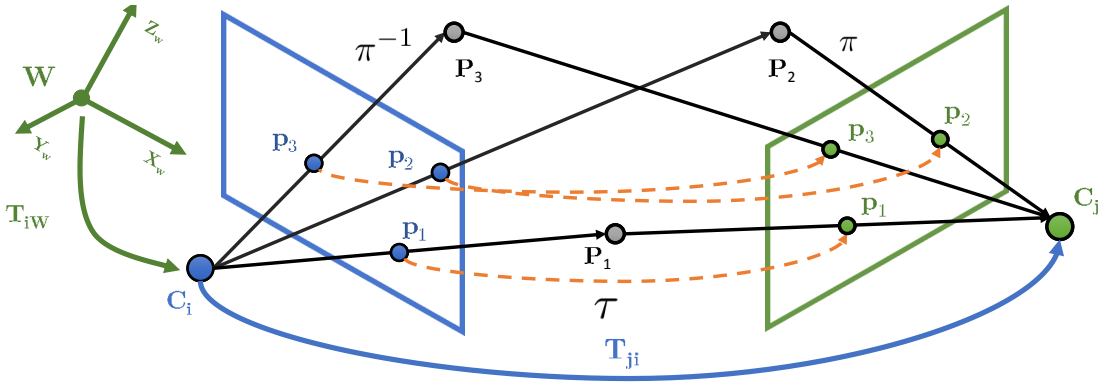


Figure 3.2: The relation between points in the image $\mathbf{p}_i$, the corresponding 3D points $\mathbf{P}_i$ and the reprojected points $\mathbf{p}'_i$, which are computed by the projection functions π, its inverse π^{-1} and the full reprojection function τ. The camera center $\mathbf{C}_i$ is related to the *world coordinate system* by a transformation $\mathbf{T}_{iw}$ and $\mathbf{T}_{ji}$ is the transformation between $\mathbf{C}_i$ and $\mathbf{C}_j$.

origin is arbitrary, we always choose the camera center of the first frame $\mathbf{C}_0$ as origin, i.e. $\mathbf{C}_0 = [0, 0, 0]$. A 3D rigid body motion between different cameras preserves distance and orientation. It is defined as a transformation matrix on the special Euclidean group $\mathbf{T} \in SE(3)$ comprising a 3×3 rotation matrix $\mathbf{R} \in SO(3)$ and a 3×1 translation vector $\mathbf{t} = [t_x, t_y, t_z]^T$:

$$\mathbf{T}_{4\times4} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} = \begin{pmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.8}$$

where $\mathbf{0} = [0, 0, 0]^T$. The rotation matrix $\mathbf{R}$ is from the special orthogonal group $SO(3)$ and satisfies (i) $\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}_3$ and (ii) $det(\mathbf{R}) = 1$. Since $\mathbf{R}^{-1} = \mathbf{R}^T$, the corresponding inverse transformation $\mathbf{T}^{-1}$ is given as:

$$\mathbf{T}^{-1}_{4\times4} = \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix}, \tag{3.9}$$

The matrix $\mathbf{T}$ can be decomposed into translations along the x-, y- and z-axis as:

$$\mathbf{T}_x = \begin{pmatrix} & t_x \\ \mathbf{I}_3 & 0 \\ & 0 \\ \mathbf{0}^T & 1 \end{pmatrix}, \quad \mathbf{T}_y = \begin{pmatrix} & 0 \\ \mathbf{I}_3 & t_y \\ & 0 \\ \mathbf{0}^T & 1 \end{pmatrix}, \quad \mathbf{T}_z = \begin{pmatrix} & 0 \\ \mathbf{I}_3 & 0 \\ & t_z \\ \mathbf{0}^T & 1 \end{pmatrix}, \tag{3.10}$$

and rotations about the x-axis by an angle ϕ, about the y-axis by an angle θ and about

the z-axis by an angle φ:

$$T(\phi) = \begin{pmatrix} \mathbf{R}_\phi & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}, \quad T(\theta) = \begin{pmatrix} \mathbf{R}_\theta & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}, \quad T(\varphi) = \begin{pmatrix} \mathbf{R}_\varphi & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}. \tag{3.11}$$

The individual rotation matrices are:

$$\mathbf{R}_\phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{pmatrix}, \mathbf{R}_\theta = \begin{pmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{pmatrix}, \mathbf{R}_\varphi = \begin{pmatrix} c_\varphi & -s_\varphi & 0 \\ s_\varphi & c_\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.12}$$

where $s_\theta = sin(\theta)$ and $c_\theta = cos(\theta)$ and analogously for ϕ and φ. A 3D point $\mathbf{P}$ defined in one coordinate system can be transformed into another by a simple vector-matrix multiplication:

$$\tilde{\mathbf{P}}' = \mathbf{T}\tilde{\mathbf{P}}. \tag{3.13}$$

The last entry of $\tilde{\mathbf{P}}'$ is always equal to 1, thus the last line of $\mathbf{T}$ is typically dropped such that:

$$\mathbf{P}' = \mathbf{T}_{3\times4}\tilde{\mathbf{P}}. \tag{3.14}$$

Homogeneous coordinates can be avoided altogether by splitting $\mathbf{T}$ into a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$:

$$\mathbf{P}' = \mathbf{R}\mathbf{P} + \mathbf{t}. \tag{3.15}$$

Since the camera center $\mathbf{C}$ in the local coordinate frame is at the origin $\mathbf{C} = [0, 0, 0]^T$, the position of the camera center in the *world coordinate system* $\mathbf{C}_W$ can be computed from

$$\mathbf{C} = [0, 0, 0]^T = \mathbf{R}\mathbf{C}_\mathbf{W} + \mathbf{t}, \tag{3.16}$$

as

$$\mathbf{C}_\mathbf{W} = -\mathbf{R}^T\mathbf{t}. \tag{3.17}$$

In this thesis, we denote the relative transformations between coordinate systems by subscripts similar to [38, 144], e.g. $\mathbf{T}_{ij}$, where camera j is the origin and i is the target. $\mathbf{T}_{ij}$ can be also be interpreted in the way that it transforms a 3D point from camera j to camera i (see Fig. 3.2). The transformation $\mathbf{T}_{ji}$ from camera i to j is the inverse of $\mathbf{T}_{ij}$

$$\mathbf{T}_{ji} = \mathbf{T}_{ij}^{-1}. \tag{3.18}$$

All relative transformations $\mathbf{T}_{ji}$ between any two cameras i and j can be computed from their respective camera to world transformations $\mathbf{T}_{wi}$ and $\mathbf{T}_{wj}$, which we refer to as **camera poses** throughout this thesis:

$$\mathbf{T}_{ji} = \mathbf{T}_{wj}^{-1}\mathbf{T}_{wi} = \mathbf{T}_{jw}\mathbf{T}_{wi}. \tag{3.19}$$

In this notation, the transformation matrices must be written in a way such that their subscripts align, e.g. in (3.19) the two ws representing the world coordinate system are next to each other. (3.15) already shows how to transform a 3D point from one coordinate system into another but in practice, it is often necessary to reproject from one image plane to the other directly:

$$\tilde{\mathbf{p}}' = \underbrace{\mathbf{K}[\mathbf{R}|\mathbf{t}]}_{\text{Projection Matrix}} \tilde{\mathbf{P}}, \tag{3.20}$$

where $[\mathbf{R}|\mathbf{t}]$ are referred to as external or extrinsic parameters as opposed to the intrinsic parameters in $\mathbf{K}$. Both are typically jointly expressed as projection matrix. The complete reprojection from one image plane to another (see Fig. 3.2) can be written as:

$$\tilde{\mathbf{p}}' = \mathbf{K}(\mathbf{R}\mathbf{K}^{-1}\tilde{\mathbf{p}}Z + \mathbf{t}) = \mathbf{K}\mathbf{R}\mathbf{K}^{-1}\tilde{\mathbf{p}}Z + \mathbf{K}\mathbf{t}. \tag{3.21}$$

or in terms of the inverse depth $\rho = \frac{1}{Z}$ [21]:

$$\tilde{\mathbf{p}}' = \mathbf{K}(\mathbf{R}\mathbf{K}^{-1}\tilde{\mathbf{p}} + \mathbf{t}\rho) = \mathbf{K}\mathbf{R}\mathbf{K}^{-1}\tilde{\mathbf{p}} + \mathbf{K}\mathbf{t}\rho, \tag{3.22}$$

followed by a dehomogenization to compute the coordinates in the image plane (3.6). For ease of notation, we define the full warping function τ to reproject from one image plane to another under a transformation $\mathbf{T}$:

$$\mathbf{p}' = \tau(\mathbf{T}, \mathbf{p}, Z) = \pi(\mathbf{R}\pi^{-1}(\mathbf{p}, Z) + \mathbf{t}), \tag{3.23}$$

where π and π^{-1} are defined as in (3.7) and (3.4). Figure 3.2 illustrates the connection between π, π^{-1} and τ by the example of three 3D points and their respective reprojections.

Since a 3×3 rotation matrix has nine entries but only three Degrees of Freedom (DOF) it is over-parameterized, which makes optimization difficult and computationally more expensive than necessary. This has lead to alternative representations with fewer parameters such as Euler angles and unit quaternions [27]. We will discuss these rotation representations in the following sections and then continue with the representation of the complete six *DOF* rigid body motion in the Lie algebra.

3.2.1 Euler Angles

A rotation from one orthogonal coordinate system to another can be described with three successive rotations parameterized by the Euler angles (θ, ϕ, φ). Each one of the three angles describes a rotation about a single coordinate axis. While there are as many as 12 rotation orderings [27], we explain it by the example of the popular ZYZ, Y, or 323 convention (see Fig. 3.3), which transforms a coordinate system xyz via the following sequence of rotations:

- Rotate around the z-axis by an angle φ to x', y', z'.

- Rotate around the y'-axis by an angle ϕ to x'', y'', z''.
- Rotate around the z''-axis by an angle θ.

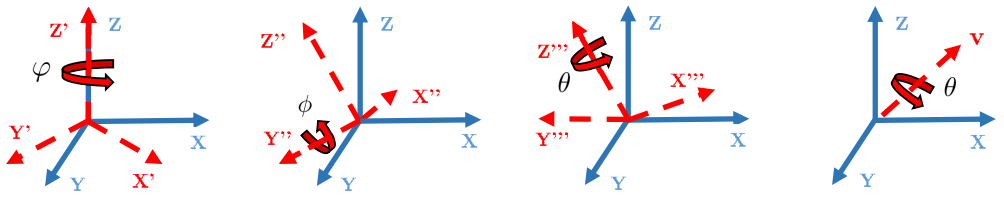
Multiplying the individual rotation matrices results in the complete rotation matrix $\mathbf{R} \in SO(3)$:

$$\begin{aligned} \mathbf{R} &= \mathbf{R}_{\varphi,z}\mathbf{R}_{\phi,y'}\mathbf{R}_{\theta,z''} \\ \mathbf{R} &= \begin{pmatrix} c_\varphi c_\phi c_\theta - s_\varphi s_\theta & -c_\varphi c_\phi s_\theta - s_\varphi c_\theta & c_\varphi s_\phi \\ s_\varphi c_\phi c_\theta - c_\varphi s_\theta & -s_\varphi c_\phi c_\theta + c_\varphi c_\theta & s_\varphi s_\phi \\ -s_\phi c_\theta & s_\phi s_\theta & c_\phi \end{pmatrix}, \end{aligned} \tag{3.24}$$

where $s_\varphi = sin(\varphi)$ and $c_\varphi = cos(\varphi)$ and analogously for the angles θ and ϕ. From $\mathbf{R}$ the angles (θ, ϕ, φ) can be computed by the inverse mapping:

$$\begin{pmatrix} \theta \\ \phi \\ \varphi \end{pmatrix} = \begin{pmatrix} \arctan\left(\frac{r_{12}}{r_{02}}\right) \\ \arctan\left(\frac{\sqrt{r_{02}^2 + r_{12}}}{r_{22}}\right) \\ \arctan\left(-\frac{r_{21}}{r_{20}}\right)) \end{pmatrix}, \tag{3.25}$$

with r_{ij} being the matrix elements of the i-th row and j-th column. One of the fundamental problems of Euler angles is the gimbal lock, a state where one of the rotations does not have any effect, e.g. angles $\phi = 0$ or $\phi = k\pi$ induce a double rotation around the z-axis This problem can only be addressed by different representations such as axis-angle or quaternions.



(a) Euler angle representation (ZYZ or Y-convention) **(b)** Axis-Angle

Figure 3.3: (a) The Euler angle representation defines three successive rotations around one of the main coordinate axes. (b) The axis-angle representation defines the rotation as an axis given by a unit vector $\mathbf{v}$ and a rotation angle θ around it.

3.2.2 Unit Quaternions

Quaternions are a popular rotation representation due to their simplicity and mathematical elegance, while not suffering from singularities like the Euler angle representation (see Sec. 3.2.1). In this section, we give a rough overview over the most important properties of quaternions and refer to [27, 57, 130] for more details. A quaternion $q \in \mathbb{H}$ is defined

as:

$$\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}, \quad q_i \in \mathbb{R}. \tag{3.26}$$

Typically, more compact notations as 4D vector or as a pair consisting of the real part q_0 and the imaginary components (q_1, q_2, q_3) are used:

$$\mathbf{q} = [q_0, q_1, q_2, q_3]^T = [q_0, (q_1, q_2, q_3)]^T. \tag{3.27}$$

For a quaternion it must hold that $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$. Its adjoint $\mathbf{q}^*$, norm $||\mathbf{q}||$ and inverse $\mathbf{q}^{-1}$ are:

$$\mathbf{q}^* = [-q_0, q_1, q_2, q_3]^T, \tag{3.28}$$

$$||\mathbf{q}|| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}, \tag{3.29}$$

$$\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{||\mathbf{q}||^2}. \tag{3.30}$$

In case of a unit quaternion, i.e. $||\mathbf{q}|| = ||\mathbf{q}^*|| = 1$, the corresponding inverse simplifies to $\mathbf{q} = \mathbf{q}^*$. A common interpretation of a unit quaternion is as rotation by an angle θ around an axis defined by a vector $\mathbf{v} = (v_x, v_y, v_z)^T \propto (q_1, q_2, q_3)$: The four components of the quaternion and θ and $\mathbf{v}$ are related as follows:

$$q_0 = \cos\left(\frac{\theta}{2}\right), \; q_1 = \sin\left(\frac{\theta}{2}\right)v_x, \; q_2 = \sin\left(\frac{\theta}{2}\right)v_y, \; q_3 = \sin\left(\frac{\theta}{2}\right)v_z. \tag{3.31}$$

If $\mathbf{q}$ is a unit quaternion, the rotation matrix $\mathbf{R}$ can be computed as:

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1q_2 - q_0q_3) & 2(q_0q_2 + q_1q_3) \\ 2(q_1q_2 + q_0q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_0q_1 + q_2q_3) & 1 - 2(q_1^2 + q_2^2) \end{bmatrix}. \tag{3.32}$$

Note that if $\mathbf{q}$ is not a unit quaternion, the inhomogeneous expression (3.32) is no longer an orthogonal matrix and the homogeneous expression (C.15) has to be used. For the derivation of the inhomogeneous from the homogeneous expression, we refer the reader to Section C.2.6. The rotation matrix $\mathbf{R}$ can be converted to a quaternion $\mathbf{q}$ as described in [130]:

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} \sqrt{1 + R_{00} + R_{11} + R_{22}} \\ sgn(r_{21} - r_{12})\sqrt{1 + r_{00} - r_{11} - r_{22}} \\ sgn(r_{02} - r_{20})\sqrt{1 - r_{00} + r_{11} - r_{22}} \\ sgn(r_{10} - r_{01})\sqrt{1 - r_{00} - r_{11} + r_{22}} \end{bmatrix}. \tag{3.33}$$

(3.33) only holds up to a certain threshold and other formulations have to be used [57, 130].

Two rotations given as unit quaternions $\mathbf{r}_{ji} = r_0 + r_1 i + r_2 j + r_3 k$ and $\mathbf{q}_{ik} = q_0 + q_1 i +$

$q_2 j + q_3 k$ are combined by multiplication:

$$\begin{aligned}\mathbf{t}_{jk} =& \mathbf{r}_{ji}\mathbf{q}_{ik} \\ =& (r_0 + r_1 i + r_2 j + r_3 k)(q_0 + q_1 i + q_2 j + q_3 k) \\ =& (r_0 q_0 - r_1 q_1 - r_2 q_2 - r_3 q_3) + i(r_0 q_1 + r_1 q_0 + r_2 q_3 - r_3 q_2) \\ &+ j(r_0 q_2 - r_1 q_3 - r_2 q_0 - r_3 q_1) + k(r_0 q_3 + r_1 q_2 - r_2 q_1 + r_3 q_0).\end{aligned} \tag{3.34}$$

A 3D vector $\mathbf{v}$ in camera i can also be written as a quaternion $\mathbf{q}_v$ without a real part:

$$\mathbf{q}_v = 0 + xi + yj + zk. \tag{3.35}$$

$\mathbf{q}_v$ can be rotated from camera i to j with the rotation given as unit quaternion $\mathbf{q}_{ji}$ [111]:

$$\begin{aligned}\mathbf{q}'_v =& \mathbf{q}_{ji}\mathbf{q}_v\mathbf{q}^*_{ji} \\ =& (q_0 + q_1 i + q_2 j + q_3 k)(xi + yj + zk)(q_0 - q_1 i - q_2 j - q_3 k).\end{aligned} \tag{3.36}$$

The complete expansion is given in [111] and Appendix C.2.7. Compared to rotation matrices, more operations are needed to rotate a vector by a unit quaternion. However, all these operations are performed on real numbers and no costly trigonometric functions are involved.

Apart from not suffering from singularities, e.g. the gimbal lock, and their simplicity, quaternions have several other advantages over rotation matrices. With quaternions it is easy to sample and interpolate in the space of rotations in a systematic way, e.g. uniform sampling distance, and it is possible to find closed-form solutions to some problems since the unit length condition can be applied through re-normalization. In contrast, due to their non-linear conditions, such problems are challenging to solve with rotation matrices. Further, less multiplications are required when multiplying two quaternions compared to two rotation matrices but more operations are need to rotate a vector by a unit quaternion. We refer to [70] for a broader discussion about advantages of quaternions.

3.2.3 Lie Group for 3D Rigid Body Motions

Since the rotational part $\mathbf{R} \in SO(3)$ of the transformation $\mathbf{T}$ is a 3×3 matrix with 9 entries but only has 3 *DOF*, it is over-parameterized. Reducing the 9 + 3 parameters of a transformation matrix to only 6 parameters for the 6 *DOFs* is essential for numerical optimization methods. In this section, we first introduce the Lie-algebra for orthogonal rotation matrices $\mathfrak{so}(3)$ and then continue with the Lie-algebra $\mathfrak{se}(3)$ for the minimal representation of 3D rigid body motions. The group $SO(3)$ has an associated Lie algebra $\mathfrak{so}(3)$ defined by three skew-symmetric matrices $\mathbf{G}_1^{\mathfrak{so}(3)}$,$\mathbf{G}_2^{\mathfrak{so}(3)}$ and $\mathbf{G}_3^{\mathfrak{so}(3)}$ corresponding to infinitesimal rotations along each axis. The infinitesimal step along an axis is the derivative

of (3.12) with respect to the angles ϕ, θ and φ evaluated at 0:

$$\begin{aligned}
\mathbf{G}_1^{\mathfrak{so}(3)} &= [\mathbf{e}_1]_\times = \frac{\mathbf{R}_\phi}{\partial\phi}\bigg|_{\phi=0} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \ \mathbf{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \\
\mathbf{G}_2^{\mathfrak{so}(3)} &= [\mathbf{e}_2]_\times = \frac{\mathbf{R}_\theta}{\partial\theta}\bigg|_{\theta=0} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \ \mathbf{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\
\mathbf{G}_2^{\mathfrak{so}(3)} &= [\mathbf{e}_2]_\times = \frac{\mathbf{R}_\varphi}{\partial\varphi}\bigg|_{\varphi=0} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \ \mathbf{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}
\end{aligned} \tag{3.37}$$

The operator $[.]_\times$ generates a skew-symmetric matrix from a vector and is defined in (C.7). Similarly, the group $SE(3)$ has an associated Lie algebra $\mathfrak{se}(3)$ formed by six 4×4 generator matrices $\mathbf{G}_i,\ i = 1..6$. $\mathbf{G}_1, \mathbf{G}_2$ and $\mathbf{G}_3$ correspond to an infinitesimal rotation and $\mathbf{G}_4, \mathbf{G}_5$ and $\mathbf{G}_6$ correspond to an infinitesimal translation along each axis. $\mathbf{G}_i, \forall\ i = 1..6$ can be computed by differentiating with respect to the individual components and evaluating at 0. $\mathbf{G}_{1,2,3}^{\mathfrak{se}(3)}$ are simply 4×4 versions of (3.37):

$$\mathbf{G}_{1,2,3}^{\mathfrak{se}(3)} = \begin{pmatrix} \mathbf{G}_{1,2,3}^{\mathfrak{so}(3)} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{pmatrix}, \tag{3.38}$$

and $\mathbf{G}_4$,$\mathbf{G}_5$ and $\mathbf{G}_6$ are defined as:

$$\begin{aligned}
\mathbf{G}_4 &= \frac{\partial\mathbf{T}_x}{\partial t_x}\bigg|_{t_x=0} = \begin{pmatrix} \mathbf{0}_3 & \mathbf{e}_1 \\ \mathbf{0} & 0 \end{pmatrix} \\
\mathbf{G}_5 &= \frac{\partial\mathbf{T}_y}{\partial t_y}\bigg|_{t_y=0} = \begin{pmatrix} \mathbf{0}_3 & \mathbf{e}_2 \\ \mathbf{0} & 0 \end{pmatrix} \\
\mathbf{G}_6 &= \frac{\partial\mathbf{T}_z}{\partial t_z}\bigg|_{t_z=0} = \begin{pmatrix} \mathbf{0}_3 & \mathbf{e}_3 \\ \mathbf{0} & 0 \end{pmatrix}
\end{aligned} \tag{3.39}$$

An arbitrary element in $\mathfrak{se}(3)$ comprises six coordinates represented as a vector in $\mathbb{R}^6$, where each coordinate multiplies with a generator matrix to form the 4×4 matrix $\mathbf{M}$:

$$\begin{aligned}
\mathbf{M}(\boldsymbol{\xi}) &= \sum_{i=1}^{6} \mathbf{G}_i\boldsymbol{\xi}_i = \mathbf{G}_1\omega_1 + \mathbf{G}_2\omega_2 + \mathbf{G}_3\omega_3 + \mathbf{G}_4\nu_1 + \mathbf{G}_5\nu_2 + \mathbf{G}_6\nu_3 \\
&= \begin{pmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{\nu} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -\omega_3 & \omega_2 & \nu_1 \\ \omega_3 & 0 & -\omega_1 & \nu_2 \\ -\omega_2 & \omega_1 & 0 & \nu_3 \\ 0 & 0 & 0 & 0 \end{pmatrix}.
\end{aligned} \tag{3.40}$$

Thus, $\boldsymbol{\xi} \in \mathfrak{se}(3)$ is the minimal representation as twist coordinates of the associated Lie-algebra comprising two separate 3-vectors, the angular velocity $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^T \in \mathfrak{so}(3)$ and the linear velocity $\boldsymbol{\nu} = (\nu_1, \nu_2, \nu_3)^T$:

$$\boldsymbol{\xi} = \begin{pmatrix} \boldsymbol{\omega} \\ \boldsymbol{\nu} \end{pmatrix} = (\omega_1, \omega_2, \omega_3, \nu_1, \nu_2, \nu_3)^T. \tag{3.41}$$

The mapping $\mathfrak{se}(3) \mapsto SE(3)$ from $\boldsymbol{\xi}$ to a transformation matrix $T(\boldsymbol{\xi})$ is then given by the matrix exponential:

$$\begin{aligned} exp : \mathfrak{se}(3) &\mapsto SE(3) \\ \boldsymbol{\xi} &\mapsto \mathbf{T}(\boldsymbol{\xi}) \end{aligned} \tag{3.42}$$

$$T(\boldsymbol{\xi}) = e^{\mathbf{M}(\boldsymbol{\xi})} = \begin{pmatrix} e^{[\boldsymbol{\omega}]_\times} & \mathbf{V}\boldsymbol{\nu} \\ \mathbf{0}^T & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix}. \tag{3.43}$$

$e^{[\omega]_\times}$ is equivalent to the axis-angle representation and can be computed by the Rodrigues' formula [148]:

$$e^{[\omega]_\times} = \mathbf{R} = \mathbf{I}_3 + \frac{sin(||\boldsymbol{\omega}||)}{||\boldsymbol{\omega}||}[\boldsymbol{\omega}]_\times + \frac{1 - cos(||\boldsymbol{\omega}||)}{||\boldsymbol{\omega}||^2}[\omega]_\times^2, \tag{3.44}$$

Since $\theta = ||\omega||$, $\frac{1}{\theta}[v]_\times$ is a normalization, we can simplify (3.44) to:

$$\mathbf{R} = \mathbf{I}_3 + \sin(\theta)[\omega]_\times + (1 - \cos\theta)[\omega]_\times^2, \tag{3.45}$$

where $\theta = ||\omega||$ and ω being a unit vector. For the rotational part there also exists a direct mapping to a unit quaternion [59]:

$$e_q^{\boldsymbol{\omega}} = \begin{cases} (1, 0, 0, 0)^T, & if \;\; \boldsymbol{\omega} = (0, 0, 0)^T \\ \left(cos\frac{||\boldsymbol{\omega}||}{2}, \frac{sin(||\boldsymbol{\omega}||)\boldsymbol{\omega}}{2||\boldsymbol{\omega}||} \right) & otherwise \end{cases} \tag{3.46}$$

The matrix $\mathbf{V}$ of the translational part $\mathbf{t} = \mathbf{V}\nu$ is given as:

$$\mathbf{V} = \mathbf{I}_3 + \frac{1 - cos(||\boldsymbol{\omega}||)}{||\boldsymbol{\omega}||^2}[\boldsymbol{\omega}]_\times + \frac{||\boldsymbol{\omega}|| - sin(||\boldsymbol{\omega}||)}{||\boldsymbol{\omega}||^3}[\boldsymbol{\omega}]_\times^2. \tag{3.47}$$

These derivations of the exponential map are based on different power series and described in great detail in [10, 32, 148] The logarithm map is the inverse of the exponential map (3.42) and maps from $SE(3)$ to $\mathfrak{se}(3)$:

$$\begin{aligned} log : SE(3) &\mapsto \mathfrak{se}(3) \\ \mathbf{T}(\boldsymbol{\xi}) &\mapsto \boldsymbol{\xi} \end{aligned} \tag{3.48}$$

The angular velocity $\boldsymbol{\omega}$ can be computed from the rotation matrix $\mathbf{R}$ as:

$$\begin{aligned} \boldsymbol{\omega} = [\log(\mathbf{R})]_\vee &= \frac{\theta}{2\sin(\theta)} \begin{pmatrix} r_{21} - r_{12} \\ r_{02} - r_{20} \\ r_{10} - r_{01} \end{pmatrix} \\ \theta &= \arccos\left(\frac{1}{2}(trace(\mathbf{R}) - 1)\right), \\ \log(\mathbf{R}) &= \frac{\theta}{2\sin(\theta)}(\mathbf{R} - \mathbf{R}^T) \end{aligned} \tag{3.49}$$

,

and linear velocity $\boldsymbol{\nu}$ ca the translation $\mathbf{t}$ as:

$$\begin{aligned} \boldsymbol{\nu} =& \mathbf{V}^{-1}\mathbf{t} \\ \mathbf{V}^{-1} =& \mathbf{I}_3 - \frac{1}{2}[\boldsymbol{\omega}]_\times + \frac{1}{\theta^2}\left(1 - \frac{\theta\cos(\frac{\theta}{2})}{2\sin(\frac{\theta}{2})}\right)[\boldsymbol{\omega}]_\times^2 \end{aligned} \tag{3.50}$$

The operator $[.]_\vee$ is the inverse of $[.]_\times$ and defined in (C.8). Finally, we define an operator $\circ$ to concatenate poses analogously to multiplying the transformation matrices (3.19):

$$\boldsymbol{\xi}_{ij} = \boldsymbol{\xi}_{iw} \circ \boldsymbol{\xi}_{wj} = \boldsymbol{\xi}_{wi}^{-1} \circ \boldsymbol{\xi}_{wj} = \log\left(\exp(\boldsymbol{\xi}_{iw})\exp(\boldsymbol{\xi}_{wj})\right), \tag{3.51}$$

and also extend the full warping function τ presented in (3.23) to the Lie Group:

$$\mathbf{p}' = \tau(\boldsymbol{\xi}_{ij}, \mathbf{p}, Z). \tag{3.52}$$

This section introduced all the mathematical tools necessary to describe transformations in 3D space. Starting with the pinhole camera model, we showed how to compute the 3D representation of a point in its respective camera. To handle multiple cameras, we defined a 3D rigid body motion to transform points in 3D space. Finally, we discussed various rotation representations such as axis-angle and quaternions and derived the Lie Algebra in 3D space, which is essential for camera pose estimation.

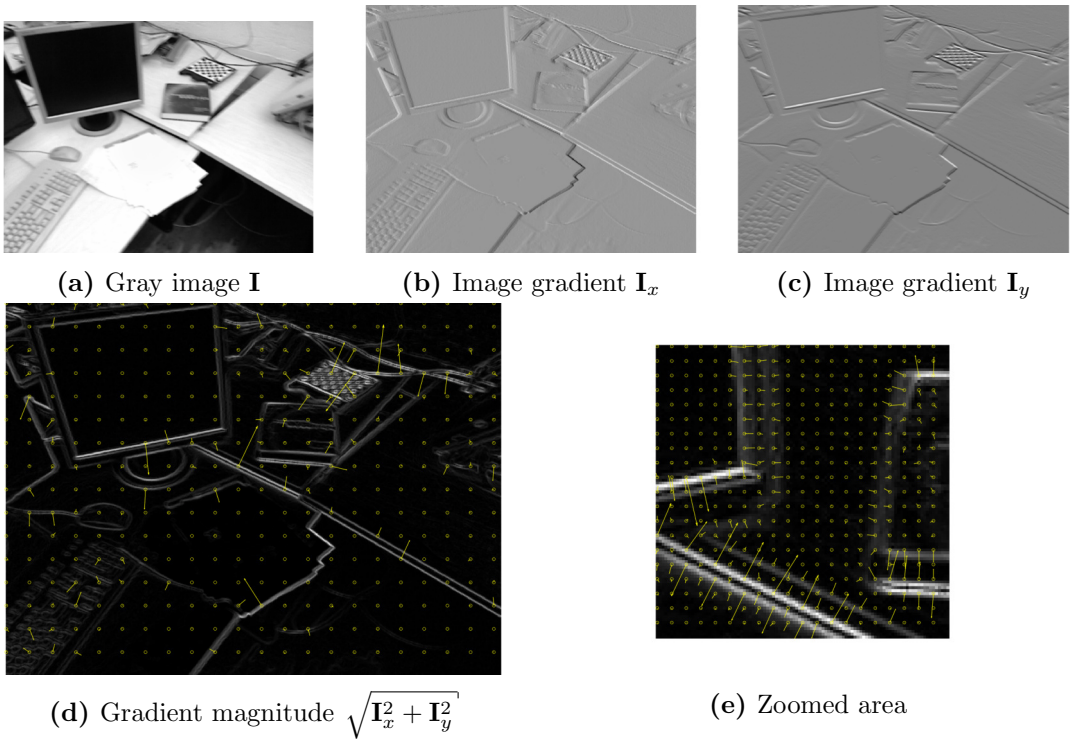
3.3 Direct Image Alignment

Probably the most intuitive way to estimate the relative motion between two cameras is to align their images directly. The basic idea of direct image alignment is that the intensity of a 3D point observed in an image $\mathbf{I}_0$ is the same as in $\mathbf{I}_1$, i.e. intensities are consistent between $\mathbf{I}_0$ and $\mathbf{I}_1$, which also coined the term photo-consistency-based methods. The two images are aligned, when their intensity differences are minimal or in other terms, their photo-consistency is maximized. The goal is to find a relative motion $\boldsymbol{\xi}_{01}$ between two

images, which minimizes the photometric error E_{photo}:

$$E_{photo} = \sum_i (\mathbf{I}_0(\mathbf{p}') - \mathbf{I}_1(\mathbf{p}))^2 = \sum_i r_{i,photo}^2, \tag{3.53}$$

where $\mathbf{p}' = \tau(\boldsymbol{\xi}_{01}, \mathbf{p}, Z)$, Z is the depth at $\mathbf{p}$ and $r_{i,photo}$ is the photometric residual. This cost function can be directly evaluated on the intensity image $\mathbf{I}$. Nearly all state-of-the-art methods minimize (3.53) with an iterative optimization method, which requires the intensity gradients of $\mathbf{I}$ in x- and y-direction, respectively $\mathbf{I}_x$ and $\mathbf{I}_y$, to compute the step δ in each iteration. The optimization problem is essentially the same as for edge-based methods and will be discussed in Section 4.1.1 in more detail. Figures 3.4a, 3.4b and 3.4c show an intensity image with its gradients, where it can be seen that in texture-less regions the gradients are very close to zero. With the gradients being close to zero, the residuals in those areas carry no valuable information for the minimization problem and can even reduce accuracy and convergence speed. Thus, only selecting areas with high information content such as high-gradient regions [23, 38, 40] or sparse points [37] instead of just the dense image [80, 81] increases both accuracy and convergence speed of the optimization. There are several shortcomings of photo-consistency-based direct methods



(a) Gray image $\mathbf{I}$ **(b)** Image gradient $\mathbf{I}_x$ **(c)** Image gradient $\mathbf{I}_y$

(d) Gradient magnitude $\sqrt{\mathbf{I}_x^2 + \mathbf{I}_y^2}$ **(e)** Zoomed area

Figure 3.4: (a) Gray-scale image $\mathbf{I}$ and the respective gradients in (b) x- and (c) y-directions computed as central differences from the TUM RGBD dataset [147]. (d) Gradient magnitude $\sqrt{\mathbf{I}_x^2 + \mathbf{I}_y^2}$ of (a) with gradients sampled over the image and a (e) zoomed part.

to be aware of. One major practical challenge is that due to illumination changes, the photo-consistency assumption is often not fulfilled or only valid in parts of the scene To mitigate this problem, [23, 37, 38] increase the model parameters in the optimization by additional brightness terms. Further, motion blur can smooth textured regions thereby reducing the number of pixels carrying valuable information. Another critical point is the initialization of photo-consistency-based methods since the image gradients only represent local information (see Fig. 3.4d). When starting too far away from the minimum, the algorithms often converge to a local minimum or do not converge at all. Thus, the convergence basin of such methods is small and the coarse-to-fine scheme applied in most approaches only partially addresses this problem as shown in [89, 133].

3.4 Robust Parameter Estimation

Most computer vision problems can be formulated as minimization of an energy function E over a set of parameters $\mathbf{x}$ to find the optimal values $\mathbf{x}^*$:

$$\mathbf{x}^* = \arg\min E(\mathbf{x}). \tag{3.54}$$

Such an energy minimization is at the core of all the methods presented in this thesis as well as of problems like camera calibration, *VO*, *SLAM* or Bundle Adjustment (BA). For example, *VO* estimates the unknown 6 *DOF* camera motion between two frames by minimizing an error measure such as the photo-consistency or reprojection error. These energy functions typically take a non-linear least-squares form:

$$E(\mathbf{x}) = \sum_i r_i(\mathbf{x})^2, \quad r_i(\mathbf{x}) = \hat{y}_i - f(\mathbf{x}, y_i) \tag{3.55}$$

where $\mathbf{x}$ are the parameters to be estimated and r_i is a non-linear function, which computes the difference between observations $\hat{\mathbf{y}}_\mathbf{i}$ and a model function $f(\mathbf{x}, \mathbf{y_i})$. In the literature, the residual functions r_i are often stacked in a residual vector $\mathbf{r}$ to write E conveniently as:

$$E(\mathbf{x}) = ||\mathbf{r}(\mathbf{x})||_2^2 = \mathbf{r}(\mathbf{x})^T \mathbf{r}(\mathbf{x}). \tag{3.56}$$

Most state-of-the-art *SLAM* and *VO* systems solve such problems in an iterative fashion. Sections 3.4.1 and 3.4.2 introduce the widely used iterative Gauss-Newton (GN) and Levenberg-Marquardt (LM) methods. Since all practical applications suffer from noise and outliers, it is necessary to weight individual residuals differently. Section 3.4.3 describes the Iteratively Reweighted Least-Squares (IRLS) method to tackle these problems and shows different weighting and influence functions.

Connection to Maximum-Likelihood: There is a connection between the maximum likelihood and the least-squares formulation [121]. In many practical cases, we can as-

sume a Gaussian distribution with constant standard deviation and zero-mean for the discrepancy ϵ_i between model and observations:

$$\epsilon_i = \hat{\mathbf{y}}_\mathbf{i} - f(\mathbf{x}, \mathbf{y_i}) \sim \mathcal{N}(0, \sigma^2) \tag{3.57}$$

The maximum likelihood is given as:

$$L(\mathbf{y}|\mathbf{x}, \sigma) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\epsilon_i^2}{2\sigma^2}\right) = (2\pi\sigma^2)^{-N/2} \exp\left(-\frac{\sum_{i=1}^{N}\epsilon_i^2}{2\sigma^2}\right) \tag{3.58}$$

Since the ln is a strictly increasing function, (3.58) can be simplified as log-likelihood:

$$\begin{aligned} l(\mathbf{y}|\mathbf{x}, \sigma) &= ln(L(\mathbf{y}|\mathbf{x}, \sigma)) \\ &= -\frac{N}{2}\ln(2\pi\sigma^2) - \frac{\sum_{i=1}^{N}(\hat{\mathbf{y}}_\mathbf{i} - f(\mathbf{x}, \mathbf{y_i}))^2}{2\sigma^2}. \end{aligned} \tag{3.59}$$

Maximizing (3.59) yields the optimal estimate $\mathbf{x}_{ML}$:

$$\begin{aligned} \mathbf{x}_{ML} &= \arg\max_{\mathbf{x}} l(\mathbf{y}|\mathbf{x}, \sigma) \\ &= \arg\max_{\mathbf{x}} -\left(\frac{N}{2}\ln(2\pi\sigma^2) + \frac{\sum_{i=1}^{N}(\hat{\mathbf{y}}_\mathbf{i} - f(\mathbf{x}, \mathbf{y_i}))^2}{2\sigma^2}\right) \end{aligned} \tag{3.60}$$

Maximizing a negative function is the same as minimizing a positive function, thus changing arg max to arg min and dropping the constants $\frac{N}{2}\ln(2\pi\sigma^2)$ and $\frac{1}{2\sigma^2}$ leads to:

$$\begin{aligned} \mathbf{x}_{ML} &= \arg\min_{\mathbf{x}} \left(\frac{N}{2}\ln(2\pi\sigma^2) + \frac{\sum_{i=1}^{N}(\hat{\mathbf{y}}_\mathbf{i} - f(\mathbf{x}, \mathbf{y_i}))^2}{2\sigma^2}\right) \\ &= \arg\min_{\mathbf{x}} \sum_{i=1}^{N}(\hat{\mathbf{y}}_\mathbf{i} - f(\mathbf{x}, \mathbf{y_i}))^2 \end{aligned} \tag{3.61}$$

Since the measurement errors are independent and follow a Gaussian distribution with constant standard deviation, the least-squares formulation (3.55) is equal to the maximum likelihood estimation of the parameters.

3.4.1 The Gauss-Newton Method

The *GN* method is a well-known algorithm to minimize non-linear least squares problems by iteratively solving quadratic approximations of the energy function E [104, 121]. It is derived by linearly approximating the residual function r with a first-order Taylor expansion around $\mathbf{x}_0$:

$$r(\mathbf{x}) \approx r(\mathbf{x}_0) + \mathbf{J}_r(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0), \tag{3.62}$$

or using $\mathbf{x} = \mathbf{x}_0 + \delta$, (3.62) can be written as:

$$r(\mathbf{x}_0 + \boldsymbol{\delta}) \approx r(\mathbf{x}_0) + \mathbf{J}_r(\mathbf{x}_0)\boldsymbol{\delta}. \tag{3.63}$$

The Jacobian $\mathbf{J}_r(\mathbf{x}_0)$ is given as:

$$\mathbf{J}_r(\mathbf{x}_0) = \frac{\partial r(\mathbf{x})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}_0} = \frac{\partial r(\mathbf{x}_0 + \boldsymbol{\delta})}{\partial \boldsymbol{\delta}}\bigg|_{\boldsymbol{\delta}=\mathbf{0}} \tag{3.64}$$

From the linearized residual (3.63), the *GN* method iteratively solves a quadratic approximation to an energy function (3.55):

$$E(\mathbf{x}_0 + \boldsymbol{\delta}) \approx ||r(\mathbf{x}_0) + \mathbf{J}_r\boldsymbol{\delta}||_2^2 = r(\mathbf{x}_0)^T r(\mathbf{x}_0) + 2\boldsymbol{\delta}^T \underbrace{\mathbf{J}_r^T r(\mathbf{x}_0)}_{\mathbf{b}} + \boldsymbol{\delta}^T \underbrace{\mathbf{J}_r^T \mathbf{J}_r}_{\mathbf{H}_{GN}} \boldsymbol{\delta} \tag{3.65}$$

In contrast to Newton's method, where the full Hessian (3.67) is required, $\mathbf{H}_{GN}$ is an approximation of the Hessian by the outer product of the Jacobians $\mathbf{H_{GN}} \approx 2\mathbf{J_r^T J_r}$. This approximation can also be easily derived from the full Hessian by assuming the residual is linear. Then the second derivative $r(\mathbf{x})''^T$ in (3.67) is equal to $\mathbf{0}$ and can be dropped.

$$\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial r(\mathbf{x})^T r(\mathbf{x})}{\partial \mathbf{x}} = 2r(\mathbf{x})'^T r(\mathbf{x}) = 2\mathbf{J_r^T} r(\mathbf{x}) \tag{3.66}$$

$$\frac{\partial E(\mathbf{x})^2}{\partial^2 \mathbf{x}} = r(\mathbf{x})''^T r(\mathbf{x}) + 2r(\mathbf{x})'^T r(\mathbf{x})' = r(\mathbf{x})''^T r(\mathbf{x}) + 2\mathbf{J_r^T J_r} \tag{3.67}$$

The next step is to find the increment $\boldsymbol{\delta}$, which minimizes (3.65). Taking the first derivative with respect to the increment $\boldsymbol{\delta}$ of (3.65) and setting it equal to $\mathbf{0}$ yields:

$$\begin{aligned} \mathbf{0} &= 2\mathbf{b} + 2\mathbf{H}_{GN}\boldsymbol{\delta}, \\ -\mathbf{b} &= \mathbf{H}_{GN}\boldsymbol{\delta}. \end{aligned} \tag{3.68}$$

The update $\boldsymbol{\delta}$ is then given as:

$$\boldsymbol{\delta} = -\mathbf{H}^{-1}\mathbf{b} = -(\mathbf{J}_r^T\mathbf{J}_r)^{-1}\mathbf{J}_r^T\mathbf{r}(\mathbf{x}_0) \tag{3.69}$$

The estimate $\mathbf{x}_{k+1}$ at iteration $k+1$ can be computed from the previous estimate $\mathbf{x}_k$ via an update-step:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\delta}, \tag{3.70}$$

until convergence, e.g. $\boldsymbol{\delta}$ is below a threshold.

In practice, it is often required to weight the residuals differently to cope with outliers or noise. This can be achieved by introducing a weight factor to (3.55),:

$$E(\mathbf{x}) = \sum_i w_i r_i(\mathbf{x})^2 = ||\mathbf{r}(\mathbf{x})||_W^2, \tag{3.71}$$

where w_i is a constant, non-negative weight and $\mathbf{W}$ the respective diagonal weight matrix, which can be directly incorporated into the updates:

$$\mathbf{H} = \mathbf{J_r^T W J_r}, \qquad \mathbf{b} = \mathbf{J_r^T W r}(\mathbf{x}_0) \tag{3.72}$$

The performance of the *GN* algorithm strongly depends on its initialization. While it offers quadratic convergence close to the minimum, it can become unstable when the initialization is far away from the minimum. In such cases, gradient descent is more reliable since it guarantees a decrease of the cost function.

3.4.2 The Levenberg-Marquardt method

The *LM* method [93, 104, 107, 121] introduces a non-negative dampening factor λ to the *GN* method (3.69) to interpolate between *GN* and gradient descent algorithm:

$$\boldsymbol{\delta} = -(\mathbf{J_r^T J_r} + \lambda \mathbf{I})^{-1} \mathbf{J_r^T r}(\mathbf{x}), \tag{3.73}$$

where $\mathbf{I}$ is the identity matrix. One *LM* iteration solves (3.73) repeatedly with different values for λ until the cost function decreases. If it is close to the minimum, λ is set to a small value such that the algorithm is closer to *GN*. In contrast, if the cost function does not decrease, λ is increased to bring the algorithm closer towards gradient descent. However, if λ is very large, inverting $(\mathbf{J_r^T J_r} + \lambda \mathbf{I})^{-1}$ approaches $\mathbf{I}\frac{1}{\lambda}$ and the Hessian is ignored. To still benefit from the Hessian in such cases, $\mathbf{I}$ is replaced by the diagonal entries of $\mathbf{H}$ $diag(\mathbf{J_r^T J_r})$:

$$\boldsymbol{\delta} = -(\mathbf{J_r^T J_r} + \lambda \operatorname{diag}(\mathbf{J_r^T J_r}))^{-1} \mathbf{J_r^T r}(\mathbf{x}). \tag{3.74}$$

The diagonal entries scale along the curvature to move further in directions with smaller gradients.

3.4.3 Iteratively Reweighted Linear Least Squares

As shown in Section 3.4, a least-squares formulation assumes a Gaussian distribution of the residuals. In nearly all practical cases, the residuals are not Gaussian distributed and there also exist outliers with large residual values. These outliers have a significant influence on the accuracy, convergence basin and speed of the optimization. To approximate other distributions and reduce the influence of outliers, the energy is re-written as M-estimator with a robust error norm ρ:

$$E(\mathbf{x}) = \sum_i \rho(r_i(\mathbf{x})), \tag{3.75}$$

M-Estimator	Robust norm $\rho(r)$	Influence $\varphi(r)$	Weight $w(r)$
L_2	$\frac{1}{2}r^2$	r	1
Huber $\begin{cases} \lvert r\rvert < \theta_H \\ \lvert r\rvert \geq \theta_H \end{cases}$	$\begin{cases} \frac{r^2}{2} \\ \theta_H(\lvert r\rvert - \frac{k}{2}) \end{cases}$	$\begin{cases} r \\ \theta_H \ sign(r) \end{cases}$	$\begin{cases} 1 \\ \frac{\theta_H}{\lvert r\rvert} \end{cases}$
Tukey $\begin{cases} \lvert r\rvert < \theta_T \\ \lvert r\rvert \geq \theta_T \end{cases}$	$\begin{cases} \frac{\theta_T^2}{6}(1-(1-(\frac{r}{\theta_T})^2)^3) \\ \frac{\theta_T^2}{6} \end{cases}$	$\begin{cases} r(1-(\frac{r}{\theta_T})^2 \\ 0 \end{cases}$	$\begin{cases} (1-(\frac{r}{\theta_T})^2 \\ 0 \end{cases}$
Cauchy	$\frac{\theta_C^2}{2} log(1+(\frac{r}{\theta_C})^2)$	$\frac{r}{1+(\frac{r}{\theta_C})^2}$	$\frac{1}{1+(\frac{r}{\theta_C})^2}$

Table 3.2: Various robust error norms $\rho(r)$, influence functions $\varphi(r)$, and weight functions $w(r)$ for a scalar residual r with θ_H, θ_T and θ_C being constants.

where r_i is the residual and $\mathbf{x}$ are the estimated parameters. The minimization requires to find a zero-crossing of the derivative:

$$0 = \frac{\partial E(\mathbf{x})}{\partial \mathbf{x}} = \sum_i \frac{\partial \rho(r_i)}{\partial r_i} \frac{\partial r_i(\mathbf{x})}{\partial \mathbf{x}}, \tag{3.76}$$

with an influence function $\varphi(r_i)$:

$$\varphi(r_i) = \frac{\partial \rho(r_i)}{\partial r_i}. \tag{3.77}$$

Plugging (3.77) into (3.75) and expanding by r_i, we can write:

$$0 = \sum_i w(r_i) r_i \frac{\partial r_i(\mathbf{x})}{\partial \mathbf{x}} = \sum_i \frac{\varphi(r_i)}{r_i} r_i \frac{\partial r_i(\mathbf{x})}{\partial \mathbf{x}} = \sum_i \varphi(r_i) \frac{\partial r_i(\mathbf{x})}{\partial \mathbf{x}} \tag{3.78}$$

with the weight function $w(r_i)$:

$$w(r_i) = \frac{\varphi(r_i)}{r_i}. \tag{3.79}$$

Table 3.2 and Figure 3.5 show an overview of different norms, influence, and weight functions. Writing (3.75) in the classical least-squares formulation with a residual dependent weight w yields:

$$E_{IRLS}(x) = \frac{1}{2} \sum_i w(r_i(\mathbf{x}_k)) r_i(\mathbf{x})^2, \tag{3.80}$$

where $\mathbf{x}_k$ is considered constant and set to the most recent estimate of $\mathbf{x}$. Thus, the weight changes in every iteration instead o f being fixed as in (3.71).

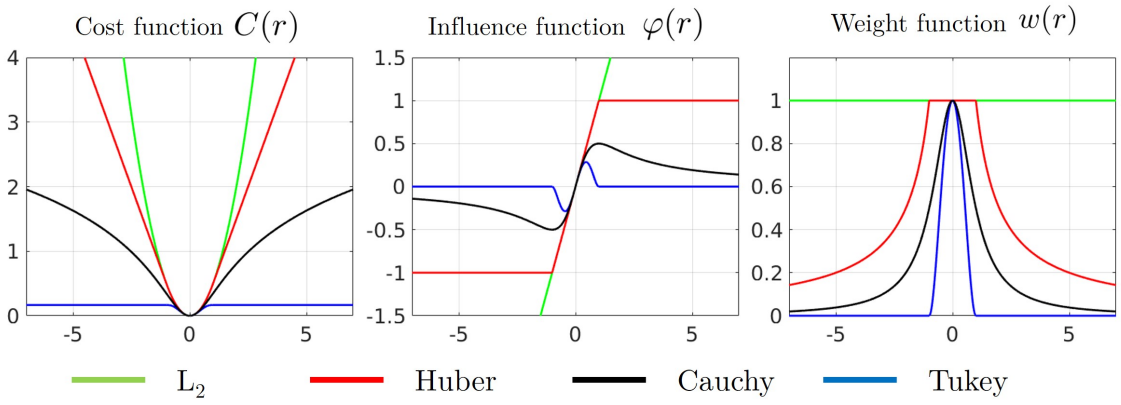


Figure 3.5: Visualizations of various robust error norms ρ, influence functions φ and weight functions $w(r)$.

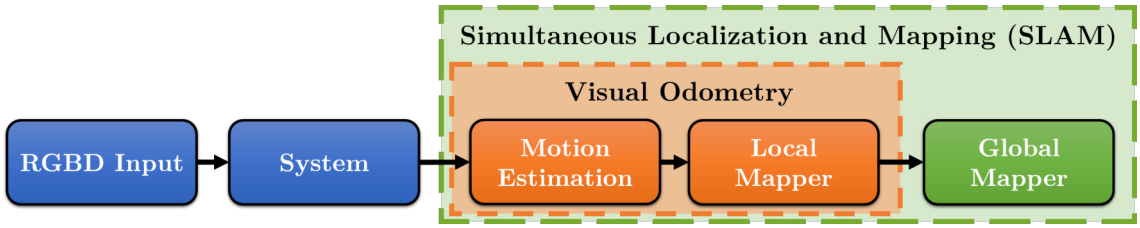
3.5 Conclusion

This chapter introduced the mathematical tools and notations required to understand the algorithms in this thesis. Further, the discussed fundamentals and concepts are essential to understand state-of-the-art literature and modern *SLAM* systems. Starting from the basic pinhole camera model, we described how to compute, transform and reproject 3D points in multi-camera systems. Then we defined a 3D rigid body motion with its associated Lie algebra and discussed various rotation representations. The final part gives an overview of parameter estimation techniques and shows how to solve problems in the presence of outliers and noise.

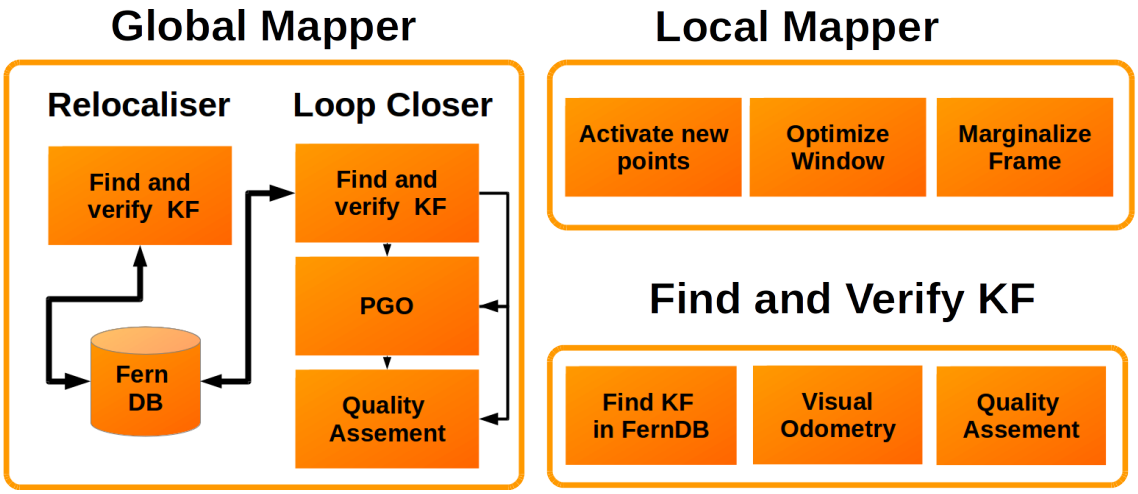
4 Edge-based Simultaneous Localization and Mapping

Contents

In this chapter, we will present the Visual Odometry (VO) and Simultaneous Localization and Mapping (SLAM) algorithms developed and published over the past years [132–134]. The main contribution of this thesis is RESLAM, a fully edge-based *SLAM* system [134] and this chapter is divided according to the components of the *SLAM* system as shown in Figure 4.1a. Section 4.1.1 builds the foundation of this chapter by introducing the edge-based camera motion estimation as an energy minimization problem as proposed in [133] and demonstrating how to solve it efficiently. To deal with challenging scenes in scarcely textured environments, Section 4.1.3 shows how to incorporate and minimize an additional geometric error term in an Iterative Closest Point (ICP) point-to-plane formulation [132]. One of the key improvements of [132, 134] compared to [133] is an additional local refinement in the form of a *Local Mapper* covered by Section 4.4.2. While in [132] the *Local Mapper* only refines the pose of the most recent Keyframe (KF), in [134] it refines all the involved model parameters such as world poses of the *KFs*, the depth of the individual edges and the intrinsic camera parameters. The *Global Mapper* is the last and probably most crucial part of the RESLAM as it builds a globally consistent map. Section 4.5 describes the *Global Mapper* and how its place-recognition module works, which is essential for loop closure and relocalization. Figure 4.1a depicts an overview of RESLAM and visualizes the connection between *VO* and *SLAM*. The individual components of the *Local Mapper* and *Global Mapper* and their respective interactions are shown in Figure 4.1b.



(a) SLAM system with its key components



(b) Individual components of the *Local Mapper* and the *Global Mapper*.

Figure 4.1: (a) Overview of the complete SLAM system comprising three main modules: (1) Motion Estimation, (2) Local Mapper and (3) Global Mapper. Motion Estimation and Local Mapper can be considered as part of *VO*, while *SLAM* requires an additional Global Mapper. (b) The Local Mapper and the Global Mapper with their respective components. Since loop closure and relocalization are both a place-recognition problem, they access the same database of Fern descriptors.

4.1 Camera Motion Estimation

The problem at the core of all *VO* and *SLAM* systems is the estimation of the camera motion. Figure 4.3 shows the typical setup of *KF*-based motion estimation, which aligns the current frame with a *KF* to estimate a rotation $\mathbf{R}$ and a translation $\mathbf{t}$. Since $\mathbf{R}$ and $\mathbf{t}$ only have 6 Degrees of Freedom (DOF), a minimal representation as twist-coordinates $\boldsymbol{\xi}$ is used (see Sec. 3.2.3). The estimation of the motion $\boldsymbol{\xi}_{ij}$ between two cameras i and j can be formulated as an energy minimization problem:

$$\boldsymbol{\xi}_{ij}^{*} = \arg\min_{\boldsymbol{\xi}_{ij}} E, \tag{4.1}$$

where E is an arbitrary energy term. Most direct methods address this problem either with a photo-consistency-based [23, 37, 38], a geometric [78, 79, 117] or a combined approach [80, 81, 156, 159].

However, since the earliest days of computer vision, edges have been recognized as important visual cues in images. They have many favorable qualities such as robustness to illumination changes compared to raw intensity values and a certain invariance to scale and orientation. While early edge-based *SLAM* systems treat and process edges similar to features [35, 85, 142], a new direct edge alignment paradigm has emerged.

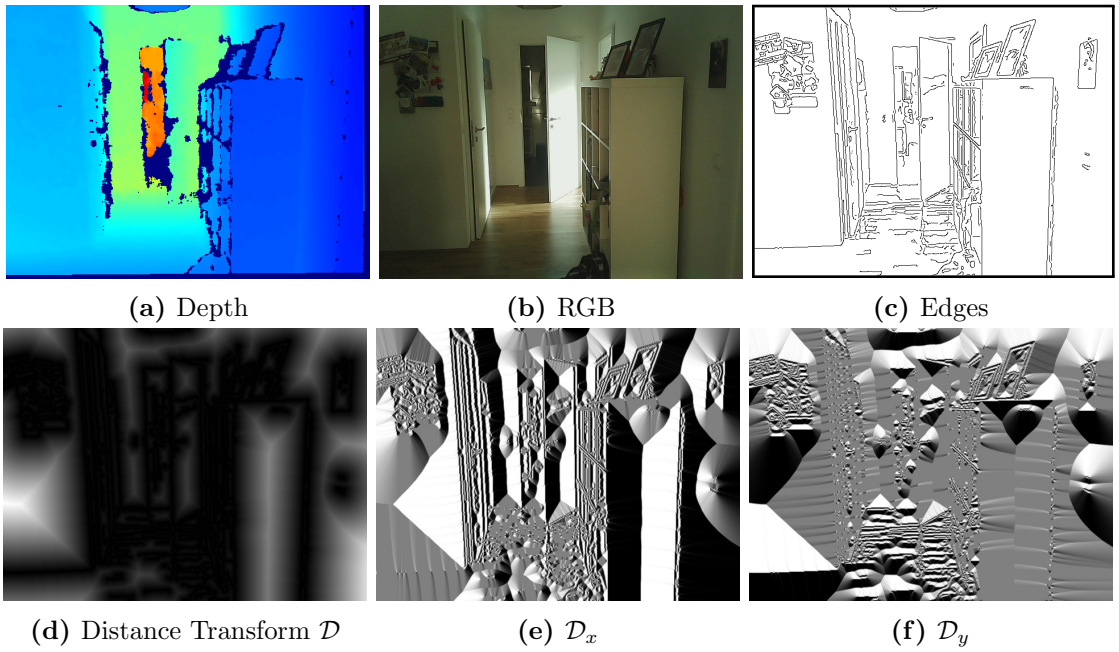
In this section, we will extensively discuss this new paradigm in Section 4.1.1 and later show how to combine it with a geometric error in an *ICP* point-to-plane formulation (see Sec. 4.1.3). We further demonstrate how to efficiently solve the edge-alignment problem and provide all the mathematical derivations necessary to implement both approaches.

4.1.1 Direct Edge-based Camera Motion Estimation

The input to the system is a sequence of RGBD frames as depicted in Figure 4.2, where a frame at time t is denoted as $\mathcal{F}_t$ and comprises an RGB image $\mathbf{I}_t$, a depth map $\mathbf{Z}_t$ and an edge detection $\mathbf{E}_t$. The edges in $\mathbf{E}_t$ are detected with an arbitrary edge detector denoted as *edge*:

$$\mathbf{E}_t = edge(\mathbf{I}_t), \tag{4.2}$$

where $\mathbf{I}_t$ is either an RGB or intensity image depending on the type of edge detector.



(a) Depth **(b)** RGB **(c)** Edges

(d) Distance Transform $\mathcal{D}$ **(e)** $\mathcal{D}_x$ **(f)** $\mathcal{D}_y$

Figure 4.2: (a,b) RGBD input recorded with an Orbbec Astra Pro, (c) the extracted edges , (d) the computed Distance Transform (DT), (e,f) gradients in x- and y-direction the $\mathcal{D}_x$ and $\mathcal{D}_y$.

We want to estimate the minimal representation of the relative camera motion $\boldsymbol{\xi}_{ij}$ between the edge detections $\mathbf{E}_i$ and $\mathbf{E}_j$ of two cameras i and j. Figure 4.3 shows this estimation, where the *KF* is in camera i and the current frame in camera j. Each frame

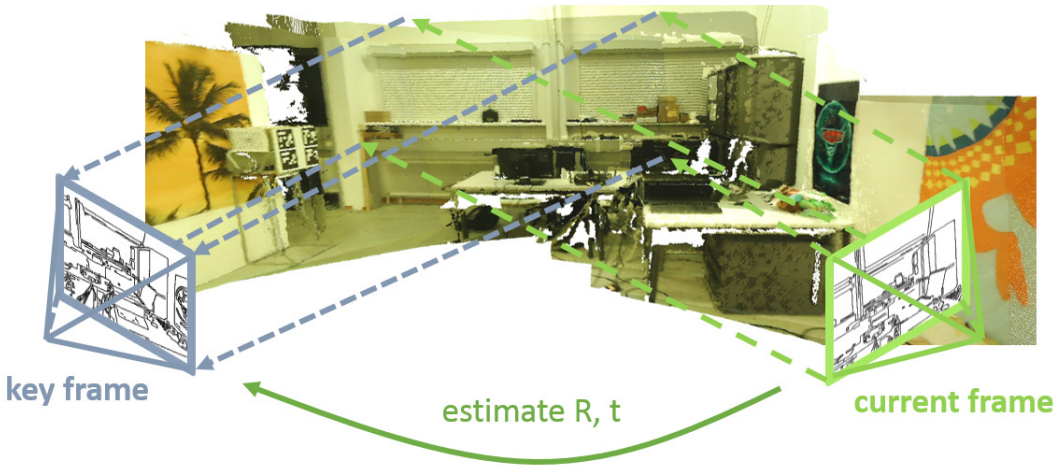


Figure 4.3: Estimating the camera motion between the current frame and the last *KF* is based on aligning their respective edge detections to compute a relative rotation $\mathbf{R}$ and translation $\mathbf{t}$ representing a 6 *DOF* rigid body motion.

$\mathcal{F}$ contains a set of edges $\mathcal{E}_t$ with a valid depth in $\mathbf{Z}_t$. Aligning edges corresponds to minimizing a distance function $d(.)$ between an edge $\mathbf{p}_j$ detected in $\mathbf{E}_j$, which is reprojected to $\mathbf{E}_i$ as $\mathbf{p}'_j = \tau(\boldsymbol{\xi}_{ij}, \mathbf{p}_j, \mathbf{Z}_j(\mathbf{p}_j))$, and the closest edge in $\mathbf{E}_i$:

$$E_{edge} = \sum_{\mathbf{p}_j \in \mathcal{E}_j} d(n(\mathbf{p}'_j), \mathbf{p}'_j)^2. \tag{4.3}$$

The function $n(.)$ returns the nearest neighbor in $\mathbf{E}_i$ depending on $\mathbf{p}'_j$ and is defined as:

$$n(\mathbf{p}'_j) = \arg\min_{\mathbf{p}_i \in \mathbf{E}_i} ||\mathbf{p}_i - \mathbf{p}'_j||. \tag{4.4}$$

Note that (4.3) is closely related to the reprojection error (2.1) computed in indirect methods [36, 84, 116], but instead of feature points, it uses individual edge points. Even though there are some indirect edge-based approaches that compute edge correspondences [33, 35, 85, 105], we focus on a direct formulation without correspondences. This direct formulation poses the challenge of finding the distance to the closest edge efficiently. A naive approach solves (4.4) by an exhaustive search through all the edge detections and keeping track of the minimum distance. This corresponds to a complexity of $O(N)$ to find a neighbor for one reprojected $\mathbf{p}'$ and $O(NM)$ to minimize (4.3), where N and M are the number of edges in $\mathbf{E}_i$ and $\mathbf{E}_j$, respectively. Another option is to search in image space for the nearest-neighbor in a window $w \times w$ around p' [76]. This reduces the complexity of one nearest neighbor search to $O(c)$ and to $O(cN)$ for the whole algorithm, where $c = w^2$ and $c << N$. Note that, while c is a constant, with reasonable search window size this still greatly affects the speed of the algorithm.

Since the edges are detected once and do not change afterward, there is a faster option

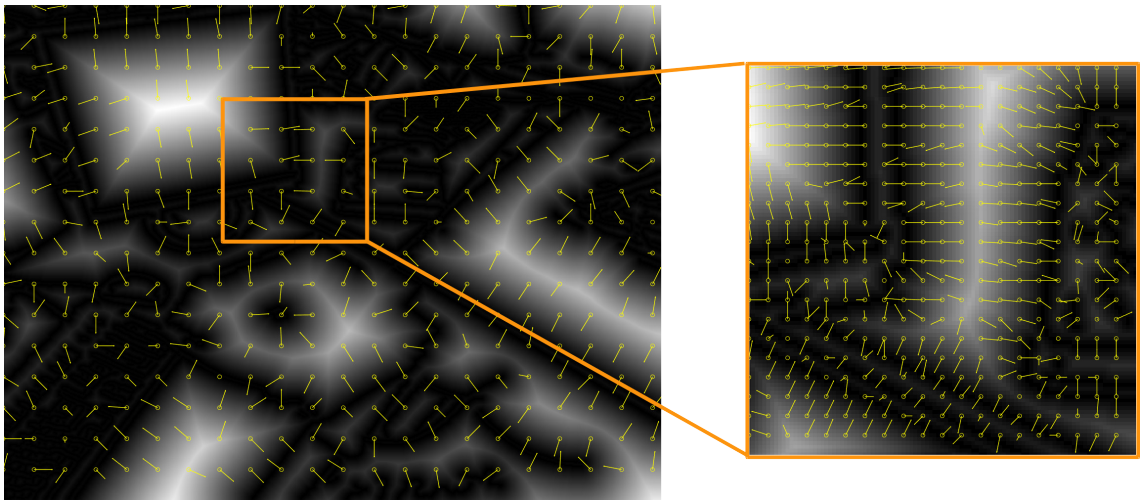


Figure 4.4: Distance transform of the edge detections from Figure 3.4a and the gradient vectors given by the derivatives of the DT in x- and y-direction and computed as in (4.17). It is clearly visible that the gradient vectors point towards the closest edge even in untextured areas as opposed to the photometric gradients presented in Figure 3.4.

to implicitly assign the nearest neighbor. We can compute a representation called DT [42, 125] containing at each pixel position the distance to the closest edge. Figures 4.2c and 4.4 show the detected edges and the DT, where bright values represent a large distance and bright values a small distance to an edge. With the DT of an edge detection $\mathbf{E}$ denoted as $\mathbf{D}$, we can reformulate (4.3) as:

$$\boldsymbol{\xi}_{ij}^{*} = \arg\min_{\boldsymbol{\xi}_{ij}} E_{edge}, \tag{4.5}$$

with the energy E_{edge} defined as:

$$\begin{aligned} E_{edge} &= \sum_{\mathbf{p}\in\mathcal{E}_j} \Theta_H r_{edge}^2 \\ &= \sum_{\mathbf{p}\in\mathcal{E}_j} \Theta_H \mathbf{D}(\tau(\xi_{ij}, \mathbf{p}, Z_j(\mathbf{p})))^2. \end{aligned} \tag{4.6}$$

This reduces the complexity for a single evaluation to $O(1)$ and for the whole set of edges to $O(M)$. There are fast algorithms to compute the DT in $O(N)$ such as the two-pass algorithm by Felzenszwalb and Huttenlocher [44]. Further, the KF-based setup has the great benefit that the costly DT computation has to be performed only once, whenever a new KF is created. As depicted in Figure 4.5, the DT is not smooth due to the spatial discretization, which could cause problems during the optimization. We address this by bi-linearly sampling the residual value r_{edge} in (4.6) from the DT at the typically non-integer position of $\mathbf{p}'$. Since the DT contains positive values, it provides only the magnitude of the residual but no sign or direction like the photometric residual. Intending to provide

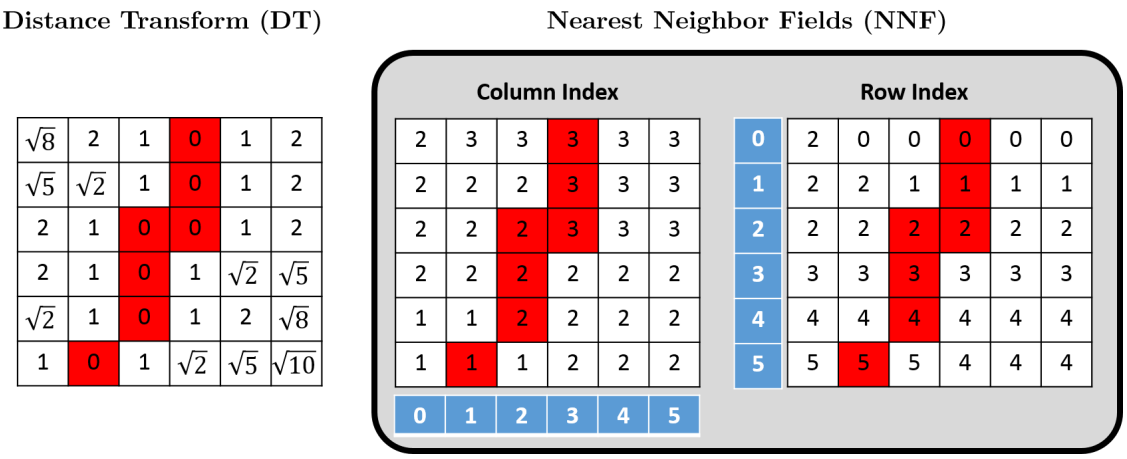


Figure 4.5: (left) Distance Transform of a 6×6 image for an edge (red) in comparison to (right) a Nearest Neighbor Field represented as column and row index as proposed in [88, 165].

	Single Edge	All Edges	Pre-Computation
Image	O(N)	O(NM)	-
Window	O(c)	O(cN)	-
DT	O(1)	O(M)	O(MN)
NNF	O(1)	O(M)	O(MN)

Table 4.1: The computational complexity for the nearest neighbor search for a single edge reprojection (*Single Edge*) and for all the edge reprojections (*All Edge*). A naive approach performs an exhaustive search in the whole image or within a window c, while the DT and NNF pre-compute the distance to the closest edge at each position and only require a constant lookup time.

a signed residual, Zhou et al. [164, 165] proposed Nearest Neighbor Fields (NNF) [164] and Oriented NNF (ONNF) [165]. The main difference to the DT is that instead of storing the distance to the closest edge, the NNF store the row and column indices of the closest edge as depicted in Figure 4.5. This makes NNFs identical to the reprojection error since there are x- and y- coordinates and not just a single distance. ONNF contain 8 different NNF, where each contains only edges falling into a certain range of angles. The angle of an individual edge point is computed from the image gradients. According to Zhou et al. [165], this increases the convergence basin but ONNF are computationally more expensive since they require the computation of eight individual NNF instead of just one. Table 4.1 shows the complexity for different edge-alignment approaches, where *Single Edge* refers to searching for the nearest neighbor of a single reprojected edge and *All Edges* for the whole set of edges. In our edge-based optimizations, we solve this problem with a Levenberg-Marquardt (LM) algorithm in a coarse-to-fine scheme. Thus, the computational complexity is a critical point since all the edge reprojections have to be evaluated for different λ values repeatedly in each iteration and on each scale level.

4.1.2 Edge-based Pose Estimation in SE(3)

The edge-based energy minimization presented in (4.5) aims to find the relative 6 *DOF* motion $\boldsymbol{\xi}_{ij}^{*}$ that minimizes E_{edge}. Our algorithms solve (4.5) with an iteratively-reweighted *LM* algorithm (see Sec. 3.4.2). This section discusses several formulations of the image- and edge-alignment and presents the derivation of the *LM* algorithm for $SE(3)$ including the update steps and the Jacobian computations.

In the literature [6, 87], there are three main formulations for the image alignment problem: *Forward Additive*, *Forward Compositional* and *Inverse Compositional*. While [6, 87] study these formulations for image alignment:

$$r_i^{photo} = \mathbf{I}_{kf}(\mathbf{p}_i') - \mathbf{I}(\mathbf{p}_i), \tag{4.7}$$

we can convert it to a similar form for edge-based case by aligning two distant transforms:

$$r_i^{edge} = \mathbf{D}_{kf}(\mathbf{p}_i') - \mathbf{D}(\mathbf{p}_i). \tag{4.8}$$

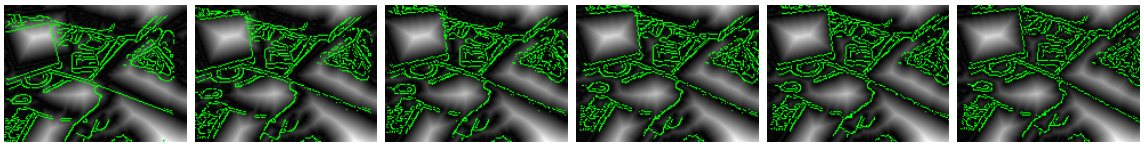
Since the set of edges $\mathcal{E}$ is selected from the current frame, i.e. $\mathbf{D}(\mathbf{p}_i) = 0,\ \forall\ \mathbf{p}_i \in \mathcal{E}$, r_i^{edge} simplifies to:

$$r_i^{edge} = \mathbf{D}_{kf}(\mathbf{p}_i'). \tag{4.9}$$

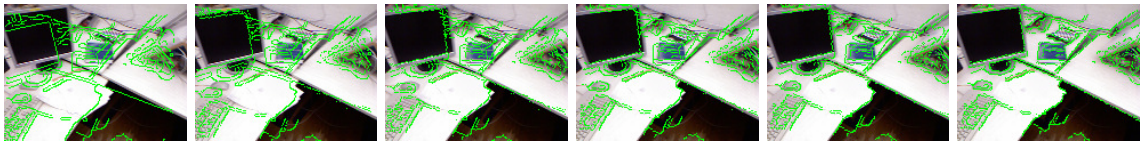
The main benefits of the *Forward Compositional* and *Inverse Compositional* approaches are that large parts of the Jacobian stay constant, avoiding the costly re-computation of the Jacobian in each iteration. However, the computationally most efficient *Inverse Compositional* formulation is not applicable since the gradients of the *DT* are computed in the frame, where the edges are selected, and therefore always equal to 0. In the *Forward Compositional* formulation, the Jacobians of the projection $\mathbf{J}_{\tau}$ have to computed only once for each edge in the *KF* and remain fixed afterward. Since the residuals are evaluated on $\mathbf{D}$ the part of the Jacobian related to the gradients, $\mathbf{J_I}$ for image alignment and $\mathbf{J_D}$ for edge alignment, change in every iteration. We instead rely on the popular *Forward Additive* approach also used by many state-of-the-art systems [37, 38, 80, 81] due to its simplicity. The main difference to the *Forward Compositional* is that all parts of the Jacobian are recomputed in each iteration. In our experiments, the complete recomputation only adds little overhead compared to the *Forward Additive*.

In the following, we will define the update rule for $SE(3)$ and then derive the Jacobians by the example of one edge residual r_i^{edge}, which is analogous to r_i^{photo} except for the gradients in the image plane. We rewrite the update step for the *LM* algorithm given in (3.69) in terms of the incremental in $\boldsymbol{\delta\xi} \in SE(3)$ as:

$$x^{k+1} = x^k + \boldsymbol{\delta} x \rightarrow \boldsymbol{\xi}^{k+1} = \boldsymbol{\xi}^k \boxplus \boldsymbol{\delta\xi}, \tag{4.10}$$



(a) Iteration 0 **(b)** Iteration 5 **(c)** Iteration 10 **(d)** Iteration 15 **(e)** Iteration 20 **(f)** Iteration 25



(g) Iteration 0 **(h)** Iteration 5 **(i)** Iteration 10 **(j)** Iteration 15 **(k)** Iteration 20 **(l)** Iteration 25

Figure 4.6: Visualization of edge alignment process over 25 iterations starting from an initial motion $\mathbf{T} = \mathbf{I}_4$. The edge energy E_{edge} is minimized on the DT (first row) with bright image parts corresponding to large residuals. The second row is just for visualization purposes and shows the progress on the RGB image.

where the $\boxplus$ operator is a manifold group operation in a left-compositional way:

$$\boldsymbol{\xi}^k \boxplus \boldsymbol{\delta\xi} \longrightarrow \exp(\boldsymbol{\delta\xi})\boldsymbol{\xi}^k. \tag{4.11}$$

Note that we only discuss the left-compositional formulation in this thesis since we use in our algorithms and refer to [10] for right-compositional formulation. Using (4.10), the residual $r(\boldsymbol{\xi}^{k+1})$ at iteration $k+1$ is defined as:

$$r(\boldsymbol{\xi}^{k+1}) = \mathbf{D}_{kf}(\tau(\mathbf{p}_i, \boldsymbol{\xi}^{k+1}, \mathbf{Z}(\mathbf{p}_i))) = \mathbf{D}_{kf}(\tau(\mathbf{p}_i, \boldsymbol{\xi}^k \boxplus \boldsymbol{\delta\xi}, \mathbf{Z}(\mathbf{p}_i))). \tag{4.12}$$

At each iteration $r(\xi^{k+1})$ is linearized around the last estimate $\boldsymbol{\xi}^k$ using the Taylor expansion similar to (3.62):

$$r(\boldsymbol{\xi}^{k+1}) \approx r(\boldsymbol{\xi}^{k+1})\big|_{\boldsymbol{\xi}=\boldsymbol{\xi}^k} = \underbrace{r(\boldsymbol{\xi}^k)}_{\mathbf{D}_{kf}(\tau(\mathbf{p}_i,\boldsymbol{\xi}^k,\mathbf{Z}(\mathbf{p}_i)))} + \mathbf{J}_i(\boldsymbol{\xi}^k)\boldsymbol{\delta\xi} \tag{4.13}$$

Figure 4.6 shows the edge-based motion estimation between two frames over 25 iterations with the DT and the RGB image (only for visualization purposes). Sections 3.4.2 and 3.4.1 already cover the optimization scheme, Thus, we will focus on the computation of the full Jacobian $\mathbf{J}_r$ for a single residual r. $\mathbf{J}_r$ is evaluated at the current estimate $\boldsymbol{\xi}^k$ and

can be split into four individual Jacobians as:

$$\begin{aligned}\mathbf{J}_r(\boldsymbol{\xi}^k) &= \frac{\partial r_{edge}}{\partial \boldsymbol{\xi}} \\ \mathbf{J}_r(\boldsymbol{\xi}^k) &= \mathbf{J_D}\mathbf{J}_\tau \\ &= \mathbf{J_D}\mathbf{J}_\pi\mathbf{J_T}\mathbf{J}_{\boldsymbol{\xi}} \\ &= \frac{\partial \mathbf{D}_2(\mathbf{p}')}{\partial \mathbf{p}'}\bigg|_{\mathbf{p}'=\pi(\mathbf{P}')} \frac{\partial \pi(\mathbf{P}')}{\partial \mathbf{P}'}\bigg|_{\mathbf{P}'=\mathbf{T}(\boldsymbol{\xi})\mathbf{P}} \frac{\partial \mathbf{T}(\boldsymbol{\xi})\mathbf{P}}{\mathbf{T}(\boldsymbol{\xi})}\bigg|_{\mathbf{P}=\mathbf{P}_i} \frac{\partial \mathbf{T}(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\bigg|_{\boldsymbol{\xi}=\boldsymbol{\xi}^k}.\end{aligned} \tag{4.14}$$

For ease of notation, the evaluation point is $\boldsymbol{\xi}^k$ is omitted from now on.

Jacobian $\mathbf{J}_D$ of the DT: The derivative in the image plane at a point $\mathbf{p}'$ is given by the gradients in x-/y-direction $\mathbf{D}_x$ and $\mathbf{D}_y$:

$$\mathbf{J}_D(\mathbf{p}') = [\mathbf{D}_x(\mathbf{p}), \mathbf{D}_y(\mathbf{p})]^T. \tag{4.15}$$

$\mathbf{D}_x$ and $\mathbf{D}_y$ are computed as the central differences of $\mathbf{D}$ in the respective direction (see Fig. 4.2e and 4.2f):

$$\mathbf{D}_x(\mathbf{p}') = \frac{1}{2}\left(\mathbf{D}(\mathbf{p}' + [1,0]^T) - \mathbf{D}(\mathbf{p}' - [1,0]^T)\right), \tag{4.16}$$

$$\mathbf{D}_y(\mathbf{p}') = \frac{1}{2}\left(\mathbf{D}(\mathbf{p}' + [0,1]^T) - \mathbf{D}(\mathbf{p}' - [0,1]^T)\right). \tag{4.17}$$

Typically, $\mathbf{p}'$ is not an integer coordinate and the exact gradients are computed by bilinear interpolation.

Jacobian $\mathbf{J}_\pi$ of the projection function π: The projection of the 3D point $\mathbf{P}'$ is defined as:

$$\mathbf{p}' = \pi(\mathbf{P}') = \left(\frac{f_x X'}{Z'} + c_x, \frac{f_y Y'}{Z'} + c_y\right)^T. \tag{4.18}$$

Its derivative with respect to $\mathbf{P}' = [X', Y', Z']^T$ is the Jacobian $\mathbf{J}_\pi$:

$$\mathbf{J}_\pi = \frac{\partial \pi(\mathbf{P}')}{\partial \mathbf{P}'} = \begin{pmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{pmatrix}. \tag{4.19}$$

Jacobian $\mathbf{J}_T$ of the transformation T: The derivative of the transformation $\mathbf{T}$ with respect to a 3D point $\mathbf{P} = [X, Y, Z]^T$ can be computed in terms of the Kronecker product

and the matrix derivative (see Sec. C.2.4 and C.2.5):

$$\begin{aligned}\mathbf{J}_T &= \left.\frac{\partial \mathbf{TP}}{\partial \mathbf{T}}\right|_{\mathbf{T}=T(\boldsymbol{\xi}^k),\mathbf{P}=\mathbf{P}_i} \\ &= \begin{pmatrix} X\mathbf{I}_3 & Y\mathbf{I}_3 & Z\mathbf{I}_3 & \mathbf{I}_3 \end{pmatrix} = \mathbf{P}^T \otimes \mathbf{I}_3 \\ &= \begin{pmatrix} X & 0 & 0 & Y & 0 & 0 & Z & 0 & 0 & 1 & 0 & 0 \\ 0 & X & 0 & 0 & Y & 0 & 0 & Z & 0 & 0 & 1 & 0 \\ 0 & 0 & X & 0 & 0 & Y & 0 & 0 & Z & 0 & 0 & 1 \end{pmatrix}\end{aligned} \tag{4.20}$$

Jacobian of the exponential map $\mathbf{J}_{\boldsymbol{\xi}}$: For infinitesimal motions, where $\xi_i << 1$, there is a linear approximation to the exponential map:

$$\exp([\boldsymbol{\xi}]_\times) = \mathbf{I}_4 + \sum_{i=1}^{6} \mathbf{G}_i \boldsymbol{\xi}_i = \mathbf{I}_4 + [\boldsymbol{\xi}]_\times \tag{4.21}$$

Since the last row is by definition always equal to $\mathbf{0}$, it is typically dropped. We can formulate the expression $vec([\boldsymbol{\xi}]_\times)$ in terms of a matrix-vector product as:

$$\exp([\boldsymbol{\xi}]_\times) = \mathbf{M}\boldsymbol{\xi} = \begin{pmatrix} \mathbf{0}_3 & -[\mathbf{e}_1]_\times \\ \mathbf{0}_3 & -[\mathbf{e}_2]_\times \\ \mathbf{0}_3 & -[\mathbf{e}_3]_\times \\ \mathbf{I}_3 & \mathbf{0}_3 \end{pmatrix} \boldsymbol{\xi}, \tag{4.22}$$

where $\mathbf{e}_1 = [1,0,0]^T$, $\mathbf{e}_2 = [0,1,0]^T$ and $\mathbf{e}_3 = [0,0,1]^T$. Using (4.21) and (4.22), the derivative of the exponential map linearized at the origin $\boldsymbol{\xi} = \mathbf{0}$ can be written as:

$$\left.\frac{\partial e^{[\boldsymbol{\xi}]_\times}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\mathbf{0}} = \frac{\partial \mathbf{I}_4 + [\boldsymbol{\xi}]_\times}{\partial \boldsymbol{\xi}} = \frac{\partial [\boldsymbol{\xi}]_\times}{\partial \boldsymbol{\xi}} = \frac{\partial \mathbf{M}\boldsymbol{\xi}}{\partial \boldsymbol{\xi}} = \mathbf{M} \tag{4.23}$$

To generalize to arbitrary transformations $\mathbf{T}(\boldsymbol{\xi})$ given $\boldsymbol{\xi} \neq \mathbf{0}$, it is linearized around a different point on the manifold.

$$\mathbf{J}_{\boldsymbol{\xi}} = \frac{\partial e^{[\boldsymbol{\delta\xi}]_\times} \mathbf{T}(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \tag{4.24}$$

Applying the chain-rule and using (C.12) and (4.23) yields:

$$\begin{aligned}\mathbf{J}_{\boldsymbol{\xi}} &= \underbrace{\frac{\partial e^{[\boldsymbol{\delta\xi}]_\times} \mathbf{T}(\boldsymbol{\xi})}{\partial e^{[\boldsymbol{\delta\xi}]_\times}}}_{\text{(C.12)}} \quad \underbrace{\left.\frac{\partial e^{[\boldsymbol{\delta\xi}]_\times}}{\partial \boldsymbol{\delta\xi}}\right|_{\boldsymbol{\delta\xi}=\mathbf{0}}}_{\text{(4.23)}} \\ &= \left(\mathbf{T}(\boldsymbol{\xi})^T \otimes \mathbf{I}_3\right) \mathbf{M}\end{aligned} \tag{4.25}$$

The final result is a 12×6 matrix:

$$\mathbf{J}_{\boldsymbol{\xi}} = \begin{pmatrix} \mathbf{0}_3 & -[\mathbf{r}_1]_\times \\ \mathbf{0}_3 & -[\mathbf{r}_2]_\times \\ \mathbf{0}_3 & -[\mathbf{r}_3]_\times \\ \mathbf{I}_3 & [\mathbf{t}]_\times \end{pmatrix} \tag{4.26}$$

The full Jacobian $\mathbf{J}_i(\boldsymbol{\xi})$: The full Jacobian for the edge-based error function (4.6) is then:

$$\begin{aligned} \mathbf{J}_r &= \mathbf{J}_\mathbf{D}\mathbf{J}_\pi\mathbf{J}_g\mathbf{J}_G \\ &= \begin{pmatrix} \mathbf{D}_x & \mathbf{D}_y \end{pmatrix} \begin{pmatrix} f_x\frac{1}{Z'} & 0 & -f_x\frac{X'}{Z'^2} & -f_x\frac{Y'X'}{Z'^2} & f_x\left(1+\frac{X^{2'}}{Z'^2}\right) & -f_x\frac{Y'}{Z'} \\ 0 & f_y\frac{1}{Z'} & -f_y\frac{Y'}{Z'^2} & -f_y\left(1+\frac{Y^{2'}}{Z'^2}\right) & f_y\frac{Y'X'}{Z'^2} & f_y\frac{X'}{Z'} \end{pmatrix} \end{aligned} \tag{4.27}$$

The final Jacobian is of dimension 6×1 and the Hessian $\mathbf{H} = \mathbf{J}_r^T\mathbf{J}_r$ is 6×6. Note that for the well-known reprojection error the direction in image space is computed directly from the residual but the edge-based residual is only an unsigned distance value. Thus, we rely on the Jacobian $\mathbf{J}_\mathbf{D}$ to compute the direction to the closest edge in image space as depicted in Figure 4.4.

The photometric case: The Jacobian for the photometric residual is the same as (4.27) except that the gradients are computed on the intensity image as $\mathbf{I}_{kf,x}$ and $\mathbf{I}_{kf,y}$ instead of the *DT* $\mathbf{D}$. Figure 3.4 shows an example intensity image and the corresponding gradients.

$$\begin{aligned} \mathbf{J}_r &= \mathbf{J}_\mathbf{I}\mathbf{J}_\pi\mathbf{J}_g\mathbf{J}_G \\ &= \begin{pmatrix} \mathbf{I}_x & \mathbf{I}_y \end{pmatrix} \begin{pmatrix} f_x\frac{1}{Z'} & 0 & -f_x\frac{X'}{Z'^2} & -f_x\frac{Y'X'}{Z'^2} & f_x\left(1+\frac{X^{2'}}{Z'^2}\right) & -f_x\frac{Y'}{Z'} \\ 0 & f_y\frac{1}{Z'} & -f_y\frac{Y'}{Z'^2} & -f_y\left(1+\frac{Y^{2'}}{Z'^2}\right) & f_y\frac{Y'X'}{Z'^2} & f_y\frac{X'}{Z'} \end{pmatrix} \end{aligned} \tag{4.28}$$

Motion Initialization A crucial point for the minimization of (4.5) is the initialization of the relative motion $\boldsymbol{\xi}_{ji}^0$. In the *KF*-based setting, we always estimate the motion from a current frame $\mathcal{F}_c$ to the *KF* $\mathcal{F}_{kf}$ (see Fig. 4.3 and 4.7a). Especially, when the cameras are farther apart, simply starting with zero motion $\boldsymbol{\xi}_{ji}^0 = [0,0,0,0,0,0]^T$ can result in slow convergence speeds, convergence to a local minimum or low accuracy. To start with an initialization close to the minimum, we use the already estimated motion of the current frame's predecessors $\mathcal{F}_{c-1}$ and $\mathcal{F}_{c-2}$ and sample several different initializations from various motion models (see Fig. 4.7b-4.7f). We compute the cost of each initialization on the *DT* (4.6) and choose the one with the lowest. Our purely *VO*-based approaches [132, 133]

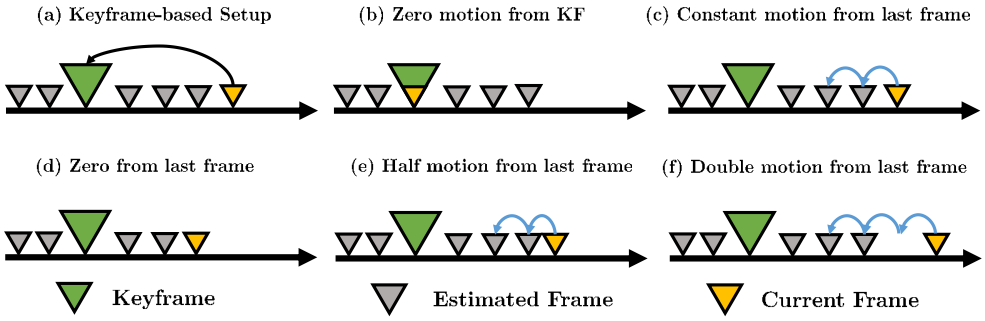


Figure 4.7: (a) shows the *KF*-based setup and (b) - (f) the various motion initializations. (b) zero motion from the last *KF* is often problematic and can cause slow convergence or convergence to a local minimum. Alternative initializations such as (c) constant motion, (d) no motion, (e) half motion and (f) double motion from the last frame address these issues.

only sample constant motion (see Fig. 4.7b):

$$\boldsymbol{\xi}^0_{kf,c} = \boldsymbol{\xi}_{kf,c-1} \circ \boldsymbol{\xi}_{c-2,c-1}, \tag{4.29}$$

where $\boldsymbol{\xi}_{c-2,c-1} = \boldsymbol{\xi}^{-1}_{kf,c-2} \circ \boldsymbol{\xi}_{kf,c-1}$, and no (zero) motion from the last *KF* (see Fig. 4.7(c)):

$$\boldsymbol{\xi}^0_{kf,c} = [0,0,0,0,0,0]^T. \tag{4.30}$$

These two initializations work well with smooth motions and high frame-rates but might slow down convergence speed when the initialization does not correspond to actual motion. Thus, RESLAM [134] considers three additional motion models (see Fig. 4.7d-f): (1) Zero motion from the last frame

$$\boldsymbol{\xi}^0_{kf,c} = \boldsymbol{\xi}_{kf,c-1}, \tag{4.31}$$

and (2) decelerated (half) motion:

$$\boldsymbol{\xi}^0_{kf,c} = \boldsymbol{\xi}_{kf,c-1} \circ \exp(0.5 \log \boldsymbol{\xi}_{c-2,c-1}), \tag{4.32}$$

(3) accelerated (double) motion

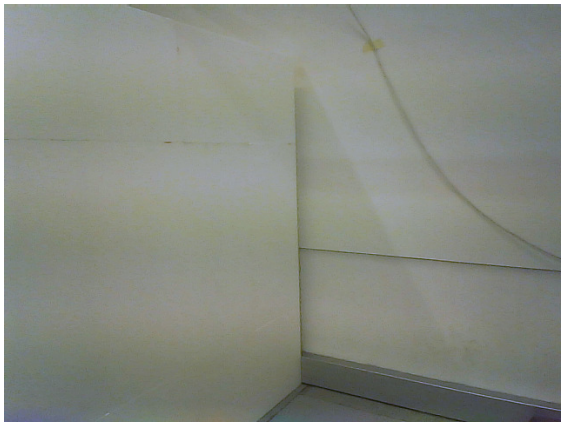
$$\boldsymbol{\xi}^0_{kf,c} = \boldsymbol{\xi}_{kf,c-1} \circ \boldsymbol{\xi}_{c-2,c-1} \circ \boldsymbol{\xi}_{c-2,c-1}. \tag{4.33}$$

These additional motion models used in RESLAM improve convergence speed, robustness and accuracy compared to REVO.

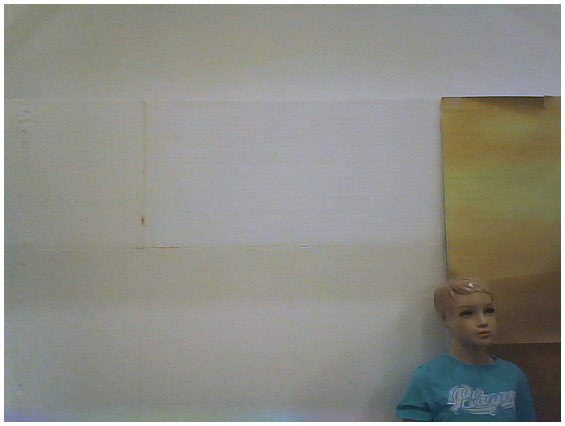
4.1.3 Edge- and ICP-based Relative Pose Estimation

Indoor scenes with texture- and structure-less surfaces like walls, floors or ceilings are one of the main error sources for common *VO* and *SLAM* methods. Figure 4.8 shows

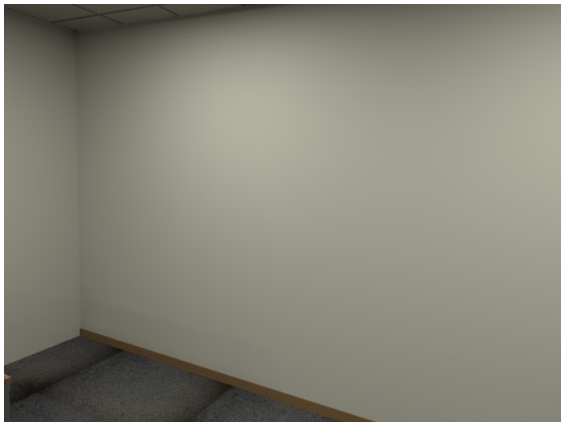
an excerpt of some challenging indoor sequences from our own recordings (see Fig. 4.8a and 4.8b) two public benchmark datasets [62, 147]. Further, illumination changes and flickering lights can cause additional problems as already discussed in Section 1.1.



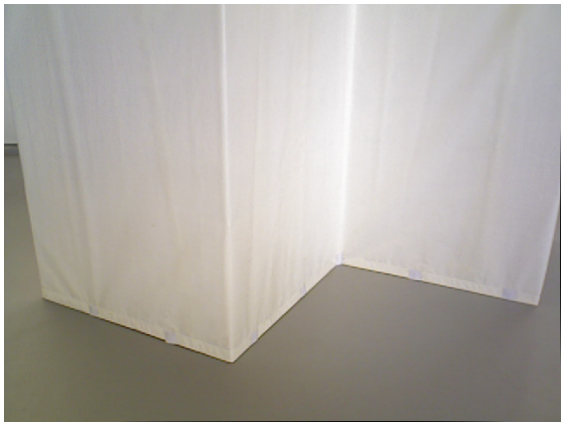
(a) Orbbec RGBD recording



(b) Orbbec RGBD recording



(c) Office scene [62].



(d) Texture-less office scene [147].

Figure 4.8: Typical indoor scene parts such as walls, ceilings and floors are scarcely textured and therefore challenging for image-based methods. In contrast, point cloud alignment methods require geometric structures and suffer when planes are dominant in the scene.

In [132], we increase the robustness of our edge-based method by additionally incorporating the depth map from the RGBD sensor directly into the motion estimation. We propose to align the point clouds generated from the depth maps with an *ICP* algorithm and extend (4.6) by a geometric error E_{geo}:

$$\boldsymbol{\xi}_{ij}^{*} = \underset{\boldsymbol{\xi}_{ij}}{\operatorname{argmin}} E_{edge} + \lambda E_{geo}, \tag{4.34}$$

where λ is a balancing factor between the two energy terms. E_{edge} is defined equal to (4.6)

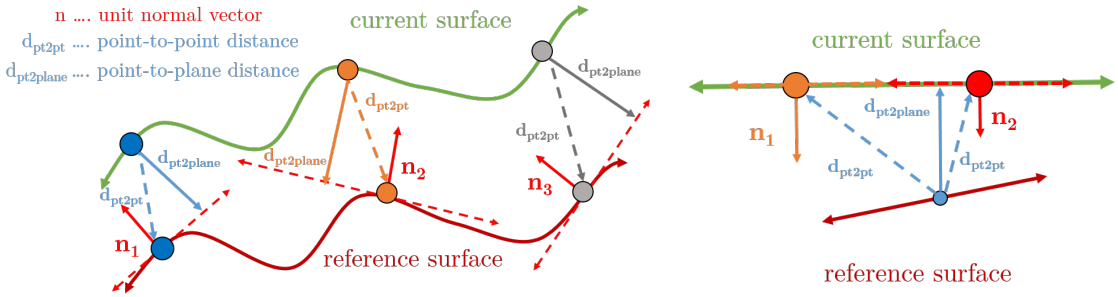
and E_{geo} can also be written in a similar least-squares formulation:

$$E_{geo} = \sum_{i \in \mathcal{Z}} w_t r_{i,geo}(\boldsymbol{\xi}_{ij})^2. \tag{4.35}$$

where w_t is a Tukey weight (see Tab. 3.2) and the residual $r_{i,geo}(\boldsymbol{\xi}_{ij})$ is given as the point-to-plane distance between the two point clouds. We minimize (4.34) again in a coarse-to-fine scheme over three pyramid levels with an iteratively reweighted *LM* method. There exist several *ICP* formulations of the geometric error [128] and if correspondences are known, there are also closed-form solutions [48, 69]. However, for most practical cases the correspondences or data associations are unknown and matching two point sets is a computationally expensive operation, which might be required in each iteration. There are approaches to compute the correspondences such as matching points according to certain properties like surface normals, colors or curvature, or relying on local descriptors, e.g. Point Feature Histograms [129]. Another option is to look for the closest point in terms of space, which involves either an exhaustive search or a costly pre-processing step to generate a kd lookup tree. Blais et al. [9] proposed *projective data association*, a very fast alternative used in most state-of-the-art systems [78, 117, 159]. *Projective data association* assigns correspondences by reprojecting a point $\mathbf{p}_i$ from $\mathbf{Z}_i$ to $\mathbf{Z}_j$ and taking the point at $\mathbf{p}' = \tau(\xi_{ji}, \mathbf{Z}(\mathbf{p}), \mathbf{p})$ as correspondence. It does not require any additional data structure or search in 3D space, which makes it very fast and also easily parallelizable. The 3D correspondences $\mathbf{Q}_j$ for a point $\mathbf{P}'_i = \mathbf{R}_{ji}\mathbf{P} + \mathbf{t}_{ji}$ can then be computed as:

$$\mathbf{Q}_j = \pi^{-1}(\mathbf{p}'_i, Z_i(\mathbf{p}'_i)). \tag{4.36}$$

After computing the correspondence, there are two different ways to define the residual r_{geo}: (i) point-to-point distance or a (ii) point-to-plane distance. Figure 4.9a visualizes the difference between the two formulations.



(a) The point-to-point (pt2pt) and point-to-plane (pt2pl) metric of the ICP method [8, 20].

(b) Two points with the same tangent plane have equal costs.

Figure 4.9: Geometric error when aligning two surfaces with point-to-point d_{pt2pt} and point-to-plane distance $d_{pt2plane}$. (b) shows that the point-to-plane formulation allows to slide along flat surfaces.

Point-to-Point Distance: The point-to-point distance is simply defined as the vector $\mathbf{d}_{p2p}$ between the transformed point $\mathbf{P}'_i = \mathbf{R}_{ji}\mathbf{P} + \mathbf{t}_{ji}$ from $\mathbf{Z}_i$ and its associated point $\mathbf{Q}_j$ in $\mathbf{Z}_j$:

$$r_{p2p} = ||\mathbf{d}_{d2p}||_2 = ||\mathbf{P}'_i - \mathbf{Q}_j||_2. \tag{4.37}$$

As depicted in Figure 4.9b, aligning planar surfaces by minimizing the point-to-point error is problematic when the initialization is not close to the minimum since it cannot *slide* along surfaces. In practice, these are crucial drawbacks because the initialization is typically not ideal and indoor scenes mainly contain planar surfaces. Thus, there are several alternatives to tackle this problem [128].

Point-to-Plane Distance: The most popular alternative is to project the point-to-point distance vector $\mathbf{d}_{p2p} = \mathbf{P}'_i - \mathbf{Q}_j$ along the direction of the surface normal such that all points lying on the same planar surface have equal distance to points on another planar surface (see Fig. 4.10c). The point-to-plane error is defined as:

$$r_{p2pl} = \langle \mathbf{d}_{p2p}, \mathbf{n}_r \rangle, \tag{4.38}$$

where $\mathbf{d}_{p2p}$ is again the same is in (4.37) and $\mathbf{n}_r$ is the surface normal. With this formulation, *sliding* along surfaces does not increase the cost opposed to the point-to-point metric (4.37). However, it is necessary to compute the surface normal vector $\mathbf{n}$ at each point $\mathbf{p}$ from its local neighborhood as visualized in Figure 4.10c. The normal vector $\mathbf{n}$ at $\mathbf{p}$ is the cross-product of $\mathbf{v}_x$ and $\mathbf{v}_y$:

$$\mathbf{n_r} = \mathbf{v}_x \times \mathbf{v}_y, \tag{4.39}$$

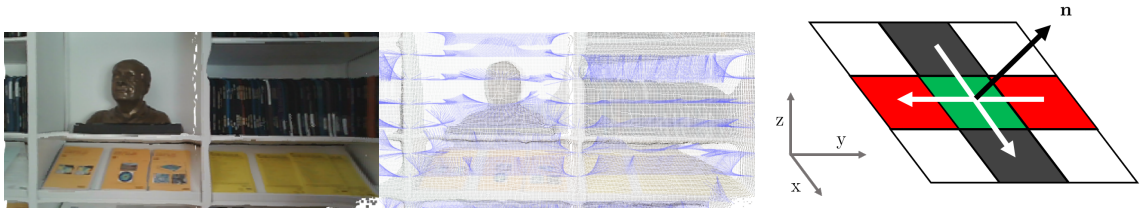
where the vectors $\mathbf{v}_x$ and $\mathbf{v}_y$ are computed from the unprojected 3D points adjacent to $\mathbf{p}$ as:

$$\mathbf{v}_x = \pi^{-1}(\mathbf{p}_{x1}, Z(\mathbf{p}_{x1})) - \pi^{-1}(\mathbf{p}_{x2}, Z(\mathbf{p}_{x2})), \tag{4.40}$$

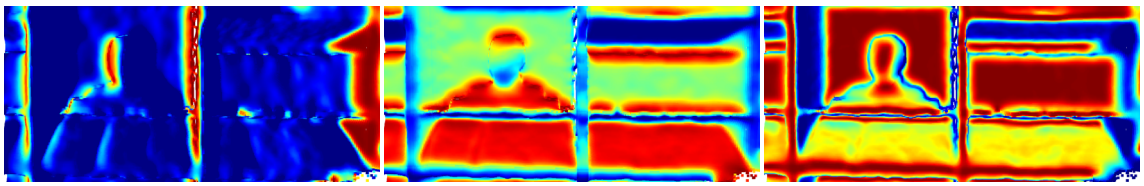
$$\mathbf{v}_y = \pi^{-1}(\mathbf{p}_{y1}, Z(\mathbf{p}_{y1})) - \pi^{-1}(\mathbf{p}_{y2}, Z(\mathbf{p}_{y2})). \tag{4.41}$$

The adjacent pixel $\mathbf{p}_{x1}$ and $\mathbf{p}_{x2}$ are $[p_x \pm 1, p_y]$ respectively and $\mathbf{p}_{y1}$ and $\mathbf{p}_{y2}$ are $[p_x, p_y \pm 1]$. Figure 4.10b visualizes the computed surface normals on a typical indoor scenes with the respective x-, y- and z-components of the normal vectors shown Figures 4.10d- 4.10f.

Balancing Factor λ_w: Jointly optimizing two energy terms requires a balancing factor λ [106]. This is especially challenging when combining energies with completely different metrics, e.g. edge distance in pixels and point-to-plane error in meters, or density, e.g. rather sparse edges compared to the dense geometric term. The choice of the λ value is not straightforward and has not been investigated for the combination of an edge and a geometric error term. We study the influence of λ by evaluating the Relative Pose Error



(a) Colored Point Cloud (PCL) **(b)** *PCL* with surface normals **(c)** Surface normal computation



(d) Normal x-component **(e)** Normal y-component **(f)** Normal z-component

Figure 4.10: The colored *PCL* (a) and the corresponding surface normals (b). (c) The surface normals at a point are computed from its neighborhood. (d,e,f) are the respective x-,y- and z-components of the surface normals.

(RPE) and Absolute Trajectory Error (ATE) on three sequences from the TUM RGBD benchmark dataset [147] (see Sec 6.1.1 for a detailed description) . Figure 4.11 depicts the *RPE* and *ATE* for *"fr1/xyz"*, *"fr1/rpy"*, *"fr1/desk"* and the average of the errors over a range of λ values from 0.5 to 18. The evaluation indicates that the correct choice of λ depends on the dataset because the errors stay nearly constant for *"fr1/xyz"* and *"fr1/rpy"*, while there are huge jumps for *"fr1/desk"*. However, Figure 4.11 shows that the simple choice of $\lambda = 1$ gives reasonable results, which we also verified by several other experiments (see Sec 6.1.1). The intuition behind the choice of $\lambda = 1$ is that when there are many edge detections in the image, they highly influence the optimization since the edge distance in pixels is typically orders of magnitudes higher than the geometric error in the millimeter range. In contrast, with only a few edge detections the geometric term is emphasized.

4.1.4 Optimizing the Geometric Error on SE(3)

This section discusses the optimization of the combined error function (4.34) and presents the derivations of the Jacobians. To solve (4.34), we again apply an iteratively re-weighted *LM* optimization method in a coarse-to-fine scheme. While it is necessary to compute an additional normal map $\mathbf{N}$ for the geometric error, in the *KF*-based setup, $\mathbf{N}$ has to be computed - similar to the *DT* - only when inserting a new *KF*. Before discussing the combined optimization (4.34), we first focus on the optimization of the E_{geo} (4.35) with its residual:

$$r_{p2pl} = \langle \mathbf{d}_{p2p}, \mathbf{n}_r \rangle = \langle \mathbf{R}_{ji}\mathbf{P} + \mathbf{t}_{ji} - \mathbf{Q}, \mathbf{n}_r \rangle, \tag{4.42}$$

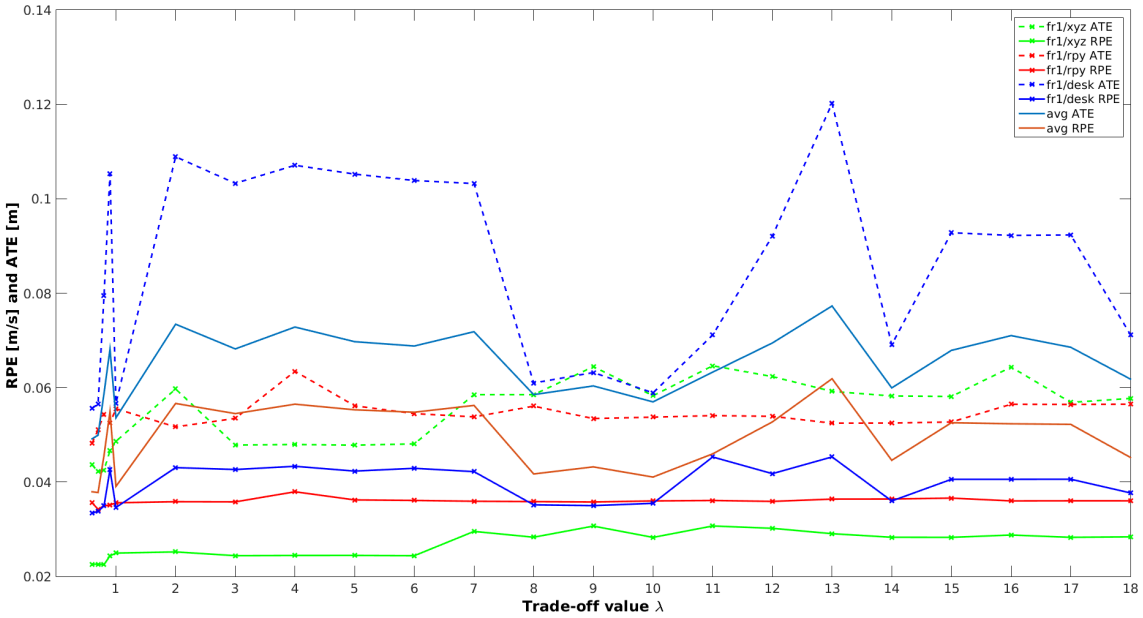


Figure 4.11: Evaluation of pose and alignment errors with varying balancing or trade-off factor λ on three sequences of the TUM RGBD benchmark [147]. The choice of λ is scene-specific but $\lambda = 1$ shows a good overall performance.

For very small rotations, the exponential matrix of $\boldsymbol{\omega}$ can be approximated analogously to the edge-based case as given in (4.21):

$$\mathbf{R} = \mathbf{I}_3 + [\boldsymbol{\omega}]_\times = \begin{pmatrix} 1 & -\omega_3 & \omega_2 \\ \omega_3 & 1 & -\omega_1 \\ -\omega_2 & \omega_1 & 1 \end{pmatrix} \tag{4.43}$$

Substituting (4.43) into (4.42) and expanding, r_{p2pl} becomes:

$$\begin{aligned} r_{p2pl} =& (\mathbf{P}' - \mathbf{Q})^T \mathbf{n} + \mathbf{t}^T \mathbf{n} + \\ & \omega_1 (Y' n_z - Z' n_y) + \omega_2 (Z' n_x - X' n_z) + \omega_3 (X' n_y - Y' n_x), \end{aligned} \tag{4.44}$$

where $\mathbf{n} = [n_x, n_y, n_z]^T$ and $\mathbf{P}' = [X', Y', Z']^T$. Using the cross-product $\mathbf{c} = \mathbf{P}' \times \mathbf{n}$, (4.44) simplifies to:

$$r_{p2pl} = (\mathbf{P}' - \mathbf{Q})^T \mathbf{n} + \mathbf{t}^T \mathbf{n} + \boldsymbol{\omega}^T \mathbf{c}. \tag{4.45}$$

The derivatives with respect to $\boldsymbol{\omega}$ and $\mathbf{t}$ are:

$$\begin{aligned} \frac{\partial r_{p2pl}}{\partial \boldsymbol{\omega}} &= \mathbf{c} \\ \frac{\partial r_{p2pl}}{\partial \mathbf{t}} &= \mathbf{n} \end{aligned} \tag{4.46}$$

.

The Jacobian $\mathbf{J}_{geo}$ for the *ICP* optimization is then:

$$\mathbf{J}_{p2pl} = (\mathbf{c}^T, \mathbf{n}^T)^T. \tag{4.47}$$

Finally, for the joint optimization, we again set up an equation system of the form

$$\mathbf{H}\boldsymbol{\delta} = \mathbf{b}, \tag{4.48}$$

where

$$\begin{aligned} \mathbf{H} &= \mathbf{J}_{edge}^T \mathbf{W}_{edge} \mathbf{J}_{edge}^T + \lambda \mathbf{J}_{geo}^T \mathbf{W}_{geo} \mathbf{J}_{geo}^T \\ \mathbf{b} &= \mathbf{J}_{edge}^T \mathbf{W}_{edge} \mathbf{r}_{edge} + \lambda \mathbf{J}_{geo}^T \mathbf{W}_{geo} \mathbf{r}_{geo}. \end{aligned} \tag{4.49}$$

In this form, the computations of the Jacobians are easily parallelizable and can be implemented as shown in Section C.1.

4.2 Edge-based Quality Assessment

One of the central problems in *SLAM* is to assess the quality or validity of an estimated relative motion. In this section, we propose a fully Edge-based Quality Assessment (EBQA), which was successfully used in REVO [133], and show how our method classifies estimates into valid and poor. Figure 4.12 visualizes the complete three-step *EBQA* pipeline with the intermediate results. Algorithm 1 summarizes the computation steps and relates them to the corresponding equations. Our *EBQA* is based on the idea that the estimated world pose $\boldsymbol{\xi}_{Wj}$ of a frame $\mathcal{F}_j$ to be assessed has to be consistent with previous estimates, i.e. the estimate must align the respective edge detections to previous frames. The first step is to reproject edges with valid depth from N previous frames denoted as $\mathcal{E}_i \in \mathcal{P}$ to the current frame $\mathcal{F}_j$ as depicted in Fig. 4.12(1). Each individual set $\mathcal{E}_i$ is reprojected to a related hit map $\mathbf{M}_i$ in $\mathcal{F}_j$:

$$\forall\ \mathcal{E}_i \in \mathcal{P}: \quad \mathbf{M}_i(\mathbf{p}') = 1, \quad \mathbf{p}' = \lfloor \tau(\mathbf{p}, \xi_{ji}, \mathbf{Z}_f(\mathbf{p})) \rfloor \ \ \forall\ \mathbf{p}\ \in \mathcal{E}_i, \tag{4.50}$$

with $\lfloor . \rfloor$ being the *floor* operation. Due to rounding, multiple reprojected edges of a single frame can map to the same coordinates. When large scale differences occur, this could lead to all the edges of one image falling on a tiny fraction of another image, e.g. $\mathcal{F}_j$ shows a whole chair and the other images close-ups of the chair's legs. Thus, coinciding reprojections only count once by setting $\mathbf{M}_i(\mathbf{p}') = 1$ as described in (4.50). The second step sums up the individual hit maps $\mathbf{M}_i$ to a joint hit map $\mathbf{M}$ containing values $[0, 1, .., N]$ (see Fig. 4.12 (2)):

$$\mathbf{M} = \sum_i \mathbf{M}_i. \tag{4.51}$$

A value of N at a certain position $\mathbf{p}$ means that edges from all N previous frames were reprojected to it. Figure 4.12(3) shows a color-coded hit map, where orange, yellow and

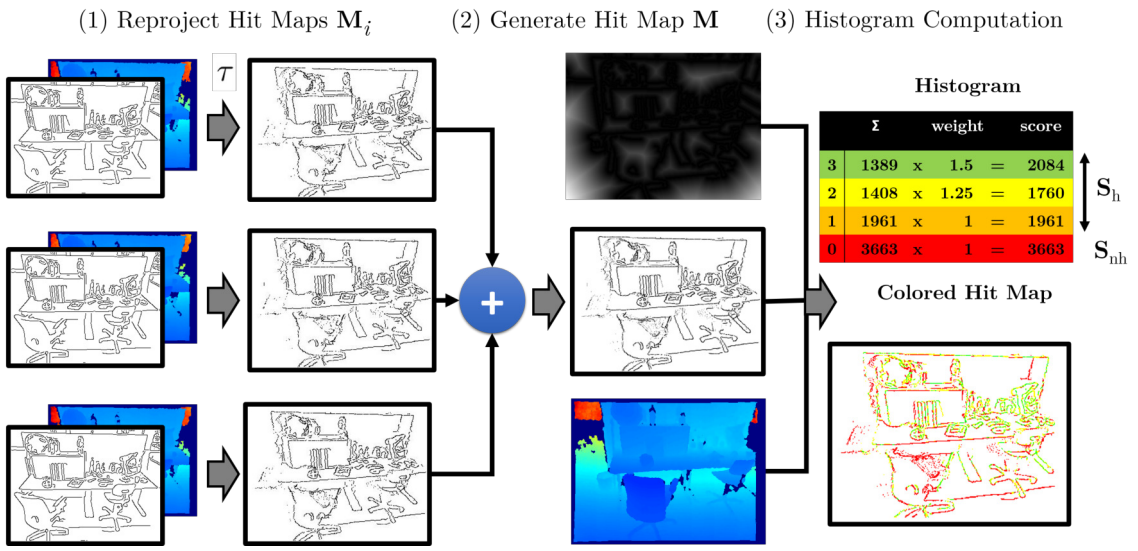


Figure 4.12: Three-step edge-based quality assessment pipeline to classify estimates into valid and poor. (1) reprojects edge detections from N previous frames to the frame to be assessed $\mathcal{F}_i$ and stores the reprojections in separate hit maps $\mathbf{M}_i$. (2) sums up all the individual hit maps to $\mathbf{M}$ and (3) computes a histogram of hits and non-hits. This can also be visualized as color-coded hit map, where red are the non-hits and orange, yellow and green are reprojections with $N = [1, 2, 3]$ hits. An estimate is considered poor if the weighted hit score S_h is greater than the non-hit score S_{nh}.

green represent that $N = [1, 2, 3]$ reprojections are close to an edge and red shows non-hits. The third step evaluates if the reprojected are close to edge detections in $\mathcal{F}_j$ and generates a histogram $\mathbf{H}$ with $N + 1$ bins as follows:

$$\mathbf{H}(n) = \sum_{\mathbf{p} \in \mathcal{Q}} \underbrace{[\![\mathbf{M}(\mathbf{p}) == n]\!]}_{\text{selection}} \underbrace{[\![\mathbf{D}_j(\mathbf{p}) \leq \theta_{qa}]\!]}_{\text{distance}} \underbrace{[\![\mathbf{Z}_j(\mathbf{p}) > 0]\!]}_{\text{geometry}}; \quad \forall n \in [0, N], \tag{4.52}$$

where $\mathcal{Q}$ is the set of pixel positions $\mathbf{p}$ of all reprojected edges, $\mathbf{M}(\mathbf{p})$ the number of reprojections at $\mathbf{p}$ with a maximum value of N, $\mathbf{D}_j$ the DT and $\mathbf{Z}_j$ the depth map. $[\![.]\!]$ is the Iverson bracket defined in Section C.2.1. In the following, we describe the three individual terms of (4.52)). For bin n, the *selection* term is only true if at point $\mathbf{p}$ there are n reprojected edges. The *distance* term is true if there is an edge detection nearby, which corresponds to the DT being smaller than a threshold θ_{qa}. Note that in [132, 133], we avoid computing the DT for $\mathcal{F}_j$ and compare directly on the edge detection $[\![\mathbf{E}_j(\mathbf{p}) == 1]\!]$. The *geometry* term restricts the edge comparison only to valid depths in both images, thereby introducing a geometric similarity constraint, which is crucial for robust loop closure candidate detection. Since the terms are joined by multiplication, they evaluate to 1 only if all three individual terms are true. The final step of our *EBQA* classifies the estimates into valid and poor. It follows the intuition that when the pose estimation or the loop candidate is valid, edge detections should strongly overlap, i.e. high

Algorithm 1: Edge-based Quality Assessment

$\mathbf{M} = \mathbf{0}$
// Compute the distribution map $\mathbf{M}$
for $\mathcal{E}_i \in \mathcal{P}$ **do**
  $\mathbf{M}_i = \mathbf{0}$
  $\mathbf{p}' = floor\big(\tau(\mathbf{p}, \boldsymbol{\xi}_{ji}, \mathbf{Z}_i(\mathbf{p}))\big)$
  if $isInImage\big(\mathbf{p}'\big)$ **then**
    $\mathbf{M}_i(\mathbf{p}') = 1$ // Equation 4.50
  $\mathbf{M} = \mathbf{M} + \mathbf{M}_i$
// Compute the histogram $\mathbf{H}$
for *each point* $\mathbf{p}$ *in* $\mathbf{M}$ **do**
  if $\mathbf{Z}_c(\mathbf{p}) > 0$// Check if depth is valid
  then
    if $\mathbf{DT}_c(\mathbf{p}) < \theta_{qa}$// Check if close to an edge
    then
      $n = \mathbf{M}(\mathbf{p})$// Number of reprojections
      $\mathbf{H}(n) = \mathbf{H}(n) + 1$
// Compute the scores S_n and S_nh
$\mathbf{w}$ =weights for each bin
$S_nh = 0$ // Non-hit score
$S_h = 0$ // Hit score
for $n \in [1..N]$ **do**
  $S_h = S_h + \mathbf{w}(i)\mathbf{H}(i)$ // Equation 4.53
$S_{nh} = \mathbf{w}(0)\mathbf{H}(0)$// Equation 4.53
$isValid = S_h > S_{nh}$// Equation 4.54

number in $\mathbf{H}(N-1)$ and $\mathbf{H}(N)$ but low number of non-overlaps in $\mathbf{H}(0)$. This can be described in the form of a hit score S_h and non-hit score S_{nh}:

$$\begin{aligned} S_h =& \sum_{n=1}^{N} \mathbf{w}(n)\mathbf{H}(n), \\ S_{nh} =& \mathbf{w}(0)\mathbf{H}(0), \end{aligned} \tag{4.53}$$

where $w(n)$ is the weight for bin n. Since its better to have multiple reprojections hitting an edge, the weights are higher for higher bin numbers, e.g. $\mathbf{w} = [1, 1, 1.25, 1.5]$ corresponding to the bins $[0, 1, 2, 3]$. The estimate is considered valid if the hit score S_h is higher than the non-hit score S_{nh} and poor otherwise:

$$S_h > S_{nh}. \tag{4.54}$$

In this section, we proposed an fast and reliable method for quality assessment, which

integrates smoothly into REVO [132, 133] and RESLAM [134] due to its edge-based nature. Our *EBQA* is versatile in the sense that it can assess the quality of pose estimates [132, 133] in the *VO* setting and it is also capable to verify loop closure and relocalization candidates in *SLAM* [134]. Another possible application of our *EBQA* is the generation of ground truth data to train neural networks for the task of edge detection, which we will discuss in Chapter 5.

4.3 Keyframe Management

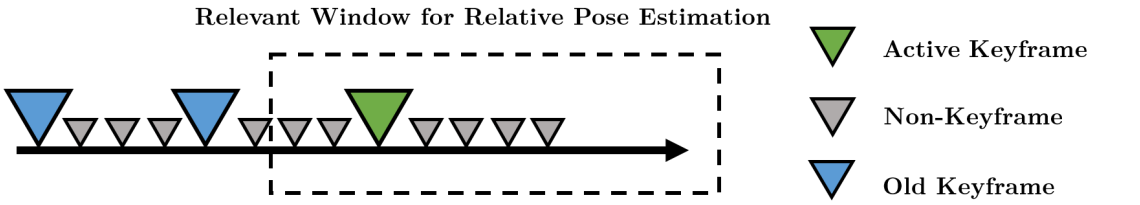
Keyframe management comprises the creation, removal or marginalization of *KFs* and is one of the central topics in *VO* and *SLAM*. The choice of a good *KF* is not trivial and typically often not the same in both cases. *VO* systems that only consider the last *KF* such as our systems [132, 133] and [81, 89], the distance to the *KF* should be kept large to reduce the drift. In contrast, systems that have access to multiple *KFs*, e.g. through a local or global map, often add an abundance of *KFs* and remove redundant or less useful ones later [37, 116, 134]. Common approaches for *KF* selection are to simply take every nth *KF* [89] or insert a new *KF* after a particular threshold is reached based on angle or distance [38], number of features [116] or an entropy-based ratio [80, 81]. In the following, we will discuss the *KF* selection strategies of our *VO* systems REVO [132, 133] and RESLAM [134].

4.3.1 Keyframe Management for VO

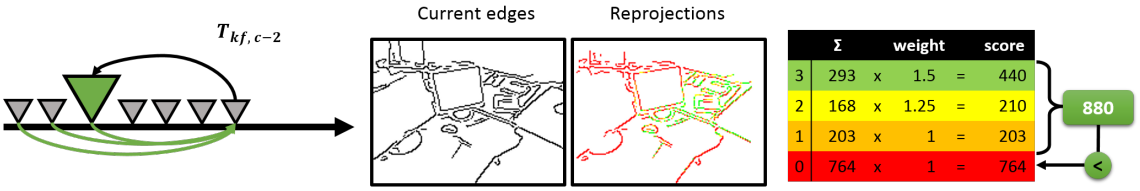
The pure *VO* case only requires the last *KF* (see Fig. 4.3). While REVO [133] simply discards an old *KF* once a new one is created, [132] stores old *KFs* with a fixed camera poses to use them at a later stage. Creating a new *KF* is a computationally demanding task since it involves computing the *DT* [133] and the normal map [132], which takes significantly longer than the rather fast relative pose estimation. Thus, the goal is to create as few *KFs* as possible, which at the same time keeps the distance to the last *KF* large to reduce drift. Figures 4.13b-4.13e show how to determine when to create a new *KF*. After estimating the relative pose of a new frame, the *EBQA* presented in Section 4.2 computes a hit-score and a non-hit score. As long as the hit-score is higher than the non-hit score as in Figures 4.13b-4.13c, no new *KF* is required. Once, the *EBQA* determines an estimate to be poor, i.e. the non-hit score is higher than the hit-score (see Fig. 4.13d), a new *KF* has to be created. Instead of simply taking the current frame as *KF*, we instead take the last frame with a valid estimate, i.e. the previous one, and use it as *KF*. Finally, the relative motion is re-estimated and re-assessed, resulting in a valid estimate (see Fig. 4.13e).

4.3.2 Keyframe Management for SLAM

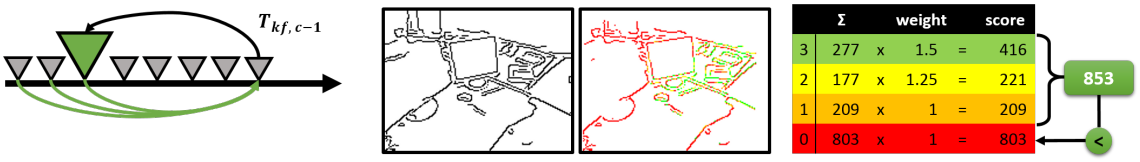
In contrast to pure *VO*, systems with a local map [37, 92] and a global one [116] often create many *KFs* and cull them later. RESLAM [134] also follows this strategy and



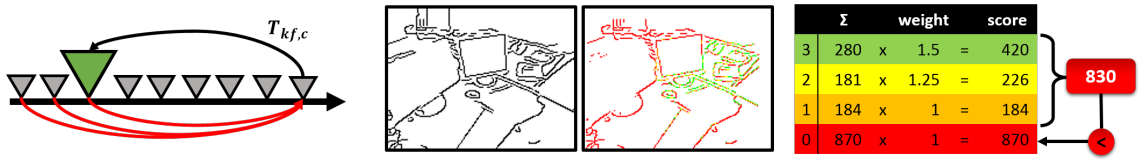
(a) Camera trajectory with non-*KFs*, old *KFs* and the most recent (active) *KF*.



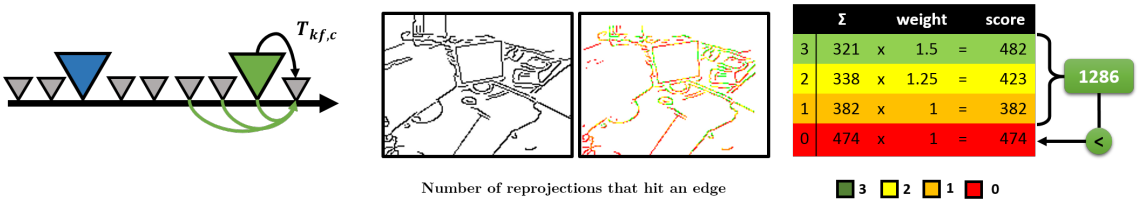
(b) Relative motion estimation $T_{kf,c-2}$, where the computed hit-score is above the non-hit-score.



(c) Relative motion estimation $T_{kf,c-1}$, where the computed hit-score is above the non-hit-score.



(d) Relative motion estimation $T_{kf,c}$, where the computed hit-score is below the non-hit-score. Thus, create a new *KF* such that the quality criteria is again fulfilled.



(e) Convert the previous frame, i.e. the last one fulfilling the quality criteria, into a new *KF* and re-estimate the relative transformation $T_{kf,c}$. Then the quality criteria should hold again.

Figure 4.13: (a) The camera trajectory for our purely *VO*-based systems [132, 133]. The *KF* creation and tracking quality assessment process is shown for several frames. After each relative motion estimation, we assess the tracking quality with the method presented in Section 4.1.1. (b, c) As long as the score of edge-hits stays above the non-hits, pose estimation continues and no new *KF* is created. (d) Once the computed hit score is below the non-hit score, (e) the last valid non-*KF* is converted into a new *KF* and the relative camera motion re-estimated such that the resulting score is again above the non-hit score.

distinguishes between two main types of *KFs* as depicted in Figure 4.14a: (i) *Active KFs* are refined in the local window and (ii) *Marginalized KFs* are removed from the local window to keep it constrained. Independent of their type, there are also map *KFs*, which build the global map and are marked by a black border in Figure 4.14a. In the following, we will discuss the life cycle of the *KFs* in RESLAM, i.e. creation, marginalization and removal of non-map *KFs*.

Keyframe Creation: RESLAM creates a new *KF* and adds it to the local window based on three different metrics: (i) the mean square optical flow C_{fov}, (ii) the mean flow without rotation C_{occ} and (iii) an edge alignment criteria. The (i) mean square optical flow C_{fov} describes changes in the field of view:

$$C_{fov} = \sqrt{\frac{1}{n}\sum ||\mathbf{p} - \mathbf{p}'||_2^2}\,, \tag{4.55}$$

and the (ii) mean flow without rotation C_{occ} measures occlusions by only considering the displacement $\mathbf{t}$:

$$C_{occ} = \sqrt{\frac{1}{n}\sum ||\mathbf{p} - \mathbf{p}^t||_2^2}\,, \tag{4.56}$$

where $\mathbf{p}' = \pi(\mathbf{R}\pi^{-1}(\mathbf{p}, \mathbf{Z}_p) + \mathbf{t})$ and $\mathbf{p}^t = \pi(\pi^{-1}(\mathbf{p}, \mathbf{Z}_p) + \mathbf{t})$. Similar to the the *EBQA* presented in Section 4.2, RESLAM also counts (iii) the number of edge reprojections close and not close to an edge detection denoted as N_{in} and N_{out}, respectively:

$$\begin{aligned} N_{in} &= \sum_{\mathbf{p}\in\mathcal{E}_c} [\![\mathbf{D}_c(\tau(\boldsymbol{\xi}, \mathbf{p}, \mathbf{Z}_c(\mathbf{p}))) \leq \theta_{in}]\!], \\ N_{out} &= \sum_{\mathbf{p}\in\mathcal{E}_c} [\![\mathbf{D}_c(\tau(\boldsymbol{\xi}, \mathbf{p}, \mathbf{Z}_c(\mathbf{p}))) > \theta_{in}]\!], \end{aligned} \tag{4.57}$$

where θ_{in} is a threshold and $[\![.]\!]$ is the Iverson bracket (see Sec. C.2.1). Finally, RESLAM creates a new *KF* if at least one of the two criteria is fulfilled:

$$C_{fov}\theta_{fov} + C_{occ} * \theta_{occ} > 1 \quad \text{or} \quad N_{in} < 2N_{out}, \tag{4.58}$$

where θ_{fov} and θ_{occ} are empirically determined factors. Whenever RESLAM creates a new *KF*, it additionally computes the similarity to all other *KFs* in the place-recognition database. If the *KF* is distinct enough from all the others, it is added to the database, i.e. becomes part of the global map.

Keyframe Marginalization Since the local window is constrained, RESLAM marginalizes old *KFs* whenever a new *KF* is added. In order to keep the *KFs* well-distributed in the local map, we apply the marginalization strategy proposed in [37]. This strategy follows three main points: (1) always keep the newest two *KFs* $\mathcal{F}_1$, $\mathcal{F}_2$,

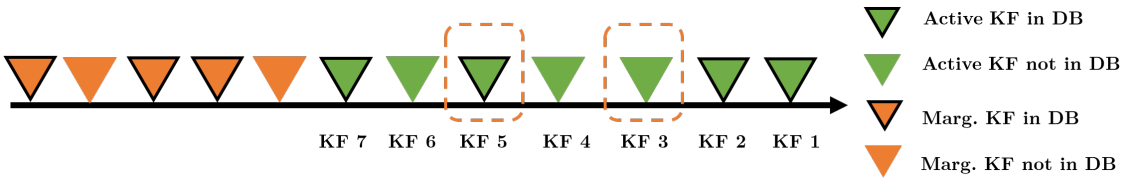
Seq.	Number of Frames	Created KF	KF After Culling
fr1/desk	573	223	90
fr1/desk2	620	299	69

Table 4.2: Creating an abundance of *KFs* increases the robustness of the system and culling the *KFs* later reduces the complexity of Pose Graph Optimization (PGO) after loop-closure.

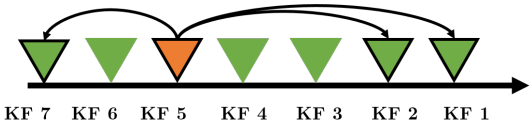
(2) marginalize *KF* with less than 5% visible points in $\mathcal{F}_1$ and (3) if no *KF* fulfills (2), marginalize the frame with the lowest distance score. The distance score s_i measures the spatial distribution of the *KFs*:

$$s_i = \sqrt{d(i,1)} \sum_{j \in [3,n] \setminus i} (d(i,j) + \epsilon)^{-1}, \tag{4.59}$$

where d is the Euclidean distance between two *KFs* $d(i,j) = ||\mathbf{t}_{ij}||_2$ and ϵ a small constant to prevent division by 0. When we marginalize a map *KF*, we store it along all its relative transformations to the other map *KFs* within the local window (see Fig. 4.14b). In contrast, when marginalizing a *KF* not in the map, we simply link it to the closest map *KF* and convert it to a non-*KF* (see Fig. 4.14c). This is essentially the culling procedure to remove the *KF* from the global map, thereby reducing the number of *KF* in the global map by a factor $2-4$ compared to the created *KFs*. Table 4.2 compares the number of created *KFs* and database *KFs* on two fast-paced sequences, where this difference is clearly visible.



(a) Active *KFs* in the local window before marginalization.



(b) Marginalization of *KF* 3 stored in the FernDB.

(c) Marginalization of *KF* 5 not stored in the FernDB

Figure 4.14: (a) Once the active window $\mathcal{K}$ becomes too large, we marginalize an old *KF* whenever a new *KF* is added. We propose two different strategies shown in (b) and (c) after marginalization depending on whether the *KF* is stored in the place-recognition database or not. (b) A *KF* not stored in the place-recognition database is linked to the closest map *KF* and converted to normal frame. (c) For a *KF* stored in the database, all its relative transformation to other map *KFs* in $\mathcal{K}$ are stored to be incorporated in the *PGO*.

4.4 Local Mapper

In contrast to taking only the last *KF* into account [81, 89, 133], typically referred to as pure *VO*, it is beneficial for accuracy and robustness to also refine with respect to N previously estimated *KFs*. There exist several strategies for this task:

- Refine the most recent *KF* with respect to N temporally close or last *KFs*
- Refine the most recent *KF* with respect to N spatially close *KFs*
- A full local Bundle Adjustment (BA) refines the pose estimates of all the active *KFs* within the window

Figure 4.15 shows a visualization of these strategies with the current *KF* in green, the other involved *KFs* in orange. Refining with respect to N temporally close *KFs* suffers from the problem that the overlap between the *KFs* is often small. This might even reduce the overall accuracy since the refinement focuses only on the small overlapping part. Thus, we will focus on the two other strategies, which are also used in our publications [132, 134]. Section 4.4.1 describes the refinement on spatially close *KFs* [132] and Section 4.4.2 the full local *BA*, which refines all the involved model parameters such as camera intrinsics, camera poses and depth of the edges [134].

4.4.1 Optimization on Spatially Close Keyframes

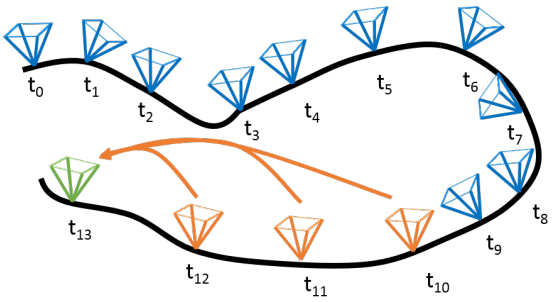
Instead of refining with respect to the last N *KFs*, we propose to optimize on *KFs* seeing similar scene parts, i.e. they are close in space and not necessarily in time. After initial relative motion estimation (see Sec. 4.1.1), we refine the current *KF*'s world pose with respect to N spatially close *KFs* (see Fig. 4.15b). This also has the benefit to implicitly close *small* loops without the need of any database. To find candidate *KFs*, we search through all previously estimated *KFs* and select the ones with camera centers close to the *KF* to be refined and a similar orientation. The distance d_c and rotation angle α between two frames $\mathcal{F}_i$ and $\mathcal{F}_j$ can be computed from their relative translation $\mathbf{t}_{ij}$ and rotation $\mathbf{R}_{ij}$:

$$d_c = ||\mathbf{t}_{ij}||_2, \quad \alpha = \arccos \frac{1}{2}(\text{trace}(\mathbf{R}_{ij}) - 1). \tag{4.60}$$

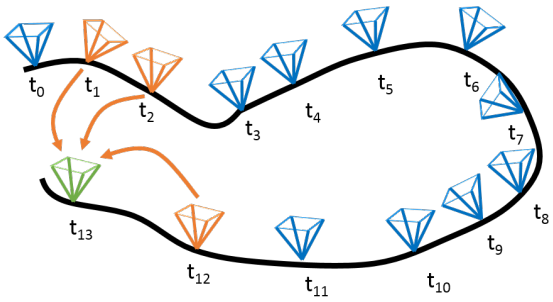
We first search through all estimated candidates and select potential candidates fulfilling $d_c < \theta_d$ and $\alpha < \theta_\alpha$ and then take the N *KFs* with lowest distance d_c. Then, we set up an optimization problem similar to (4.34) to refine the world pose of the current *KF* $\boldsymbol{\xi}_{jW}$ with respect to a set $\mathcal{S}$ of spatially close *KFs*:

$$\boldsymbol{\xi}_{jW}^* = \arg\min_{\boldsymbol{\xi}_{jW}} \sum_{\boldsymbol{\xi}_{Wi} \in \mathcal{S}} E_{edge}(\boldsymbol{\xi}_{ji})) + \lambda E_{icp}(\boldsymbol{\xi}_{ji}), \quad \boldsymbol{\xi}_{ji} = \boldsymbol{\xi}_{jW} \circ \boldsymbol{\xi}_{Wi}. \tag{4.61}$$

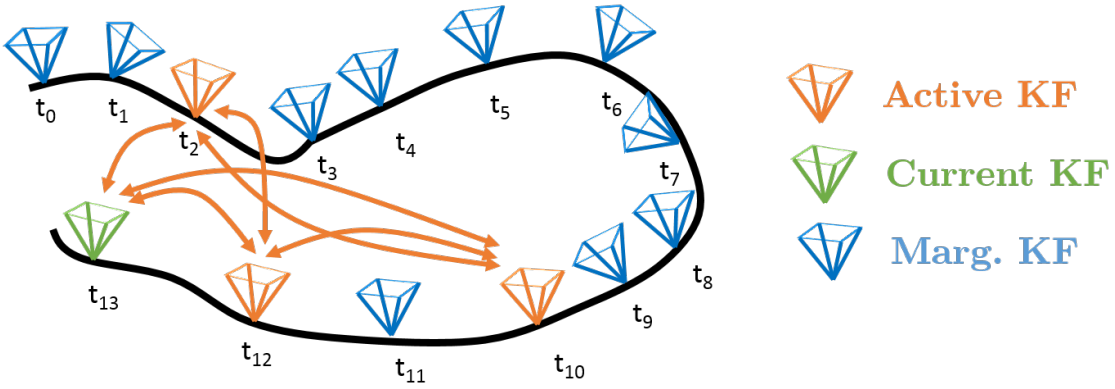
Note that (4.61) only refines the world pose of the current *KF* and all other poses remain fixed.



(a) Refine with last N *KFs*



(b) Refine with N spatially close frames *KFs*



(c) Full local *BA* over all N *KFs* in a window.

Figure 4.15: Visualization of different local optimization strategies. (a) Classical refinement with the last N *KFs* and our proposed methods: (b) refinement on spatially close *KFs* and (c) full local bundle adjustment over all *KFs* in a sliding window.

In our experiments, we found that this method shows definite improvements especially in smaller scenes but is no replacement for global loop closure. In longer sequences, drift can cause a problem since the estimate to older *KFs* can be poor and refining with respect to those can even degrade the overall accuracy in some cases. Further, this method degenerates into refining only over the last N *KFs* in purely exploratory trajectories, i.e. when not revisiting previously seen areas. New measurements have the potential to improve old estimates but in this formulation, only the most recent estimate is refined. Thus, the next step is to rely on a full local *BA* to refine all involved model parameters.

4.4.2 Local Bundle Adjustment

Instead of refining only the pose of the most recent *KF* as discussed in the previous section, in RESLAM [134], we set up a full local *BA* system in a sliding window similar to [37, 92]. Figure 4.15c shows this *BA* setup with four active *KFs*, where the current one is depicted in green and the others in orange. Within this sliding window $\mathcal{K}$, we jointly refine the depths of all the active edges, the camera poses and the camera intrinsics. Further, we marginalize old *KFs* poses and edges to keep the windowed optimization at a bounded

complexity.

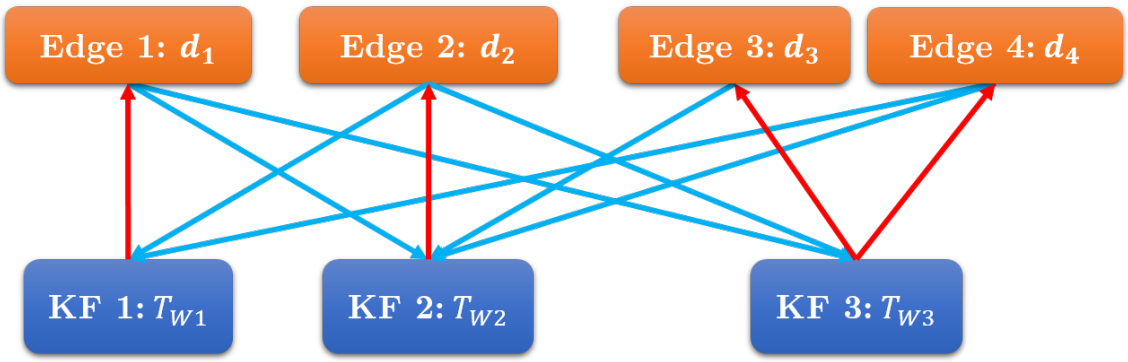
BA is a well-known technique and at the basis of most feature-based Structure from Motion (SfM) [65, 150], *VO* and *SLAM* [92, 114] systems. Such systems build a factor graph, where individual features are assigned to the *KF* they were first detected in and then linked to their correspondences or measurements in other *KFs*. Figure 4.16a shows the factor graph of a toy problem with three *KFs* and four points (or edge points in our case). Edge points are associated with their host *KF*, i.e. the *KF* they were first detected in (red), and parametrized by the depth d_i in this *KF*. An edge point can only have one host *KF* but is typically visible in multiple *KFs* as depicted by the blue arrows. From the factor graph, the full energy for the sliding window $\mathcal{K}$ can be written as:

$$E_{\mathcal{K}} = \sum_{i \in \mathcal{K}} \sum_{\mathbf{p}_e \in \mathcal{A}_i} \sum_{j \in \mathcal{K}, j \neq i} E_{\mathbf{p}_e, j} \tag{4.62}$$

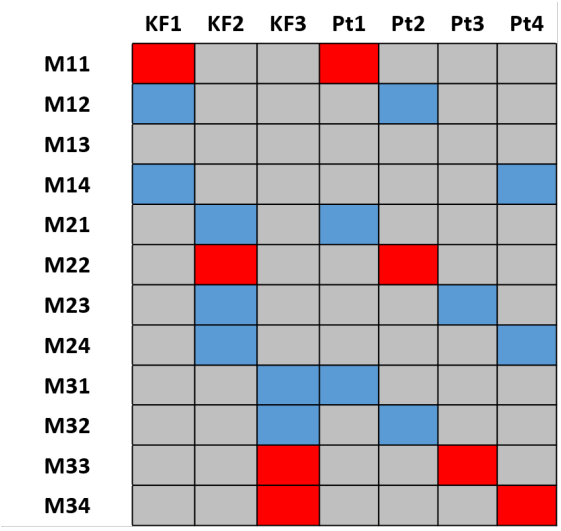
where $\mathcal{A}_i$ the set of active edge points in each *KF* and $E_{\mathbf{p}_e,j}$ represents the energy of one single edge point from *KF* i reprojected to *KF* j. Note that this graph is simple to build when correspondences are known as is the case for feature-based approaches. However, for the direct formulation this process is not straightforward and requires special consideration. In this section, we first describe how to build and maintain a factor graph in the direct formulation and discuss when to activate, drop and marginalize edge points within the window. We then present how to optimize all the model parameters and finally discuss the marginalization of *KFs* and edge points.

4.4.2.1 Building the Factor Graph

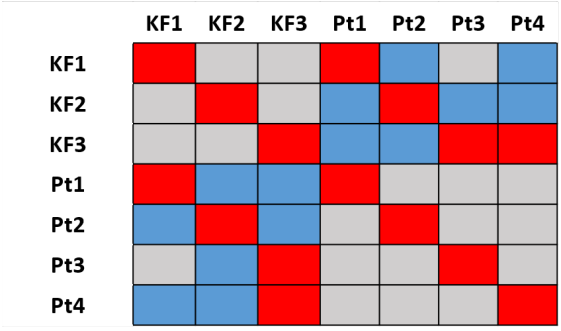
We build a factor graph for the windowed optimization by maintaining an active set of edges $\mathcal{A}_i$ for each *KF* within the window, which is inspired by [37]. In contrast to the feature-based approaches, where the depth of a point is defined in the world coordinate system, in our factor graph, the depth depends on the world pose of the edge point's host *KF* (see Fig. 4.16a). Apart from that, the factor graph is basically the same as in feature-based methods and can be solved similarly to [77, 145]. Each *KF* in the window has a set of around 10k - 20k detected edge pixels with valid depth $\mathcal{E}_i$. Due to the depth initialization provided by the RGBD sensor, this set is significantly larger than the number of points in monocular approaches [37, 92], where only points with already estimated depth are of interest. However, optimizing all detected edge points from each *KF* in $\mathcal{K}$ is not possible in real-time on a CPU. Thus, strategies to maintain a computable number of edge points are of great importance. Unlike [37], we do not limit the size of the active edges but only the number of *KFs* in $\mathcal{K}$. Similar to the *KFs* (see Sec. 4.3.2), also the edge points follow a certain life-cycle: (1) activation, (2) marginalization and (3) removal, which will be discussed in the following.



(a) Factor graph of a toy problem with three *KFs* and four edges.



(b) Jacobian Structure



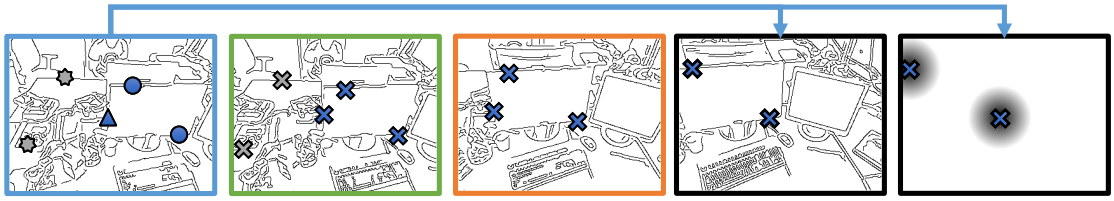
(c) Hessian Structure

Figure 4.16: The structure of the Jacobian (left) and the Hessian (right) for the toy problem presented in Figure 4.16a, where **KF1** - **KF3** are the *KFs*, $\mathbf{Pt_j}$ the edge points and **Mij** are the measurements, i.e. where the edge points are observed. Red blocks represent that this is the edge's host frame, while blue ones are observations.

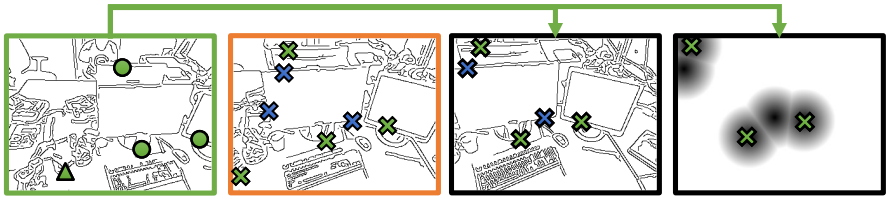
Edge Activation: Whenever edge points are marginalized or removed, new ones are activated to replace them. To keep the edge points spatially well-distributed over the image, RESLAM maintains a distance map $\mathbf{M}_f$ for $KF4$. It reprojects all active edge points $\mathcal{A}_i$ from $KF1$ - $KF3$ to $KF4$ and inserts them into the activation map (see Fig. 4.17b-4.17d). Active edge points which are still visible and valid are depicted as colored circles in their host *KFs*, e.g. two for $KF1$, three for $KF2$ and four for $KF3$, and their respective observations in other *KFs* as $\times$. RESLAM activates a new edge point if it fulfills the following conditions: (i) it is visible in $KF4$, (ii) its reprojection $\mathbf{p}'$ is close to an edge detection in $KF4$:

$$\mathbf{D}_4(\mathbf{p}') < \Theta_A, \tag{4.63}$$

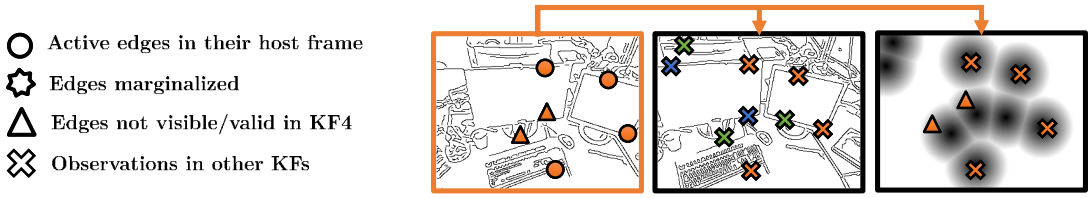
(a) Edge point activation for a sliding window with 4 *KFs*, where $KF4$ is the newest.



(b) Reproject the valid edges points $\mathcal{A}_1$ (blue) from $KF1$ to $KF4$ and insert them into $\mathbf{M}_4$.



(c) Reproject the active edges points $\mathcal{A}_2$ (green) from $KF2$ to $KF4$ and insert them into $\mathbf{M}_4$.



(d) Try to activate new edges points $\mathcal{A}_3$ (orange) for $KF3$ by reprojection to the act. map $\mathbf{M}_4$.

Figure 4.17: The edge activation procedure for a window of size $N = 4$, where the most recent $KF4$ maintains an activation map $\mathbf{M}_4$, which is essentially a DT. Active edges are depicted as filled circles in their respective host frames, e.g. **KF 1** has 2 active edge points, while the respective observations are depicted as $\times$. (a, b) Already active edge points in $KF1$ and $KF2$ are projected into the activation map $\mathbf{M}_4$ and subsequently updated, i.e. a new observation is added. With every new KF new scene parts become visible and previously active edge points can fall out of view. (c) The second newest **KF 3** has no active edge points yet, thus we reproject all edges with valid depth to $\mathbf{M}_4$ and activate an edge point if it fulfills the three activation criteria. The edge points depicted as stars are either not sufficiently far away from already active edge points or not close enough to an edge detection to be activated.

and (iii) it is not close to an already activated edge point:

$$\mathbf{M}(\mathbf{p}') > \Theta_M, \tag{4.64}$$

where Θ_A and Θ_M are thresholds and $\mathbf{D}$ is the DT computed from the edge detections. Figures 4.17b and 4.17d visualize edges that do not fulfill these criteria as triangles, e.g. for $KF1$ one edge is not visible in $KF4$ and for $KF3$ there are already activated edge points close by. Once a new edge point is activated, RESLAM adds it to $\mathbf{M}$ and searches for observations in the other *KFs* to build up a factor graph with sufficient connections.

Edge Marginalization: To keep the number of active edges at an amount, which allows processing in real-time, we aim to marginalize edges not contributing information to the currently visible scene part. RESLAM marginalizes an edge in either one of two cases: (i) its host KF is marginalized or (ii) it is not visible in the newest two *KFs*. For instance, one edge in Figure 4.17b (gray) is not visible in $KF3$ and $KF4$ and is therefore marginalized.

Edge Removal: Whenever RESLAM marginalizes a KF, it also removes all its observations (see Fig. 4.20b). Thus, active edge points only visible in their host KF and the just marginalized KF end up without observations. Since these edge points do not contribute to the local BA, RESLAM also removes them from the system and all activation maps to make space for new edge points to be activated.

4.4.2.2 Windowed Optimization Problem

Within the local window, we aim to optimize all the involved model parameters, namely the camera intrinsics, the *KFs* poses and the depth of all the active edges. In this section, we denote all the model parameters as $\mathbf{x} \in SE(3)^n \times \mathbf{R}^m$ and formulate the windowed optimization problem in terms of the sliding window energy $E_{\mathcal{K}}$ (4.62) as:

$$\begin{aligned} \mathbf{x}^* &= \arg\min_{\mathbf{x}} E_{\mathcal{K}}(\mathbf{x}) \\ &= \arg\min_{\mathbf{x}} \sum_{i \in \mathcal{K}} \sum_{\mathbf{p}_e \in \mathcal{A}_i} \sum_{j \in \mathcal{K}, j \neq i} E_{\mathbf{p}_e, j}(\mathbf{x}). \end{aligned} \tag{4.65}$$

Since the estimates from the *Motion Estimation* (see Sec. 4.1.1) are typically already close to the minimum, we solve (4.65) with an iteratively reweighted Gauss-Newton (GN) scheme instead of applying the slower LM damping. The parameter update can be written in terms of the current estimate $\mathbf{x}^k$ and the $\boldsymbol{\delta}$-update:

$$\mathbf{x}^{k+1} = \mathbf{x}^k \boxplus \boldsymbol{\delta}, \tag{4.66}$$

where we generalize the $\boxplus$ operator such that it is defined as in (4.11) for $SE(3)$ and as conventional addition for Euclidean space. We then set up a *GN* equation system:

$$\mathbf{H}\boldsymbol{\delta} = \mathbf{b}, \tag{4.67}$$

where $\mathbf{H} = \mathbf{J}^T\mathbf{W}\mathbf{J}$ and $\mathbf{b} = -\mathbf{J}^T\mathbf{W}\mathbf{r}$ with $\mathbf{W}$ being a weight-matrix. Even for the local window, the size of $\mathbf{H}$ is typically quite large and since solving (4.67) involves inverting $\mathbf{H}$, this would take too long for real-time processing. However, $\mathbf{H}$ has a special block structure (see Fig. 4.16) with the *KFs* in the top-left corner, the edge points in the bottom-right corner and the rest are the measurements or observations of the edge points in the *KFs*. By taking advantage of this special structure, we can efficiently solve (4.67) with the Schur complement [65, 150]. This is a well-known trick used in many *SfM* systems to reduces the complexity of classical *BA*. The Schur complement (SC) reduces the complexity of inverting an $O((m+n)^3)$ matrix to $O((m+n)^2)$. For example, often large parts of a matrix are sparse and can be inverted faster than dense parts. For the *SC*, we divide $\mathbf{H}$,

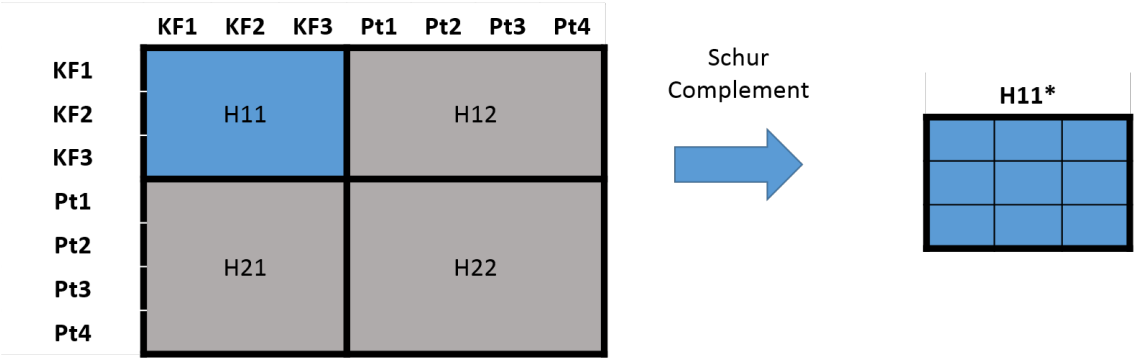


Figure 4.18: The Schur complement reduces the computational complexity by splitting the Hessian $\mathbf{H}$ into four blocks and treating them separately. Instead of inverting the complete matrix $\mathbf{H}$, first only the top left block is computed to get δ_{kf} and then δ_ρ can be computed by resubstitution.

$\boldsymbol{\delta}$ and $\mathbf{b}$ into blocks as follows:

$$\begin{pmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21} & \mathbf{H}_{22} \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta}_1 \\ \boldsymbol{\delta}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix} \tag{4.68}$$

where $\mathbf{H}_{11}$ corresponds to the *KFs*, $\mathbf{H}_{22}$ to the edge points and $\mathbf{H}_{12} = \mathbf{H}_{21}^T$ to the observations (see Fig. 4.18). Computing $\mathbf{b}_1$ and $\mathbf{b}_2$ yields:

$$\mathbf{b}_1 = \mathbf{H}_{11}\boldsymbol{\delta}_1 + \mathbf{H}_{12}\boldsymbol{\delta}_2, \quad \mathbf{b}_2 = \mathbf{H}_{21}\boldsymbol{\delta}_1 + \mathbf{H}_{22}\boldsymbol{\delta}_2. \tag{4.69}$$

Re-arranging for $\boldsymbol{\delta}_1$ and $\boldsymbol{\delta}_2$:

$$\boldsymbol{\delta}_1 = \mathbf{H}_{11}^{-1}(\mathbf{b}_1 - \mathbf{H}_{12}\boldsymbol{\delta}_2), \quad \boldsymbol{\delta}_2 = \mathbf{H}_{22}^{-1}(\mathbf{b}_2 - \mathbf{H}_{21}\boldsymbol{\delta}_1). \tag{4.70}$$

Solving (4.70) for $\boldsymbol{\delta}_1$ gives:

$$\begin{aligned}(\mathbf{H}_{11} - \mathbf{H}_{12}\mathbf{H}_{22}^{-1}\mathbf{H}_{21})\boldsymbol{\delta}_1 &= \mathbf{b}_1 - \mathbf{H}_{12}\mathbf{H}_{22}^{-1}\mathbf{b}_2, \\ \mathbf{H}_{11}^{*}\boldsymbol{\delta}_1 &= \mathbf{b}_1^{*}.\end{aligned} \tag{4.71}$$

This formulation only requires inverting $\mathbf{H}_{22}^{-1}$ instead of the larger $\mathbf{H}$ matrix. The other $\boldsymbol{\delta}_2$ can be easily obtained by re-substituting into (4.70). In the following, we will explain the various steps of the optimization and to keep it simple, only focus on a single residual and its corresponding row in the Jacobian. In each iteration, we evaluate the residual around the current estimate $\mathbf{x}^k$ similar to the motion estimation (see Sec. 4.1.1). The energy $E_{\mathbf{p}_e,j}(\mathbf{x}^k)$ from (4.65) for two *KFs* i and j is defined as:

$$E_{\mathbf{p}_e,j}(\mathbf{x}^k) = \Theta_H r_k^2(\mathbf{x}^k), \tag{4.72}$$

with Θ_H the Huber weight function. The residual r_k is defined as:

$$r_k(\mathbf{x}^k) = D(\mathbf{p}_j'(\mathbf{T}_{iW}, \mathbf{T}_{jW}, \mathcal{C}, \rho)), \tag{4.73}$$

where $\mathbf{p}_j'$ is the reprojection of $\mathbf{p}_e$ into *KF* j. Note that in contrast to the motion estimation (see Sec. 4.1.1), (4.73) relies on the inverse depth ρ [21] defined in the host *KF* i. The Jacobian is defined with respect to an increment $\boldsymbol{\delta}$ already discussed in Section 3.4.1:

$$\mathbf{J}_k = \left.\frac{\partial r_k(\boldsymbol{\delta} \boxplus \mathbf{x})}{\partial \boldsymbol{\delta}}\right|_{\boldsymbol{\delta}=\mathbf{0}}. \tag{4.74}$$

$\mathbf{J}_k$ comprises two parts, the derivatives of the *DT* denoted as $\mathbf{J}_{\mathbf{D}_j}$ and the geometric parameters $\mathbf{J}_{geo}$ containing the world poses $\mathbf{T}_{iW}$, $\mathbf{T}_{jW}$, the inverse depth ρ and the intrinsic camera parameters $\mathcal{C}$:

$$\begin{aligned}\mathbf{J}_k &= \left[\frac{\partial \mathbf{D}_j}{\partial \mathbf{p}'}\frac{\partial \mathbf{p}'(\boldsymbol{\delta} \boxplus \mathbf{x})}{\partial \boldsymbol{\delta}_{geo}}\right], \\ &= [\mathbf{J}_{\mathbf{D}_j}\mathbf{J}_{geo}.]\end{aligned} \tag{4.75}$$

When all the involved Jacobians are evaluated at the current estimate in each iteration like in the relative motion estimation (see Sec. 4.1.2) also the tangent space moves to this new evaluation point. However, once *marginalization* is introduced into the system, the evaluation points require special consideration. *Marginalization* essentially fixes the evaluation point $\mathbf{x}_0$ on which its future $\boldsymbol{\delta}$-updates are accumulated. In the case of the non-linear nullspace of the world poses, adding linearizations around different points can have the effect of eliminating the nullspaces. This problem has been extensively discussed in the literature [37, 71, 74, 92]. We tackle this problem by *First Estimate Jacobians* introduced by Huang et al. [74], which has also been successfully applied in [37, 92]. The *First Estimate Jacobians* technique evaluates the Jacobians for the marginalized terms

not at the current estimate $\mathbf{x}$ but applies the $\boldsymbol{\delta}$ to the fixed linearization point $\mathbf{x}_0$:

$$\mathbf{J}_{geo} = \left.\frac{\partial \mathbf{p}'(\boldsymbol{\delta} \boxplus \mathbf{x}_0)}{\partial \boldsymbol{\delta}}\right|_{\boldsymbol{\delta}=\mathbf{0}}. \tag{4.76}$$

Even though the linearization points for the Jacobians are suboptimal and could lead to errors, this approximation works well in practice since the Jacobian $\mathbf{J}_{geo}$ is smooth and the increment very small. The other Jacobian $\mathbf{J}_{\mathbf{D}_j}$ is much less smooth but in contrast to $\mathbf{J}_{geo}$ does not affect the nullspaces. Thus, we evaluate it at the current estimate $\mathbf{x}$. As visualized in Figure 4.16a, a single residual (4.73) depends on the world pose of its host *KF* and the pose of the *KF* it is reprojected to. To reduce the computational burden by the increased number of dependencies, the Jacobians with respect to the two *KFs* can be linearly related through the adjoint of their relative pose [32, 37, 144].

4.4.2.3 Marginalization

Since a robot or agent continuously explores the environment, the size of the state vector $\mathbf{x}$ increases over time. This introduces the problem that at some point the state vector is too large to be computed in real-time, which happens in practice after only a small number of around 10 *KFs*. In this section, we describe our marginalization process, which closely follows the ideas presented by Engel et al. [37, 41] and Leutenegger et al. [92] and refer to [30, 124, 140] for further reading. The basic idea of marginalization is to divide $\mathbf{x}$ into three separate state vectors:

$$\mathbf{x} = (\mathbf{x}_m, \mathbf{x}_a, \mathbf{x}_n), \tag{4.77}$$

where $\mathbf{x}_m$ contains marginalized states, $\mathbf{x}_a$ active states and $\mathbf{x}_n$ new states. The cost function $E(\mathbf{x})$ in terms of the three state vectors is then:

$$E(\mathbf{x}) = E(\mathbf{x}_m, \mathbf{x}_a, \mathbf{x}_n). \tag{4.78}$$

Since the marginalized states $\mathbf{x}_m$ are only linked to the active ones $\mathbf{x}_a$ but do not participate in any new measurements $\mathbf{x}_n$, it is straightforward to separate the cost function:

$$E(\mathbf{x}) = E(\mathbf{x}_m, \mathbf{x}_a, \mathbf{x}_n) = E_n(\mathbf{x}_a, \mathbf{x}_n) + E_m(\mathbf{x}_m, \mathbf{x}_a). \tag{4.79}$$

$E_n(\mathbf{x_a}, \mathbf{x_n})$ is equal to the window energy $E_{\mathcal{K}}$ (4.62) and depends on terms that either involve only $\mathbf{x}_a$, only $\mathbf{x}_n$ or both. In contrast, $E_m(\mathbf{x}_m, \mathbf{x}_a)$ does not involve any terms from $\mathbf{x}_n$ but depends on terms that either involve only $\mathbf{x}_m$ or both, $\mathbf{x}_m$ and $\mathbf{x}_a$. The marginalized cost function E_m is simply added to E_n in all subsequent optimization. We compute the part of the energy E_m containing all residuals that depend on state variables, which should be marginalized and add it to the cost function $E_{\mathcal{K}}$ in all following optimization and marginalization operations. For E_m we compute one *GN* approximation around a

current estimate $\mathbf{x}_0$ (the evaluation point of the residual r_k) with $\mathbf{b} = \mathbf{J}(\mathbf{x}_0)\mathbf{W}\mathbf{r}(\mathbf{x}_0)$ and $\mathbf{H} = \mathbf{J}(\mathbf{x}_0)^T\mathbf{W}\mathbf{J}(\mathbf{x}_0)$:

$$\begin{aligned} E_m(\boldsymbol{\delta} \boxplus \mathbf{x}_0) &\approx ||\mathbf{r}(\mathbf{x}_0) + \mathbf{J}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)||_2^2, \\ &= \mathbf{r}(\mathbf{x}_0)^T\mathbf{r}(\mathbf{x}_0) + 2(\mathbf{x} - \mathbf{x}_0)^T\mathbf{b} + (\mathbf{x} - \mathbf{x}_0)^T\mathbf{H}(\mathbf{x} - \mathbf{x}_0), \\ &= \underbrace{\mathbf{r}(\mathbf{x}_0)^T\mathbf{r}(\mathbf{x}_0) + \mathbf{x}_0^T\mathbf{H}\mathbf{x}_0 - \mathbf{x}_0^T\mathbf{b}}_{=\text{ const}} + 2\mathbf{x}^T(\mathbf{b} - \mathbf{H}\mathbf{x}_0) + \mathbf{x}^T\mathbf{H}\mathbf{x}. \end{aligned} \tag{4.80}$$

Dropping the constant terms yields a quadratic function on $\mathbf{x}$:

$$E_m \approx 2\mathbf{x}^T \underbrace{(\mathbf{b} - \mathbf{H}\mathbf{x}_0)}_{=\mathbf{b}'} + \mathbf{x}^T\mathbf{H}\mathbf{x}. \tag{4.81}$$

where $\mathbf{b}'$ is a linearization of $\mathbf{b}$, which is also written in this manner in [92]. Computing the derivative of E_m (4.81) with respect to $\mathbf{x}$:

$$\frac{\partial E_m}{\partial \mathbf{x}} = \mathbf{H}\mathbf{x} + \mathbf{b}' = \mathbf{0}, \tag{4.82}$$

results in a typical *GN*-system (3.68) of the form:

$$\mathbf{H}\mathbf{x} = -\mathbf{b}'. \tag{4.83}$$

We marginalize variables with the *SC* similar to [37, 41, 92] and drop any terms that would influence the sparsity pattern of $\mathbf{H}$. To apply the *SC*, we rewrite (4.83) in terms of the states we would like to marginalize $\mathbf{x}_m$ and the ones we would like to keep $\mathbf{x}_a$ and also split $\mathbf{H}$ and $\mathbf{b}$ accordingly:

$$\begin{bmatrix} \mathbf{H}_{aa} & \mathbf{H}_{am} \\ \mathbf{H}_{ma} & \mathbf{H}_{mm} \end{bmatrix} \begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_m \end{bmatrix} = \begin{bmatrix} \mathbf{b}'_a \\ \mathbf{b}'_m \end{bmatrix}. \tag{4.84}$$

We then apply the *SC* to arrive at:

$$\underbrace{(\mathbf{H}_{aa} - \mathbf{H}_{am}\mathbf{H}_{mm}^{-1}\mathbf{H}_{ma})}_{\mathbf{H}^*_{aa}}\mathbf{x}_a = \underbrace{\mathbf{b}'_a - \mathbf{H}_{am}\mathbf{H}_{mm}^{-1}\mathbf{b}'_m}_{\mathbf{b}'^*_a}, \tag{4.85}$$

and write in short form:

$$\mathbf{H}^*_{aa}\mathbf{x}_a = \mathbf{b}'^*_a, \tag{4.86}$$

By plugging (4.86) into (4.81), the final energy E_m around the current evaluation point $\mathbf{x}_a$ can be written as:

$$E_m(\mathbf{x}_a) = 2\mathbf{x}_a^T\mathbf{b}'^*_a + \mathbf{x}_a^T\mathbf{H}^*_{aa}\mathbf{x}_a, \tag{4.87}$$

where (4.87) only depends on $\mathbf{b}'^*_a$, $\mathbf{H}^*_{aa}$ and the states $\mathbf{x}_a$ remaining in the window but not on any of the marginalized states $\mathbf{x}_m$. Thus, we can discard all the marginalized

states $\mathbf{x}_m$ and just store $\mathbf{b}_a'^*$ and $\mathbf{H}_{aa}^*$ after marginalization. In all subsequent *GN* and marginalization steps, we add the energy E_m to the overall edge-based window energy $E_\mathcal{K}$, which also requires the tangent space of the linearization $\mathbf{x}_0$ to stay the same for all variables in E_m. Note that the *SC* is an exact representation and the only approximation comes from linearization in (4.80).

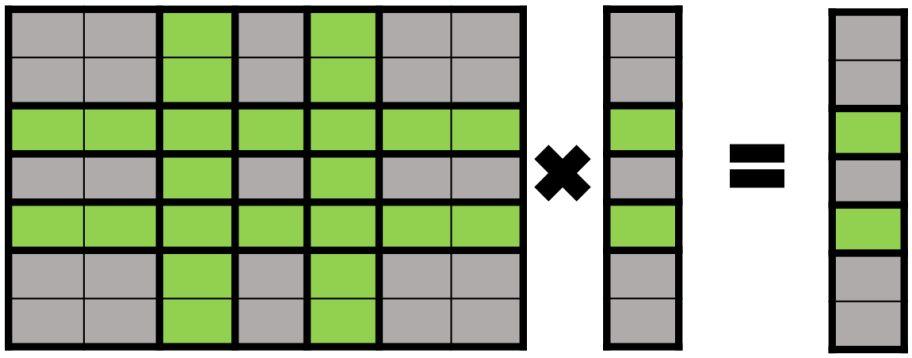
In the following, we will explain how the marginalization process works in practice for arbitrary rows and columns and then describe it for edge points and *KFs*. When working with a standard *GN* system $\mathbf{Hx} = \mathbf{b}$, the rows and columns we want to marginalize are typically at arbitrary positions in $\mathbf{H}$, $\mathbf{b}$ and $\mathbf{x}$ as shown in green in Figure 4.19a. In such cases, we move the respective elements to the lower right part of $\mathbf{H}$ and to the bottom of $\mathbf{x}$ and $\mathbf{b}$ (see Fig. 4.19b) and split them into blocks according to the states we want to keep and to marginalize. Before we marginalize a *KF*, we marginalize all its active edge points, then all the edge points not visible in the last two *KFs*, then remove all the *KF*'s observations completely from the system and finally marginalize the *KF*. Figure 4.20 shows the marginalization of $KF2$ and its active edge point E_2. To marginalize E_2, we follow the scheme presented in Figure 4.19 and move the respective row and column such that it can be marginalized. This marginalization also includes E_2's observations and reduces the matrix size to $(N-1)\times(N-1)$. Marginalizing a $KF2$ and including its observations would corrupt the sparsity of the lower right block by introducing off-diagonal blocks. Thus, we first drop all the observations to keep the sparsity pattern (see Fig. 4.20b) Similar to an edge, we follow the marginalization procedure (see Fig. 4.19) and shift the $KF2$'s row and column to the end. Then we split into blocks and marginalize to reduce the matrix size to $(N-2)\times(N-2)$ compared to the original matrix.

4.5 Global Mapper

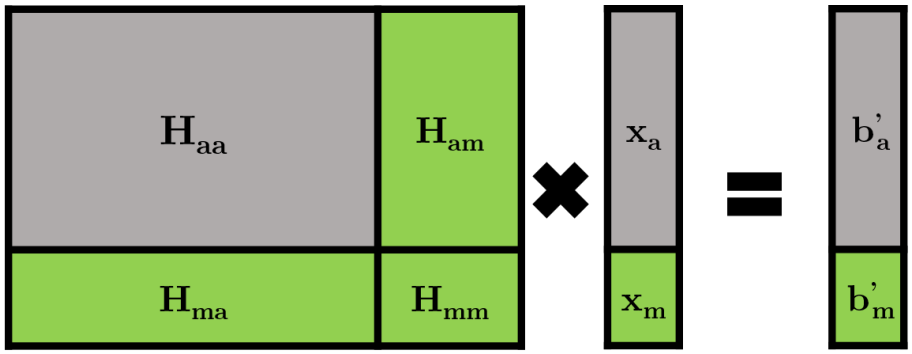
The main difference between *SLAM* and *VO* is that the former keeps a global map of the scene (see Fig. 4.1). RESLAM [134] implements a global mapper, which represents the global map in the form of a pose graph, where the nodes are *KFs* and the edges in between are relative constraints. One of the central challenges for the global mapper is to find and verify potential candidates for loop closure and relocalization. It comprises the following three components:

- Fern-based place recognition, which computes, matches and keeps a list binary descriptors for *KF*.
- Relocalization to continue after a stopping the system
- Loop closer to reduce the global drift when a place is revisited

In the next sections, we will describe these three components in greater detail.



(a) Marginalize two arbitrary rows and columns.



(b) Shift to the end and divide into blocks.

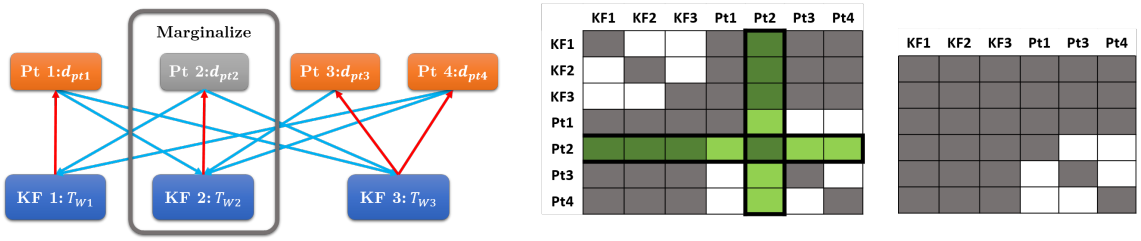
Figure 4.19: The typical *GN* system $\mathbf{H}x = b$. (a) To marginalize the rows and columns depicted in green, (b) they are moved to the lower right part of $\mathbf{H}$ and the bottom of $\mathbf{b}$ and $\mathbf{x}$.

4.5.1 Fern-based Place Recognition

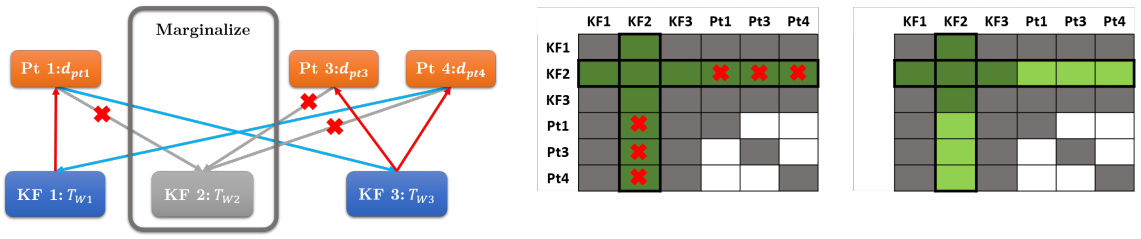
The Fern database is the central point of the global mapper and accessed by both, the relocalizer and loop closing modules. In RESLAM [134], we follow the place-recognition approach presented by Glocker et al. [55], which is also used in many state-of-the-art systems [78, 159] and based upon *Random Ferns* [122]. There are many alternative place recognition approaches and we refer to [103, 160] for a detailed survey.

To compute a Fern descriptor, we first sample at N_f random image coordinates, then evaluate if the value at this coordinate is below a threshold θ_f and finally combine all those evaluations into a binary string. Figure 4.22 visualizes the Fern descriptor computation with one code block at one position. For each KF, we compute a Fern code block at these N_f randomly sampled image coordinates separately for the red, green and blue channel and optionally for the depth image (see Fig. 4.22(2)). Note that the N_f image coordinates are randomly chosen once in the beginning and remain constant throughout the sequence. The n-th code block $\mathbf{b}_p$ at a position $\mathbf{p}$ for a frame's f RGB image is computed as depicted in Figure 4.22(2) and can be mathematically described as:

$$\mathbf{b}_{f,n}(i) = [\![\mathbf{I}_i(\mathbf{p}) < \theta_f]\!]; \quad i = 0, 1, 2, \tag{4.88}$$



(a) Marginalize **Pt 2**.



(b) Drop all measurements of **KF 2**.

(c) Marginalize **KF 2**.

Figure 4.20: Marginalization of **Pt 2** and **KF 2**: (a) Marginalize **Pt 2**, then (b) drop all the measurements (observations) of **KF 2** by setting them equal to 0 and finally (c) marginalize **KF 2**.

where the binary nature of the Fern descriptor comes from the Iverson bracket $[\![.]\!]$ (see Sec. C.2.1) and $I_i,\ i = 0, 1, 2$ represents the RGB channels. In practice, we choose $N_f = 500$ and evaluate on a down-sampled 40×30 RGB image and optionally also on the depth map. One code block is depicted in Figure 4.22(3), where the channels are marked in their respective colors. Finally, the complete Fern descriptor b_f is the concatenation of the individual blocks $\mathbf{b}_p, n$:

$$\mathbf{b}_f = \mathbf{b}_{p,0} ... \mathbf{b}_{p,N_f}. \tag{4.89}$$

From this compact representation, the similarity between two *KFs* i and j is given by the block-wise Hamming distance [55]:

$$\text{BlockHD}(\mathbf{b}_i, \mathbf{b}_j) = \frac{1}{N_f} \sum_{k=1}^{N_f} \mathbf{b}_{i,k} \equiv \mathbf{b}_{j,k}, \tag{4.90}$$

where $\equiv$ returns 0 if there is a difference between two blocks and 1 otherwise. (4.90) counts

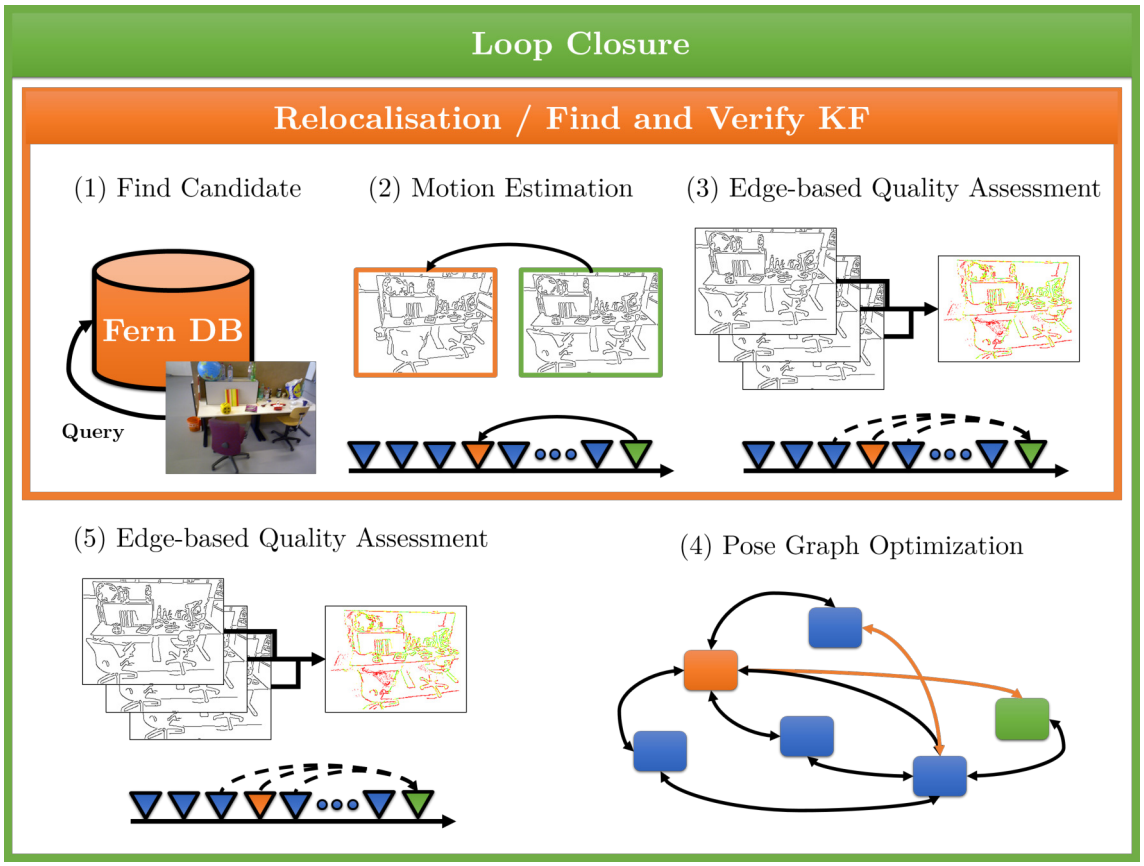


Figure 4.21: Relocalization and Loop Closing are two closely related task in the as both have a *Find and Verify KF* step. Loop Closing additionally requires *PGO* and one *EBQA* step.

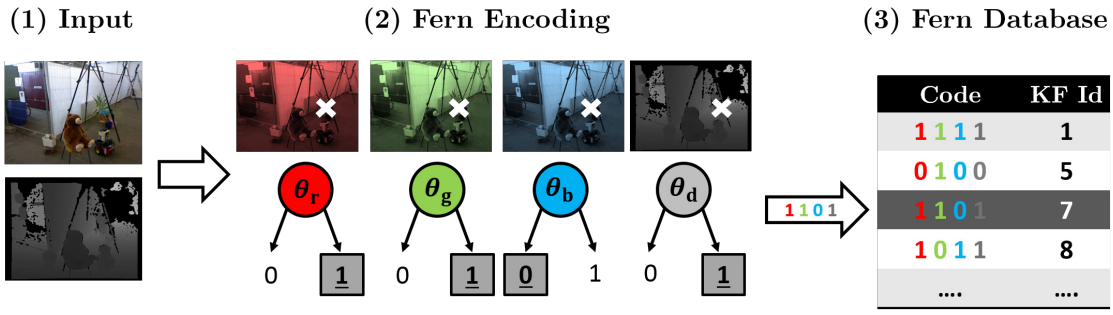


Figure 4.22: (1) Each *KF* is sampled at N positions separately for the red, green and blue channel and optionally for the depth image (2) If the sampled value is below a threshold θ, the computed binary descriptor is 0 and otherwise 1 at the respective position. (3) This descriptor is then be matched to all descriptors in the database.

the number of different blocks and then normalizes it by the total number of blocks N_f to map it to a similarity score in the range $[0, 1]$, i.e. a percentage. The higher the score, the greater the similarity between the two *KFs*.

Instead of keeping all the *KFs* ever created, we instead aim to maintain a database of distinct *KFs* to keep the global map optimizable in real-time. For each new *KF*, we compute the similarity score to each *KF* already stored in the database using (4.90) and add it if all the scores are smaller than the harvest threshold θ_h. In our experiments, we found that $\theta_h = 0.2$, i.e. 20 % difference to all *KFs* in the database, is a reasonable choice to maintain the *KFs* well-distributed in the scene. Table 4.2 shows the difference between the number of stored *KFs* and the total number of created *KFs*, where a typical office scene has only 90 - 100 *KFs*.

Speed is one of the greatest advantages of the Fern-based method [55]. Computing the similarity score between two *KFs* is a very fast process and going through a database of thousands of *KFs* is possible within several milliseconds. Further, the descriptor only involves thresholding intensity values, which is computationally far less expensive than more sophisticated descriptors such as ORB [127] used in [115]. Another benefit of [55] is that it adapts to the environment since the database is built based on the current scene and does not require any pre-trained vocabulary as others [56, 115]. Further, the generated Fern database can be easily stored and reloaded to enable map relocalization at a later stage. One downside of the discussed method is that the location accuracy varies with the random choice of sample positions in each run and the ideal sampling pattern is specific to the scene. In practice, we found this to be a negligible problem because our number of sampled positions N_f is roughly 40% of the downscaled 40×30 image.

4.5.2 Relocalization

Our proposed methods are very robust on the benchmark datasets but in practice, several situations can occur that cause relative motion estimation to fail. Section 1.1 already discusses some potential problems and Figure 1.4 depicts some typical causes, where large parts of a scene do not have sufficient texture for edge detection or depth information is missing, e.g. due to sunlight (see Fig. 1.4f), or the sensor is fully or partially covered or the environment is are not static, e.g. a person walks in front of the scene. For the system to continue in such cases, relocalization capabilities are essential. However, relocalization is not only relevant in failure cases but also during tasks such as 3D reconstruction, e.g. for a person it is difficult to record a scene and while simultaneously inspecting the generated global map. Thus, the capability to stop and continue at a later point in time is critical for practical use.

Due to the lack of a global mapper in our *VO* systems, RESLAM is the only one with relocalization capabilities. RESLAM [134] considers the pose estimation as failed if either the average reprojection error is higher than θ_{reloc} and/or the number of reprojected edges that are inliers is lower than N_{inlier}. Relative pose estimation follows a coarse-to-

fine scheme and whenever it fails on one of the lower levels, it restarts the estimation on the next higher level with the previous initialization. Only if it fails on the highest level, RESLAM considers it as failed and processes the next frame. In case the pose estimation fails again, the system switches to relocalization mode (see Fig. 4.1). Since the motion estimation typically does not fail from one frame to the next but rather degrades over several frames, it is also required to remove all the frames in the local window. If the system is paused, no frames are removed.

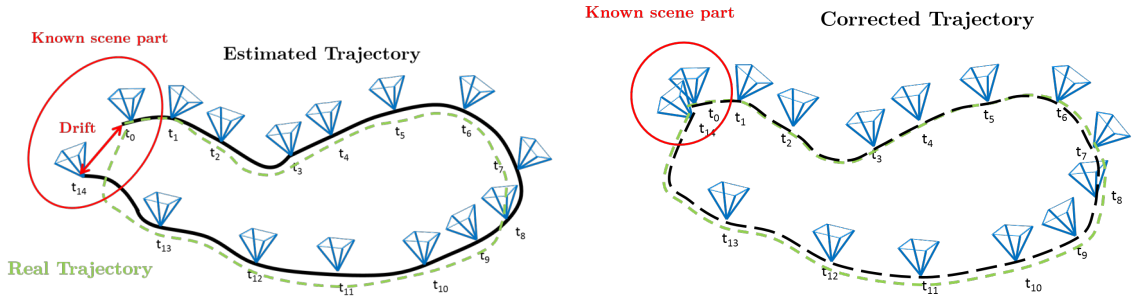
Relocalization proceeds in three steps as shown in Figure 4.21. (1) At the core of the loop closure is the Fern-based place-recognition introduced in Section 4.5.1. The main difference between our implementation and [55, 78, 159] is that they find a global world pose associated with the most similar *KF* and utilize it as initialization for global model alignment, while we query the database to find a candidate *KF* $\mathcal{F}_{cand}$ (see Fig. 4.21(1)). A *KF* is a potential candidate if the similarity score is below θ_{lc} and in case multiple *KFs* fulfill the criteria, we select the top match. (2) We then estimate the relative motion $\mathbf{T}_{curr,cand}$ by minimizing the edge-based error (4.6) presented in Section 4.1.1 starting from $\mathbf{T}_{curr,cand} = \mathbf{I}_4$ (see Fig. 4.23a) (3) To reduce wrong matches from the Fern database, we immediately assess the motion quality using the adjacent *KFs* to $\mathcal{F}_{cand}$ and apply the *EBQA* introduced in Section 4.2. Only if the estimate is valid, relocalization is considered successful. We then apply the *EBQA* introduced in Section 4.2 and only continue if the estimate is valid (see Fig. 4.23b).

(a) Relative camera motion estimation **(b)** *EBQA* verification

Figure 4.23: For a candidate *KF* from the Fern database, (a) estimate the relative camera motion between the current *KF* (green) and the candidate *KF* (orange) and (b) verify the estimated camera motion with our *EBQA* using the candidate and its adjacent *KFs*.

4.5.3 Loop Closure

Even with the additional local *BA* drift still accumulates over the whole sequence introducing a difference between *estimated* and *real* trajectory (see Fig. 4.24a). Thus, it is necessary to correct the drift through loop closure, whenever a previously seen scene part is revisited (see Fig. 4.24b). Loop closure is a long-standing problem in SLAM and in this section, we describe how we address loop closure in RESLAM [134] with an edge-based algorithm. Figure 4.21 shows our five-step loop closure pipeline, which (1) finds a candidate *KF* in the database, (2) then estimates the relative motion between the candidate and query *KF*, (3) assess the quality of the estimate, (4) performs *PGO* and (5) again assesses the quality. Since the first three steps are identical to the relocalization, we refer to Section 4.5.2 for a detailed description. In (4), we set up a pose graph, where the *KFs*



(a) Global drift without loop closure **(b)** Trajectory after loop closure

Figure 4.24: Whenever we find a candidate KF in the Fern database, (a) we estimate the relative camera motion between the current *KF* (green) and the candidate *KF* (orange). (b) Is the edge-based verification using the *EBQA* method presented in Section 4.2 with the candidate and its adjacent *KFs*.

with their world poses are the nodes and the edges are relative transformations between *KFs*. *PGO* is a well-studied problem and discussed in many publications [17, 60, 64, 144]. Figure 4.25a shows the pose graph for a toy problem, with the current *KF* (green), the candidate *KF* (orange), other *KFs* (blue) and the relative transformations from local *BA* (black) and loop closure constraints (orange). In practice, the pose graph is typically many times larger with hundreds of *KFs* and thousands of connections (see Fig. 4.25b). The *PGO* optimization aims to distribute the drift to all the *KF* such that their refined world poses minimize the relative transformations. For two *KFs* i and j we aim to refine their world poses $\hat{\mathbf{T}}_{wj}$ and $\hat{\mathbf{T}}_{wi}$. The constraint or *constant* measurement is a relative transformation $\mathbf{T}_{ji}$ between them coming either from a loop closure or from the windowed optimization (see Sec. 4.4.2). In the ideal case, the measurements and world poses are identical:

$$\mathbf{I}_4 = \mathbf{T}_{ij}\hat{\mathbf{T}}_{wj}^{-1}\hat{\mathbf{T}}_{wi} \tag{4.91}$$

Moving $\mathbf{T}_{ij}$ to the other side yields:

$$\begin{aligned}
\mathbf{T}_{ij}^{-1} &= \hat{\mathbf{T}}_{wj}^{-1}\hat{\mathbf{T}}_{wi} \\
\mathbf{T}_{ij}^{-1} &= \begin{bmatrix} \hat{\mathbf{R}}_{wj} & \mathbf{t}_{wj} \\ \mathbf{0}^T & 1 \end{bmatrix}^{-1} \begin{bmatrix} \hat{\mathbf{R}}_{wi} & \mathbf{t}_{wi} \\ \mathbf{0}^T & 1 \end{bmatrix} \\
\mathbf{T}_{ij}^{-1} &= \begin{bmatrix} \hat{\mathbf{R}}_{wj}^T & -\hat{\mathbf{R}}_{wj}^T\mathbf{t}_{wj} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{R}}_{wi} & \mathbf{t}_{wi} \\ \mathbf{0}^T & 1 \end{bmatrix} \\
\mathbf{T}_{ij}^{-1} &= \begin{bmatrix} \hat{\mathbf{R}}_{wj}^T\hat{\mathbf{R}}_{wi} & \hat{\mathbf{R}}_{wj}^T(\mathbf{t}_{wi} - \mathbf{t}_{wj}) \\ \mathbf{0}^T & 1 \end{bmatrix} \\
\begin{bmatrix} \mathbf{R}_{ij}^T & -\mathbf{R}_{ij}^T\mathbf{t}_{ij} \\ \mathbf{0}^T & 1 \end{bmatrix} &= \begin{bmatrix} \hat{\mathbf{R}}_{ji} & \hat{\mathbf{R}}_{wj}^T(\mathbf{t}_{wi} - \mathbf{t}_{wj}) \\ \mathbf{0}^T & 1 \end{bmatrix}
\end{aligned} \tag{4.92}$$

We can rewrite the rotational part of (4.92) as:

$$\begin{aligned} \mathbf{R}_{ij}^T &= \hat{\mathbf{R}}_{ji} \\ \mathbf{I}_3 &= \mathbf{R}_{ji}^T \hat{\mathbf{R}}_{ji} \end{aligned} \tag{4.93}$$

and the translational part as:

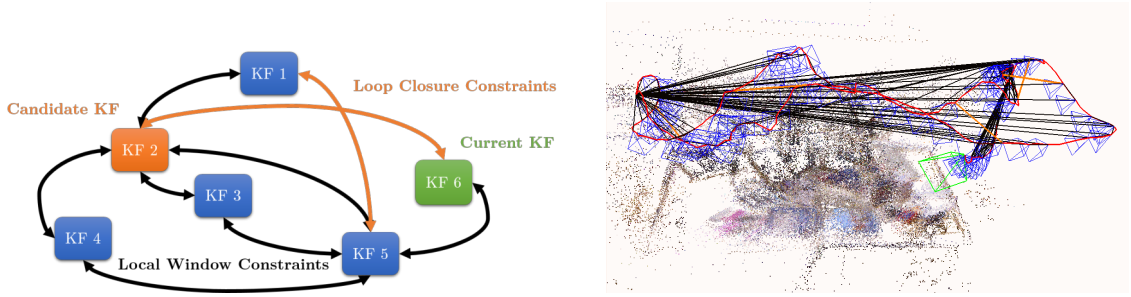
$$\begin{aligned} -\mathbf{R}_{ij}^T \mathbf{t}_{ij} &= \hat{\mathbf{R}}_{wj}^T (\mathbf{t}_{wi} - \mathbf{t}_{wj}) \\ \mathbf{0} &= \hat{\mathbf{R}}_{wj}^T (\mathbf{t}_{wi} - \mathbf{t}_{wj}) + \mathbf{R}_{ij}^T \mathbf{t}_{ij} \end{aligned} \tag{4.94}$$

Since the rotation matrices are over-parametrized, we rely on a unit quaternion parametrization (see Sec. 3.2.2). Following [64], we define the residual errors as:

$$\begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} 2\mathbf{q}_{ji}^{-1} \hat{\mathbf{q}}_{ji} \\ \hat{\mathbf{R}}_{wj}^T (\mathbf{t}_{wi} - \mathbf{t}_{wj}) + \mathbf{R}_{ij}^T \mathbf{t}_{ij} \end{bmatrix}, \tag{4.95}$$

where the rotation matrices are computed from the quaternion as shown in Section 3.2.2. We solve the *PGO* with the Ceres [2] using a *LM* scheme [93, 107, 121], a sparse Cholesky factorization [19] and a robust Huber-loss function. In contrast to other approaches that keep the first one or two *KFs* fixed, we instead fix the world poses of the *KFs* that are currently active in the local window.

Since even one wrong loop closure can seriously degrade the overall quality of the map, it is crucial to again assess the quality of the loop candidate identical to step (3). Only after this final assessment considers the loop closure as valid, the global map is updated with the refined world poses. Throughout our experiments, we found that two *EBQA* verifications before and after loop closure is very successful in preventing wrong loop closures.



(a) Simple Pose Graph **(b)** Complete Pose Graph

Figure 4.25: (a) A simple pose graph with 6 *KFs*, where the *KFs* are the nodes and the edges are the relative motions from either the local map (black) or from a loop closure (orange). (b) The pose graph created from the *fr1/desk* sequence from the TUM RGBD dataset [147].

Loop closure is one of the most important components of a *SLAM* as it is able to greatly reduce the global drift, which improves overall map quality and consistency. However, one

of the downside of loop closure is that it distributes the drift or error over all the *KFs*, which might reduce the quality in well-estimated scene parts. This problem can be mitigated by incorporating measurements from other sensors, e.g. an IMU, as an additional constraint in the *PGO*. Chapter 6 presents the influence of the loop closure on the accuracy of RESLAM.

4.6 Conclusion

This chapter summarizes our most important contributions to the field of *VO* [132, 133] and *SLAM* [134]. At the beginning of this chapter, we propose a direct edge-based relative motion estimation, which aligns the edge detections of two frames (see Section 4.1.1). In contrast to direct methods based on photo-consistency, edges are less influenced by illumination changes compared to raw intensity values and the sparse representation offers a large convergence basin. A great advantage over classical feature-based methods is that no costly descriptor computation is necessary and the direct nature of our method requires no error-prone matching step. We formulate the edge-based relative motion estimation as an energy minimization problem on the DT, which allows the evaluation of a single residual in $O(1)$ time making the whole system able to run at around 50 - 60 fps on a laptop computer. Section 4.1.2 describes how to efficiently solve this energy minimization with an Iteratively Reweighted Least-Squares (IRLS) method including also the derivations of the Jacobians. Relying solely on edges can be problematic in indoor scenes, since they often contain scarcely textured areas, e.g. walls and ceilings, where no or only very few edges are detected. However, the RGBD sensor still records a dense depth map in these cases. Thus, we propose to additionally utilize the depth map in the form of a geometric error for relative motion estimation (see Section 4.1.3). The geometric error is formulated as point-to-plane distance between the surfaces of two depth maps and minimized with an *ICP* algorithm. In Section 4.1.4, we again derive the Jacobians and then show to jointly optimize and edge-based and geometric error. This joint optimization greatly increases accuracy and robustness in difficult scenes mainly for two reasons: (1) the geometric term typically produces dense residuals over the whole image, thereby enforcing structural consistency when edges are concentrated in one part of the image and (2) if there are mostly planar surfaces, moving along planes is nearly cost-free for the geometric error but costly for edge-based term.

One of our central contributions is the Edge-based Quality Assessment presented in Section 4.2, which classifies relative motion estimates into valid and poor. It reprojects edges from previously estimated frames into the current frame with the motion estimate be assessed and generates a histogram of overlapping edges. Whenever many more edges overlap than not, the estimate is considered valid and otherwise poor. While the idea is simple, our *EBQA* proved to be very robust, fast and versatile, e.g.in [134] it verifies loop closure and relocalization candidates and in [132, 133] it decides when to insert a new *KF* . The creation and management of *KFs* are critical points in most system and we

propose two different approaches for our *VO* (see Sec. 4.3.1) and *SLAM* (see Sec. 4.3.2) systems. For *VO*, we aim to keep a *KF* as long as possible to reduce the drift and also avoid too many costly *KF* creations. In contrast, for *SLAM* refines all the *KFs* within a local window and many close *KFs* increase accuracy and stability. Therefore, initially many *KFs* are created and culled later.

Section 4.4 proposes edge-based local mapping, which greatly improves accuracy and consistency over the whole trajectory. It estimates the relative motion with respect to several *KFs* and not just the last frame or *KF* as in the pure *VO* setting. We first discuss the optimization of the most recent *KF* with respect to spatially close *KFs*, which closes small loops and is successfully used in [132]. However, since it only refines the most recent *KF*, new measurements cannot improve old estimates. Thus, in Section 4.4.2, we propose a sliding window *BA* over all model parameters, including camera intrinsics, *KF* poses and the depth of the individual edge points. We first show how to build up a factor graph for a direct edge-based method without known correspondences on the example of a toy problem. Then, we demonstrate how to efficiently solve the rather large but sparse equation system with a *GN* method and the *SC*. Since a robot or agent continuously explores new areas, the number of states in the window increases quickly. To keep it at a size, which is still computable in real-time, we marginalize old *KF* and edge points, while keeping the sparsity of the equation system. We discuss the individual steps of the marginalization process and derive the respective equations.

The main difference between *VO* and *SLAM* is the capability of the latter to recognize previously visited areas, close loops and perform relocalization. Section 4.5 describes all three tasks in the form of a global mapper as implemented in [134]. A Fern-based place recognition [55] (see Sec. 4.5.1) computes a binary descriptor from an RGB image and optionally a depth map. These descriptors can be matched for thousands of frames within a few milliseconds to find similar *KFs*, which are then candidates for loop closure or relocalization (see Sec. 4.5.1). Similar frames are candidates for relocalization or loop closure and both work nearly identical. Relocalization estimates a relative motion between the candidate and the current frame and verifies with the previously discussed *EBQA*. This step is identical to the first part of the loop closure procedure. After verifying the candidate, the global mapper generates a pose graph containing *KFs* as nodes and relative constraints from the local mapping, previous loop closures and the current estimate as edges. *PGO* distributes the accumulated drift over the graph and refines all the *KF* poses. Since even one wrong loop closure often destroys the complete model, we propose to again verify the result after *PGO* with the *EBQA* and only incorporate the loop closure if the estimate is valid.

5 Edge Detection

Contents

Edge detection is one of the most fundamental challenges in computer vision and has been extensively researched for the past decades. While humans can easily find meaningful or natural edges such as object boundaries, this is still a very challenging task for a computer and several techniques exist. There are several requirements for an edge detector: robustness to (i) illumination changes, (ii) motion blur and (iii) noise, as well as (iv) repeatability, i.e. detecting the same edges in consecutive frames. Of course, for real-time systems the computation speed is the deciding factor.

The choice of the appropriate edge detector for our proposed edge-based methods is still an open topic. This chapter will give a brief overview of various edge detection methods and we refer to the surveys [58, 166] for a broader overview. Since our proposed edge-based methods typically run in a coarse-to-fine scheme, we first show why simply detecting edges on every scale level can lead to problems. To tackle this problem, we propose edge enhancement, which transfers edges from higher pyramid levels to lower ones to improve edge detections and then introduce an even faster but equally robust technique, which only requires the costly edge detection on the highest scale and then down-scales the Distance Transform (DT). We conclude this chapter with qualitative and quantitative evaluations of various edge detectors.

Edge detectors can be roughly divided into traditional methods, learning- and deep-learning-based methods. Traditional methods such as Sobel [83], zero-crossings [108], Pb [109], gPb [3] or Canny [16] detect edges based on intensity, color or texture changes and typically require careful threshold setting. These detectors find an abundance of edges in highly textured regions, which often do not correspond to object boundaries

but to fine-grained texture and can bias pose estimation. Even though Canny [16] has some shortcomings, it is still one of the most used edge detectors mainly due to the widely available implementations, its fast speed of around 400 fps and the reasonable performance.

Another type are learning-based detectors that rely on supervision and hand-crafted features. Dollar and Zitnick [28, 29] introduced Structured Edges (SE) that delivered state-of-the-art results in terms of meaningful edge detection while being relatively fast (around 12 fps on a CPU). SE comprise (i) a large number of manually designed features, (ii) the fusion of multi-scale responses and (iii) the incorporation of structural information. While SE's results were impressive at the time, the manually designed features make it difficult to adapt to new problems and require expert knowledge for in-depth modifications.

Deep learning has revolutionized many fields of computer vision and has also found its way into edge detection. Xie et al. [161] showed impressive improvements with their *Holistically-Nested Edge Detection (HED)*, which combines (i) holistic image training and prediction and (2) nested multi-scale feature learning performing deep layer supervision to guide early classification results. They proposed an RGB model trained on BSDS500 [3] and an RGBD model with an additional depth input channel trained on NYUv2 [141]. Even with the recent advances in hard- and software the inference speed still is far below real-time In *Deep Contour (DC)* [139], Shen et al. divide the Ground Truth (GT) from the BSDS500 [3] dataset into different shape classes and train a Convolutional Neural Network (CNN) to predict those classes. They divide an image into small 45×45 RGB patches and pass them through the *CNN*, which is computationally expensive and renders this approach unusable for real-time application. *Bi-Directional Cascade Network for Perceptual Edge Detection (BDCN)* [66] employs a shallow *CNN* structure making it one of the fastest method. With its bi-directional cascade structure, it enforces each layer to learn edges specific to a scale. To also exploit cues on multiple scales with their shallow *CNN* structure, there is a scale enhancement mode, which comprises multiple parallel convolutions with various dilation rates. *Richer Convolutional Features for Edge Detection (RCF)* [100, 101] is an accurate and fast method running at up to 30 fps. RCF applies a novel deep structure, which encapsulates semantic and fine detail features from all the convolutional layers instead of just the last ones. Even though deep learning methods do not require hand-crafted features like SE, the network architecture still has to be carefully designed and probably the biggest disadvantage is the huge amount of annotated data to train the model. Another issue is that these methods are often too slow for real-time applications even when using a GPU.

5.1 Edge Detection on Multiple-Scales

Direct methods typically optimize in a coarse-to-fine scheme over several pyramid levels to cope with large inter-frame motions. Most edge-based methods [89, 154, 165] follow this approach and detect edges on each pyramid level, which we refer to as Multi-Level Edge Detection (MLD).While typically an abundance of edges is detected at the highest level of

the image pyramid, fine-grained parts are smoothed over in lower levels and do not produce edge responses anymore. Figure 5.1 shows that even in scarcely textured areas edges are detected on the highest resolution, i.e. Level 0 at a resolution of 640×480. However, after down-scaling to the next lower resolution, i.e. Level 1 at a resolution of 320×240, there is already a significant decline in the number of detected edges. Large edge-less areas introduce a bias in the motion estimation since only the areas with edge detections are aligned. In indoor scenes, edge detectors rely on fine texture in areas with little intensity differences, e.g. walls, floors. While some of the machine-learned edge detectors discussed in this chapter suffer to a much lesser degree, this is still an important topic. In this section, we propose two methods to address this problem: (1) Edge Enhancement (EE) to transfer edge detections from a higher to a lower resolution [132] and (2) Single-Level Edge Detection (SLD) [134], which only requires an edge detection on the highest resolution, then computes the DT and finally down-scales it.

5.1.1 Multi-Level Edge Detection with Edge Enhancement (MLD+EE)

The goal of EE is to transfer edges from a higher pyramid level to a lower one to compensate for edges lost due to smoothing. Figure 5.1 shows our proposed algorithm [132], which first computes a covering image and then copies edges from a higher resolution into insufficiently covered region. One of the key components is to find such regions on all but the highest resolution level. For this purpose, we split the image into a discrete grid of $N \times N$ patches and count the number of edge detections in each patch. A patch is insufficiently covered if the number of edge detections within it is below Θ_E. The result is a covering image, which represents the spatial distribution of edges in terms of sufficiently (green) and insufficiently covered areas (see Fig. 5.1). If a patch is insufficiently covered, we enhance the edge image by transferring detections from a higher resolution image into the lower resolution image. A comparison between *MLD* with and without EE in Figures 5.1 and 5.2 demonstrates the increase in the number of detected edges.

5.1.2 Single Level Edge Detection (SLD)

Detecting edges on each pyramid level (*MLD*) does not just introduce problems regarding robustness but also comes at significant computational cost. While the computational cost is very low for fast edge detectors such as Canny [16], modern machine-learned edge detectors are far more complex and often barely work in real-time. Other computations such as DT for the Keyframes (KFs) and EE for every single frame additionally slow down the system. We propose a novel Single-Level Edge Detection, which is significantly faster than *MLD* and *MLD*+EE and simultaneously addresses the robustness issue. Instead of detecting edges on each pyramid level separately, *SLD* only detects edges and explicitly computes the DT on the highest level. For edge-based motion estimation (see Sec. 4.1.1), the edges are then reprojected to lower levels via the intrinsic camera parameters, i.e. to go from highest to second highest, divide the intrinsic camera parameters by a factor of

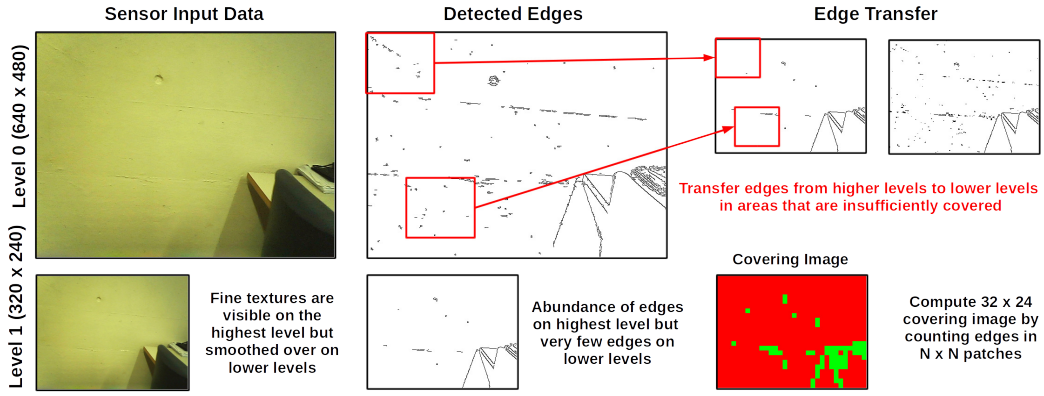


Figure 5.1: Edge enhancement improves the spatial distribution of the detected edges by computing a covering image to find insufficiently covered regions (depicted in red). These regions are then enhanced by transferring edge detections from a higher to the lower resolution level.

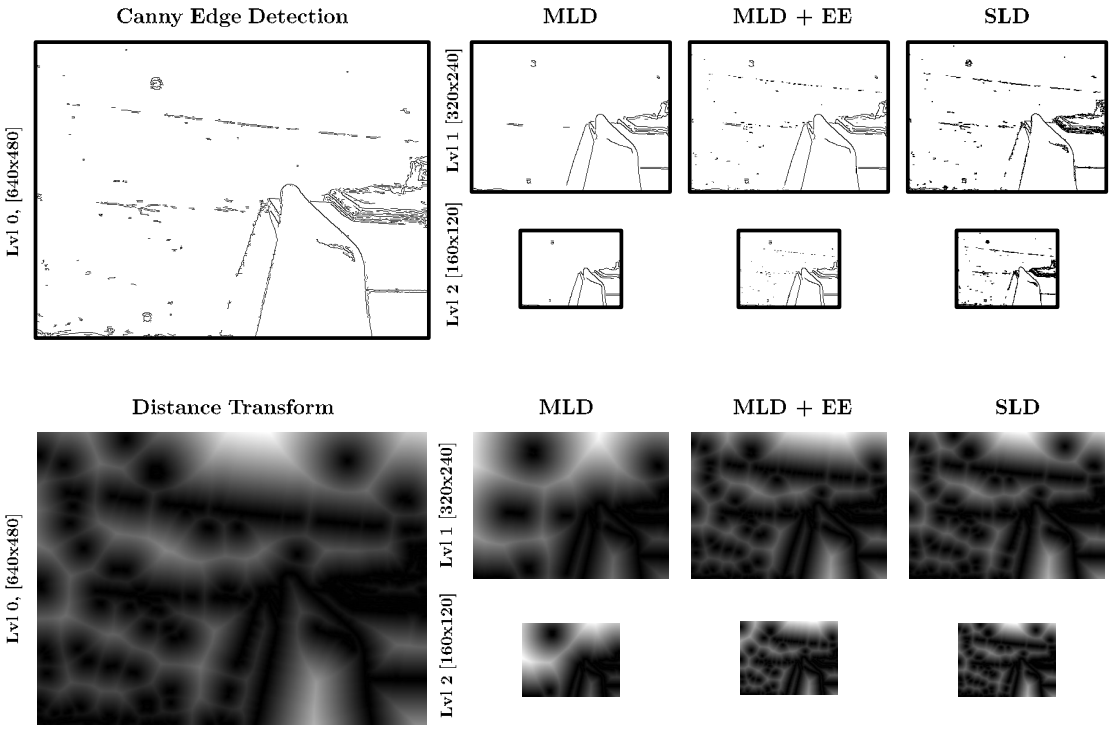


Figure 5.2: Edges and *DT* for Multi-Level Edge Detection (*MLD*), *MLD* + Edge Enhancement (*EE*) [132] and Single-Level Edge Detection (*SLD*) [134]. The edges of *SLD* are just for visualization purposes and never computed since the *DT* is down-scaled directly (see Sec. 5.1.2). The improvement reached by *MLD* + Edge Enhancement (*EE*) over *MLD* is evident since many more regions have edge detections even on the lower resolution levels. The most recent *SLD* shows nearly the same results as *MLD* + Edge Enhancement (*EE*) while being significantly faster.

2 in the projection function π. To avoid processing an abundance of edges on the lower levels, only every second edge is used on the lower levels.

Further, *SLD* avoids the costly explicit re-computation of the DT on each pyramid level $\mathbf{D}_n$ by down-scaling it from a higher resolution level $\mathbf{D}_{n-1}$. Assuming three scale levels $N = 3$, the DT at the highest resolution level $\mathbf{D}_0$ is simply computed from the edge detection: For the lower levels $n = 1, 2$, $\mathbf{D}_n$ can be computed as the mean over a pixel patch of size N_p from $\mathcal{D}_{n-1}$:

$$\mathbf{D}_n(x, y) = 0.5\frac{1}{N_p}\sum_{i=0}^{1}\sum_{j=0}^{1}\mathcal{D}_{n-1}(2x + i, 2y + j), \qquad n = 1, 2 \tag{5.1}$$

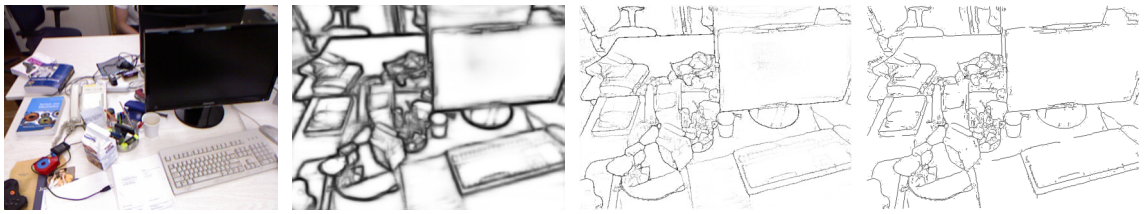
where N_p is the size of the patch at the higher resolution level and 0.5 an additional scale factor since the pixel distances halve between levels. To avoid processing an abundance of edges on the lower levels, we only use every second edge on the lower levels and process all of them on the highest.

Figure 5.2 shows a comparison between classical *MLD*, i.e. edge detection on every pyramid level, *MLD* + *EE* and *SLD*. The difference between *MLD* and our proposed methods is easily recognizable in the DT and the edge detections. Note that for *SLD* the edge detections on the lower levels are just for visualization purposes and never computed. It is interesting to see that there is hardly any difference in the DT between *MLD+EE* compared to our recent *SLD*, even though the latter comes at much lower computational cost. The strength of our method to only detect edges on one level becomes apparent, when computationally more expensive machine-learned edge detectors are used.

5.2 Evaluation of Edge Detectors

In this section, we show qualitative and quantitative results for various edge detectors We select the still popular Canny [16] as an example of a traditional edge detector. It is also used in our proposed methods whenever real-time processing is required. From the vast amount of learning-based detectors, we choose SE [28, 29] since it was the best performing detector before the advent of deep learning, and RCF [100, 101], BDCN [66], HED [161] and DC [139] from the field of deep learning. A typical edge detection is binary, i.e. edge and not edge, but all the learning-based methods compute a probability map (see Fig. 5.3b). Since this probability map is unusable for our proposed edge-based methods, we first perform non-maximum suppression as depicted in Figure 5.3c to get a reduced probability map with values between 0 and 1. We then threshold the map at a probability value $\Theta_{ml} = 0.2$ to get binary edge detections, remove objects smaller than 5 px and finally apply a thinning algorithm [90] (see Figure 5.3d).

The Canny implementation is the one released in OpenCV [13]. Throughout our experiments, we run with a kernel size of 3 and lower and upper hysteresis threshold of 100 and 150 , except for Table 5.1, where we also run with 60 and 80. For the learning-based



(a) RGB Image **(b)** Probability map **(c)** Non-max suppression **(d)** Thresholding and thinning

Figure 5.3: (a) Input RGB image from which learning-based edge detectors such as HED [161] (b) compute a probability map of edges. This probability map is refined (c) by non-maximum suppression and (d) thresholding to a binary image followed by thinning to remove very small edge detections.

detectors, we rely on the implementations published alongside the respective publications but run the RGB-only models to enable a fair comparison.

5.2.1 Qualitative Evaluations

In this section, we show qualitative results to study the strengths and weaknesses of the various edge detectors. For this purpose, we present a total of four qualitative experiments, where **Experiment 1** and **Experiment 2** show the influence of illumination changes on the edge detections and **Experiment 3** and **Experiment 4** study the effects of motion blur. In the following, we will describe and discuss **Experiment 1** and **Experiment 2** as well as **Experiment 3** and **Experiment 4** jointly.

For **Experiment 1**, we sample four images with various levels of illumination from the *flicker* sequence in HDRFusion dataset [96] (see Fig. 5.4). It is a synthetic dataset with high dynamic range, which makes it suitable for illumination evaluations. In **Experiment 2**, we set up a standard desktop scene, record with a statically mounted Orbbec Astra Pro sensor and vary illumination with dimmable light source. Illumination smoothly decreases over a second while the sensor records at 30 fps. We again sample four images at different illuminations and present the results in Figure 5.5. In Figures 5.5 and 5.4, the first row shows the RGB images, which change from bright to very dark, and the other rows depict the detections of the respective edge detectors. Canny performs well for the first three images in **Experiment 1** but only detects a reasonable amount of edges for the first image in **Experiment 2** and fails for the others. The main reason for the poor performance in the darker images is that Canny is susceptible to threshold settings, which are kept fixed throughout both experiments. Adapting the thresholds to the current illumination of the scene is not straightforward and slows down edge detection [43]. Similarly, HED and DC work well in the first three images but also suffer greatly once the illumination is low (see *column 4*). In **Experiment 2** their edge detections on all but the first image can also be considered unusable. For these cases, more variation in the training data could help the networks to learn a more robust edge detection. The best performing algorithms

in both experiments are SF, BDCN, and RCF, which detect edges in all the images in **Experiment 1** and also for the first three images in **Experiment 2**. The last and darkest image in **Experiment 2** is difficult for all methods because its signal-to-noise ratio is very low.

Experiment 3 is based on the *basement* sequence of the NYUv2 dataset [141] with only mild motion blur and a slowly moving person in the background (see Fig. 5.6) Qualitatively, there are not many differences and all detectors cope well with light motion blur. It is interesting to see that SE, HED, and DC all have problems in the left part of the image (see *column 1-3*). All three are not able to detect the corner, while Canny, RCF, BDCN have edge detections for the complete room geometry. Further, SE and DC have problems with segmenting the person standing on the left side behind the column. The problems with the left side of the image most likely stem from the low illumination in this part of the scene. Our final **Experiment 4** comprises a sequence of four images the fast-paced *fr1/desk* sequence from the TUM RGBD dataset [147]. Figure 5.7 depicts the selected RGB images in the first row and the results of the respective edge detectors in the other rows. Throughout the sequence, motion blur continuously increases from no motion blur in *column 1* until heavy motion blur in *column 4*, where object borders are nearly unrecognizable. The significant impact of motion blur on the edge detections becomes apparent when comparing *column 1* and *column 2* to the others. While all detectors show reasonable performance in *column 1* without motion blur, Canny, SE, and DC greatly suffer as depicted in *column 4*. For example, the of the screens completely disappear for Canny in *column 3* and *column 4*, and one side of the screen in the back for DC and SE. Not only do they detect very few edges, they also elongate edges along the motion directory. This is because edges that are oriented close to the motion direction are not blurred as much as the ones oriented normal to it, e.g. the sides of the screens. Elongated edges could lead to problem during motion estimation, since it is difficult to align them with edges from non-blurred images, e.g. aligning *column 1* with *column 3*. Another problem is that motion blur can introduce double detections at object borders, which makes alignment in these regions ambiguous. However, the state-of-the-art machine-learned edge detectors RCF, BDCN and the older HED show far more stable detections (see *column 2-4*).

Throughout our experiments, we observed that Canny is prone to detecting fine-grained structures that can clutter the scene and influence motion estimation. In **Experiment 1** *column 1-3*, Canny detects an abundance of edges on the front page of the magazine and the paintings and in *column 4* even the individual keys of the keyboard. In contrast, the other detectors mostly find the outlines of objects, which is desirable for motion estimation.

Figure 5.4: Four different RGB images from the HDRFusion dataset [96] with varying illumination and the corresponding edge detections. While all methods work well for the first three images, only SE, RCF, and BDCN deliver relatively constant edge detections even with varying illumination.

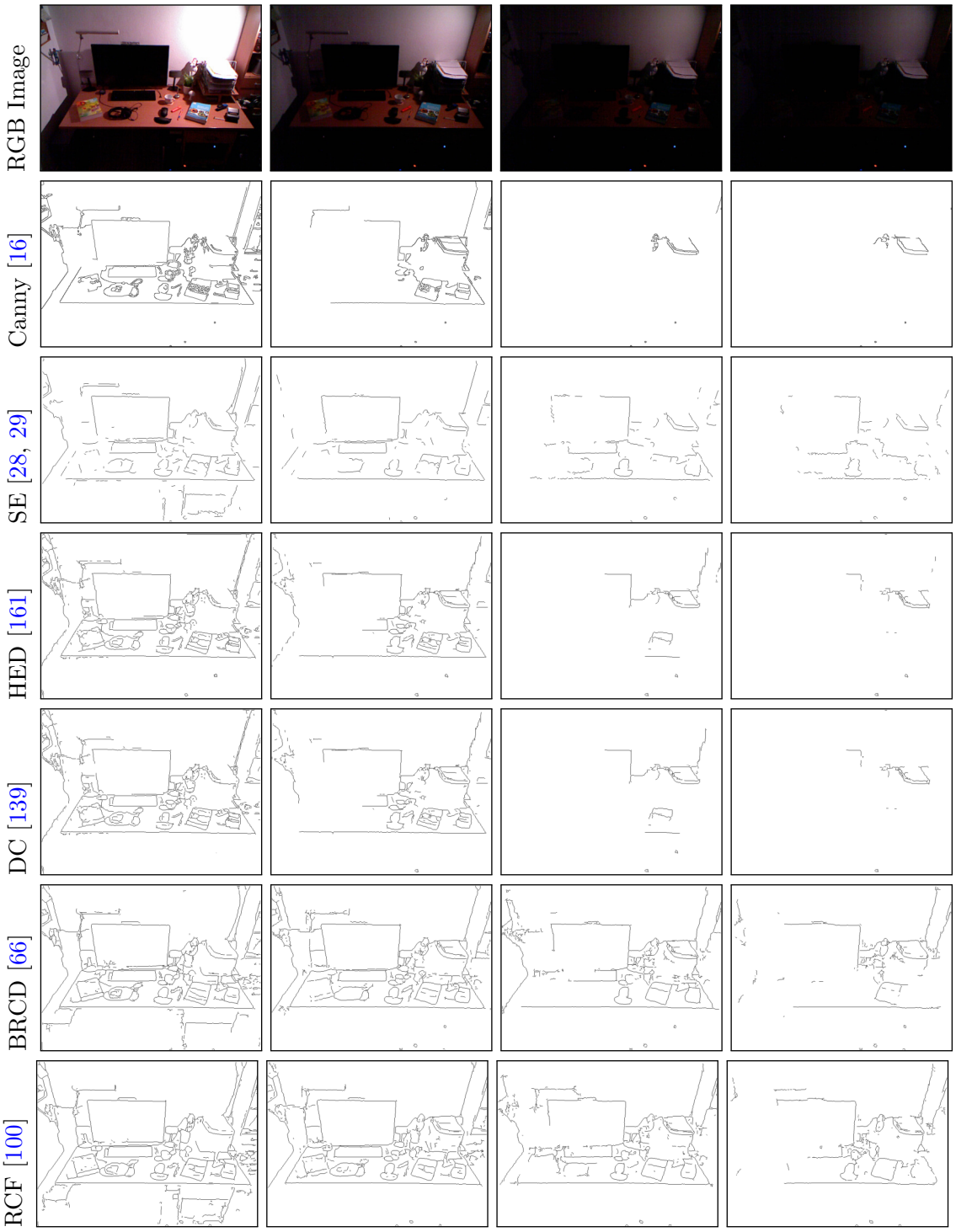


Figure 5.5: Four different RGB images with varying illuminations and the corresponding edge detections. The setup is a static desktop scene with a dimmable light source, which reduces the illumination over several seconds. While all methods work well for the first image images, only SE, RCF, and BDCN deliver relatively constant edge detections even for the second and third image. All detectors fail in the very dark last image.

Figure 5.6: The *basement* sequence of the NYUv2 dataset [141] with no motion blur for *columns 1-2* and mild motion blur in *columns 3-4*. While all the detectors are not influenced by the motion blur, SE and DC have problems with the dark left half of the image (see *columns 2-4*).

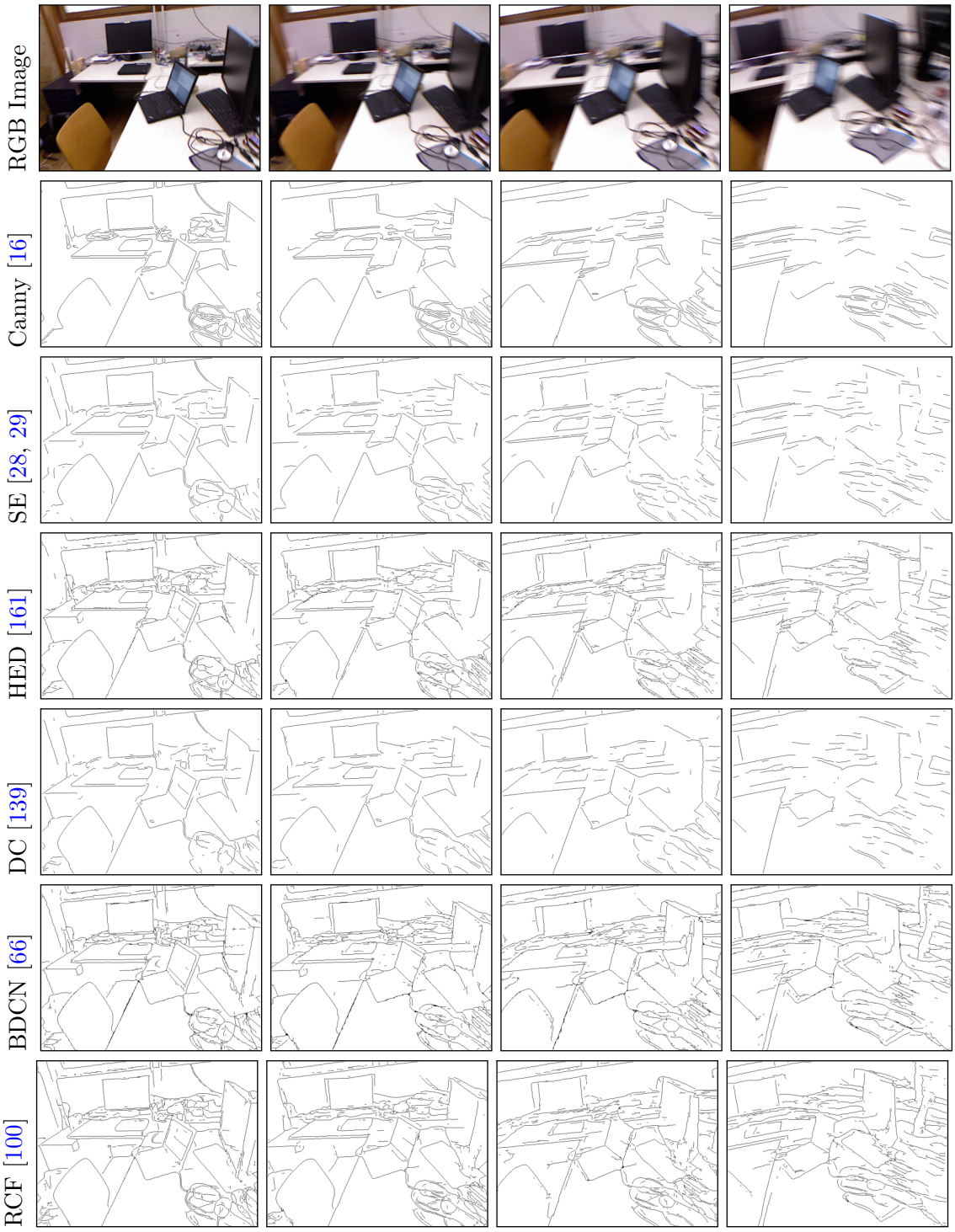


Figure 5.7: Four images from the *fr1/desk* sequence of the TUM RGBD dataset [147] starting with no motion blur and increasing until the object boundaries are nearly unrecognizable. All detectors deliver good results for the first two images but the performance of Canny already declines in *column 3*. In *column 4*, Canny, DC and SE detect mostly elongated lines along the motion direction, which are difficult to align. In contrast, HED, RCF, and BDCN are still able to find edges in this difficult situation.

5.2.2 Quantitative Evaluations

In this section, we present a qualitative analysis of the same six edge detectors as in Section 5.2.1. We design two experiments: **Experiment 5** evaluates the repeatability of the edge detections under varying illumination conditions and **Experiment 6** compares the Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) on a few sequences.

For **Experiment 5**, we again use the same dataset as in **Experiment 2** with a static camera and a dimmable light source. It is crucial for edge-based motion estimation that the number of detected edges remains relatively constant between consecutive frames. With a static camera, a perfect edge detector should always detect the same amount of edges independent of the illumination of the scene. In this experiment, we count the number of detected edges over a full light-dark-light cycle with a fixed camera. Figure 5.8 shows a sequence of 100 frames with a few sampled images along the x-axis and the curves represent the number of detected edges at a certain frame. From $frame30$ until $frame75$ illumination is so low that all the image-based edge detectors only detect noise and are completely unusable. However, in the other sequence parts, there are significant differences between the various edge detectors. For example, the number of edges for Canny drops rapidly starting at $frame$ 10 and only recovers around $frame$ $80-85$. We already qualitatively showed Canny's poor performance under illumination changes in **Experiment 1** and **Experiment 2**. Similar to Canny, DC and HED only remain stable for the first few frames, which is an expected result given their poor performance in previous experiments. SE has by far the lowest edge detections of all methods even in the beginning of the scene but remains fairly stable until $frame$ 30, then drops to 0 and returns around $frame$ 75. Finally, BDCN and RCF are by far the best edge detectors since they detect around 8000 edges up near-complete darkness at $frame30$ and return quickly to that number even with low illumination.

Our final **Experiment 6** evaluates the performance in terms of the ATE and RPE of all the edge detectors on nine selected sequences of the RGBD TUM dataset [147]. We refer to Section 6.1.1 for a detailed description of the metrics and the dataset. To evaluate solely the performance of the different edge detectors, we run only REVO with 3 pyramid levels without local or global optimization. Since the threshold settings greatly influence Canny, we run it twice but with different upper and lower values for the hysteresis, 60/80 and 100/150, respectively. Table 5.1 presents the results of the different edge detectors with the ATE and RPE in separated. The results indicate that the threshold settings for Canny influence the accuracy but are specific to the scene since *Canny 2* is better than *Canny 1* in only six out of nine sequences for the ATE. The overall performance of Canny is surprisingly high given its poor performance in some of the qualitative experiments. This is mostly because the robustness to illumination is not so important in the well-illuminated office scenes of the TUM RGBD benchmark [147]. All detectors except *Canny 2* have problems in the fast motion scene *fr1/desk2*, which contains a short part with highly blurred frames. The difference in accuracy comes from these few images and depends

Comparison of the Absolute Trajectory Error (ATE) [cm]							
	Traditional		Learning-based				
Seq.	Canny 1	Canny 2	SF [28, 29]	HED [161]	RCF [100, 101]	BDCN [66]	DC [139]
fr1/xyz	**5.09**	6.78	6.72	6.67	5.16	5.51	7.19
fr1/rpy	5.16	4.95	7.28	**4.55**	6.64	6.72	5.99
fr1/desk	7.53	**6.09**	7.32	6.61	18.31	45.4	7.10
fr1/desk2	20.31	**8.22**	14.88	15.98	35.94	19.67	17.68
fr1/plant	7.17	6.71	**4.66**	5.34	5.39	5.41	7.51
fr1/room	33.22	30.36	27.62	25.10	22.62	**21.37**	31.95
fr2/desk	**3.06**	8.86	8.54	16.96	7.10	13.39	6.4
fr2/xyz	1.38	0.89	1.00	0.89	1.44	1.07	**0.87**
fr3/long_off_hous	**5.1**	7.83	11.49	28.53	15.94	20.00	11.79
Comparison of the Relative Pose Error (RPE) in [cm/s]							
fr1/xyz	2.14	3.04	2.45	2.70	**2.11**	2.82	3.22
fr1/rpy	3.55	3.56	4.35	3.84	3.63	3.44	**3.28**
fr1/desk	3.67	**3.33**	3.48	3.70	11.77	23.43	3.92
fr1/desk2	11.76	**6.33**	11.30	11.62	20.75	14.77	11.97
fr1/plant	**2.68**	2.83	2.81	2.77	2.82	3.19	2.97
fr1/room	**4.85**	5.09	5.77	5.64	5.33	5.87	5.17
fr2/desk	1.44	1.43	1.27	1.66	**1.02**	1.25	1.16
fr2/xyz	**0.39**	0.40	0.44	0.40	0.45	0.45	0.44
fr3/long_off_hous	**1.29**	1.42	1.26	2.26	1.37	1.49	1.33

Table 5.1: Comparison of the ATE in [cm] and the RPE in [$\frac{cm}{s}$] achieved with REVO with the traditional Canny and various learning-based edge detectors. For Canny, we run two different lower and upper threshold settings for the hysteresis computation: *Canny* 1 with 60 and 80 and *Canny* 2 with 100 and 150.

mainly on the choice of the *KF*, i.e. whether a *KF* is created directly before the critical part or not. Apart from this sequence, the learned detectors show reasonable results on most of the other sequences. The *RPE* in Table 5.1 differs only in the *mm* range between the methods, which is likely below the accuracy of the ground truth.

5.3 Conclusion

In this chapter, we studied traditional and learning-based edge detection methods and presented qualitative and quantitative experiments. One of the most important features of an edge detector is its repeatability, i.e. detecting the same edges in consecutive frames, under difficult conditions. **Experiment 1** and **Experiment 2** show the superiority of the learned edge detectors under varying illumination conditions. Even in very dark conditions, RCF and BDCN can detect edges, where Canny, SE, HED and DC fail. In the

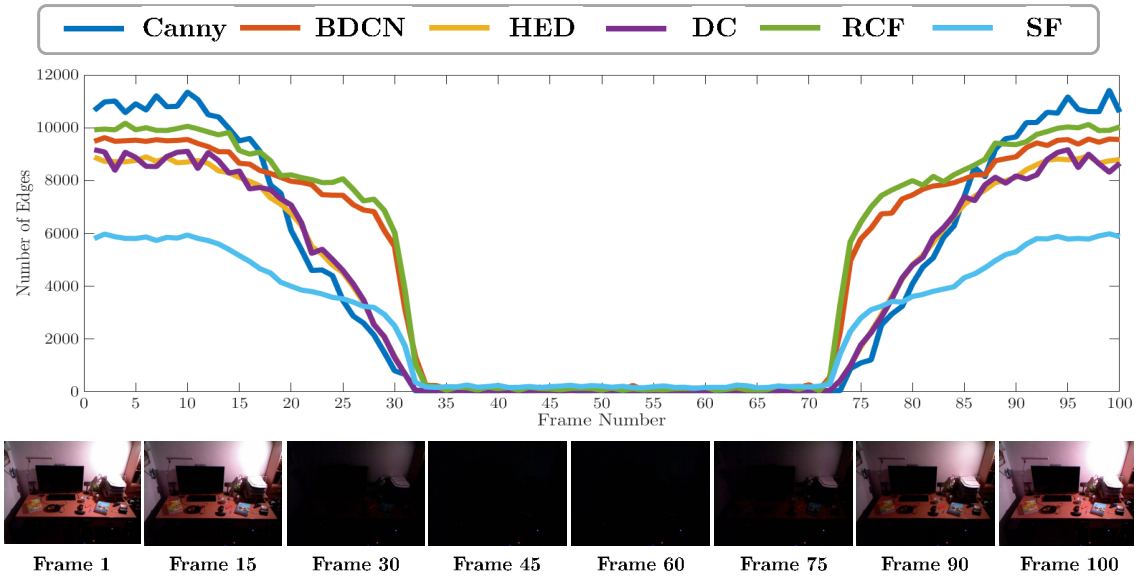


Figure 5.8: The number of detected edges over a sequence of 100 frames in which illumination first decreases and then increases for various edge detectors. For better visualization, sampled images from the sequence are depicted along the x-axis. While the number of edges rapidly drops for Canny [16] and HED [161] when the illumination of the scene is reduced, SF [28], RCF [100], and BDCN [66] are still able to detect edges until near darkness.

related quantitative **Experiment 5**, BDCN and RCF also demonstrate their exceptional performance, detect a rather stable amount of edges up until near-complete darkness. Robustness under motion blur is another quality criteria for an edge detector. In **Experiment 3** and **Experiment 4**, we show that all evaluated methods are can cope with mild motion blur as depicted in Figure 1.4a. However, during the fast motions in **Experiment 5**, Canny, SE and DC elongate edges along the motion direction but are unable to find edges in other directions. In contrast, RCF, BDCN, and also HED detect all the edges even under heavy motion blur (see Fig. 1.4b). In the last **Experiment 6**, we evaluate all the edge detectors on nine sequences of the TUM RGBD dataset [147] in terms of the *ATE* and *RPE* and present the results in Table 5.1. Due to the rich texture and stable illumination, all the detectors show a strong performance.

To summarize, we showed that Canny works well in standard scenes with sufficient texture and stable illumination but that its repeatability greatly suffers under motion blur and low illumination. Further, Canny requires careful threshold settings depending on the scene and the sensor. Our experiments indicate that modern machine-learned detectors are a promising alternative due to their robustness even in very difficult conditions. They show an impressive performance regarding repeatability, which is the most crucial part for edge-based motion estimation. The learned detectors can cope with heavy motion blur as depicted in Figures 5.6 and 5.7 and as shown in Figure 5.8 remain stable even

in low illumination conditions, where Canny fails very early on. However, for standard scenes, the benefits of learned detectors are currently minor since none have been trained for the motion estimation task. Training an edge detector specifically for edge-based motion estimation is an important step to increase the accuracy and stability even further. However, when working under real-time constraints on a single CPU, there is no way around fast traditional detectors. Currently, RCF and BDCN already run at around 25 fps on a strong GPU and with the rapid development of dedicated hard- and software, learned detectors definitely become an option for real-time applications in the future.

6 Experiments and Results

In this thesis, we proposed several contributions to the fields of RGBD edge-based Visual Odometry (VO) and Simultaneous Localization and Mapping (SLAM) as discussed in Chapter 4. On the *VO* side, our contributions are the purely edge-based REVO [133] (see Sec. 4.1.1) and its extension REVO+ICP [132], which adds a geometric term to the motion estimation (see Sec. 4.1.3). Our major contribution to the area of *SLAM* is RESLAM [134], which uses edges for motion estimation, local and global mapping (see Sec. 4.4-4.5). We already studied the influence of various edge detectors on the edge-based motion estimation in the previous Chapter 5 but a broader comparison of our systems to state-of-the-art works is still missing. In this chapter, we present extensive qualitative and quantitative evaluations of all our contributions and compare them to well-known *VO* and *SLAM* methods. Throughout the evaluations, we rely on a multitude of benchmark datasets and challenging sequences recorded with our RGBD sensors. We introduce the datasets and evaluation metrics in Section 6.1.1. In Section 6.2, we first compare all our contributions and study the influence of the individual components such as loop closure or Bundle Adjustment (BA). We then compare our systems to state-of-the-art *VO* methods in Section 6.3 and complete the experiments by an extensive *SLAM* evaluation.

6.1 Experimental Setup

This section introduces the datasets and evaluation metrics used for quantitative and qualitative evaluations and then summarizes all the thresholds and other implementation details for our methods.

6.1.1 Benchmark Datasets

Since the release of RGBD sensors, many benchmark datasets have been released and we refer to [46] for an extensive summary. However, only very few have gained enough popularity to be interesting for comparing RGBD *VO* and *SLAM* systems. In this section,

we will discuss the popular TUM RGBD [147], the synthetic ICL-NUIM [62] and the recent ETH3D [136] benchmark datasets. These datasets cover a wide variety of scenes and camera motions and all of the relevant state-of-the-art methods are evaluated on at least one of them. Figure 6.1 shows example RGB images and depth maps from all three datasets. Additionally, we recorded RGBD sequences to show large-scale loop closure capabilities and robustness in scarcely textured scenes.

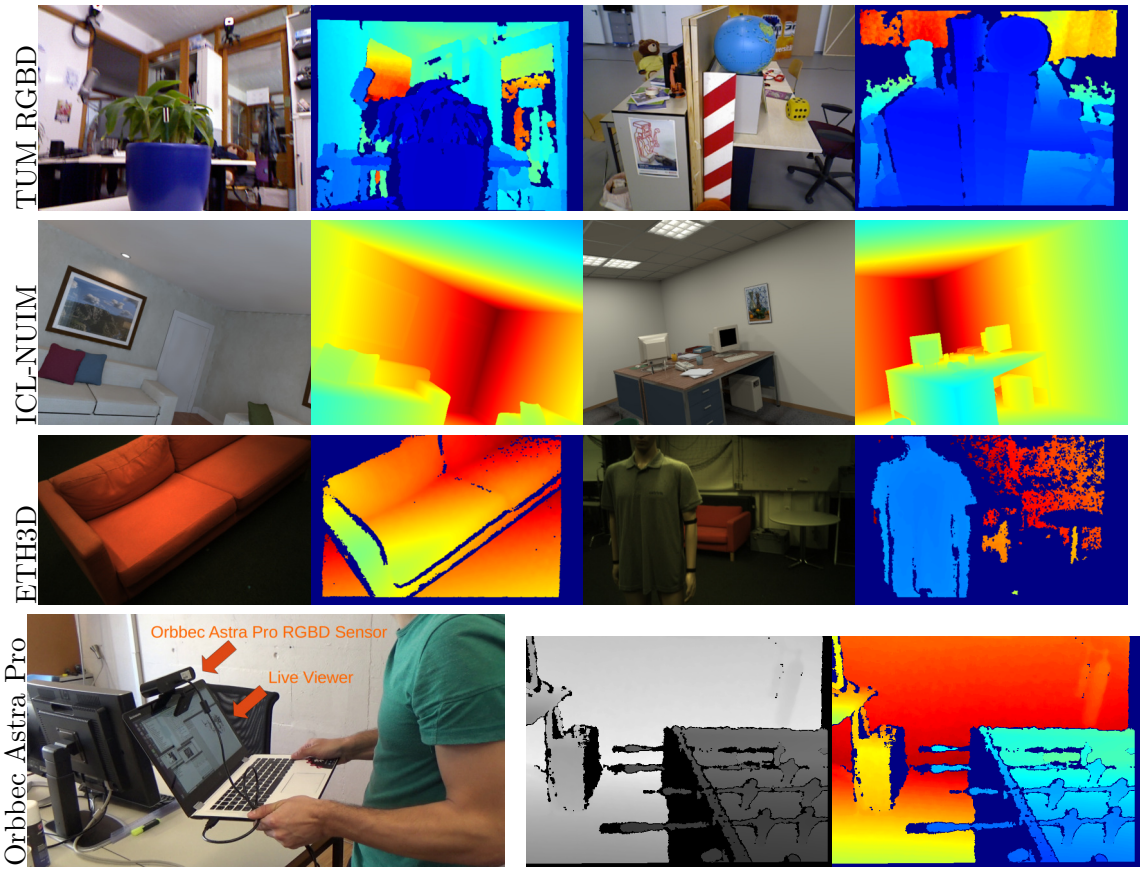


Figure 6.1: Excerpts from the benchmark sequences evaluated in this thesis. The TUM RGBD [147] dataset recorded with a Microsoft Kinect, the synthetic ICL-NUIM [62] dataset with perfectly synchronized RGB image and depth map and the recent and well-calibrated ETH3D [136] dataset. In the last row, our recording setup with an Orbbec Astra Pro RGBD sensor mounted on a laptop and the recorded data.

TUM RGBD The TUM RGBD dataset [147] is the most widely used dataset for quantitative evaluations of RGBD *VO* and *SLAM* systems. It comprises indoor RGBD sequences with the corresponding ground truth poses from a motion capture system. The sequences are recorded with a Microsoft Kinect at 30 fps with a resolution of 640×480 for RGB images and depth maps, while the ground truth poses are obtained at 100 Hz. The individual sequences in this dataset vary in length and difficulty and are divided into

three groups *freiburg 1 (fr1)*, *freiburg 2 (fr2)* and *freiburg 3 (fr3)*. What makes this dataset so interesting is the wide variety of indoor scenes, the convenient evaluation framework and the ground truth from the motion capture system. However, downsides are the relatively low image resolution, the inaccurate calibration, the rolling shutter camera, the poor synchronization between depth and RGB camera and the lack of large-scale recordings.

ICL NUIM The ICL NUIM dataset [62] is a synthetic dataset with noise-free images and exact ground truth poses. The dataset itself is small and covers only two rooms, an office and a living room, both depicted in Figure 6.1. It uses the same data format and evaluation metrics as the TUM RGBD dataset and can therefore be smoothly integrated in existing evaluation frameworks and systems. Due to synthetic nature, it does not suffer from synchronization or calibration issues like the TUM dataset. This makes it well-suited to test methods or new features under perfect conditions without issues caused by the sensor but it is difficult to predict the performance under real conditions from just synthetic experiments.

ETH3D The very recent ETH3D dataset as published together with BAD-SLAM by Schöps et al. [136]. It is a novel, well-calibrated and synchronized benchmark dataset comprising stereo and depth images alongside Inertial Measurement Unit (IMU) measurements. The authors built a setup with synchronized global shutter cameras and the infrared emitter from an Asus Xtion Live Pro. Through active stereo, they computed the depth map from the recorded infrared pattern, thereby solving the synchronization issue present in most RGBD sensors. The sequences are recorded at a framerate of 30 fps, a resolution of 739×458 and without auto-exposure. Two example RGBD pairs from this dataset are shown in Figure 6.1. For the indoor scenes, the Ground Truth (GT) trajectory comes from a motion capture system and for outdoor scenes from an offline Structure from Motion (SfM) method. The dataset is divided into training and test sequences, where the ground truth is only publicly available for the training part.

Orbbec Astra Pro Recordings To cover even more motions and scenes, we recorded additional RGBD sequences with an Orbbec Astra Pro sensor. We mounted it onto a laptop computer (see Fig. 6.1) and recorded while running a live viewer showing reconstruction and trajectory. Depth maps and RGB images are both recorded at a resolution of 640×480. Resolution and image quality of the Orbbec Astra Pro is comparable to the Microsoft Kinect but the depth map is denser and contains measurements up to around a distance of 8 m.

6.1.2 Evaluation Metrics

Alongside their TUM RGBD dataset, Sturm et al. [147] propose two metrics to evaluate the performance of *VO* and *SLAM* systems. For *VO* system, they propose to measure the

drift over a fixed time interval δt between a set of poses Q from the GT trajectory and a set of poses P from the estimated trajectory. They refer to this error as Relative Pose Error (RPE) and define it at a time step i as:

$$\mathbf{RPE}_i = (\mathbf{Q}_i^{-1}\mathbf{Q}_{i+\delta t})^{-1}(\mathbf{P}_i^{-1}\mathbf{P}_{i+\delta t}) \tag{6.1}$$

where δt is the time distance between poses. Computing (6.1) over a sequence with N estimated poses has, gives $M = N - \delta$ individual $RPEs$. Following [147], we compute the Root Mean Squared Error (RMSE) over the translational component:

$$RMSE(\mathbf{RPE}_{1:n}, \delta_t) = \sqrt{\frac{1}{M}\sum_{i=0}^{M} ||trans(\mathbf{RPE}_i)||^2}\,, \tag{6.2}$$

where $trans(\mathbf{RPE}_i)$ is the translational part of the RPE in (6.1). The RPE is well-suited to measure accuracy in a local area but cannot give a global accuracy estimate.

To assess the global consistency, Sturm et al. [147] propose to measure the absolute distance between the estimated and the ground truth trajectory. Since the coordinate systems of the two trajectories are arbitrary, they are first aligned by a rigid body transformation $\mathbf{S}$ computed with Horn's method [69]. The Absolute Trajectory Error (ATE) at time step i is defined as:

$$\mathbf{ATE}_i = \mathbf{Q}_i^{-1}\mathbf{S}\mathbf{P}_i, \tag{6.3}$$

where Q and P are again the set of poses of the GT and estimated trajectory. Like for the RPE, again the $RMSE$ of the translational component is evaluated:

$$RMSE(\mathbf{ATE}_{1:n}) = \sqrt{\frac{1}{N}\sum_{i=0}^{N} ||trans(\mathbf{ATE}_i)||^2}\,. \tag{6.4}$$

6.1.3 Implementation Details

Like all state-of-the-art systems, also ours depend on various thresholds and only show their best performance with certain configurations. In this section, we give all the values and details to reproduce the results presented in this chapter. Throughout all the evaluations, REVO detects edges with the Canny algorithm from the OpenCV [13] with a kernel size of 3 and a hysteresis thresholds depending on the sensor input. For the TUM RGBD dataset [147], the lower and upper thresholds are 100 and 150 and for the ICL-NUIM dataset [62] 20 and 40. The relative pose estimation runs over three pyramid levels with the highest resolution being the resolution of the dataset. On each pyramid level, edges are considered outliers, when they are 10, 20 or 30 pixels away from the closest edge detection and the inliers in E_{edge} (4.1.1) weighted by a Huber weight function with $\Theta_H = 0.3$. REVO selects Keyframes (KFs) based on the Edge-based Quality Assessment (EBQA) proposed in Section 4.2 running it on the second pyramid level with histogram weights

$\mathbf{w} = [1, 1, 1.25, 1.5]$. In the combined motion estimation REVO+ICP 4.1.3, the edge-based optimization and *KF* selection are identical to REVO including the hysteresis thresholds. The only difference in the edge-based part is the Edge Enhancement (EE) for the edge detections discussed in Section 5.1.1. We choose patch sizes of $N = [20, 10, 5]$ px for each respective pyramid level and consider a patch insufficiently covered when less 5% of its area are edge detections. The geometric term is weighted with a Tukey weight function with $\theta_T = \sqrt{1.5}$ (see Tab. 3.2). The local mapper refines over $N = 4$ spatially close *KFs* within a search range of $\Theta_d < 0.1m$ and $\Theta_\alpha < 30°$.

The relative pose estimation in RESLAM is again identical to REVO with three pyramid levels, a Huber weight function with $\Theta_H = 0.3$ and outlier rejection at 10, 20 and 30 pixels. We found that for RESLAM, good choices for lower and upper thresholds are: 60 and 80 for the TUM RGBD datasets [147], 20 and 40 for the ICL-NUIM dataset [62], 20 and 40 for the ETH3D [136]. New *KFs* are selected with an optical flow criteria instead of *EBQA* with the constant thresholds $\theta_{fov} = 0.15$ and $\theta_{occ} = 0.1$. In the local mapper, edges are activated if they are within a maximum distance of $\Theta_A = 5$ px to an edge detection and at least $\Theta_M = 5$ px away from the closest activated edge. The global mapper adds *KFs* to the database if their block-wise Hamming distance (4.90) to all the *KFs* in the database is above $\theta_h = 0.2$. It considers a *KF* as potential LC candidate if at least one similarity is below $\theta_{lc} = 0.25$. To verify a loop, again the same histogram weights $\mathbf{w} = [1, 1, 1.25, 1.5]$ are used and a loop constraint has three times the weight of a relative constraint from local bundle adjustment.

All the systems are real-time capable on a laptop CPU for a typical RGBD recording at 30 fps with a resolution of 640x480. However, there are differences due to the increased number of computations for REVO+ICP and RESLAM. Further, other factors such as the number of detected edges, the inter-frame motion and of course the hardware influence the speed. For the TUM RGBD sequences, REVO runs at around 40 - 60 fps, REVO+ICP at 20-30 fps and RESLAM at 25-30 fps.

6.2 Comparison of our Contributions

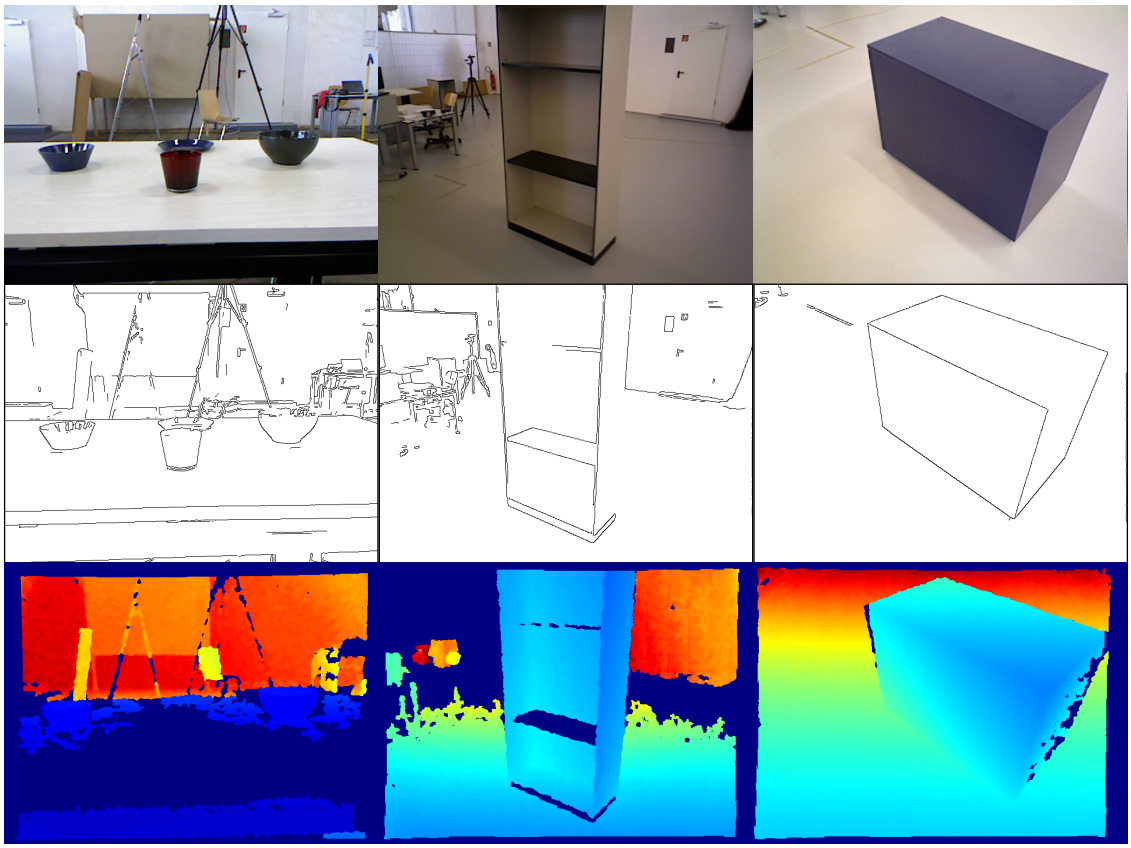
In this section, we focus on the methods proposed in this thesis and show the influence of the individual components to highlight the continuous development throughout the last years. The first part of this section presents quantitative evaluations on several sequences from the popular TUM RGBD and ICL-NUIM datasets as well as qualitative results in the form of trajectories and reconstructions (see Tab. 6.1 and 6.2). The second part is a purely qualitative evaluation and shows reconstructions of difficult scenes recorded with an Orbbec Astra Pro RGBD sensor.

The methods in this thesis can be grouped together in three different systems. Our most basic edge-based *VO* system REVO (see Sec. 4.1.1), the extension to include also a geometric term in the pose estimation REVO+ICP [132] and finally the RESLAM, a complete *SLAM* system with a full local *BA* and LC capabilities. By the example of

REVO, we show the benefits of *KF*-based *VO* over frame-to-frame *VO* (see Sec. 4.1.1). For REVO+ICP, we evaluate edge-based motion estimation with the proposed *EE* (E+EE) as presented in Section 5.1.1. We then study the Iterative Closest Point (ICP)-only version, the combined relative pose estimation (E+ICP) (see Sec. 4.1.3) and the optimization with respect to spatially close *KFs* (Opt) as shown in Section 4.4.1. RESLAM runs in four different configurations: edge-based motion estimation and Single-Level Edge Detection (SLD) (E+SLD) (see Sec. 5.1.2), with full local *BA* (E+BA) presented in Section 4.4.2, with LC but without *BA* and finally as full *SLAM* system with LC and *BA*. All modules in REVO, REVO+ICP, and also E+SLD and BA of RESLAM fall into the *VO* category but only LC and SLAM in RESLAM can be considered *SLAM* systems. In Tables 6.1 and 6.2 the results are given in terms of the *ATE* in $[cm]$ and *RPE* in $[cm/s]$. The best overall result is depicted in **bold**, the best *VO* result is underlined and tracking losses or sequences with an *ATE* above 80 *cm* are marked as failed (✗).

From the **TUM RGBD dataset**, we select 15 sequences from all three groups and summarize the results Table 6.1 When looking at the REVO evaluations, it is evident that the *KF*-based setting is superior to the frame-to-frame setting. Overall REVO performs well in all the sequences and even achieves the best *RPE* for *fr1/desk* and *fr1/360*, which is surprising considering that it only optimizes with respect to the last *KF*. However, without a local or global optimization it cannot compete with the other methods in terms of the *ATE*. The TUM RGBD dataset is mostly well-textured and therefore there is no significant improvement of *EE* and *SLD* over the traditional edge detection on multiple scales. In contrast to the edge-based methods, the *ICP*-only method performs poorly in all sequences except *fr1/xyz* and *fr1/desk* and even fails for *fr2/dishes*, *fr3/large-cabinet* and *fr3/cabinet* (see Fig. 6.2). One reason for failure in *fr2/dishes* is that large parts of the depth map are unusable, either due to invalid values caused by reflection on the table or unreliable far away measurements on the back wall. Another problem is that the mostly planar structures in *fr3/large-cabinet* and *fr3/cabinet* render the alignment ambiguous. The combined pose estimation ("E+ICP") does not improve the accuracy compared to only the edge-based term but increases the robustness. The largest improvement comes from the local refinement on spatially close *KFs*, which improves the results significantly in short sequences such as *fr1/xyz*, *fr1/rpy*, *fr1/desk*, and *fr1/plant*. RESLAM shows impressive overall results and scores the best *ATE* and *RPE* for 10 out of the 15 sequences. As mentioned before, the *SLD* does not show any improvement over *EE* or traditional edge detection. The evaluations show that the greatest improvement comes from the full local *BA* presented in Section 4.4.2. Especially in the longer *fr2* sequences, the local optimization improves the *ATE* by up to a factor $2 - 3$ in some cases, while loop closure ("LC") is only significantly better in the heavy rotation scene *fr1/rpy*. Due to the nature of the TUM sequences, where start and endpoints are not the same or have different viewpoints, the LC mostly closes loops rather on the small than the large scale. Thus, the LC degenerates to a local mapping with very few *KFs*, which is less accurate than the refinement over spatially close *KFs* or the local *BA*. Finally, the complete *SLAM* system

shows a consistent performance over all sequences and it is either the best or very close to the best throughout the evaluations. A common problem for edge-based methods is when edge detections are concentrated on object borders only, e.g. the cabinet depicted in Figure 6.2. Then the depth measurements for the edges can be invalid or inaccurate, e.g. the depth measurement corresponds to the distance to the back wall instead of the cabinet. This is problematic for the alignment, especially when no additional geometric term or local *BA* is used.



(a) *fr2/dishes* **(b)** *fr3/large_cabinet* **(c)** *fr3/cabinet*

Figure 6.2: Example images from three different TUM RGBD sequences that caused problems during the evaluations. *fr2/dishes* is especially challenging for the ICP-only method since parts of the depth map on the table are invalid and the distance to the back wall is above the range in which depth measurements are considered valid. In *fr3/large-cabinet* the ICP-only method suffers from a similar problem on its back wall. For the other methods, the challenges in *fr3/large-cabinet* and *fr3/cabinet* are that most edges are on the cabinet's boundaries, where depth measurements are unreliable.

Figure 6.3 shows six point clouds reconstructed with RESLAM from the TUM RGBD dataset. Since all the edge points are handled and optimized as individual points without any knowledge of their surroundings, the point clouds are not as regular as the ones directly from the RGBD sensor. However, in Figures 6.3 (a),(b) and (c) the borders of the tables

Comparison of the Absolute Trajectory Error (ATE) [cm]

	REVO [133]		REVO+ICP [132]				RESLAM [134]			
Seq.	FF	KF	E+EE	ICP	E+ICP	Opt	E+SLD	BA	LC	SLAM
fr1/xyz	13.31	6.78	5.81	3.25	4.86	**1.55**	7.26	2.45	1.92	2.25
fr1/rpy	12.30	4.95	5.21	16.72	5.55	2.26	6.18	7.32	**2.25**	2.82
fr1/desk	10.54	6.09	6.09	7.20	5.67	**2.96**	5.04	3.17	4.11	3.15
fr1/desk2	16.87	8.22	7.53	16.97	7.68	5.99	7.09	**4.91**	5.57	**4.91**
fr1/plant	7.30	6.71	7.75	20.95	6.62	**3.87**	6.53	5.78	5.42	5.76
fr1/360	21.32	13.56	14.60	22.10	13.82	16.18	**12.04**	13.61	**12.04**	13.60
fr2/desk	32.90	8.86	8.90	16.77	9.28	9.51	3.49	2.36	2.31	**2.08**
fr2/xyz	7.90	0.89	0.89	13.48	1.92	1.71	0.87	**0.44**	0.70	0.54
fr2/rpy	4.24	1.68	0.98	24.09	1.36	0.98	0.88	0.62	**0.55**	0.61
fr2/desk_w_pers	16.45	10.49	10.69	78.34	8.33	7.10	8.57	**2.83**	7.54	3.56
fr2/dishes	14.87	6.77	6.92	✗	6.60	7.21	8.58	**3.74**	6.22	3.75
fr3/large-cabinet	50.89	56.88	53.33	✗	**35.60**	50.16	✗	35.82	✗	35.82
fr3/cabinet	✗	19.75	27.57	✗	24.19	27.48	29.21	**11.81**	29.21	**11.81**
fr3/str-ntex-far	21.89	9.40	7.21	24.33	5.55	**2.18**	7.35	6.41	7.35	6.41
fr3/long_off_hous	48.93	7.83	7.49	52.31	14.89	9.38	15.24	9.81	4.46	**4.23**
Comparison of the Relative Pose Error (RPE) in [cm/s]										
fr1/xyz	4.23	3.04	2.89	2.86	2.49	**2.25**	3.52	2.27	2.85	2.61
fr1/rpy	4.06	3.56	3.49	9.08	3.56	3.30	3.46	3.86	**3.24**	3.89
fr1/desk	4.80	**3.33**	3.38	5.58	3.46	3.44	4.31	3.38	3.98	3.40
fr1/desk2	7.47	6.33	6.34	7.69	6.04	6.30	5.87	**4.63**	5.36	4.93
fr1/plant	3.58	2.83	3.04	7.34	2.82	2.82	2.82	2.64	3.00	**2.63**
fr1/360	8.11	**6.79**	6.82	15.58	8.54	12.60	7.89	7.74	7.89	7.75
fr2/desk	2.17	1.43	1.48	3.05	1.50	1.83	1.00	**0.92**	1.11	1.30
fr2/xyz	1.13	0.40	0.40	1.09	0.49	0.55	**0.32**	0.36	0.37	0.49
fr2/rpy	0.93	0.50	0.38	3.21	0.39	0.41	**0.30**	0.37	**0.30**	0.37
fr2/desk_w_pers	3.41	2.73	2.74	5.97	1.74	1.62	1.40	**1.05**	1.40	1.21
fr2/dishes	3.28	2.06	2.54	✗	2.49	3.01	1.82	1.05	1.77	**1.04**
fr3/large-cab	✗	22.89	21.95	✗	16.11	**15.22**	✗	23.80	✗	23.80
fr3/cabinet	9.30	5.97	6.89	✗	**5.71**	7.06	7.38	**4.53**	7.38	**4.53**
fr3/str-ntex-far	7.40	4.08	3.78	7.39	2.29	**2.17**	4.77	4.91	4.77	4.91
fr3/long_off_hous	3.04	1.42	1.42	3.11	2.02	4.20	1.38	**1.04**	1.27	1.10

Table 6.1: Comparison of our methods with their individual components on the TUM RGBD dataset: REVO in a frame-to-frame (FF) and *KF*-based setting, REVO+ICP with edge enhancement "E+EE", "ICP" only, a combination of both (REVO+ICP) and REVO+ICP and refinement on spatially close *KFs* ("Opt"). RESLAM in the *VO* setting with single level edge detection ("E+SLD"), with local mapping (*BA*), with global mapping (LC) and the full *SLAM* system with LC and *BA*. Best *VO* result underlined and best overall result in **bold**.

are still clearly visible and also Figure 6.3 (d) and (f) give a good impression of the scene.

The second evaluation compares our contributions on the synthetic **ICL-NUIM dataset**, which covers two different rooms, a living room and an office. We select the first two sequences *kt*0 and *kt*1 from each and summarize the results in Table 6.2. In general, the ICL-NUIM sequences are low textured scenes without noise, which poses a challenge for edge- and image-based methods. For REVO, again the *KF*-based setting is superior for *icl/living_room-kt1*, *icl/office-kt0*, and *icl/office-kt1*. REVO with the standard multi-scale edge detection fails for *icl/living_room-kt0* but the new *EE* and *SLD* both manage to track the sequence. In contrast to the previous evaluation, due to the perfectly synchronized and noise-free depth map REVO+ICP outperforms the other methods in 3 out of 4 sequences. For *icl/living_room-kt1*, the ICP-only version shows very high accuracy, which can be explained by the presence of major planes on the floor, wall and ceiling throughout the sequence. The spatial refinement increases the accuracy especially in the *icl/office-kt1* sequence, which contains many small loops. RESLAM only shows the highest accuracy for *icl/living_room-kt0*, where the loop closure gives the biggest improvement. One interesting finding is that with very few edge detections, the overlap between the *KFs* is concentrated in specific areas and the local *BA* can even decrease the accuracy as shown for *icl/living_room-kt1* and *icl/office-kt1*.

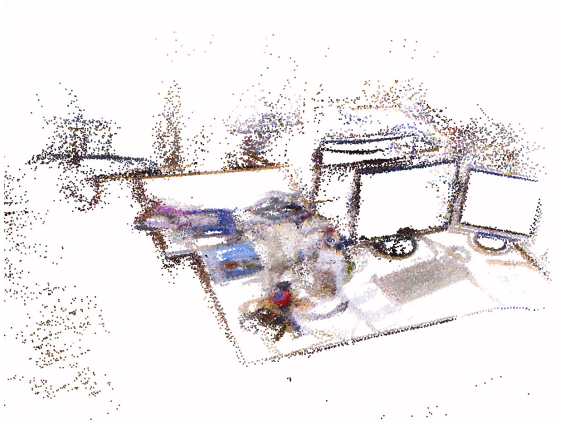
Figure 6.5 visualizes the trajectories for the *icl/living_room-kt0* and *icl/office-kt1* sequence, where REVO+ICP is always very close to the ground truth. Qualitative results achieved by REVO+ICP are depicted in Figure 6.6, where the walls in (a) area aligned correctly at a right angle and even the tiles on the ceiling in (b) are visible.

For the **Orbbec Astra Pro** recordings, we present purely qualitative results. Figure 6.7 demonstrates the capabilities of REVO+ICP, which in (a) accurately reconstructs a room with many white walls and in (b) manages to track movements in front of scarcely textured walls. Sample images from the respective sequences are shown on the left of the reconstructions. To illustrate the LC capabilities, we record a trajectory with a length of approximately 150 m with the same start and endpoints (see Fig. 6.8) Due to the many invalid depth values caused by far away surfaces, the drift is large and start and endpoints are not in the same area. With the loop closure, we can correct the drift and correctly estimate the trajectory.

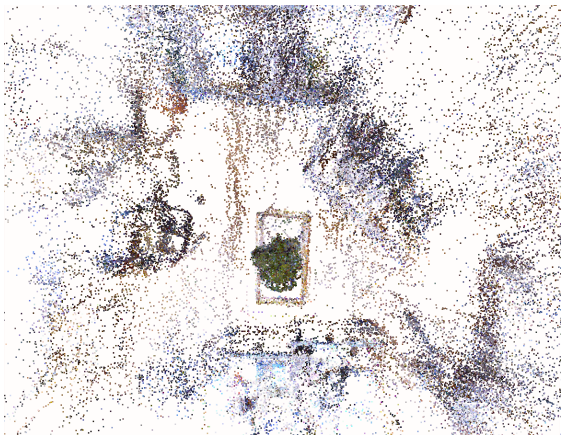
6.3 Visual Odometry Evaluations

In this section, we benchmark our *VO* methods and state-of-the-art RGBD *VO* systems on TUM RGBD [147] and ICL-NUIM [62] sequences and visualize trajectories of selected sequences. Our *VO* methods are REVO in the *KF*-based setting, REVO+ICP with refinement of spatially close *KFs*, and RESLAM with the full local *BA*, to state-of-the-art RGBD *VO* systems. Since there is an abundance of *VO* systems, we select six popular ones: an ICP-only implementation called *ICPCUDA* [157], DVO [81] with a com-

(a) Sequence *fr1/desk*

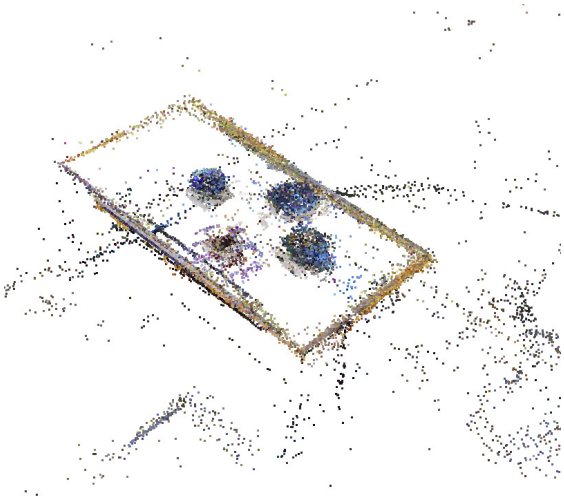


(b) Sequence *fr1/xyz*



(c) Sequence *fr1/plant*

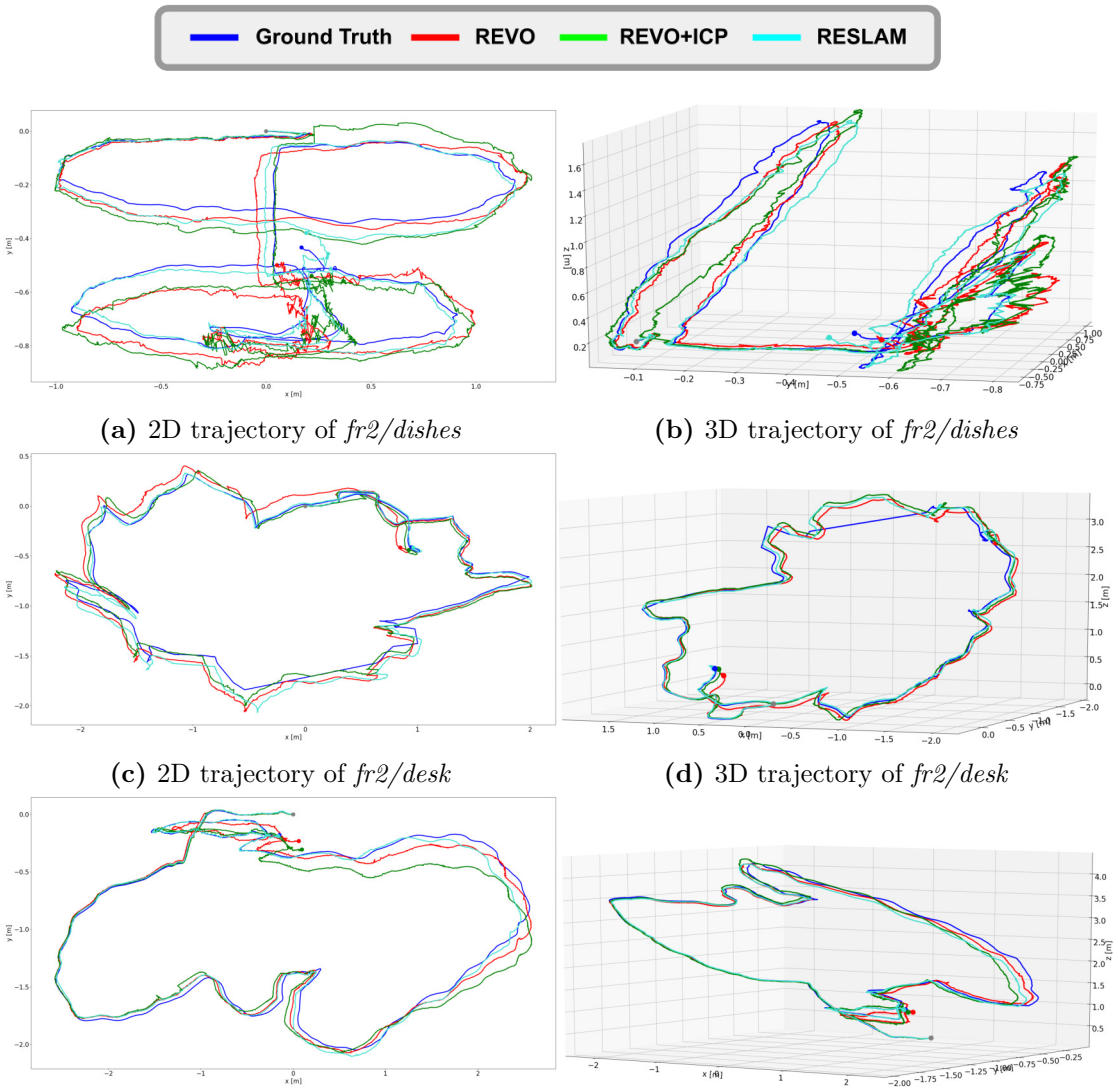
(d) Sequence *fr2/desk*



(e) Sequence *fr2/dishes*

(f) Sequence *fr3/long_off_household*

Figure 6.3: Six point clouds computed by RESLAM covering all three groups of the TUM RGBD dataset, with (c) viewed from the top and the rest from the side.



(a) 2D trajectory of *fr2/dishes* **(b)** 3D trajectory of *fr2/dishes*

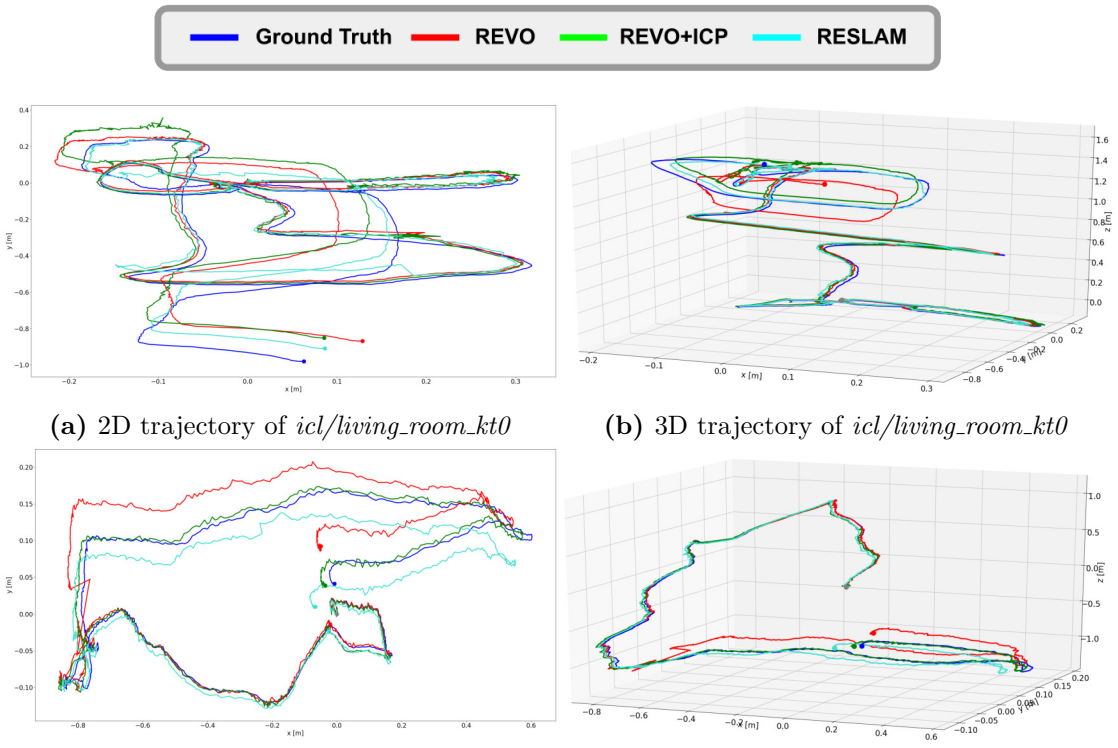
(c) 2D trajectory of *fr2/desk* **(d)** 3D trajectory of *fr2/desk*

(e) 2D trajectory of *fr3/long_office_household* **(f)** 3D trajectory of *fr3/long_office_household*

Figure 6.4: A variety of 2D and 3D trajectories from longer TUM RGBD sequences, where the starting point of all the trajectories is marked by a grey dot and their respective endpoints by colored dots. In all the visualized trajectories, RESLAM (turquoise) is closest to the ground truth (blue).

bined photometric and geometric error but without *KFs*, FOVIS [73] with FAST features, ORB-SLAM2 [116] running in *VO* mode and two edge-based methods with nearest Approximate Nearest Neighbor Fields (ANNF) [164] and Oriented Nearest Neighbor Fields (ONNF) [165] discussed in 4.1.1. *ICPCUDA* is a publicly available high-speed implementation[1] of the point-to-plane *ICP* algorithm to estimate the relative pose between two consecutive frames. We use it without any algorithmic changes but adapt the intrinsic

[1] https://github.com/mp3guy/ICPCUDA



(a) 2D trajectory of *icl/living_room_kt0*

(b) 3D trajectory of *icl/living_room_kt0*

(c) 2D trajectory of *icl/office_kt1*

(d) 3D trajectory of *icl/office_kt1*

Figure 6.5: 2D and 3D trajectories from two ICL-NUIM sequences, where the starting point of all the trajectories is marked by a grey dot and their respective endpoints by colored dots. In all the visualized trajectories, REVO+ICP performs best due to the perfect depth map.

(a) Sequence *icl/living_room_kt0*

(b) Sequence *icl/office_kt0*

Figure 6.6: Point clouds from the ICL-NUIM dataset reconstructed with REVO+ICP and refinement on spatially close *KFs*. (a) For *icl/living_room-kt0*, the walls of the room are accurately reconstructed shown by the right angle between them and in (b) *icl/office-kt0*, the office furniture and even the tiles on the ceiling are reconstructed.

Comparison of the Absolute Trajectory Error (ATE) [*cm*]

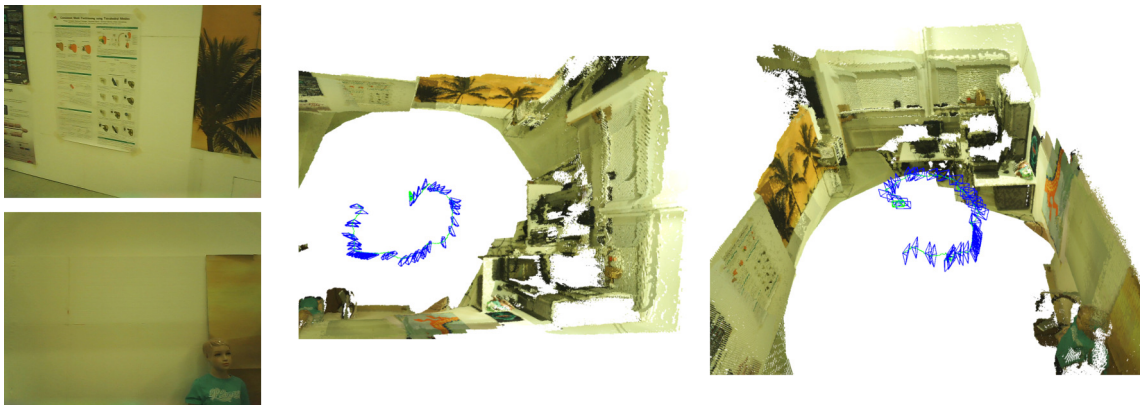
	REVO [133]		REVO+ICP [132]				RESLAM [134]			
Seq.	FF	KF	E+EE	ICP	E+ICP	Opt	E+SLD	BA	LC	SLAM
icl/living_room-kt0	✗	✗	8.82	12.31	6.33	5.46	6.96	3.99	**2.06**	3.36
icl/living_room-kt1	7.33	2.16	2.47	**0.07**	2.26	1.23	1.01	1.40	0.86	1.47
icl/office-kt0	7.39	1.80	1.91	22.77	0.72	**0.66**	6.39	4.65	1.49	2.22
icl/office-kt1	5.84	1.05	3.53	12.25	5.85	**0.75**	1.92	2.38	1.84	2.72
Comparison of the Relative Pose Error (RPE) in [*cm*/*s*]										
icl/living_room-kt0	✗	✗	2.41	3.45	2.21	1.89	1.79	**1.53**	1.54	1.98
icl/living_room-kt1	2.22	0.86	0.88	**0.05**	1.23	1.36	0.58	0.86	0.57	0.89
icl/office-kt0	1.98	0.58	0.63	6.32	**0.48**	0.51	1.25	1.03	1.06	2.45
icl/office-kt1	1.93	0.69	2.00	6.24	2.75	**0.53**	0.59	1.01	0.73	1.09

Table 6.2: Comparison of our methods with their individual components on the TUM RGBD dataset: REVO in a frame-to-frame (FF) and *KF*-based setting, REVO+ICP with edge enhancement "E+EE", "ICP" only, a combination of both (REVO+ICP) and REVO+ICP and refinement on spatially close *KFs* ("Opt"). RESLAM in the *VO* setting with single level edge detection ("E+SLD"), with local mapping (*BA*), with global mapping (LC) and the full *SLAM* system with LC and *BA*. Best *VO* result underlined and best overall result in **bold**.

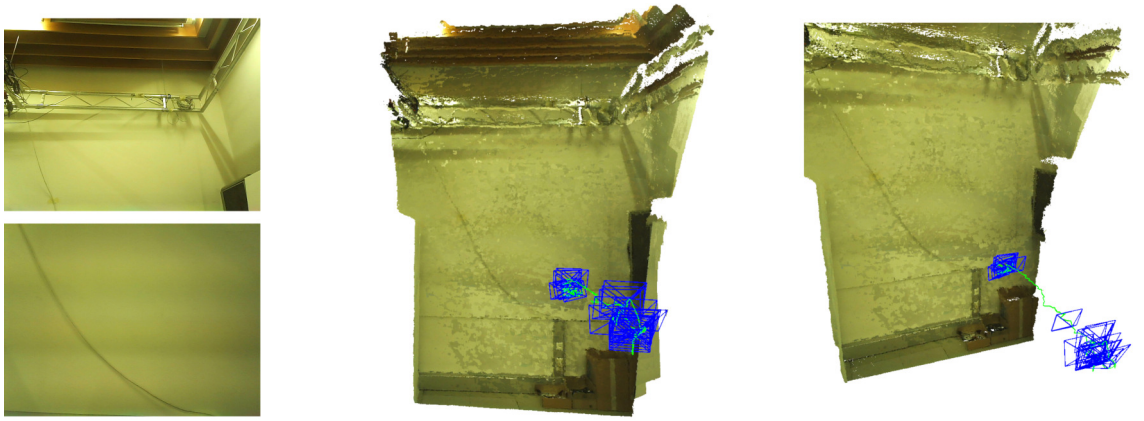
camera parameters for the respective datasets. For DVO [81] and FOVIS [73] we run the evaluation framework[2] without any modifications. We take the TUM evaluations for ORB-SLAM2 in *VO* mode from [165] and compute the missing results for two ICL-NUIM sequences by setting $mbOnlyTracking = true$ (marked by * in Tab. 6.4). There is no publicly available implementation for ANNF and ONNF, thus we copy the results from [165]. In Tables 6.3 and 6.4, we again show the *ATE* in [*cm*] and the *RPE* in [*cm*/*s*]. The best overall score is marked in **bold**, the best score from an edge-based method is underlined and tracking failures are shown as an ✗.

We evaluate on the same 15 **TUM RGBD** sequences [147] as in the previous section and visualize the trajectories for three longer sequences in Figure 6.9. Regarding the *ATE* in Table 6.3, our methods show the highest accuracy in 9 out of 15 sequences and are the best in 12 out of 15 for the edge-based methods only. Our proposed local optimizations reduce the overall drift, which lowers the *ATE*, but as expected increases the *RPE*. This is evident from Table 6.3, where our methods have the overall lowest error on three sequences and when considering only edge-based systems on seven sequences. However, even in the cases where ANNF or ONNF are the best, e.g. in the *fr2* sequences, the difference to our methods is in the [*mm*] range, while the difference in the *ATE* can be several [*cm*], e.g. *fr2/desk_with_person*. Feature-based methods such as ORB-SLAM2 and FOVIS work best, when there is little motion and the same features are seen throughout the sequence

[2] https://github.com/mpizenberg/rgbd-tracking-evaluation



(a) 360° RGBD sequence in a low-texture environment.



(b) RGBD sequence recorded in front of a white wall.

Figure 6.7: Two dense reconstructions computed with REVO+ICP. (a) is a 360° sequence recorded in a room with many scarcely textured walls and (b) primarily contains movements in front of a white wall. On the left side are sample images from the respective sequences.

like in *fr1_xyz*, *fr2_xyz*, and *fr2_rpy* but both suffer when the scene changes. The ICP-only method fails frequently since it only aligns with respect to the last frame and does not use a frame-to-model alignment like KinectFusion [117] or InfiniTAM [79]. DVO does not fail but without *KFs* the drift is significant and even problematic the short time range of 1s used for *RPE* computation. This is can also be seen in the trajectories depicted in Figure 6.9, where especially for *fr2/dishes* DVO drifts significantly compared to the others.

We benchmark on four sequences from the synthetic **ICL-NUIM** dataset and present the results in Table 6.4. Again our methods show high accuracy and perform best in three out of four sequences for both, the *ATE* and the *RPE*. In the scarcely textured ICL-NUIM sequences, FOVIS [73] fails and ORB2-SLAM [116] performs significantly worse than our methods. The ICP-only method and the combined geometric and photometric DVO both run in the frame-to-frame setting and show a very high drift.

Comparison of the Absolute Trajectory Error (ATE) [cm]

	Features		Features		Edge-based		Our methods [134]		
Seq.	ICP	DVO	FOVIS	ORB2	NNF		REVO	R+ICP	RESLAM
Seq.	D	RGBD	Fast	VO	ANNF	ONNF	KF	Opt	BA
fr1/xyz	4.17	9.34	4.68	**0.90**	13.70	4.30	6.78	1.55	2.45
fr1/rpy	10.74	6.97	8.59	6.60	20.50	4.70	4.95	**2.26**	7.32
fr1/desk	14.39	6.05	27.94	6.50	21.20	4.40	6.09	**2.96**	3.17
fr1/desk2	27.27	7.77	15.44	9.30	38.10	18.70	8.22	5.99	**4.91**
fr1/plant	41.12	4.68	16.26	6.70	13.30	5.90	6.71	**3.87**	5.78
fr1/360	28.00	29.64	14.19	13.90	31.50	24.20	**13.56**	16.18	13.61
fr2/desk	✗	42.69	10.75	27.40	3.90	3.70	8.86	9.51	**2.36**
fr2/xyz	21.95	24.75	1.42	0.80	1.00	0.80	0.89	1.71	**0.44**
fr2/rpy	24.51	25.49	1.36	**0.40**	0.70	0.70	1.68	0.98	0.62
fr2/desk_w_pers	66.05	18.83	11.60	13.5	4.70	6.90	10.49	7.10	**2.83**
fr2/dishes	✗	40.91	16.97	10.40	3.40	**3.30**	6.77	7.21	3.74
fr3/large-cabinet	✗	58.66	30.21	**15.40**	34.90	31.70	56.88	50.16	35.82
fr3/cabinet	75.62	67.12	59.56	**5.70**	10.30	**5.70**	19.75	27.48	11.81
fr3/str-notex-far	17.07	35.41	18.23	**0.80**	2.60	3.10	9.40	2.18	6.41
fr3/long_off_hous	✗	34.25	20.93	27.60	9.00	8.50	**7.83**	9.38	9.81
Comparison of the Relative Pose Error (RPE) in [cm/s]									
fr1/xyz	3.14	3.22	2.71	**1.40**	4.50	1.90	3.04	2.25	2.27
fr1/rpy	11.45	3.34	5.41	3.70	6.30	3.40	3.56	**3.30**	3.86
fr1/desk	10.01	3.78	5.20	5.10	7.50	**3.10**	3.33	3.44	3.38
fr1/desk2	16.42	5.79	5.60	7.40	15.60	13.10	6.33	6.30	**4.63**
fr1/plant	16.62	**2.56**	6.72	4.40	5.00	3.60	2.83	2.82	2.64
fr1/360	14.33	16.01	8.15	**6.50**	21.10	12.10	6.79	12.60	7.74
fr2/desk	✗	2.43	1.41	3.00	**0.80**	**0.80**	1.43	1.83	0.92
fr2/xyz	2.68	1.31	0.44	0.50	**0.30**	0.40	0.40	0.55	0.36
fr2/rpy	5.16	1.55	0.46	0.40	**0.30**	0.40	0.50	0.41	0.37
fr2/desk_w_pers	9.85	3.01	3.11	5.60	**0.80**	0.90	2.73	1.62	1.05
fr2/dishes	✗	3.93	3.88	3.50	1.60	1.20	2.06	3.01	**1.05**
fr3/large-cab	✗	21.45	**9.89**	10.00	19.00	16.70	22.89	15.22	23.80
fr3/cabinet	16.97	14.83	11.39	7.10	5.80	**3.60**	5.97	7.06	4.53
fr3/str-ntex-far	10.98	11.71	9.32	**1.30**	14.40	2.70	4.08	2.17	4.91
fr3/long_off_hous	✗	2.29	1.46	1.90	1.40	**1.00**	1.42	4.20	1.04

Table 6.3: *VO* evaluation on the TUM RGBD dataset for: an *ICP* implementation [157] and DVO [81] in a frame-to-frame setup, the feature-based FOVIS [73] and ORB-SLAM2 [116] (in *VO* mode), two other edge-based methods using ANNF [164] and ONNF [165] compared to our REVO (KF), REVO+ICP with all optimizations and RESLAM with the full local *BA*. The best overall results in **bold** and the best result of edge-based methods is underlined.

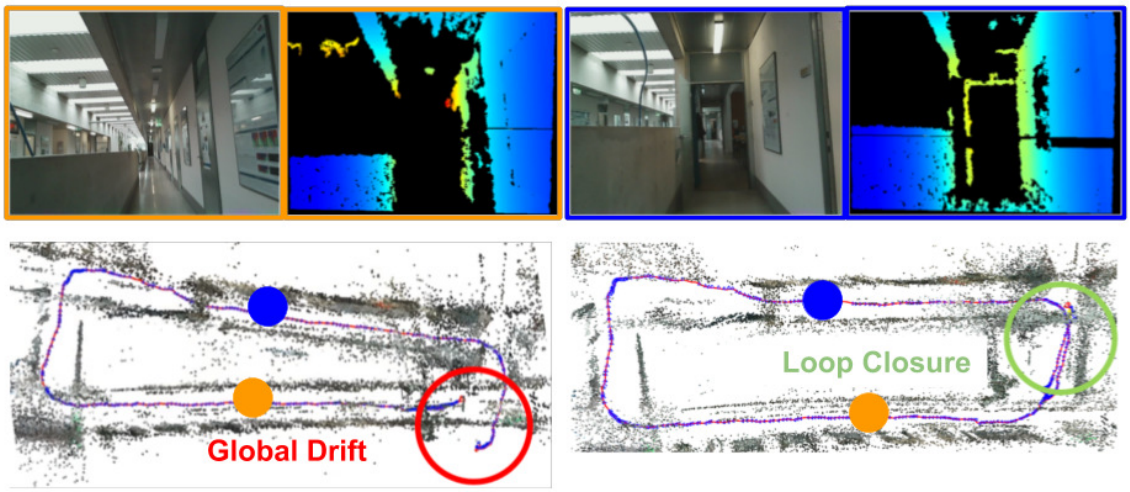


Figure 6.8: Reconstruction of a long sequence with significant drift before and after loop closure. The drift is mainly caused by the many invalid values in the depth map. Two problematic RGBD frames are depicted with their locations in the sequence marked by orange and blue dots.

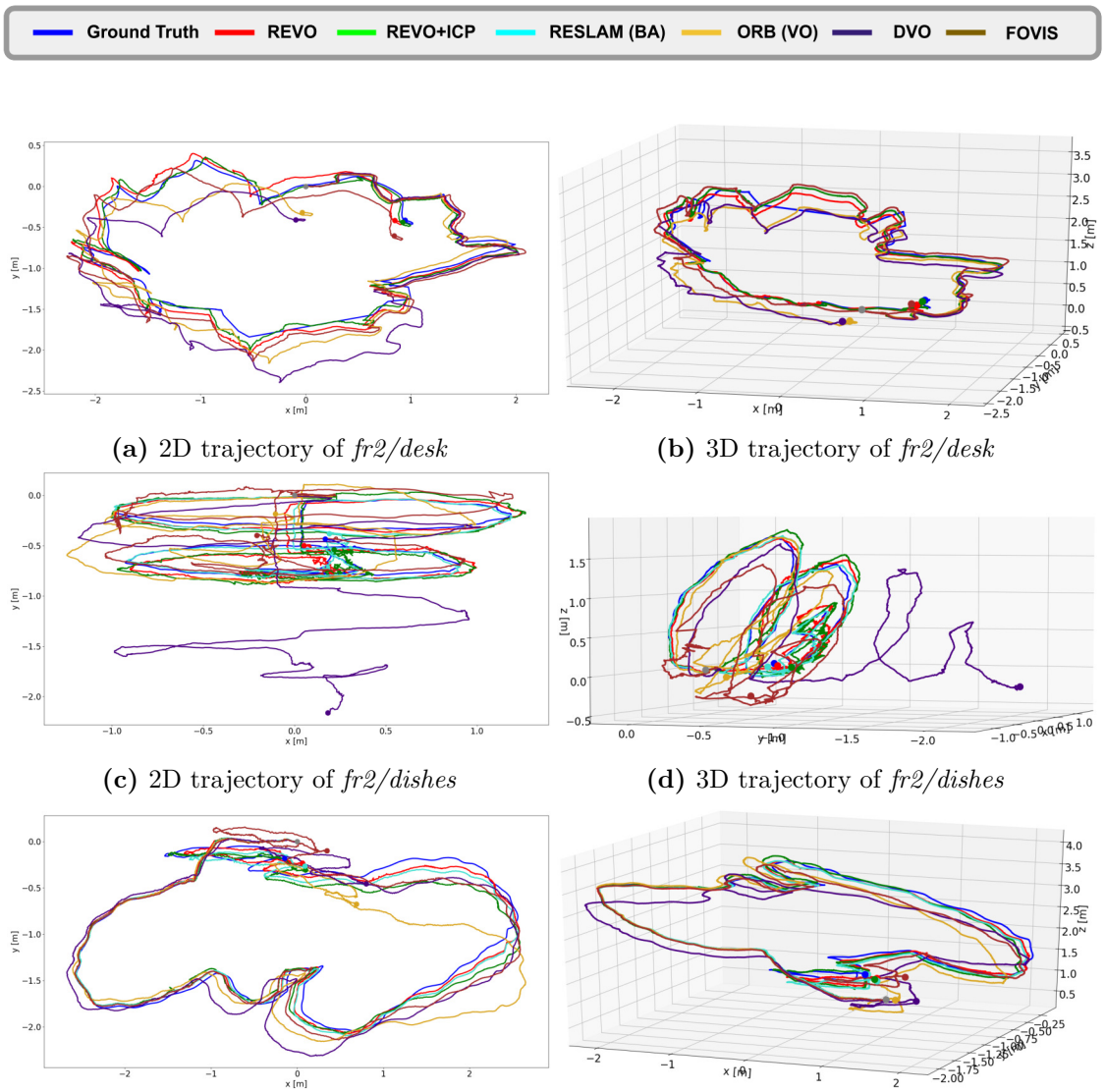
Comparison of the Absolute Trajectory Error (ATE) [cm]

			Features		Edge-based		Our methods [134]		
Seq.	ICP	DVO	FOVIS	ORB2	NNF		REVO	R+ICP	RESLAM
Seq.	D	RGBD	Fast	VO	ANNF	ONNF	KF	Opt	BA
icl/living_room-kt0	69.74	21.17	✗	4.30	7.40	**3.50**	✗	5.46	3.99
icl/living_room-kt1	4.54	23.40	✗	8.20	11.90	2.30	2.16	**1.23**	1.40
icl/office-kt0	30.31	43.19	✗	13.09*	-	-	1.80	**0.66**	4.65
icl/office-kt1	27.52	47.77	✗	6.63*	-	-	1.05	**0.75**	2.38
Comparison of the Relative Pose Error (RPE) in [cm/s]									
icl/living_room-kt0	15.77	9.01	✗	3.00	4.70	**1.40**	✗	1.89	1.53
icl/living_room-kt1	1.77	9.14	✗	2.20	5.90	0.90	**0.86**	1.36	**0.86**
icl/office-kt0	16.91	10.67	✗	3.74*	-	-	0.58	**0.51**	1.03
icl/office-kt1	16.06	19.04	✗	3.11*	-	-	0.69	**0.53**	1.01

Table 6.4: *VO* evaluation on the ICL-NUIM dataset for: an *ICP* implementation [157] and DVO [81] in a frame-to-frame setup, the feature-based FOVIS [73] and ORB-SLAM2 [116] (in *VO* mode), two other edge-based methods using ANNF [164] and ONNF [165] compared to our REVO (KF), REVO+ICP with local refinement and RESLAM with the full local *BA*. The best overall results in **bold**, the best results of edge-based methods are underlined and ORB2 results achieved with the available implementation are marked by *.

6.4 SLAM Evaluations

In this section, we present an extensive evaluations of RGBD *SLAM* systems and demonstrate their performance on the well-known TUM RGBD and ICL-NUIM datasets as



(a) 2D trajectory of *fr2/desk* **(b)** 3D trajectory of *fr2/desk*

(c) 2D trajectory of *fr2/dishes* **(d)** 3D trajectory of *fr2/dishes*

(e) 2D trajectory of *fr3/long_office_household* **(f)** 3D trajectory of *fr3/long_office_household*

Figure 6.9: 2D and 3D trajectories from three TUM RGBD sequences, where the starting point of all the trajectories is marked by a grey dot and their respective endpoints by colored dots.

well as on the recently released ETH3D dataset (see Sec. 6.1.1 for more details). For the TUM RGBD dataset, we select nine different RGBD SLAM systems and compare them to RESLAM with LC and with LC and *BA*. The selected systems are the feature-based RGBDSLAM (RS) [36] and ORB-SLAM2 [116], the combined direct and indirect approaches RKSLAM (RK) [99] and BundleFusion (BF) [24], and five direct RGBD methods DVO-SLAM [80], Kintinuous (Kint) [157], BAD-SLAM (BS) [136], ElasticFusion (EF) [159] and RGBDTAM (TAM) [23]. BAD-SLAM and RGBDTAM do not present results for ICL-NUIM in their papers [23, 136], thus we do not list them but show scores

for InfiniTAM (ITM) [79] instead. A detailed review of the various works is given in Section 2. To avoid any threshold bias for the other systems, we copy the results from the respective papers and where not available, mark the sequence by a −. The only exception is ORB-SLAM2, where we run the system with the provided config files for the missing sequences in TUM RGBD and all the sequences in ICL-NUIM. Due to random component introduced by RANSAC, we run ORB-SLAM2 five times and take the median result. Those sequences are marked by * in Tables 6.5 and 6.6. Since most state-of-the-art systems only provide results for the four living-room sequences, we additionally evaluate *icl/living_room-kt2* and *icl/living_room-kt3* with our methods.

Table 6.5 shows scores on 15 **TUM RGBD** sequences. ORB-SLAM2 [116] and RGB-DTAM [23] show an impressive overall performance, which stems mainly from their ability to generate points even without an initialization from the RGBD sensor and to refine the depth of the points. Especially in sequences, where there is one central object and far away background with mostly invalid depths, e.g. *fr2/dishes*, *fr2/desk*, this increases the overall accuracy. However, ORB-SLAM2 fails in the three nearly feature-less *fr3* scenes, where RESLAM still manages to estimate the trajectory. Even though RESLAM is not the most accurate method on the TUM RGBD datasets it is in the range of other state-of-the-art methods and shows convincing results on a wide variety of sequences.

Table 6.6 summarizes the *ATE* achieved on six different sequences from the synthetic ICL-NUIM dataset. Due to the perfect synchronization and noise free depth maps, methods that also process depth and align against a model such as InfiniTAM (ITM) [79], BundleFusion (BF) [24], and ElasticFusion (EF) [159] show a higher accuracy than purely vision based methods, e.g. ORB-SLAM2 or RESLAM. However, RESLAM again achieves accuracies comparable to other state-of-the-art methods, while relying solely on visual information and only using the depth map for initialization.

For the recent **ETH3D** dataset, we select 17 datasets from the training set and use the results for BAD-SLAM [136], DVO-SLAM [80], ElasticFusion (EF) [159], BundleFusion (BF) [24] and ORB-SLAM2 [116] as given by Schöps et al. in [136] and on their website[3]. We only run RESLAM with LC and *BA* but additionally present results achieved with two machine-learned edge detectors BDCN [66] and RCF [100, 101]. In general, the **ETH3D** sequences are more difficult than the TUM RGBD sequences due to the low illumination and and often uniform texture, e.g. the sofa in *sofa_1*. Judging from the evaluations scores in Table 6.7, BAD-SLAM is the best overall method but RESLAM also performs well on all of the sequences. It is interesting to see that the RESLAM with the machine-learned detectors shows higher accuracy than Canny for most of the sequences. While *einstein_1*, *mannequin_3-4* and *sofa_1-2* fail with Canny even after adapting the threshold for this specific scene, RESLAM manages to estimate the trajectory with the learned detectors. Some of those sequences also fail with other methods and only RESLAM and ORB-SLAM2 work for *mannequin_4*. This is mainly due to the increased robustness in low

[3] `https://www.eth3d.net/slam_benchmark`

Comparison of the Absolute Trajectory Error (ATE) [*cm*]

	RS	ORB	RK	BF	DVO	Kint	BS	EF	TAM	RESLAM	
Seq.	Features		RGBD+Feat.		Direct - RGBD					LC	SLAM
fr1/xyz	1.4	1.0*	**0.7**	1.6	1.1	1.7	1.6	1.1	1.0	1.9	2.3
fr1/rpy	2.6	2.5*	3.7	-	**2.0**	2.8	-	2.5	2.1	2.3	2.8
fr1/desk	2.3	**1.6**	2.1	-	2.1	3.7	-	2.0	2.7	4.1	3.2
fr1/desk2	4.4	**2.2**	2.4	-	4.6	7.1	-	4.8	4.2	5.6	4.9
fr1/plant	9.1	**1.4***	3.8	-	2.8	4.7	-	2.2	2.5	5.4	5.8
fr1/360	-	18.6*	10.9	-	**8.3**	-	-	10.8	10.1	12.0	13.6
fr2/desk	5.7	**0.9**	7.1	-	1.7	3.4	-	7.1	2.7	2.3	2.1
fr2/xyz	0.8	**0.4**	1.2	1.7	-	2.9	1.1	1.1	0.7	0.7	0.5
fr2/rpy	-	0.3*	0.6	-	-	-	-	1.5	**0.2**	0.6	0.6
fr2/desk_w_pers	-	**0.8***	4.5	-	-	-	-	-	-	7.6	3.6
fr2/dishes	-	5.6*	7.9	-	-	-	-	-	**3.6**	6.2	3.8
fr3/large-cabinet	-	✗*	14.8	-	-	-	-	9.9	**7.0**	✗	35.8
fr3/cabinet	-	✗*	7.9	-	-	-	-	-	**5.7**	29.2	11.8
fr3/str-ntex-far	-	✗*	-	-	-	-	-	7.4	**2.6**	7.4	6.41
fr3/long_off_hous	3.2	1.0	2.8	2.2	3.5	3.0	1.7	**1.7**	2.7	4.5	4.2

Table 6.5: *SLAM* evaluations on the TUM RGBD dataset for the feature-based RGBDSLAM (RS) [36] and ORB-SLAM2 [116], the combined approaches RKSLAM (RK) [99] and BundleFusion (BF) [24], and five direct RGBD methods DVO-SLAM [80], Kintinuous (Kint.) [157], BAD-SLAM (BS) [136], ElasticFusion (EF) [159] and RGBDTAM (TAM) [23] compared to our RESLAM with LC and with LC and *BA*. Results computed with publicly available code are marked by * and best results in **bold**.

illumination scenes and stability during fast-paced motions as depicted for *mannequin_2* and *plant_scene_2* in Figure 6.10. Another strength of the learned detectors is that they focus on object borders and not on every little texture detail like Canny. For example, in the *planar_2* sequence depicted in Figure 6.10, Canny detects an abundance of edges, while RCF and BDCN detect significantly less. A high number of edges not just increases the time required for optimization but can also introduce a bias into the motion-estimation, whenever the detections are concentrated in specific scene parts.

Comparison of the Absolute Trajectory Error (ATE) [*cm*]

	RS	ORB	RK	BF	DVO	Kint	EF	ITM	RESLAM	
Seq.	Features		RGBD+Feat.		Direct - RGBD			ICP	LC	SLAM
icl/living_room-kt0	2.6	0.7*	1.8	**0.6**	10.4	7.2	0.9	0.9	2.1	3.4
icl/living_room-kt1	0.8	13.8*	1.6	0.4	2.9	0.5	0.9	**0.3**	0.9	1.5
icl/living_room-kt2	1.8	2.3*	3.2	**0.6**	19.1	1.0	1.4	0.9	2.1	2.0
icl/living_room-kt3	43.3	1.4*	-	**1.1**	15.2	35.5	10.6	4.1	11.7	10.8
icl/office-kt0	-	2.8*	-	-	-	-		**0.6**	1.5	2.2
icl/office-kt1	-	68.8*	-	-	-	-		2.8	**1.8**	2.7

Table 6.6: *SLAM* evaluations on the ICL-NUIM dataset for the feature-based RGBDSLAM (RS) [36] and ORB-SLAM2 [116], the combined approaches RKSLAM (RK) [99] and BundleFusion (BF) [24], and four direct RGBD methods DVO-SLAM [80], Kintinuous (Kint.) [157] ElasticFusion (EF) [159], and ICP-only InfiniTAM (ITM) [79] compared to our RESLAM with LC and with LC and *BA*. Results computed with publicly available code are marked by * and best results in **bold**.

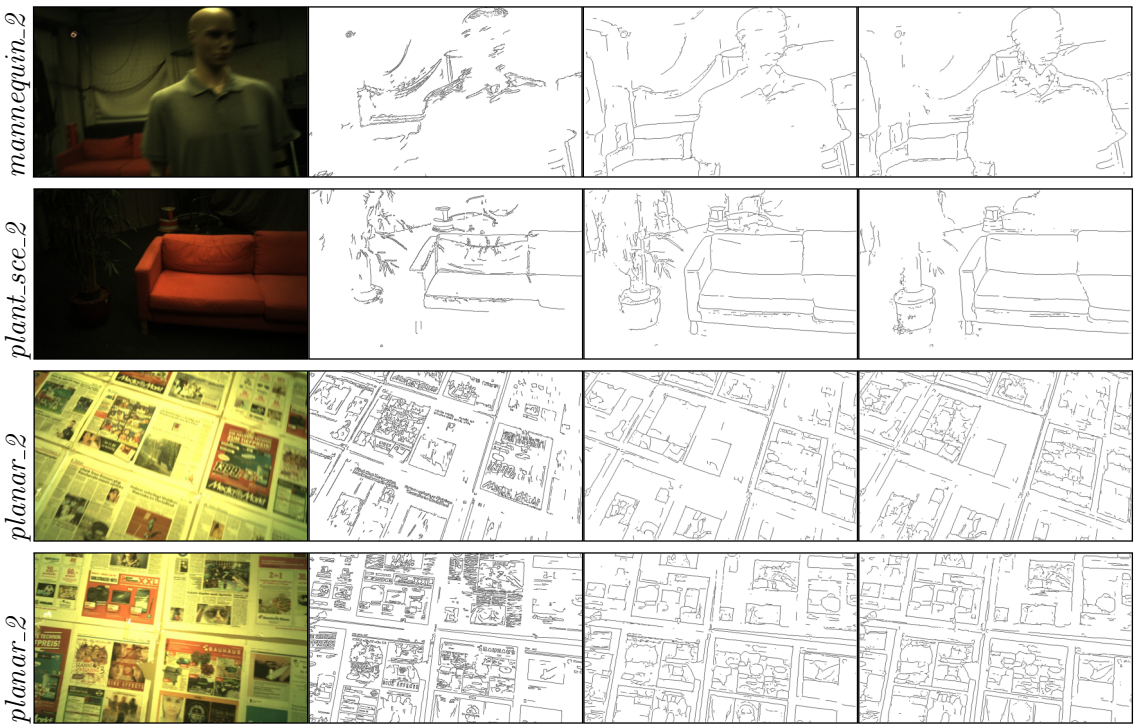


Figure 6.10: Sample images from the ETH3D sequences *mannequin_4*,*plant_scene_2* and *planar_2* with edge detections from Canny [16] and state-of-the-art learned detectors RCF [100, 101] and BDCN [66]. The learned detectors are more robust against the motion blur and low illumination as shown for *mannequin_4* and *plant_scene_2*. In *planar_2*, the learned detectors focus on object boundaries instead of the fine-grained textures as Canny.

Comparison of the Absolute Trajectory Error (ATE) [*cm*]

	BS [136]	DVO [80]	EF [159]	BF [24]	ORB-SLAM [116]	RESLAM		
Seq.	Geometric+Photometric				Features	Canny	BDCN	RCF
cables_1	0.68	**0.44**	1.18	2.24	0.74	0.93	1.18	1.13
cables_2	**0.51**	✗	1.51	9.61	0.80	0.63	0.59	0.87
einstein_1	**0.30**	0.51	2.83	2.89	0.40	0.97	1.23	1.78
einstein_2	**0.58**	1.54	✗	✗	✗	✗	9.00	4.19
ein_g_l_c_2	**0.29**	0.86	1.11	1.62	0.86	0.89	0.83	1.06
mann_3	**0.55**	2.05	✗	✗	1.52	✗	14.13	7.87
mann_4	✗	✗	✗	✗	4.98	✗	7.93	14.85
mann_face_1	0.39	0.52	✗	1.34	**0.37**	3.19	0.69	0.39
mann_face_2	**0.10**	0.34	1.00	0.91	1.00	0.38	0.37	0.16
planar_2	0.30	**0.24**	1.06	0.34	0.52	19.99	1.46	0.38
plant_scene_1	**1.09**	2.72	✗	3.35	✗	6.70	2.11	3.24
plant_scene_2	**1.52**	4.04	✗	✗	1.59	15.1	5.93	1.90
sofa_1	**0.22**	0.67	8.36	2.59	✗	✗	5.11	3.63
sofa_2	**0.36**	0.82	✗	✗	✗	✗	9.65	8.38
table_3	**0.24**	0.82	✗	1.73	0.56	1.92	2.47	2.42
table_4	**0.22**	1.82	1.25	✗	0.83	2.64	3.04	2.25
table_7	**0.28**	0.69	✗	1.02	0.69	1.51	1.67	1.29

Table 6.7: *SLAM* evaluations on the ETH3D dataset for four direct RGBD methods BAD-SLAM (BS), DVO-SLAM (DVO) [80], ElasticFusion (EF) [159] and BundleFusion (BF) [24], and the feature-based ORB-SLAM2 [116] compared to our RESLAM with LC and with LC and *BA*. Best results in **bold** and best edge-based result underlined.

6.5 Conclusion

In this chapter, we presented extensive quantitative and qualitative experiments on three different benchmark datasets covering a wide variety of scenes and camera motions. In the first part, we study the difference between all our proposed methods and the improvements given by each of their individual components. The results show that by far the biggest boost in accuracy comes from switching from frame-to-frame *VO* to a *KF*-based setting. The additional geometric term increases robustness but only has negligible effects on the overall accuracy, while introducing local optimization strategies either on spatially close *KFs* or a full local *BA*, greatly reduces the local drift. In our most recent work, we additionally perform loop closure to also correct drift on a larger scale (see. Fig. 6.8).

In the second part, we demonstrate that edge-based methods generalize well to various scenes and that they are a viable choice for the *VO* task. The results show that our edge-based methods can deal with fast-paced motions, e.g. *fr1/desk* and *fr1/desk2*, and also the scarcely textured environments *icl/living_room_0 - 1*. Our methods outperform several well-known systems such as DVO [81] or FOVIS [73], while running at a high frame rate on a CPU.

In the final part of this chapter, we compare our recent RESLAM to various well-known *SLAM* systems. Even though RESLAM does not achieve the best overall accuracy, it performs in the range of other state-of-the-art methods. In the case of the new and challenging ETH3D datasets, RESLAM even works, where many well-established systems fail. The results on ETH3D indicate that there is a huge potential for machine learned edge detectors especially in scenes with low illumination or during fast-paced motions. Compared to the traditional Canny, these new methods are far more robust even though they were not trained for the *SLAM* task.

Throughout the experiments, our edge-based methods showed impressive results for the *VO* task and performed comparable to other state-of-the-art systems for *SLAM*. All our methods run in real-time one a CPU, while the best performing *SLAM* systems require one [78, 136, 157, 159] or even two [24] GPUs.

7 Conclusion

Modern technologies such as autonomous driving, Unmanned Aerial Vehicles (UAVs) and Augmented Reality (AR) drive the demand for new, fast and robust algorithms to solve the Simultaneous Localization and Mapping (SLAM) or Visual Odometry (VO) problem. While most state-of-the-art approaches either apply an indirect formulation using features or a direct formulation based on a photometric and/or geometric error, we focused on a new family of direct algorithms in this thesis. We propose several novel direct edge-based algorithms to tackle the challenges of *VO* and *SLAM* with an RGBD sensor in indoor environments.

7.1 Summary

Our first contribution is REVO, one of the first edge-based RGBD *VO* algorithms, which is faster and more accurate than previous ones [89]. We show how to formulate the edge-based relative pose estimation problem and demonstrate how to solve it efficiently, including all the derivatives and optimization tricks. Since our main focus is on indoor scenes, which often contain nearly edge-less areas, we increase the robustness by combining our edge-based relative pose estimation with an additional geometric term, thereby using all the information from the RGBD sensor. This combination is the first of its kind and manages to even work in nearly texture-free scenes without any problems. With an additional local refinement on spatially close Keyframes (KFs), we manage to increase the accuracy even further.

These ideas culminated in RESLAM, which was to the best of our knowledge the first edge-based RGBD SLAM at the time of its release. While at its core, there is still the relative pose estimation presented in REVO, it contains several enhancements to make it more robust and to reduce the computational effort of edge detection. Instead of just refining the pose of the most recent *KF*, RESLAM introduces a full local Bundle Adjustment (BA) to refine the poses, initial depth of edge points and camera intrinsics within a sliding window. RESLAM also runs a global mapping module to handle relocalization and loop

closure, relying solely on the Edge-based Quality Assessment (EBQA) as a verification step.

The evaluations show that our methods are the most accurate compared to many well-established state-of-the-art algorithms for the *VO* problem. Further, RESLAM also achieves convincing results for the *SLAM* task, performing similar to well-known systems, while only requiring a CPU instead of one or two GPUs as other approaches. This is especially promising since it is the first system of its kind and the others follow well-studied techniques often known for decades. Another important point is that for difficult scenes with low illumination or fast-paced motions, RESLAM with machine-learned edge detectors even runs in sequences, where others fail.

7.2 Outlook

In this thesis, we focused on edge-based algorithms, which are relatively new in the field of *VO* and *SLAM*. Thus, there are still many open questions to be addressed in future research on edge-based methods as well as for *VO* and *SLAM* in general. To facilitate progress in the area of edge-based *VO* and *SLAM*, REVO and RESLAM are available as open-source software.

Robust Edge Detection

Throughout the past years, machine-learning algorithms have fundamentally changed all areas of computer vision. As shown in our experiments in Chapter 5 and Section 6.4, there is huge potential for learned edge detectors especially in scarcely textured or low illumination scenes, where classical Canny has severe problems. A possible future research direction is the training of an edge detector specifically for edge-based relative pose estimation. Our proposed *EBQA* counts the number of overlapping edges by reprojecting between multiple frames, where strong edges are likely to be seen in all the frames, while weak edges are only present in one or very few frames. This implicitly generates a score for the individual edges, which can then be used to create training data to learn a robust edge detector.

Depth Estimation

While RGBD sensors work reasonably well in small rooms, they have problems with reflective and far away surfaces and glass (see Fig. 1.4). Currently, our algorithms do not use edges without a valid depth initialization, which can lead to problems in certain scenes. Adding the capability to estimate the depth of these edges by triangulating between different views, would add stability and accuracy to the overall system. This could even be taken one step further, by completely abandoning RGBD sensors and switching to a monocular camera, which is available in all common handheld devices such as smartphones or tablets. The high frame rates even on a single CPU make our methods well-suited to

run even on devices with limited resources. Instead of abandoning the initial depth completely, learned single image depth estimation is a very interesting alternative because it could also work for degenerated motions, e.g. pure rotation, where a purely monocular setup fails.

Loop Closing Strategy

The only part of RESLAM that is currently not edge-based but uses Random Ferns [55], is the search for potential loop closure or relocalization candidates (see Sec. 4.5.3). Developing an edge-based descriptor or even a direct matching approach would be an interesting research area. Using the edges directly would avoid the additional computations for the Random Fern descriptor and edges also have the benefit of also encoding the geometric properties of the scene.

Optimization Algorithm

In their recent works, Liu et al. [98, 99] proposed several potential speed-ups for local and global *BA*. While their most recent work [98] relies on features, the underlying simplifications in the optimization can also be applied to our full local *BA* and potentially also to the Pose Graph Optimization (PGO). This could make our direct edge-based methods even faster and might facilitate the use in low-cost or onboard devices.

Additional Sensor Information

Some of the RGBD sensor released in the last two years such as the Intel RealSense D435i and T265 come with an integrated, calibrated and synchronized Inertial Measurement Unit (IMU). A logical next step is to integrate the *IMU* into RESLAM to improve the overall accuracy. Further, the *IMU* measurements are a good prior to initialize the relative motion between consecutive frames especially during fast-paced movements but can also be used to skip frames without sufficient motion to save computation time. Apart from the initialization, *IMU* measurements can prevent the incorporation of potentially wrong loop closures, which would otherwise pass a purely image-based verification step.

A

List of Acronyms and Symbols

AR	Augmented Reality
ATE	Absolute Trajectory Error
BA	Bundle Adjustment
CNN	Convolutional Neural Network
DOF	Degrees of Freedom
DT	Distance Transform
EBQA	Edge-based Quality Assessment
EE	Edge Enhancement
GN	Gauss-Newton
GPU	Graphics Processing Unit
GT	Ground Truth
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
IRLS	Iteratively Reweighted Least-Squares
KF	Keyframe
LM	Levenberg-Marquardt
MLD	Multi-Level Edge Detection
NN	Nearest Neighbor

PCL	Point Cloud
PGO	Pose Graph Optimization
RMSE	Root Mean Squared Error
RPE	Relative Pose Error
SC	Schur complement
SfM	Structure from Motion
SLAM	Simultaneous Localization and Mapping
SLD	Single-Level Edge Detection
UAV	Unmanned Aerial Vehicle
VO	Visual Odometry

B

List of Publications

During my work at the Institute for Computer Graphics and Vision, I had the opportunity to work on various topics, which led to the following peer-reviewed publications. The publications and their respective abstracts are listed in chronological order starting with the most recent work.

B.1 Publications related to this Thesis

RESLAM: A robust edge-based SLAM System

Fabian Schenk, Friedrich Fraundorfer
In: *International Conference on Robotics and Automation (ICRA)*
May, 2019, Montreal, Canada
Accepted for poster presentation

Abstract: Simultaneous Localization and Mapping is a key requirement for many practical applications in robotics. In this work, we present RESLAM, a novel edge-based SLAM system for RGBD sensors. Due to their sparse representation, larger convergence basin and stability under illumination changes, edges are a promising alternative to feature-based or other direct approaches. We build a complete SLAM pipeline with camera pose estimation, sliding window optimization, loop closure and relocalization capabilities that utilizes edges throughout all steps. In our system, we additionally refine the initial depth from the sensor, the camera poses and the camera intrinsics in a sliding window to increase accuracy. Further, we introduce an edge-based verification for loop closures that can also be applied for relocalization. We evaluate RESLAM on wide variety of benchmark datasets that include difficult scenes and camera motions and also present qualitative results. We show that this novel edge-based SLAM system performs comparable to state-of-the-art methods, while running in real-time on a CPU. RESLAM is available as open-source software.

Combining Edge Images and Depth Maps for Robust Visual Odometry

Fabian Schenk, Friedrich Fraundorfer
In: *British Machine Vision Conference (BMVC)*
September 2017, London, United Kingdom
Accepted for poster presentation

Abstract: In this work, we propose a robust visual odometry system for RGBD sensors. The core of our method is a combination of edge images and depth maps for joint camera pose estimation. Edges are more stable under varying lighting conditions than raw intensity values and depth maps further add stability in poorly textured environments. This leads to higher accuracy and robustness in scenes, where feature- or photo-consistency-based approaches often fail. We demonstrate the robustness of our method under challenging conditions on various real-world scenarios recorded with our own RGBD sensor. Further, we evaluate on several sequences from standard benchmark datasets covering a wide variety of scenes and camera motions. The results show that our method performs best in terms of trajectory accuracy for most of the sequences indicating that the chosen combination of edge and depth terms in the cost function is suitable for a multitude of scenes.

Robust Edge-based Visual Odometry using Machine-Learned Edges

Fabian Schenk, Friedrich Fraundorfer
In: *International Conference on Intelligent Robots and Systems (IROS)*
September 2017, Vancouver, Canada
Accepted for oral presentation

Abstract: In this work, we present a real-time robust edge-based visual odometry framework for RGBD sensors (REVO). Even though our method is independent of the edge detection algorithm, we show that the use of state-of-the-art machine-learned edges gives significant improvements in terms of robustness and accuracy compared to standard edge detection methods. In contrast to approaches that heavily rely on the photo-consistency assumption, edges are less influenced by lighting changes and the sparse edge representation offers a larger convergence basin while the pose estimates are also very fast to compute. Further, we introduce a measure for tracking quality, which we use to determine when to insert a new key frame. We show the feasibility of our system on real world datasets and extensively evaluate on standard benchmark sequences to demonstrate the performance in a wide variety of scenes and camera motions. Our framework runs in real-time on the CPU of a laptop computer and is available online.

B.2 Other Publications

Globally Consistent Dense Real-Time 3D Reconstruction from RGBD Data

Rafael Jakob Weilharter, **Fabian Schenk**, Friedrich Fraundorfer
In: *Austrian Association for Pattern Recognition (OAGM) Workshop*
June 2018, Hall, Austria
Accepted for oral presentation, **Best Paper**

Abstract: In this work, we present a dense 3D reconstruction framework for RGBD data that can handle loop closure and pose updates online. Handling updates online is essential to get a globally consistent 3D reconstruction in real-time. We also introduce fused depth maps for each keyframe that contain the fused depths of all associated frames to greatly increase the speed for model updates. Furthermore, we show how we can use integration and de-integration in a volumetric fusion system to adjust our model to online updated camera poses. We build our system on top of the InfiniTAM framework to generate a model from the semi-dense, keyframe based ORB-SLAM2. We extensively evaluate our system on real world and synthetic generated RGBD data regarding tracking accuracy and surface reconstruction.

Automatic Muck Pile Characterization from UAV Images

Fabian Schenk, Alexander Tscharf, Gerhard Mayer, Friedrich Fraundorfer
In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*
June, 2019, Enschede, Netherlands
Accepted for oral presentation

Abstract: In open pit mining it is essential for processing and production scheduling to receive fast and accurate information about the fragmentation of a muck pile after a blast. In this work, we propose a novel machine-learning method that characterizes the muck pile directly from UAV images. In contrast to state-of-the-art approaches, that require heavy user interaction, expert knowledge and careful threshold settings, our method works fully automatically. We compute segmentation masks, bounding boxes and confidence values for each individual fragment in the muck pile on multiple scales to generate a globally consistent segmentation. Additionally, we recorded lab and real-world images to generate our own dataset for training the network. Our method shows very promising quantitative and qualitative results in all our experiments. Further, the results clearly indicate that our method generalizes to previously unseen data.

Guided Sparse Camera Pose Estimation

Fabian Schenk, Ludwig Mohr, Matthias Rüther, Friedrich Fraundorfer, Horst Bischof
In: *Proceedings of the OAGM Workshop*
May 2016, Wels, Austria
Accepted for oral presentation

Abstract: In this paper, we present an idea for a sparse approach to calculate camera poses from RGB images and laser distance measurements to perform subsequent facade reconstruction. The core idea is to guide the image recording process by choosing distinctive features with the laser range finder, e.g. building or window corners. From these distinctive features, we can establish correspondences between views to compute metrically accurate camera poses from just a few precise measurements. In our experiments, we achieve reasonable results in building facade reconstruction with only a fraction of features compared to standard structure from motion.

Automated Segmentation of the Walkable Area from Aerial Images for Evacuation Simulation

Fabian Schenk, Matthias Rüther, Horst Bischof
In: *Proceedings of the 2nd International Conference on Geographical Information Systems Theory, Applications and Management (GISTAM)*
May 2016, Rome, Italy
Accepted for oral presentation

Abstract: Computer-aided evacuation simulation is a very import preliminary step when planning safety measures for major public events. We propose a novel, efficient and fast method to extract the walkable area from high resolution aerial images for the purpose of evacuation simulation. In contrast to previous work, where the authors only extracted streets and roads or worked on indoor scenarios, we present an approach to accurately segment the walkable area of large outdoor areas. For this task we use a sophisticated seeded region growing (SRG) algorithm incorporating the information of digital surface models, true-orthophotos and inclination maps calculated from aerial images. Further, we introduce a new annotation and evaluation scheme especially designed for assessing the segmentation quality of evacuation maps. An extensive qualitative and quantitative evaluation, where we study various combinations of SRG methods and parameter settings by the example of different real-world scenarios, shows the feasibility of our approach.

Automated Walkable Area Segmentation from Aerial Images for Evacuation Simulation

Fabian Schenk, Matthias Rüther, Horst Bischof
In: *International Conference on Geographical Information Systems Theory, Applications and Management (GISTAM)*
Accepted for Journal presentation

Abstract: In this paper, we propose a novel, efficient and fast method to extract the walkable area from high-resolution aerial images for the purpose of computer-aided evacuation simulation for major public events. Compared to previous work, where authors only extracted roads and streets or solely focused on indoor scenarios, we present an approach to fully segment the walkable area of large outdoor environments. We address this challenge by modeling human movements in the terrain with a sophisticated seeded region growing algorithm (SRG), which utilizes digital surface models, true-orthophotos and inclination maps computed from aerial images. Further, we propose a novel annotation and scoring scheme especially developed for assessing the quality of the extracted evacuation maps. Finally, we present an extensive quantitative and qualitative evaluation, where we show the feasibility of our approach by evaluating different combinations of SRG methods and parameter settings on several real-world scenarios.

C Appendix

C.1 Efficient Jacobian and Hessian computations

The speed of our method depends largely on the performance of the optimization backend. Apart from the residual evaluation, the computation of the Jacobians $\mathbf{J}$ and the approximation of the Hessian $\mathbf{H} \approx \mathbf{J^T J}$ are the most computationally demanding parts. We will discuss their efficient implementations by the example of an equation system for the relative motion estimation problem:

$$\mathbf{H}_{6\times6}\mathbf{x}_{6\times1} = \mathbf{b}_{6\times1} = \mathbf{J}_{6\times N}\mathbf{r}_{N\times1}, \tag{C.1}$$

where $\mathbf{r}$ is a vector of stacked residuals, $\mathbf{J}$ the corresponding Jacobians and $\mathbf{H}$ the approximated Hessian.

In a toy example, we assume 3 residuals and 2 variables:

$$\begin{aligned}\mathbf{J} &= \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} = \begin{pmatrix} \mathbf{J}_0 & \mathbf{J}_1 & \mathbf{J}_2 \end{pmatrix} \\ \mathbf{J}^T &= \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix} = \begin{pmatrix} \mathbf{J}_0 \\ \mathbf{J}_1 \\ \mathbf{J}_2 \end{pmatrix}.\end{aligned} \tag{C.2}$$

Stacking all residuals together and computing one large matrix is neither memory

efficient nor easy to parallelize. Instead, we split (C.3) into several computations:

$$\begin{aligned}
\mathbf{H} &= \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix} \\
&= \begin{pmatrix} a \\ d \end{pmatrix} \begin{pmatrix} a & d \end{pmatrix} + \begin{pmatrix} b \\ e \end{pmatrix} \begin{pmatrix} b & e \end{pmatrix} + \begin{pmatrix} c \\ f \end{pmatrix} \begin{pmatrix} c & f \end{pmatrix} \\
&= \begin{pmatrix} a^2 & ad \\ ad & d^2 \end{pmatrix} + \begin{pmatrix} b^2 & be \\ be & e^2 \end{pmatrix} + \begin{pmatrix} c^2 & cf \\ cf & f^2 \end{pmatrix} \\
&= \begin{pmatrix} a^2+b^2+c^2 & ad+be+cf \\ ad+be+cf & a^2+b^2+c^2 \end{pmatrix}
\end{aligned} \tag{C.3}$$

With this formulation, we can compute $\mathbf{H}_i = \mathbf{J}_i^T \mathbf{J}_i$ for all r_i separately and finally the overall Hessian $\mathbf{H}$ as:

$$\mathbf{H} = \sum_i \mathbf{H}_i. \tag{C.4}$$

This makes it easily possible to distribute the computations to several threads and then sum up the resulting Hessians. Since $\mathbf{H}_i = \mathbf{J}_i^T \mathbf{J}_i$ is symmetric, it suffices to only compute the upper triangular part.

C.2 Additional Mathematical Definitions

C.2.1 The Iverson Bracket

The Iverson bracket $[\![.]\!]$ is very useful to describe Boolean evaluations mathematically. It is defined as:

$$[\![C]\!] = \begin{cases} 1 & \text{if C is true,} \\ 0 & \text{otherwise.} \end{cases} \tag{C.5}$$

C.2.2 The *vec* Operator

The *vec* operator expands a matrix into a vector by stacking all columns of an $N \times M$ matrix into an $NM \times 1$ vector. For a matrix $\mathbf{A}$, the *vec* operator gives:

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \equiv vec(\mathbf{A}) = (a, c, b, d)^T \tag{C.6}$$

C.2.3 The Skew-symmetric Matrix

The operator $[.]_\times$ generates a 3×3 *skew-symmetric* or *cross-product* matrix from a vector $\mathbf{x} = [x, y, z]^T$ as:

$$[\mathbf{x}]_\times = \begin{pmatrix} x \\ y \\ z \end{pmatrix}_\times = \begin{pmatrix} 0 & -z & y \\ z & 0 & x \\ -y & -x & 0 \end{pmatrix} \tag{C.7}$$

Its inverse operator $[.]_\vee$ generates a vector from a *skew-symmetric* matrix:

$$\begin{bmatrix} 0 & -z & y \\ z & 0 & x \\ -y & -x & 0 \end{bmatrix}_\vee = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{C.8}$$

C.2.4 The Kronecker Product

The Kronecker product or operator, sometimes also matrix direct product, is denoted by $\otimes$ and gives the tensor product of two matrices $\mathbf{A} \otimes \mathbf{B}$:

$$\mathbf{K}_{AB} = \mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a\mathbf{B} & b\mathbf{B} \\ c\mathbf{B} & d\mathbf{B} \end{pmatrix} = \begin{pmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cg & ch & dg & dh \end{pmatrix}. \tag{C.9}$$

where $\mathbf{K}_{AB}$ is of dimensions $N_A M_B \times N_A M_B$, where N and M are the respective rows and columns. A special case is the Kronecker product with an $N \times N$ identity matrix I_N:

$$\mathbf{A} \otimes I_2 = \begin{pmatrix} a & 0 & b & 0 \\ 0 & a & 0 & b \\ c & 0 & d & 0 \\ 0 & c & 0 & d \end{pmatrix}. \tag{C.10}$$

C.2.5 Matrix Derivatives

For simpler notation, some derivations in this thesis are written in terms of a derivative with respect to matrix. We follow the assumption of [10] that whenever derivatives of

matrices occur, they are implicitly expanded via the *vec* operator.

$$\begin{aligned}\mathbf{AB} &= \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix} \\ &\equiv vec(\mathbf{AB}) = (ae+bg, ce+dg, af+bh, cf+dh)^T\end{aligned} \tag{C.11}$$

$$\frac{\partial \mathbf{AB}}{\partial \mathbf{A}} \equiv \frac{\partial vec(\mathbf{AB})}{\partial vec(\mathbf{A})} = \begin{pmatrix} e & 0 & g & 0 \\ 0 & e & 0 & g \\ f & 0 & h & 0 \\ 0 & f & 0 & h \end{pmatrix} \tag{C.12}$$

For example, the first column of (C.12) corresponds to:

$$\frac{\partial (ae+bg, ce+dg, af+bh, cf+dh)^T}{\partial a} = (e, 0, f, 0)^T. \tag{C.13}$$

The result in (C.12) can also be expressed in terms of the Kronecker product [10]:

$$\frac{\partial \mathbf{M_1 M_2}}{\partial \mathbf{M_1}} = \mathbf{M_2}^T \oplus I_N, \tag{C.14}$$

where $\mathbf{M_1}$ and $\mathbf{M_2}$ are $N \times N$ matrices and I_N is the $N \times N$ identity matrix.

C.2.6 General Conversion of a Quaternion to a Rotation Matrix

The homogeneous expression of the rotation matrix $\mathbf{R}$ from a quaternion $\mathbf{q} = [q_0, q_1, q_2, q_3]^T$ is given as:

$$\mathbf{R} = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_0q_2+q_1q_3) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_0q_1+q_2q_3) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix}. \tag{C.15}$$

If $\mathbf{q}$ is a unit quaternion, we can simplify the expression R in (C.15). Since $||\mathbf{q}|| = 1$, $||\mathbf{q}||^2 = (q_0^2+q_1^2+q_2^2+q_3^2) = 1$ and we can rewrite r_{00} as:

$$q_0^2+q_1^2-q_2^2-q_3^2 = q_0^2+q_1^2+q_2^2+q_3^2-q_2^2-q_3^2-q_2^2-q_3^2 = 1-2(q_2^2+q_3^2) \tag{C.16}$$

The same simplification for r_{11} and r_{22} gives a much simpler inhomogeneous expression (3.32) for R.

C.2.7 Rotate a 3D Vector by a Quaternion

A 3D vector in camera is represented as quaternion $\mathbf{q}_v = 0 + xi + yj + zk$ and rotated from camera i to j by a unit quaternion $\mathbf{q}_{ji}$ as given in [111]:

$$\begin{aligned}\mathbf{q}'_v =& \mathbf{q}_{ji}\mathbf{q}_v\mathbf{q}^*_{ji} \\ =& (q_0 + q_1 i + q_2 j + q_3 k)(xi + yj + zk)(q_0 - q_1 i - q_2 j - q_3 k) \\ =& (x(q_0^2 + q_1^2 - q_2^2 - q_3^2) + 2y(q_1 q_2 - q_0 q_3) + 2z(q_0 q_2 + q_1 q_3))i + \\ & (2x(q_0 q_3 + q_1 q_2) + y(q_0^2 - q_1^2 + q_2^2 - q_3^2) + 2z(q_2 q_3 - q_0 q_1))j + \\ & (2x(q_1 q_3 - q_0 q_2) + 2y(q_0 q_1 + q_2 q_3) + z(q_0^2 - q_1^2 - q_2^2 + q_3^2))k.\end{aligned} \tag{C.17}$$

All the involved operations are performed only on real numbers and do not require any costly trigonometric functions.

Bibliography

[1] Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S. M., and Szeliski, R. (2011). Building rome in a day. *Communications of the ACM*, 54(10):105–112. (page 13)

[2] Agarwal, S., Mierle, K., et al. (2012). Ceres solver. *http://ceres-solver.org*. (page 88)

[3] Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. (2010). Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(5):898–916. (page 91, 92)

[4] Audras, C., Comport, A., Meilland, M., and Rives, P. (2011). Real-time dense appearance-based slam for rgb-d sensors. In *Australasian Conference on Robotics and Automation*, volume 2, pages 2–2. (page 18)

[5] Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117. (page 11)

[6] Baker, S. and Matthews, I. (2004). Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision (IJCV)*, 56(3):221–255. (page 53)

[7] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 404–417. (page 13)

[8] Besl, P. J. and McKay, N. D. (1992). Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–606. (page 60)

[9] Blais, G. and Levine, M. D. (1995). Registering multiview range data to create 3d computer objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 17(8):820–824. (page 60)

[10] Blanco, J.-L. (2010). A tutorial on se (3) transformation parameterizations and on-manifold optimization. *University of Malaga, Technical Report*, 3. (page 38, 54, 143, 144)

[11] Bouguet, J.-Y. (2004). Camera calibration toolbox for matlab. *http://www. vision. caltech. edu/bouguetj/calib_doc/index. html*. (page 29)

[12] Boyd, S., Xiao, L., and Mutapcic, A. (2003). Subgradient methods. *Lecture notes of EE392o, Stanford University, Autumn Quarter*, 2004. (page 22)

[13] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. (page 95, 110)

[14] Bu, S., Zhao, Y., Wan, G., Li, K., Cheng, G., and Liu, Z. (2017). Semi-direct tracking and mapping with rgb-d camera for mav. *Multimedia Tools and Applications*, 76(3):4445–4469. (page 22, 23)

[15] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics (T-RO)*, 32(6):1309–1332. (page 2, 5, 11)

[16] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, (6):679–698. (page 15, 18, 19, 91, 92, 93, 95, 98, 99, 100, 101, 104, 126)

[17] Carlone, L., Tron, R., Daniilidis, K., and Dellaert, F. (2015). Initialization techniques for 3d slam: a survey on rotation estimation and its use in pose graph optimization. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 4597–4604. (page 87)

[18] Caruso, D., Engel, J., and Cremers, D. (2015). Large-scale direct slam for omnidirectional cameras. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 141–148. (page 17)

[19] Chen, Y., Davis, T. A., Hager, W. W., and Rajamanickam, S. (2008). Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):22. (page 88)

[20] Chen, Y. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155. (page 60)

[21] Civera, J., Davison, A. J., and Montiel, J. M. (2008). Inverse depth parametrization for monocular slam. *IEEE Transactions on Robotics (T-RO)*, 24(5):932–945. (page 14, 33, 78)

[22] Comport, A. I., Malis, E., and Rives, P. (2007). Accurate quadrifocal tracking for robust 3d visual odometry. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 40–45. (page 17)

[23] Concha, A. and Civera, J. (2017). Rgbdtam: A cost-effective and accurate rgb-d tracking and mapping system. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 6756–6763. (page 18, 19, 20, 24, 40, 41, 48, 123, 124, 125)

[24] Dai, A., Nießner, M., Zollöfer, M., Izadi, S., and Theobalt, C. (2017). Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration. *ACM Transactions on Graphics (TOG)*. (page 17, 18, 19, 21, 24, 123, 124, 125, 126, 127, 128)

[25] Davison, A. J. (2003). Real-time simultaneous localization and mapping with a single camera. In *Proceedings of International Conference on Computer Vision (ICCV)*, volume 3, pages 1403–1410. (page 14)

[26] Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, (6):1052–1067. (page 13, 14)

[27] Diebel, J. (2006). Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35. (page 33, 34)

[28] Dollár, P. and Zitnick, C. L. (2013). Structured forests for fast edge detection. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 1841–1848. (page 92, 95, 98, 99, 100, 101, 103, 104)

[29] Dollár, P. and Zitnick, C. L. (2015). Fast edge detection using structured forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 37(8):1558–1570. (page 92, 95, 98, 99, 100, 101, 103)

[30] Dong-Si, T.-C. and Mourikis, A. I. (2011). Motion tracking with fixed-lag smoothing: Algorithm and consistency analysis. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 5655–5662. (page 79)

[31] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE Robotics & Automation Magazine*, 13(2):99–110. (page 2, 11)

[32] Eade, E. (2013). Lie groups for 2d and 3d transformations. *http://ethaneade.com/lie.pdf*. (page 38, 79)

[33] Eade, E. and Drummond, T. (2006a). Edge landmarks in monocular slam. In *Proceedings of British Machine Vision Conference (BMVC)*, pages 7–16. (page 15, 23, 50)

[34] Eade, E. and Drummond, T. (2006b). Scalable monocular slam. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 469–476. (page 14, 15)

[35] Eade, E. and Drummond, T. (2009). Edge landmarks in monocular slam. *Image and Vision Computing*, 27(5):588–596. (page 49, 50)

[36] Endres, F., Hess, J., Sturm, J., Cremers, D., and Burgard, W. (2014). 3-d mapping with an rgb-d camera. *IEEE Transactions on Robotics (T-RO)*, 30(1):177–187. (page 11, 13, 14, 16, 24, 50, 123, 125, 126)

[37] Engel, J., Koltun, V., and Cremers, D. (2018). Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(3):611–625. (page 12, 17, 21, 23, 24, 27, 40, 41, 48, 53, 67, 69, 72, 73, 78, 79, 80)

[38] Engel, J., Schöps, T., and Cremers, D. (2014). Lsd-slam: Large-scale direct monocular slam. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 834–849. (page 12, 17, 21, 24, 27, 32, 40, 41, 48, 53, 67)

[39] Engel, J., Stückler, J., and Cremers, D. (2015). Large-scale direct slam with stereo cameras. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 1935–1942. (page 17)

[40] Engel, J., Sturm, J., and Cremers, D. (2013). Semi-dense visual odometry for a monocular camera. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 1449–1456. (page 17, 21, 40)

[41] Engel, J.-J. (2017). *Large-Scale Direct SLAM and 3D Reconstruction in Real-Time*. PhD thesis, Technische Universität München. (page 79, 80)

[42] Fabbri, R., Costa, L. D. F., Torelli, J. C., and Bruno, O. M. (2008). 2d euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys (CSUR)*, 40(1):2. (page 51)

[43] Fang, M., Yue, G., and Yu, Q. (2009). The study on an application of otsu method in canny operator. In *Proceedings of the International Symposium on Information Processing (ISIP)*, page 109. (page 96)

[44] Felzenszwalb, P. F. and Huttenlocher, D. P. (2012). Distance transforms of sampled functions. *Theory of Computing*, 8:415–428. (page 18, 51)

[45] Ferstl, D., Reinbacher, C., Riegler, G., Rüther, M., and Bischof, H. (2015). Learning depth calibration of time-of-flight cameras. In *Proceedings of British Machine Vision Conference (BMVC)*. (page 29)

[46] Firman, M. (2016). Rgbd datasets: Past, present and future. In *Conference on Computer Vision and Pattern Recognition (CVPR) – Workshops*, pages 19–31. (page 107)

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

[48] Fitzgibbon, A. W. (2003). Robust registration of 2d and 3d point sets. *Image and Vision Computing*, 21(13-14):1145–1153. (page 60)

[49] Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). Svo: Fast semi-direct monocular visual odometry. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 15–22. (page 21, 22, 23)

[50] Forster, C., Zhang, Z., Gassner, M., Werlberger, M., and Scaramuzza, D. (2016). Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics (T-RO)*, 33(2):249–265. (page 21, 23, 24)

[51] Fraundorfer, F. and Scaramuzza, D. (2012). Visual odometry: Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics & Automation Magazine*, 19(2):78–90. (page 3, 11)

[52] Gálvez-López, D. and Tardos, J. D. (2012). Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics (T-RO)*, 28(5):1188–1197. (page 16, 19)

[53] Gao, X., Wang, R., Demmel, N., and Cremers, D. (2018). Ldso: Direct sparse odometry with loop closure. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 2198–2204. (page 22)

[54] Geneva, P., Maley, J., and Huang, G. (2019). An efficient schmidt-ekf for 3d visual-inertial slam. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 12105–12115. (page 14)

[55] Glocker, B., Shotton, J., Criminisi, A., and Izadi, S. (2015). Real-time rgb-d camera relocalization via randomized ferns for keyframe encoding. *IEEE Transactions on Visualization and Computer Graphics*, 21(5):571–583. (page 19, 20, 24, 82, 83, 85, 86, 90, 131)

[56] Glover, A., Maddern, W., Warren, M., Reid, S., Milford, M., and Wyeth, G. (2012). Openfabmap: An open source toolbox for appearance-based loop closure detection. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 4730–4735. (page 21, 24, 85)

[57] Goldman, R. (2010). Rethinking quaternions. *Synthesis Lectures on Computer Graphics and Animation*, 4(1):1–157. (page 34, 35)

[58] Gong, X.-Y., Su, H., Xu, D., Zhang, Z.-T., Shen, F., and Yang, H.-B. (2018). An overview of contour detection approaches. *International Journal of Automation and Computing*, 15(6):656–672. (page 91)

[59] Grassia, F. S. (1998). Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, 3(3):29–48. (page 38)

[60] Grisetti, G., Kummerle, R., Stachniss, C., and Burgard, W. (2010). A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43. (page 87)

[61] Han, J., Shao, L., Xu, D., and Shotton, J. (2013). Enhanced computer vision with microsoft kinect sensor: A review. *IEEE Transactions on Cybernetics*, 43(5):1318–1334. (page 5)

[62] Handa, A., Whelan, T., McDonald, J., and Davison, A. J. (2014). A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 1524–1531. (page 59, 108, 109, 110, 111, 115)

[63] Harris, C. and Stennett, C. (1990). Rapid-a video rate object tracker. In *Proceedings of British Machine Vision Conference (BMVC)*, pages 1–6. (page 13)

[64] Hartley, R., Trumpf, J., Dai, Y., and Li, H. (2013). Rotation averaging. *International Journal of Computer Vision (IJCV)*, 103(3):267–305. (page 87, 88)

[65] Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry*. Cambridge University Press. (page 13, 28, 29, 73, 77)

[66] He, J., Zhang, S., Yang, M., Shan, Y., and Huang, T. (2019). Bi-directional cascade network for perceptual edge detection. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 3828–3837. (page 92, 95, 98, 99, 100, 101, 103, 104, 124, 126)

[67] Heinly, J., Schonberger, J. L., Dunn, E., and Frahm, J.-M. (2015). Reconstructing the world* in six days*(as captured by the yahoo 100 million image dataset). In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 3287–3295. (page 13)

[68] Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2012). Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research (IJRR)*, 31(5):647–663. (page 14)

[69] Horn, B. K. (1987). Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642. (page 60, 110)

[70] Horn, B. K. (2001). Some notes on unit quaternions and rotation. *Lecture Handouts*. (page 36)

[71] Hsiung, S.-C. J. (2018). Toward invariant visual-inertial state estimation using information sparsification. Master's thesis, Carnegie Mellon University Pittsburgh, PA. (page 78)

[72] Hu, M.-K. (1962). Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8(2):179–187. (page 16)

[73] Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D., and Roy, N. (2017). Visual odometry and mapping for autonomous flight using an rgb-d camera. In *The International Journal of Robotics Research (IJRR)*, pages 235–252. Springer. (page 14, 24, 117, 119, 120, 121, 122, 128)

[74] Huang, G. P., Mourikis, A. I., and Roumeliotis, S. I. (2009). A first-estimates jacobian ekf for improving slam consistency. In *Experimental Robotics*, pages 373–382. (page 78)

[75] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., et al. (2011). Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of ACM Symposium on User Interface Software and Technology*, pages 559–568. (page 17)

[76] Jose Tarrio, J. and Pedre, S. (2015). Realtime edge-based visual odometry for a monocular camera. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 702–710. (page 18, 50)

[77] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J., and Dellaert, F. (2012). isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research (IJRR)*, 31(2):216–235. (page 73)

[78] Kähler, O., Prisacariu, V. A., and Murray, D. W. (2016). Real-time large-scale dense 3d reconstruction with loop closure. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 500–516. (page 11, 17, 19, 20, 21, 24, 48, 60, 82, 86, 128)

[79] Kähler, O., Prisacariu, V. A., Ren, C. Y., Sun, X., Torr, P., and Murray, D. (2015). Very high frame rate volumetric integration of depth images on mobile devices. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1241–1250. (page 11, 17, 20, 21, 25, 48, 120, 124, 126)

[80] Kerl, C., Sturm, J., and Cremers, D. (2013a). Dense visual slam for rgb-d cameras. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 2100–2106. (page 17, 18, 20, 24, 40, 48, 53, 67, 123, 124, 125, 126, 127)

[81] Kerl, C., Sturm, J., and Cremers, D. (2013b). Robust odometry estimation for rgb-d cameras. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 3748–3754. (page 18, 21, 22, 24, 40, 48, 53, 67, 71, 115, 119, 121, 122, 128)

[82] Keselman, L., Iselin Woodfill, J., Grunnet-Jepsen, A., and Bhowmik, A. (2017). Intel realsense stereoscopic depth cameras. In *Conference on Computer Vision and Pattern Recognition (CVPR) – Workshops*, pages 1–10. (page 5)

[83] Kittler, J. (1983). On the accuracy of the sobel edge detector. *Image and Vision Computing*, 1(1):37–42. (page 91)

[84] Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 1–10. (page 4, 13, 14, 15, 16, 23, 24, 50)

[85] Klein, G. and Murray, D. (2008). Improving the agility of keyframe-based slam. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 802–815. (page 13, 14, 15, 49, 50)

[86] Klein, G. S. and Drummond, T. W. (2004). Tightly integrated sensor fusion for robust visual tracking. *Image and Vision Computing*, 22(10):769–776. (page 15)

[87] Klose, S., Heise, P., and Knoll, A. (2013). Efficient compositional approaches for real-time robust direct visual odometry from rgb-d data. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 1100–1106. (page 53)

[88] Kneip, L., Yi, Z., and Li, H. (2015). Sdicp: Semi-dense tracking based on iterative closest points. In *Proceedings of British Machine Vision Conference (BMVC)*, pages 100–1. (page 22, 52)

[89] Kuse, M. P. and Shen, S. (2016). Robust camera motion estimation using direct edge alignment and sub-gradient method. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*. (page 18, 22, 24, 41, 67, 71, 92, 129)

[90] Lam, L., Lee, S.-W., and Suen, C. Y. (1992). Thinning methodologies-a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 14(9):869–885. (page 95)

[91] Lee, S. H. and Civera, J. (2018). Loosely-coupled semi-direct monocular slam. *IEEE Robotics and Automation Letters (R-AL)*, 4(2):399–406. (page 21, 23, 24)

[92] Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P. (2015). Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research (IJRR)*, 34(3):314–334. (page 13, 14, 21, 67, 72, 73, 78, 79, 80)

[93] Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168. (page 44, 88)

[94] Li, M. and Mourikis, A. I. (2012). Improving the accuracy of ekf-based visual-inertial odometry. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 828–835. (page 14)

[95] Li, M. and Mourikis, A. I. (2013). High-precision, consistent ekf-based visual-inertial odometry. *The International Journal of Robotics Research (IJRR)*, 32(6):690–711. (page 14)

[96] Li, S., Handa, A., Zhang, Y., and Calway, A. (2016). Hdrfusion: Hdr slam using a low-cost auto-exposure rgb-d sensor. In *International Conference on 3D Vision (3DV)*, pages 314–322. (page 96, 98)

[97] Li, S.-p., Zhang, T., Gao, X., Wang, D., and Xian, Y. (2019). Semi-direct monocular visual and visual-inertial slam with loop closure detection. *Robotics and Autonomous Systems*, 112:201–210. (page 23)

[98] Liu, H., Chen, M., Zhang, G., Bao, H., and Bao, Y. (2018). Ice-ba: Incremental, consistent and efficient bundle adjustment for visual-inertial slam. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 1974–1982. (page 131)

[99] Liu, H., Li, C., Chen, G., Zhang, G., Kaess, M., and Bao, H. (2017a). Robust keyframe-based dense slam with an rgb-d camera. *arXiv preprint arXiv:1711.05166*. (page 18, 21, 98, 123, 125, 126, 131)

[100] Liu, Y., Cheng, M.-M., Hu, X., Bian, J.-W., Zhang, L., Bai, X., and Tang, J. (2019). Richer convolutional features for edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*. (page 92, 95, 98, 99, 100, 101, 103, 104, 124, 126)

[101] Liu, Y., Cheng, M.-M., Hu, X., Wang, K., and Bai, X. (2017b). Richer convolutional features for edge detection. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 3000–3009. (page 92, 95, 103, 124, 126)

[102] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110. (page 13, 15)

[103] Lowry, S., Sünderhauf, N., Newman, P., Leonard, J. J., Cox, D., Corke, P., and Milford, M. J. (2015). Visual place recognition: A survey. *IEEE Transactions on Robotics (T-RO)*, 32(1):1–19. (page 82)

[104] Madsen, K., Nielsen, H. B., and Tingleff, O. (1999). Methods for non-linear least squares problems. In *Informatics and Mathematical Modelling*. (page 42, 44)

[105] Maity, S., Saha, A., and Bhowmick, B. (2017). Edge slam: Edge points based monocular visual slam. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 2408–2417. (page 14, 16, 50)

[106] Marler, R. T. and Arora, J. S. (2010). The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 41(6):853–862. (page 61)

[107] Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441. (page 44, 88)

[108] Marr, D. and Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217. (page 91)

[109] Martin, D. R., Fowlkes, C. C., and Malik, J. (2004). Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, (5):530–549. (page 91)

[110] Maurer, M., Puerta, J. P., Fraundorfer, F., and Bischof, H. (2018). Towards an autonomous vision-based inventory drone. In *International Conference on Intelligent Robots and Systems (IROS)–Workshop on Robotics for Logistics*. (page 7)

[111] Mordechai, B.-A. (2014). A tutorial on euler angles and quaternions. *Weizmann Institute of Science, Tutorial*, 1. (page 36, 145)

[112] Mourikis, A. I. and Roumeliotis, S. I. (2007). A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 3565–3572. (page 14)

[113] Mourikis, A. I., Trawny, N., Roumeliotis, S. I., Johnson, A. E., Ansar, A., and Matthies, L. (2009). Vision-aided inertial navigation for spacecraft entry, descent, and landing. *IEEE Transactions on Robotics (T-RO)*, 25(2):264–280. (page 14)

[114] Mur-Artal, R., Montiel, J., and Tardós, J. D. (2015). Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics (T-RO)*, 31(5):1147–1163. (page 14, 15, 23, 73)

[115] Mur-Artal, R. and Tardós, J. D. (2014). Fast relocalization and loop closing in keyframe-based slam. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 846–853. (page 21, 24, 85)

[116] Mur-Artal, R. and Tardós, J. D. (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics (T-RO)*, 33(5):1255–1262. (page 11, 13, 14, 15, 16, 19, 20, 24, 50, 67, 117, 120, 121, 122, 123, 124, 125, 126, 127)

[117] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011a). Kinectfusion: Real-time dense surface mapping and tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136. (page 11, 17, 19, 20, 21, 48, 60, 120)

[118] Newcombe, R. A., Lovegrove, S. J., and Davison, A. J. (2011b). Dtam: Dense tracking and mapping in real-time. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 2320–2327. (page 25)

[119] Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. (2013). Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*, 32(6):169. (page 17, 20)

[120] Nistér, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–652. (page 3, 13, 14)

[121] Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer, New York, NY, USA, second edition. (page 41, 42, 44, 88)

[122] Ozuysal, M., Calonder, M., Lepetit, V., and Fua, P. (2009). Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(3):448–461. (page 82)

[123] Paz, L. M., Piniés, P., Tardós, J. D., and Neira, J. (2008). Large-scale 6-dof slam with stereo-in-hand. *IEEE Transactions on Robotics (T-RO)*, 24(5):946–957. (page 15)

[124] Ranganathan, A., Kaess, M., and Dellaert, F. (2007). Fast 3d pose estimation with out-of-sequence measurements. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 2486–2493. (page 79)

[125] Rosenfeld, A. and Pfaltz, J. L. (1968). Distance functions on digital pictures. *Pattern Recognition*, 1(1):33–61. (page 51)

[126] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 430–443. (page 13, 15)

[127] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 2564–2571. (page 13, 15, 21, 85)

[128] Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling*, pages 145–152. (page 60, 61)

[129] Rusu, R. B. (2009). *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science Department, Technische Universität Muenchen, Germany. (page 60)

[130] Sarabandi, S. and Thomas, F. (2018). Accurate computation of quaternions from rotation matrices. In *International Symposium on Advances in Robot Kinematics*, pages 39–46. (page 34, 35)

[131] Scaramuzza, D. and Fraundorfer, F. (2011). Visual odometry [tutorial]. *IEEE Robotics & Automation Magazine*, 18(4):80–92. (page 3, 11)

[132] Schenk, F. and Fraundorfer, F. (2017a). Combining edge images and depth maps for robust visual odometry. In *Proceedings of British Machine Vision Conference (BMVC)*. (page 8, 9, 12, 18, 24, 47, 57, 59, 65, 67, 68, 71, 89, 90, 93, 94, 107, 111, 114, 119)

[133] Schenk, F. and Fraundorfer, F. (2017b). Robust edge-based visual odometry using machine learned edges. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 1297–1304. (page 8, 9, 12, 18, 22, 24, 41, 47, 57, 64, 65, 67, 68, 71, 89, 107, 114, 119)

[134] Schenk, F. and Fraundorfer, F. (2019). Reslam: A robust edge-based slam system. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 154–160. (page 8, 9, 20, 24, 47, 58, 67, 71, 72, 81, 82, 85, 86, 89, 90, 93, 94, 107, 114, 119, 121, 122)

[135] Schonberger, J. L. and Frahm, J.-M. (2016). Structure-from-motion revisited. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113. (page 13)

[136] Schops, T., Sattler, T., and Pollefeys, M. (2019). Bad slam: Bundle adjusted direct rgb-d slam. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 134–144. (page 17, 18, 20, 24, 108, 109, 111, 123, 124, 125, 127, 128)

[137] Schubert, D., Demmel, N., Usenko, V., Stuckler, J., and Cremers, D. (2018). Direct sparse odometry with rolling shutter. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 682–697. (page 22)

[138] Segal, A., Haehnel, D., and Thrun, S. (2009). Generalized-icp. In *Robotics: Science and Systems (RSS)*, volume 2, page 435. (page 14)

[139] Shen, W., Wang, X., Wang, Y., Bai, X., and Zhang, Z. (2015). Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 3982–3991. (page 92, 95, 98, 99, 100, 101, 103)

[140] Sibley, G. (2006). Sliding window filters for slam. *Technical Report CRES-06–004, University of Southern California, Center for Robotics and Embedded Systems.* (page 79)

[141] Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor segmentation and support inference from rgbd images. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 746–760. (page 92, 97, 100)

[142] Smith, P., Reid, I., and Davison, A. (2006). Real-time monocular slam with straight lines. *Proceedings of British Machine Vision Conference (BMVC)*, pages 17–26. (page 49)

[143] Steinbrücker, F., Sturm, J., and Cremers, D. (2011). Real-time visual odometry from dense rgb-d images. In *International Conference on Computer Vision (ICCV) – Workshops*, pages 719–722. (page 18)

[144] Strasdat, H. (2012). *Local accuracy and global consistency for efficient visual SLAM*. PhD thesis, Department of Computing, Imperial College London. (page 32, 79, 87)

[145] Strasdat, H., Davison, A. J., Montiel, J. M., and Konolige, K. (2011). Double window optimisation for constant time visual slam. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 2352–2359. (page 15, 73)

[146] Strasdat, H., Montiel, J. M., and Davison, A. J. (2012). Visual slam: why filter? *Image and Vision Computing*, 30(2):65–77. (page 14)

[147] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A benchmark for the evaluation of rgb-d slam systems. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 573–580. (page 40, 59, 62, 63, 88, 97, 101, 102, 104, 108, 109, 110, 111, 115, 119)

[148] Szeliski, R. (2010). *Computer vision: algorithms and applications*. (page 29, 38)

[149] Taketomi, T., Uchiyama, H., and Ikeda, S. (2017). Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16. (page 3, 11)

[150] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (1999). Bundle adjustment–a modern synthesis. In *International Workshop on Vision Algorithms*, pages 298–372. (page 13, 73, 77)

[151] Tykkälä, T., Audras, C., and Comport, A. I. (2011). Direct iterative closest point for real-time visual odometry. In *International Conference on Computer Vision–Workshops*, pages 2050–2056. (page 18)

[152] Unknown Author, W. C. (2020). Camera obscura. *https://commons.wikimedia.org/wiki/File:001_a01_camera_obscura_abrazolas.jpg*. (page 29)

[153] Usenko, V., Engel, J., Stückler, J., and Cremers, D. (2016). Direct visual-inertial odometry with stereo cameras. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*. (page 17)

[154] Wang, X., Wei, D., Zhou, M., Li, R., Zha, H., and Beijing, C. (2016). Edge enhanced direct visual odometry. In *Proceedings of British Machine Vision Conference (BMVC)*. (page 18, 22, 92)

[155] Weilharter, R. J., Schenk, F., and Fraundorfer, F. (2018). Globally consistent dense real-time 3d reconstruction from rgbd data. In *OAGM Workshop 2018: Medical Image Analysis*. (page 21)

[156] Whelan, T., Johannsson, H., Kaess, M., Leonard, J. J., and McDonald, J. (2013). Robust real-time visual odometry for dense rgb-d mapping. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 5724–5731. (page 17, 19, 20, 21, 24, 48)

[157] Whelan, T., Kaess, M., Fallon, M., Johannsson, H., Leonard, J., and McDonald, J. (2012). Kintinuous: Spatially extended kinectfusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia. (page 17, 115, 121, 122, 123, 125, 126, 128)

[158] Whelan, T., Kaess, M., Johannsson, H., Fallon, M., Leonard, J. J., and McDonald, J. (2015). Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research (IJRR)*, 34(4-5):598–626. (page 17, 18, 24, 25)

[159] Whelan, T., Salas-Moreno, R. F., Glocker, B., Davison, A. J., and Leutenegger, S. (2016). Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research (IJRR)*, 35(14):1697–1716. (page 17, 18, 19, 20, 24, 48, 60, 82, 86, 123, 124, 125, 126, 127, 128)

[160] Williams, B., Cummins, M., Neira, J., Newman, P., Reid, I., and Tardós, J. (2009). A comparison of loop closing techniques in monocular slam. *Robotics and Autonomous Systems*, 57(12):1188–1197. (page 82)

[161] Xie, S. and Tu, Z. (2015). Holistically-nested edge detection. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 1395–1403. (page 92, 95, 96, 98, 99, 100, 101, 103, 104)

[162] Yousif, K., Bab-Hadiashar, A., and Hoseinnezhad, R. (2015). An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311. (page 11)

[163] Zhang, Z. (2012). Microsoft kinect sensor and its effect. *IEEE Multimedia*, 19(2):4–10. (page 5)

[164] Zhou, Y., Kneip, L., and Li, H. (2017). Semi-dense visual odometry for rgb-d cameras using approximate nearest neighbour fields. In *Proceedings of the International Conference for Robotics and Automation (ICRA)*, pages 6261–6268. (page 22, 24, 52, 117, 121, 122)

[165] Zhou, Y., Li, H., and Kneip, L. (2019). Canny-vo: Visual odometry with rgb-d cameras based on geometric 3-d–2-d edge alignment. *IEEE Transactions on Robotics (T-RO)*, 35(1):184–199. (page 22, 24, 52, 92, 117, 119, 121, 122)

[166] Ziou, D. and Tabbone, S. (1998). Edge detection techniques-an overview. *Pattern Recognition and Image Analysis*, 8:537–559. (page 91)