ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN
Ingegneria Elettronica, Informatica e delle Telecomunicazioni
CICLO 27

Settore Concorsuale di afferenza: 09/H1
Settore Scientifico disciplinare: ING-INF/05

SCENE RECONSTRUCTION AND UNDERSTANDING BY RGB-D SENSORS

Presentata da: NICOLA FIORAIO

Coordinatore Dottorato:
Chiar.mo Prof. Alessandro Vanelli-Coralli

Relatore:
Chiar.mo Prof. Luigi Di Stefano

ESAME FINALE ANNO 2015

To my parents, my sister Michela
and Serena

may my heart always be open to little
birds who are the secrets of living
whatever they sing is better than to know
and if men should not hear them men are old

may my mind stroll about hungry
and fearless and thirsty and supple
and even if it's sunday may i be wrong
for whenever men are right they are not young

and may myself do nothing usefully
and love yourself so more than truly
there's never been quite such a fool who could fail
pulling all the sky over him with one smile

E. E. Cummings

Acknowledgements

Questa tesi di dottorato è il risultato di tre anni di lavoro di ricerca al Computer Vision Lab dell'Università di Bologna, sotto la determinante supervisione del prof. Luigi Di Stefano, i cui insegnamenti, intuizioni e parole di incoraggiamento anche nei momenti più faticosi mi hanno permesso di raggiungere questo traguardo. Non meno importanti sono stati anche i suggerimenti e l'esperienza di tutti i membri del CVLab, passati e presenti: Samuele Salti, Federico Tombari, Alessandro Franchi, Alioscia Petrelli, Alessandro Lanza, Tommaso Cavallari, Fabio Bruè. La stima che ho nei loro confronti va oltre il semplice rapporto lavorativo.

In questi anni di grandi cambiamenti ho trovato sempre un sostegno sicuro in mio padre, mia madre e mia sorella Michela. La loro presenza, il loro interesse ed il loro affetto sono sempre stati per me uno stimolo a proseguire e non li ringrazierò mai abbastanza per questo. Allo stesso modo, Serena è stata sempre al mio fianco condividendo tutti i momenti felici e anche quelli più impegnativi. Per questi motivi, a loro dedico questa tesi.

I would like also to thank Dr. Shahram Izadi, who enthusiastically accepted my application for an internship at Microsoft Research. His intuitions, deep knowledge and competence made me grow as a PhD student. The time spent there has been invaluable and I would like to thank all the people at Microsoft Research Cambridge for their scientific advice and friendship, especially Yani Ioannou, Sean Ryan Fanello, Qinxun "Jerry" Bai, Mohammad Rastegari, Sameh Khamis, Mingsong

Dou, Jonathan Taylor, Andrew Fitzgibbon, Sebastian Nowozin, Peter Kontschieder, Antonio Criminisi.

Finally, I would like to thank prof. Luca Iocchi and prof. Juergen Gall for having reviewed this thesis, giving helpful comments and suggestions.

Abstract

This thesis investigates interactive scene reconstruction and understanding using RGB-D data only. Indeed, we believe that depth cameras will still be in the near future a cheap and low-power 3D sensing alternative suitable for mobile devices too. Therefore, our contributions build on top of state-of-the-art approaches to achieve advances in three main challenging scenarios, namely mobile mapping, large scale surface reconstruction and semantic modeling.

First, we will describe an effective approach dealing with Simultaneous Localization And Mapping (SLAM) on platforms with limited resources, such as a tablet device. Unlike previous methods, dense reconstruction is achieved by reprojection of RGB-D frames, while local consistency is maintained by deploying relative bundle adjustment principles. We will show quantitative results comparing our technique to the state-of-the-art as well as detailed reconstruction of various environments ranging from rooms to small apartments.

Then, we will address large scale surface modeling from depth maps exploiting parallel GPU computing. We will develop a real-time camera tracking method based on the popular KinectFusion system and an online surface alignment technique capable of counteracting drift errors and closing small loops. We will show very high quality meshes outperforming existing methods on publicly available datasets as well as on data recorded with our RGB-D camera even in complete darkness.

Finally, we will move to our Semantic Bundle Adjustment framework to effectively combine object detection and SLAM in a unified system.

Though the mathematical framework we will describe does not restrict to a particular sensing technology, in the experimental section we will refer, again, only to RGB-D sensing. We will discuss successful implementations of our algorithm showing the benefit of a joint object detection, camera tracking and environment mapping.

Being SLAM an interactive problem, we provide additional video results for the described approaches at this web address:

http://vision.deis.unibo.it/research/104-nfioraio-thesis

Contents

List of Figures

Chapter 1

Introduction

Perceiving and understanding the world through a sensor or a combination of sensing devices is a challenging task which includes localization of the sensing platform w.r.t. some global reference, reconstruction of the environment and semantic segmentation of the scene. Classically, these tasks have been addressed separately, as localization within a known map, reconstruction by fusing a set of calibrated measurements and semantic interpretation of static scenes as a batch process. Nevertheless, certain applications, such as autonomous robot exploration, call for a joint solution. Indeed, while moving within an unknown environment, a robot constantly needs, on one hand, an up-to-date map to plan its motion and, on the other hand, an accurate and real-time localization of itself within that map. When motion estimation is performed alongside with mapping, the problem is called *Simultaneous Localization And Mapping* (SLAM).

The SLAM problem introduces a number of issues, mostly due to the inevitable propagation of estimation errors. Indeed, sensor position can be estimated from a partial map only with some uncertainty, which then propagates into the same map again when updating the reconstruction by fusing measurements according to the current pose estimate. Therefore, many techniques have been investigated to tackle such issues, so that in the last decades many technological advances dealing

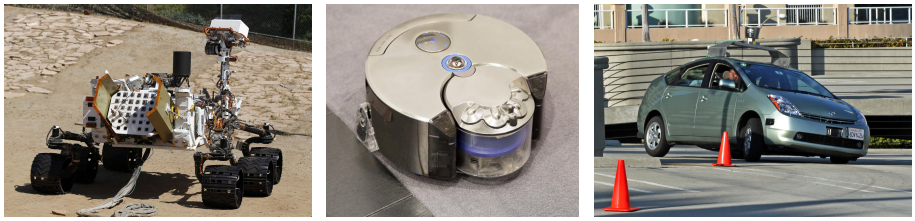


Figure 1.0.1: Nowadays, a number of mobile platforms need to solve the SLAM problem for enhanced operations. Left: the NASA rover "Curiosity", landed on Mars on 6th August 2012, equipped with sensors for obstacle avoidance and autonomous navigation. Middle: the Dyson 360 Eye™ vacuum cleaner, a first example of an autonomous robot at home using a 360° vision system for mapping and navigation. Right: the Google's driverless car uses a 3D laser range scanner and a set of cameras for autonomous driving.

Figure 1.0.2: The Microsoft Kinect (left) and the Asus Xtion PRO Live (right) are RGB-D cameras based on structured light technology.

Figure 1.0.3: Smartphones (left), tablets (middle) and smart glasses (right) are personal devices equipped with cameras and positioning sensors useful for perceiving and understanding the world.

Figure 1.0.4: The Structure sensor (left) is an active depth camera devise for enhancing tablet vision capability, while Google Project Tango (right) aims at integrating such sensors into the device.

with SLAM problems have come to light, including toy robots, vacuum cleaners, space exploration rovers and autonomous cars (see Fig. 1.0.1), and many more are expected in the near future. Moreover, if nowadays robots are designed for solving specific problems, there is a trend in the research community towards general purpose platforms, which could effectively interact with the surrounding world [60, 99]. This requires detailed geometry reconstruction and semantic understanding through, *e.g.*, object detection and localization. Therefore, many efforts are being made today to achieve, on one hand, consistent online surface reconstruction at large scale and, on the other hand, a joint approach to the detection and mapping problems. Recently, impressive results have been obtained by deploying RGB-D cameras only (see Fig. 1.0.2) for such tasks and, due to their wide availability, good performance and low cost, in this thesis we will pursue this direction. Accordingly, we will generally assume to work with a RGB-D sensor freely moving in an unknown environment, with no other information, such as, *e.g.*, wheel odometry or inertial data.

At the other end of the spectrum, the emerging field of personal mobile devices, such as smartphones, tablets and smart glasses (see Fig. 1.0.3), opens new challenging opportunities, from real-time scene reconstruction to user-centered context-specific Augmented Reality (AR). However, existing approaches usually does not suit the limited resources available on such platforms, so that new strategies have to be investigated. Again, RGB-D cameras are promising tools which, in the future, could be integrated into mobile devices for improved sensing capabilities. First examples recently announced are, *e.g.*, the Structure sensor [68] and Google Project Tango [33] (see 1.0.4), which add depth estimation to the usual color vision. In Chap. 2 we will show preliminary results obtained by attaching a Structure camera to a tablet device. Moreover, smartphones with 3D capabilities have been already launched, *e.g.* the HTC One M8, and other 3D technologies are also emerging, such as light-field cameras[1].

In this thesis we will focus on how a RGB-D camera can be used to effectively *perceive* the world. Accordingly, we will address three main open challenges, namely mobile mapping, large scale surface reconstruction and semantic understanding. As for the mobile mapping, in Chap. 2 we will describe SlamDunk [111, 108], a scalable and lightweight RGB-D SLAM system, developed both for desktop and Android platforms. We will show that, unlike existing approaches, SlamDunk gracefully handle even large workspaces without significant hindering of the overall performances. However, to comply with the strict requirements of a mobile device, SlamDunk does not reconstruct the sensed surface, but simply creates a dense point cloud by 3D projection of significant frames. Therefore, in Chap. 3 we will move to more powerful architectures, *i.e.* GPU accelerators, to address large scale surface modeling. To this end, we will describe an effective approach which outperforms the state-of-the-art in terms of real-time camera tracking and high-quality, online surface reconstruction especially in presence of loop closures.

[1] *e.g.* the Pelican array camera http://www.pelicanimaging.com [96]

This work is the result of a six-month internship at Microsoft Research Cambridge (UK) with the supervision of Dr. Shahram Izadi and it has been recently accepted for publication at the upcoming 2015 Computer Vision and Pattern Recognition international conference (CVPR) [112]. Finally, the synergistic integration of SLAM and semantic information is addressed in Chap. 4. Most of the previous findings will be here exploited to achieve joint object detection, camera tracking and environment mapping by means of a novel Semantic Bundle Adjustment framework [110]. Also, we will adapt the KinectFusion camera tracker so as to include semantic information [109]. Indeed, we believe that a tight integration of these three processes will result more and more in the forthcoming future a major research topic in both the computer vision and robotics communities.

In the next section we will briefly review the visual SLAM literature, emphasizing the milestones on the path from early researches to today. More details on relevant works will be given at the beginning of every chapter.

1.1 Camera Tracking And Mapping

The origins of visual SLAM can be dated back to the seventies [91, 12] with early works about min-max error bounds for representing spatial uncertainty. The first *probabilistic* approach deploying a multivariate representation of both position and orientation is probably due to Smith *et al*. [83, 82], where an Extended Kalman Filter (EKF) performs state estimation of camera pose as well as landmark positions, *i.e*. features tracked in different acquired images. Then, together with the increasing availability of digital cameras, filtering methods became the *de facto* standard for SLAM applications [51, 9, 21, 62, 92, 23]. The idea was to estimate a joint state probability with Gaussian uncertainty of both the current 6-DOF pose of the sensor and the 3D locations of a set of interest points detected in different image frames, so that repeated observations

would eventually shrink such uncertainties.

Due to the filtering approach, previous poses on the camera path are marginalized out in the probability formulation, while other techniques from different fields, such as, *e.g.*, photogrammetry and structure-from-motion, suggest that a joint estimation is possible and may even improve the final solution. The former field is related to the extraction of geometric information from aerial photographs, while the latter perform 3D reconstruction from a set of images. To this end, overlapping pictures have to be aligned together by finding corresponding points and a point cloud is reconstructed by point triangulation. Key is the bundle adjustment technique [95], a non-linear iterative optimization method which aims at minimizing a cost function defined as a sum of squares of reprojection errors. Recent achievements include the "Rome in a day" project [1], where an entire city is built from an unorganized collections of images gathered from internet services such as Flickr.com. However, the task is cast as an offline process and may take many hours to complete. Conversely, SLAM typically deals with incremental estimation and online refinement as soon as new measurements become available. While structure-from-motion is usually solved from scratch using batch approaches focused on accuracy, SLAM iteratively updates a probability distribution over the current pose and landmark map. Nevertheless, both methods minimize a similar cost function, *i.e.* a sum of squares of reprojection errors, and in the last decade this close relationship has been discovered and exploited [24, 40, 80]. In particular, Nister *et al.* [67] introduced the concept of "Visual Odometry", where bundle adjustment is applied over a sliding window.

A breakthrough contribution in the SLAM field has been Parallel Tracking And Mapping (PTAM), presented in [45]. The authors devised a novel system by decoupling the camera tracking and the mapping tasks in two concurrent threads. Given a 3D model of the scene, *i.e.* a collection of landmarks, incoming frames are aligned by inexpensive feature tracking. In this way, a pose for the current camera is estimated

in real-time from hundreds of observations. Then, *keyframes* are selected among all the acquired images to cover the workspace under exploration and, whenever possible, bundle adjustment optimization is performed to jointly estimate keyframe poses and landmark positions. Since this optimization need not be carried out in real-time, the framework has been effectively adopted for AR games, even on smartphones [46], while continuously updating the map. However, a large number of keyframes still set forth a complex optimization problem for the back-end so that, eventually, the computational requirements restrict the area of operation to room-size environments. Some alternatives have been proposed, based either on reducing the feature match cost terms to simpler pose-pose constraints [47] or deploying locality principles [81, 87].

Strasdat *et al*. [88] made a rigorous comparison between filtering and keyframe bundle adjustment, showing a preference for the latter. Indeed, for increasing number of points, Gaussian filtering reaches bundle adjustment accuracy, while the more keyframes are used the more robustness is achieved. However, the former shows a cubic cost in the number of landmarks, while the same cost is linear for the latter. Moreover, [88] showed that there exist particular situations where Gaussian filters perform constantly worse than bundle adjustment, therefore in this thesis we will always use a BA-style formulation for camera path estimation.

Lately, the arrival of the Microsoft Kinect camera in November 2010 paved the way for novel real-time approaches to the camera tracking and mapping problems based on the RGB-D frames delivered by the sensor. Indeed, a RGB-D frame comprises two VGA images acquired at 30 Hz: a standard RGB image and a depth map giving the metric distance of every pixel from the camera. Due to their low cost, low power and wide availability, RGB-D cameras have quickly gained interest from the scientific community, since 3D colored point clouds can be easily computed from every acquired frame while freely moving the sensor through the scene. Beside feature based methods [35, 30], high-

quality dense surface reconstruction has been demonstrated by KinectFusion [64, 38], which will be discussed in detail in Chap. 3. While dense reconstruction with monocular camera setting had already been proposed before [63, 65], they did not attain instant surface modeling. Moreover, the KinectFusion framework allows for reconstructing even in completely dark environment by using only active depth measurements, thus improving over other methods based on color information [35, 30, 43, 84].

The relative maturity of visual SLAM has encouraged the community to move towards new directions. On one hand, mobile platforms are seen as the next step for combining visual SLAM with on-board sensors, such as GPS and inertial data [46, 90]. We will discuss this topic in Chap. 2, though mainly addressing a RGB-D SLAM scenario without any form of sensor fusion. On the other hand, the semantic understanding of the scene is a valued achievement and, recently, results have shown tight correlation with the SLAM problem. From a structure-from-motion perspective, Bao *et al*. [6, 4, 5] have successfully detected object categories and jointly estimated 6-DOF camera poses with object localization on image plane, while from a SLAM perspective few works try to integrate these two tasks, *e.g*. [19, 17]. In Chap. 4 we will discuss all these approaches, highlighting also the differences between our Semantic Bundle Adjustment method [110] and SLAM++ [76], both published at the 2013 Computer Vision and Pattern Recognition international conference (CVPR).

Chapter 2

RGB-D SLAM For Mobile Devices

Over the years, the problem of exploring and simultaneously mapping an unknown environment has been addressed from a variety of approaches. Results on laser scanner data feature high precision and detailed reconstructions [61, 11], although the size and cost of the devices as well as computational issues significantly reduce the application scenarios. On the other hand, monocular and stereo visual SLAM [67, 22, 45, 47, 87] exploits only color images and has proved to be reliable in a diversity of contexts, *e.g*. augmented reality or large outdoor navigation. However, most systems still rely on sparse feature tracking techniques and require hardware acceleration to attain 3D surface reconstruction [63, 65]. We believe that both approaches, *i.e*. 3D laser scanner-based and RGB visual SLAM, are suitable for robotic platforms and desktop computers, but they have not been conceived for mobile devices, such as smart phones and tablets. Therefore, we investigated a new approach designed to be scalable, efficient and with low computational requirements. Due to its versatility and ease of use, we dub it SlamDunk [111, 108].

A key aspect of the developed algorithm is the use of RGB-D data,

Figure 2.0.1: SlamDunk allows a user with commodity hardware and a RGB-D camera to scan in real-time small objects and various indoor environments.

i.e. frame pairs including a standard RGB image and a *Depth* map which encodes, pixel-by-pixel, the distance from the camera to the nearest surface. Indeed, low-cost RGB-D sensors such as the Microsoft Kinect or the Asus Xtion PRO Live provides VGA frames at 30 Hz with minimal power consumption, thus making possible to retrieve 3D measurements at high frame rate on a mobile platform. Nevertheless, these devices, which are based on a structured light system, introduce a significant amount of noise in the estimation of pixel depths, thus harming an accurate reconstruction, though impressive results have been reported with dedicated hardware [64]. We propose to reduce complexity and improve scalability by exploiting the local mapping paradigm [81, 87, 10], whereby a global adjustment is always avoided in favor to a local optimization of the estimated camera poses. To this aim, the camera path is simplified into a skeleton of *keyframes* [47], representing the map, and a subset is selected for real-time localization of the camera sensor. Therefore, our multi-view feature tracking scheme with robust pose estimation naturally adapts to both very loopy trajectories in small workspaces, such as AR applications, and large exploratory sequences. Then, if a tracked frame brings new information about the scene, a new keyframe is spawn and a local optimization is performed. This way, we achieve local consistency, while reducing problem complexity, making SlamDunk suitable for both object reconstruction and indoor mapping (see Fig. 2.0.1).

In this chapter we will describe the SlamDunk framework, its mobile

implementation and show the results obtained on various datasets. First, in Sec. 2.1 and 2.2 we will review the main proposals and highlight similarities and differences of our approach. Then, in Sec. 2.3, we will discuss our real-time solution for desktop platforms [111]. In Sec. 2.4, we will present how the same algorithm has been implemented on an Android platform for interactive and effective 3D reconstruction on a mobile platform [108]. Finally, we close the chapter with experimental results for both the architectures in Sec. 2.5. Though in these experiments we will always make use of either the Kinect or the Xtion sensors, we point out that the recent availability of depth cameras designed for tablet devices, *e.g*. the Structure sensor [68] and the Google Project Tango [33], may pave the way, in the near future, for the integration of similar capabilities on mobile devices. As we believe that the SlamDunk framework could be a first attempt in that direction, we are going to release under an open-source license both our desktop and Android implementations.

2.1 Real-time RGB-D SLAM

Beside 3D reconstruction from laser scans integrated with the help of GPS and/or IMU data, the field of visual SLAM, *i.e*. SLAM based only on visual input, has addressed for many years the monocular camera setting, where a single RGB sensor is freely moved through the environment. Most techniques, based on filtering approaches [23, 22, 27, 18], are able to produce and refine in real-time a sparse feature map, with no notion about the dense 3D geometry of the scene. Indeed, real-time dense reconstruction from a monocular camera has been achieved only leveraging on GPU acceleration [63, 65].

The introduction of the RGB-D cameras, most notably the Microsoft Kinect, has immediately drawn the attention of the research community for its potential in 3D real-time scanning. Dense tracking and mapping has been deployed on GPU [64, 38, 102, 101, 98, 13], while few tech-

niques have been proposed for standard commodity hardware. RGB-D Mapping [35] and RGB-D SLAM [30] fill this gap with two similar visual mapping systems. Visual features [57, 103, 7, 73] are extracted from the incoming frame and matched to find correspondences with the previous one (or the previous keyframe). Then, 2D point features are projected in the 3D space by exploiting the known relationship between the color and depth image, and camera pose is robustly estimated by a RANSAC-based absolute orientation algorithm [3]. Then, [35] further refines the estimated pose by an additional ICP-like alignment, yielding improvements in low-light conditions or texture-less scenes. So, both systems rely on frame-to-frame matching, which may fail if the reference frame is not informative enough, while SlamDunk adopts a multi-view approach to increase tracking accuracy. Furthermore, we foster the creation of links by selecting neighboring keyframes on a metric distance basis, rather than following the actual camera path (see Sec. 2.3.1). We will show how this technique is also useful for metric loop closure handling, while [35] and [30] explicitly look for similar keyframes by frame-to-frame feature matching. Pose graph relaxation is carried out to reduce the overall reconstruction error. However, while [35] and [30] reduce the problem to a sum of pose-pose error terms, we always consider all the 3D point-point constraints associated to the relevant feature matches, with no marginalization or approximation to a single lighter connection per frame pair. Finally, they achieve real-time operation only when using binary features [73] or GPU acceleration [103].

Other relevant approaches for RGB-D SLAM include the dense visual odometry estimation proposed by Kerl *et al*. [44] and Steinbruecker *et al*. [86]. In these works camera pose estimation is addressed as a pairwise frame tracking problem, by densely minimizing a cost function defined over the whole image. The method has proven to reduce drift error compared to state-of-the-art camera tracking approaches, but it lacks a full-fledge SLAM framework. Keyframes are created by look-

ing at the entropy of the distribution of pose parameters, thus possibly wasting memory on loopy browsing. Moreover, metric loop closure detection is handled as a separate step of the pipeline by considering a subset of candidate keyframes, while SlamDunk, as already mentioned, foster the creation of links between distant frames as part of the camera tracking step.

As described in Sec. 2.3.3, we developed a local optimization of camera poses inspired by Sibley *et al*. [81] and Strasdat *et al*. [87]. The former proposes a relative bundle adjustment approach which is then used by the latter within a double window framework. However, camera tracking relies on matching keyframes along the camera path and metric loop closure is detected by a different matching stage.

As for other real-time RGB-D SLAM systems running on a CPU system, Scherer and Zell [77] embeds PTAM [45] in a relative bundle adjustment framework [81]. They developed a pairwise frame registration approach between the incoming camera frame and the best keyframe selected by visual feature reprojection. Although the pose is robustly estimated by a sparse optical flow and RANSAC, we argue that using multiple keyframes, as in SlamDunk, leads to higher accuracy. Moreover, while both [77] and [45] rely on monocular camera frames, we leverage on RGB-D measurements. Dryanovski *et al*. [26] align each frame to a sparse map of 3D landmarks by means of an ICP-like approach. The proposed system runs in real-time, though with QVGA image resolution, while SlamDunk deploys full VGA images. Furthermore, landmarks' positions are integrated with new measurements through Kalman filtering, which is usually less accurate than least-squares optimization [88].

2.2 Mobile RGB-D SLAM

The SLAM and reconstruction problems have been always perceived as computationally demanding, hence not suitable for mobile devices.

However, recent technological advances have fostered the development of *lightweight* approaches specifically designed to exploit all the available resources. Klein and Murray [46] proposed an implementation of PTAM [45] on an Apple iPhone 3G. Purposely, they simplified the original algorithm by extracting less visual features and requiring rich textures in the scene to robustly estimate camera movements. Nevertheless, the bundle adjustment step may require seconds to complete. Conversely, SlamDunk reduces the problem complexity by reprojecting 2D matching features according to the measured depth and estimating camera poses only. Wendel *et al*. [97] propose a similar system, but the processing is distributed between a tablet and a dedicated server pc. This is very different from our settings, where all the computation is carried out on the device. Similarly, [50] relies on a remote server for pose estimation and instant feedback. Large 3D reconstruction on a mobile phone has been demonstrated by Pan *et al*. [69]. However, the system is conceived for interactive AR scenarios and the reconstruction pipeline does not run in real-time. Finally, Tanskanen *et al*. [90] combines RGB images with inertial measurements, thus achieving real-time tracking. However, the system is designed for capturing small objects and, also, the 3D model reconstruction runs at a much lower frame rate, *i.e*. 0.3 - 0.5 Hz. Instead, SlamDunk leverages on RGB-D data and provides 3D reconstruction of the explored scene at interactive frame rate.

2.3 The SlamDunk Algorithm

Our proposed pipeline, sketched in Fig. 2.3.1, includes three different modules: Local Mapping (Sec. 2.3.1), Camera Tracking (Sec. 2.3.2) and Local Optimization (Sec. 2.3.3). RGB-D frames, retrieved from the sensing device, are aligned to the current map model, which includes a selected subset of camera frames, *i.e*. *keyframes*. Moreover, following [81, 87], we consider only the local workspace for tracking and dynamically maintain an *active window* of neighboring keyframes, which

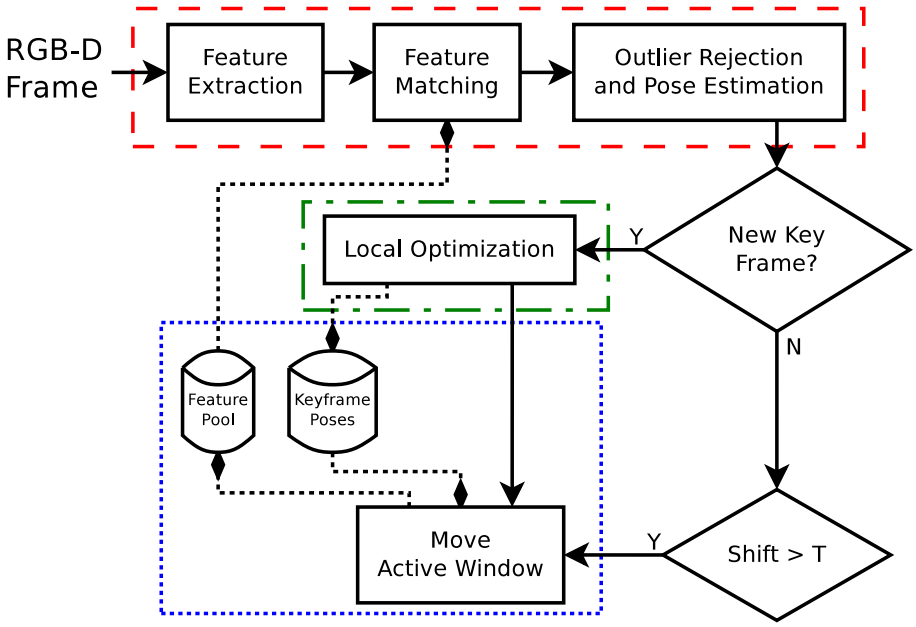


Figure 2.3.1: The SlamDunk pipeline encompasses three main modules: Local Mapping (blue dotted line, see Sec. 2.3.1), Camera Tracking (red dashed line, see Sec. 2.3.2) and Local Optimization (green dash-dot line, see Sec. 2.3.3).

enables a robust multi-view feature tracking scheme (Sec. 2.3.2). If the processed frame is classified as a *keyframe*, it is saved and added to the map, immediately refined by the Local Optimization module (Sec. 2.3.3). Again, camera path is optimized only locally, keeping problem complexity low, while reducing drift and tracking errors. Finally, the Local Mapping module (Sec. 2.3.1) moves the active window and update the local map model.

At each time stamp i there exist a 3-channel color image C_i and a depth map D_i such that, for each pixel (u,v), the sensed surface is represented in camera reference frame as

$$\mathbf{p} = \mathrm{K}^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} D_i(u,v) \tag{2.3.1}$$

where K is a projection matrix, *e.g.*

$$\mathrm{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.3.2}$$

being (f_x, f_y) and (c_x, c_y), respectively, the focal lengths and the optical centers of the camera sensor. We also assume that the two images have been registered together, so that the depth map matches the color image pixel-by-pixel.

We reconstruct the explored environment by online estimation of the camera path. Thus, we aim at recover the 3D camera pose at each time stamp i as a rigid transformation mapping 3D points from the camera reference frame to the world reference frame. Such a transformation

can be defined as a 4×4 matrix

$$\mathrm{T}_i = \begin{pmatrix} \mathrm{R}_i & \mathbf{t}_i \\ \overline{\mathbf{0}} & 1 \end{pmatrix}, \tag{2.3.3}$$

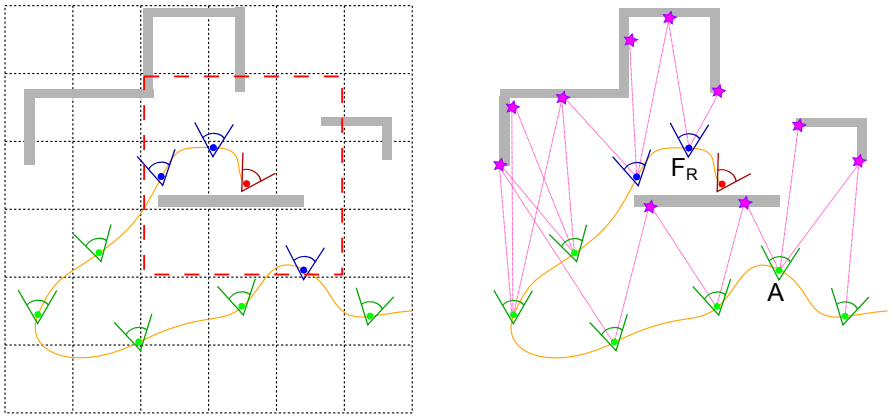
where R_i is a 3×3 rotation matrix and t_i is a 3×1 translation vector. Therefore, we can apply T_i to a given $\mathbf{x} \in \mathbb{R}^3$ as

$$\pi_3 \left(\mathrm{T}_i \left[\mathbf{x} \right] \right) \triangleq \pi_3 \left(\mathrm{T}_i \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} \right) = \mathrm{R}_i \mathbf{x} + \mathbf{t}_i, \tag{2.3.4}$$

where $[\cdot] : \mathbb{R}^3 \to \mathbb{R}^4$ is the *homogeneous* operator and $\pi_3 : \mathbb{R}^4 \to \mathbb{R}^3$ is such that $\pi_3 \left(x, y, z, w \right) := \left(x/w, y/w, z/w \right)$. In the following, to improve readability, we will generally omit both the homogeneous operator and its inverse. Finally, we observe that $\mathbf{y} = \mathrm{R}\mathbf{x} + \mathbf{t}$ implies $\mathbf{x} = \mathrm{R}^\top \mathbf{y} - \mathrm{R}^\top \mathbf{t}$, so that we define the inverse of a camera pose as

$$\mathrm{T}_i^{-1} = \begin{pmatrix} \mathrm{R}_i^\top & -\mathrm{R}_i^\top \mathbf{t}_i \\ \overline{\mathbf{0}} & 1 \end{pmatrix}. \tag{2.3.5}$$

A real-time visualization of the map is generated by reprojection of the RGB-D keyframes and transformation of the point cloud then obtained (Eq. (2.3.1) and (2.3.4)). On one hand, point density of the final model cannot be easily controlled by this method, no surface lattice is produced and, moreover, the final reconstruction does not filter noise artifacts from the original keyframe collection. A common solution is to integrate the RGB-D frames by pixel ray-tracing through an octree-based probabilistic 3D representation of the environment [37, 30], which inherently enables noise reduction, fixed resolution and efficient memory occupancy. However, our frequent pose updates due to the online optimization of the camera path would often invalidate the reconstruction and require a new integration of the keyframes. On the other hand, a map created as the reprojection of every keyframe is modeled by sim-



(a) SlamDunk indexes keyframes' poses within a quadtree structure. A local map (blue cameras) is built by querying a squared window (red square) centered at a given pose (red camera).

(b) Strasdat *et al*. [87] propose to consider all the keyframes connected to the reference keyframe F_R (blue cameras). However, this approach cannot directly handle small loop closures, *e.g*., by including the keyframe A into the local map.

Figure 2.3.2: Camera tracking is often addressed by building a local map. Here we compare our quadtree-based method (a) with the window-approach proposed by Strasdat *et al*. [87] (b).

ply update of the camera poses in the visualization pipeline. Therefore, this approach is more suited to show a camera path which is constantly refined by a SLAM engine, as in SlamDunk. Also, the reconstruction quality is not reduced, as vouched by the experimental results (see Sec. 2.5).

In the next sections we will discuss the various steps of the pipeline, namely Local Mapping, Camera Tracking, Local Optimization and metric loop closure detection. Then, in Sec. 2.4 we will present a mobile implementation on a Android platform, highlighting the limitations of the architecture and the adopted solutions. Finally, we will report quantitative and qualitative experiments in Sec. 2.5.

2.3.1 Local Mapping

SlamDunk tracks camera movements by means of a subset of the keyframe map. The creation of this local map is key for real-time operation and correct estimation of camera pose, so we developed a method which, unlike previous proposals [81, 87], does not take into account the specific camera path nor the particular problem configuration as defined in Sec. 2.3.3. Instead, as shown in Fig. 2.3.2a, we index two coordinates of the estimated camera translation into a quadtree structure approximately parallel to the ground floor. Then, we include into our local map all the keyframes within a squared window, *i.e*. the *active* window.

A quadtree data structure enables both faster insertion and retrieval than other indexes, such as octrees or KD-Trees, and allows for efficient handling of loopy trajectory. Different approaches based on the specific path followed by the camera, see, *e.g*., Fig. 2.3.2b, usually need to explicitly detect small loop closures in order to find connection with previous keyframes. The query is performed whenever a new keyframe is spawn, or if the camera pose estimate is too far from the center of the current active window. Indeed, when the sensor explores an area which has been already mapped, usually we only need to localize the camera and the active window would follow its movements.

As described in Sec. 2.3.2, camera tracking is based on a multi-view feature matching approach. Therefore, every time a new keyframe is inserted into the quadtree, we save also visual features and corresponding 3D points; then, when the active window is updated, these features are gathered from all the selected keyframes and indexed into a single KD-Tree structure, *i.e*. the *Feature Pool* in Fig. 2.3.1. Though, in principle, this task may be carried out after every frame, the creation of the tree is usually costly and, if the active window is large enough, it may not give any improvements to the tracking process. Accordingly, we update the *Feature Pool* only upon keyframe spawning or when the relative translation between the currently estimated pose and the active window center is large, *i.e*. above a threshold.

2.3.2 Camera Tracking

We estimate camera pose for each frame grabbed from the sensor using a multi-view feature tracking scheme. Visual features, such as SURF [7] or SIFT [57], are extracted from the RGB image and matched to the *Feature Pool* which indexes features from the keyframes in the active window (see Sec. 2.3.1). To improve matching reliability, we have adopted the ratio criterion presented in [57], *i.e.* we accept a match with distance d_0 *iff* the distance d_1 to the next-closest feature is such that d_0/d_1 is below a given threshold θ. However, since our KD-Tree index is built from different images, the first two matches could refer to the same feature detected from two different point of views, so that the d_0/d_1 ratio almost equals 1 even in presence of a perfect match. Therefore, we retrieve the parent keyframe of the first match and look for the next-closest feature extracted from the *same* keyframe.

The ratio threshold θ can be tuned for high confidence matching or to get a large number of correspondences. In both cases, the presence of outliers must be explicitly addressed to avoid incorrect estimation of the camera pose. Therefore, first we project each 2D point feature according to its measured depth and camera parameters (see Eq. 2.3.1), then we filter the list of 3D point matches by means of a RANSAC-based Absolute Orientation procedure [3], as outlined in Alg. 2.1. Every iteration three point pairs are randomly drawn and a unique transformation is computed by SVD least-squares fitting. The transformation is applied to each match in order to count the number of pairs belonging to the consensus set. The consensus set related to the most voted pose if finally used to refine the estimate by SVD least-squares fitting.

The multi-view feature matching scheme followed by the 3D RANSAC-based pose estimation procedure has proven to be fast, lightweight and accurate enough for tracking camera movements in a small mapped workspace. As long as new information about the scene is carried by the tracked RGB-D frame, a new keyframe should be spawn and added to our model of the environment. On one hand, a fine keyframe sampling

Algorithm 2.1 Incorrect correspondences are detected and discarded by a RANSAC-based outlier rejection algorithm. Then, correct matches provide the camera pose estimate.

Require: $I_{\max}$: max number of iteration
Require: τ: max distance between two correct matches
Require: M: set of 3D point matches
Require: p_M: probability of drawing an inlier from M
Ensure: $\widetilde{M}$ contains correct 3D point matches

$N \leftarrow \text{SizeOf}(M)$, $i \leftarrow 0$, $i_{\max} \leftarrow I_{\max}$
$S_{\text{best}} \leftarrow 0$, $\mathrm{T}_{\text{best}} \leftarrow \mathrm{I}_{4\times 4}$
while $i < i_{\max}$ **do** ▷ *RANSAC outlier rejection*
  $S_{\text{current}} \leftarrow 0$, $\mathrm{T}_{\text{current}} \leftarrow \mathrm{I}_{4\times 4}$
  Draw three matches $(\mathbf{p}_0, \mathbf{q}_0), (\mathbf{p}_1, \mathbf{q}_1), (\mathbf{p}_2, \mathbf{q}_2) \in M$
  $\hat{\mathbf{p}} \leftarrow \frac{\mathbf{p}_0+\mathbf{p}_1+\mathbf{p}_2}{3}$
  $\hat{\mathbf{q}} \leftarrow \frac{\mathbf{q}_0+\mathbf{q}_1+\mathbf{q}_2}{3}$
  $\mathrm{H} \leftarrow (\mathbf{q}_0-\hat{\mathbf{q}}) \cdot (\mathbf{p}_0-\hat{\mathbf{p}})^\top + (\mathbf{q}_1-\hat{\mathbf{q}}) \cdot (\mathbf{p}_1-\hat{\mathbf{p}})^\top + (\mathbf{q}_2-\hat{\mathbf{q}}) \cdot (\mathbf{p}_2-\hat{\mathbf{p}})^\top$
  Find matrices $\mathrm{U}, \mathrm{D}, \mathrm{V}$ such that $\mathrm{H} = \mathrm{U} \cdot \mathrm{D} \cdot \mathrm{V}^\top$
  $\mathrm{R} \leftarrow \mathrm{V} \cdot \mathrm{U}^\top$
  if $|\mathrm{R}| < 0$ **then**
    $\mathrm{V} \leftarrow [v_0, v_1, -v_2]$, where v_j is the j^{th} column of matrix V
    $\mathrm{R} \leftarrow \mathrm{V} \cdot \mathrm{U}^\top$
  end if
  $\mathrm{T}_{\text{current}} \leftarrow \begin{bmatrix} \mathrm{R} & \hat{\mathbf{p}} - \mathrm{R}\hat{\mathbf{q}} \\ \overline{\mathbf{0}} & 1 \end{bmatrix}$
  for all $(\mathbf{p}, \mathbf{q}) \in M$ **do**
    if $\|\mathrm{T}_{\text{current}}[\mathbf{q}] - \mathbf{p}\| < \tau$ **then**
      $S_{\text{current}} \leftarrow S_{\text{current}} + 1$
    end if
  end for
  if $S_{\text{current}} > S_{\text{best}}$ **then**
    $S_{\text{best}} \leftarrow S_{\text{current}}$, $\mathrm{T}_{\text{best}} \leftarrow \mathrm{T}_{\text{current}}$
    if $S_{\text{best}} = N$ **then**
      $i_{\max} \leftarrow 0$
    else
      $i_{\max} \leftarrow \min\left\{ I_{\max}, \frac{\log(1-p_M)}{\log\left(1-(S_{\text{best}}/N)^3\right)} \right\}$
    end if
  end if
  $i \leftarrow i + 1$
end while
for all $(\mathbf{p}, \mathbf{q}) \in M$ **do**
  if $\|\mathrm{T}_{\text{best}}[\mathbf{q}] - \mathbf{p}\| < \tau$ **then**
    $\widetilde{M} \leftarrow \widetilde{M} \cup \{(\mathbf{p}, \mathbf{q})\}$
  end if
end for

strategy could waste memory with no real reconstruction or localization improvements, while keeping too few keyframes may lead to a tracking failure [45, 47, 30]. Being our approach based on feature matching, we decided to rely on the amount of overlapping between the RGB-D frame and the local map. More precisely, we consider $R \times C$ cells of equal size on the RGB image, *e.g.* $R = C = 4$, and count how many such cells contain more than F matched features classified as inliers: if this number is below a threshold, we create a new keyframe from the current RGB-D frame.

2.3.3 Local Optimization

SlamDunk performs camera tracking as a localization task within a model of the local neighborhood, *i.e.* the active window. This model is fixed and it does not integrate new information from tracked frames in order to improve the overall reconstruction. Though, when a new keyframe is detected and added to the map, matched features bring in new constraints that can be used to update the camera path and reduce drift error [95, 45, 47, 87, 31, 49, 29]. Following Kümmerle *et al.* [49], we define a cost function as the sum of weighted squared errors

$$F\left(\mathrm{T}_0, \ldots, \mathrm{T}_{n-1}\right) = \sum_{i=0}^{|\mathscr{C}|-1} w_i \left\|\mathbf{e}_i\right\|^2 \tag{2.3.6}$$

where $\{\mathrm{T}_0, \ldots, \mathrm{T}_{n-1}\}$ are the unknown camera poses and $\mathscr{C}$ is the list of constraints, *i.e.* the feature matches. The cost term $w_i \left\|\mathbf{e}_i\right\|^2$ represents the error introduced by the i^{th} feature match, with $\mathbf{e}_i \in \mathscr{C}$ and $w_i \in [0,1]$ weighting the confidence of the match, *e.g.* $(1 - d_0/d_1)$ (see Sec. 2.3.2). Knowing the mapping function from the RGB to the depth frame, we can turn each feature match (f_a, f_b) in a 3D point pairs $(\mathbf{p}_a, \mathbf{p}_b)$ expressed in their own camera reference frame (*cfr.* Eq. 2.3.1). Then, we compute the reconstruction error by applying the estimates of the cor-

responding camera poses

$$\mathbf{e}_i = \mathbf{e}\left(\mathrm{T}_{i_a}, \mathrm{T}_{i_b}\right) = \mathbf{p}_{i_a} - \mathrm{T}_{i_a}^{-1}\mathrm{T}_{i_b}\mathbf{p}_{i_b}. \tag{2.3.7}$$

Finally, using Eq. (2.3.6) and (2.3.7) we state our optimization problem as

$$\begin{aligned}
\underset{\{\mathrm{T}_0,\ldots,\mathrm{T}_{n-1}\}}{\arg\min}\ F\left(\mathrm{T}_0,\ldots,\mathrm{T}_{n-1}\right) &= \underset{\{\mathrm{T}_0,\ldots,\mathrm{T}_{n-1}\}}{\arg\min} \sum_{i=0}^{|\mathscr{C}|-1} w_i \left\|\mathbf{e}_i\right\|^2 \\
&= \underset{\{\mathrm{T}_0,\ldots,\mathrm{T}_{n-1}\}}{\arg\min} \sum_{i=0}^{|\mathscr{C}|-1} w_i \left\|\mathbf{e}\left(\mathrm{T}_{i_a}, \mathrm{T}_{i_b}\right)\right\|^2 \\
&= \underset{\{\mathrm{T}_0,\ldots,\mathrm{T}_{n-1}\}}{\arg\min} \sum_{i=0}^{|\mathscr{C}|-1} w_i \left\|\mathbf{p}_{i_a} - \mathrm{T}_{i_a}^{-1}\mathrm{T}_{i_b}\mathbf{p}_{i_b}\right\|^2.
\end{aligned} \tag{2.3.8}$$

The minimization problem in Eq. (2.3.8) is usually solved by finding the point where the derivative of the cost function is zero, *i.e.*

$$\left\{\mathrm{T}_0,\ldots,\mathrm{T}_{n-1}\right\} \quad \text{such that} \quad \frac{\partial F}{\partial\left(\mathrm{T}_0,\ldots,\mathrm{T}_{n-1}\right)} = 0. \tag{2.3.9}$$

Therefore, given a good initial guess $\left\{\hat{\mathrm{T}}_0,\ldots,\hat{\mathrm{T}}_{n-1}\right\}$ such as the poses estimated by the camera tracking module, we linearize each error term around such point by its first order Taylor expansion

$$\mathbf{e}\left(\hat{\mathrm{T}}_{i_a} + \Delta\mathrm{T}_{i_a}, \hat{\mathrm{T}}_{i_b} + \Delta\mathrm{T}_{i_b}\right) \simeq \hat{\mathbf{e}}_i + \mathbf{J}_{i_a}\Delta\mathrm{T}_{i_a} + \mathbf{J}_{i_b}\Delta\mathrm{T}_{i_b}, \tag{2.3.10}$$

where $\hat{\mathbf{e}}_i = \mathbf{e}\left(\hat{\mathrm{T}}_{i_a}, \hat{\mathrm{T}}_{i_b}\right)$, $\mathbf{J}_{i_a}, \mathbf{J}_{i_b}$ are the Jacobian of $\mathbf{e}$ computed at, respectively, $\hat{\mathrm{T}}_{i_a}$ and $\hat{\mathrm{T}}_{i_b}$. Substituting in Eq. (2.3.6) the linear approximation in Eq. (2.3.10), we obtain

$$\begin{aligned}
F\left(\hat{\mathrm{T}}_0 + \Delta\mathrm{T}_0,\ldots,\hat{\mathrm{T}}_{n-1} + \Delta\mathrm{T}_{n-1}\right) &= \sum_{i=0}^{|\mathscr{C}|-1} w_i \left\|\mathbf{e}\left(\hat{\mathrm{T}}_{i_a} + \Delta\mathrm{T}_{i_a}, \hat{\mathrm{T}}_{i_b} + \Delta\mathrm{T}_{i_b}\right)\right\|^2 \\
&\simeq \sum_{i=0}^{|\mathscr{C}|-1} w_i \left\|\hat{\mathbf{e}}_i\right\|^2 + 2w_i\hat{\mathbf{e}}_i^\top \mathbf{J}_{i_a}\Delta\mathrm{T}_{i_a}
\end{aligned}$$

$$
\begin{aligned}
&+ 2w_i\hat{\mathbf{e}}_i^\top \mathbf{J}_{i_b}\Delta\mathrm{T}_{i_b} \\
&+ w_i\Delta\mathrm{T}_{i_a}^\top \mathbf{J}_{i_a}^\top \mathbf{J}_{i_a}\Delta\mathrm{T}_{i_a} \\
&+ w_i\Delta\mathrm{T}_{i_b}^\top \mathbf{J}_{i_b}^\top \mathbf{J}_{i_b}\Delta\mathrm{T}_{i_b} \\
&+ w_i\Delta\mathrm{T}_{i_a}^\top \mathbf{J}_{i_a}^\top \mathbf{J}_{i_b}\Delta\mathrm{T}_{i_b} \\
&+ w_i\Delta\mathrm{T}_{i_b}^\top \mathbf{J}_{i_b}^\top \mathbf{J}_{i_a}\Delta\mathrm{T}_{i_a} \\
&= c + 2\mathbf{b}^\top\Delta\mathrm{T} + \Delta\mathrm{T}^\top\mathrm{H}\Delta\mathrm{T} \qquad (2.3.11)
\end{aligned}
$$

where $c = \sum_{i=0}^{|\mathscr{C}|-1} w_i \|\hat{\mathbf{e}}_i\|^2$, $\Delta\mathrm{T} = \left(\Delta\mathrm{T}_0^\top, \ldots, \Delta\mathrm{T}_{n-1}^\top\right)^\top$ concatenates the increments for each camera pose, $\mathbf{b}$ is the cumulative vector

$$
\mathbf{b} = \begin{pmatrix} \left(\sum_{i=0}^{|\mathscr{C}|-1} w_i\hat{\mathbf{e}}_i^\top \mathbf{J}_{i_a}\delta_{i_a,0} + w_i\hat{\mathbf{e}}_i^\top \mathbf{J}_{i_b}\delta_{i_b,0}\right)^\top \\ \vdots \\ \left(\sum_{i=0}^{|\mathscr{C}|-1} w_i\hat{\mathbf{e}}_i^\top \mathbf{J}_{i_a}\delta_{i_a,n-1} + w_i\hat{\mathbf{e}}_i^\top \mathbf{J}_{i_b}\delta_{i_b,n-1}\right)^\top \end{pmatrix} \qquad (2.3.12)
$$

with $\delta_{j,k} = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{ow} \end{cases}$ and H is a matrix of $n \times n$ blocks such that the block h_{jk} is defined as

$$
\begin{aligned}
h_{jk} = \sum_{i=0}^{|\mathscr{C}|-1} &\left(w_i\mathbf{J}_{i_a}^\top \mathbf{J}_{i_a}\delta_{i_a,j} + w_i\mathbf{J}_{i_b}^\top \mathbf{J}_{i_b}\delta_{i_b,j}\right)\delta_{j,k} \\
&+ w_i\mathbf{J}_{i_a}^\top \mathbf{J}_{i_b}\delta_{i_a,j}\delta_{i_b,k} + w_i\mathbf{J}_{i_b}^\top \mathbf{J}_{i_a}\delta_{i_b,j}\delta_{i_a,k}. \qquad (2.3.13)
\end{aligned}
$$

Applying the result in Eq. (2.3.11) to Eq. (2.3.9) we get the linear system

$$
\mathrm{H}\Delta\mathrm{T} = -\mathbf{b}. \qquad (2.3.14)
$$

with solution $\Delta\mathrm{T}^\star$. The popular Gauss-Newton method iteratively linearizes the cost function (Eq. (2.3.11)), solve the linear system (Eq. (2.3.14)) and update the solution estimate to

$$
\mathrm{T}^\star = \hat{\mathrm{T}} + \Delta\mathrm{T}^\star, \qquad (2.3.15)
$$

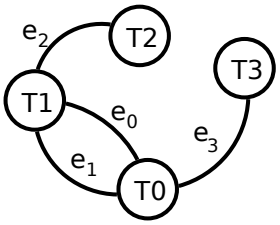
which is used as the new linearization point for the next iteration. To control the convergence of the algorithm, we deploy the Levenberg-Marquardt (LM) variation [53, 59, 56] by modifying Eq. (2.3.14) in

$$(\mathrm{H} + \lambda \mathrm{I})\,\Delta\mathrm{T} = -\mathbf{b}, \tag{2.3.16}$$

where λ is a dumping factor which controls the increment $\Delta\mathrm{T}^{\star}$: if the new solution is lower than the previous one, λ is decreased for the next step, otherwise the update is reverted and λ is increased.

We find the iterative LM solution by means of the *G2O hyper-graph solver* [49]. To deal with the non-Euclidean $\mathbb{SO}_3$ state variables, [49] represents the rotational part of the incremental solutions $\Delta\mathrm{T}^{\star}$ as a 3D vector, *e.g.* the complex part of a normalized quaternion. Indeed, using an over-parametrized representation in Eq. (2.3.14) and (2.3.15) would break the required normalization constraints. However, a minimal representation does not correctly encode the connectivity of the manifold and may lead to singular values. While $\Delta\mathrm{T}^{\star}$ is usually small and thus far from singularities, this assumption may not be true for the state variable $\mathrm{T}^{\star}$, whose rotation is therefore expressed in an over-parametrized space, *e.g.* as a 3×3 orthogonal matrix. Then, before the update step in Eq. (2.3.15) $\Delta\mathrm{T}^{\star}$ is mapped to the same over-parametrized space of $\mathrm{T}^{\star}$ and the sum operator is implemented as the usual motion composition operator.

Fig. 2.3.3 shows how the problem stated in Eq. (2.3.8) can be visualized as an undirected graph. We create a node for every unknown and map every cost term to an edge connected to the corresponding camera poses. Solving for the whole camera path has a linear to cubic complexity, depending on the topology of the graph, hence it is not suitable for online mapping. Instead, we argue that a better user experience can be achieved by considering only the local area currently under exploration and ignoring the relationships beyond a certain distance. Inspired by [81, 87], we perform a breadth-first search from the last added keyframe (the "root" node in Fig. 2.3.4) and include into the cost function all the



$$F(\mathrm{T}_0,\mathrm{T}_1,\mathrm{T}_2,\mathrm{T}_3) = \|\mathbf{e}_0(\mathrm{T}_0,\mathrm{T}_1)\|^2 + \|\mathbf{e}_1(\mathrm{T}_0,\mathrm{T}_1)\|^2 + \|\mathbf{e}_2(\mathrm{T}_1,\mathrm{T}_2)\|^2 + \|\mathbf{e}_3(\mathrm{T}_0,\mathrm{T}_3)\|^2$$

Figure 2.3.3: We can define one-to-one mapping between a cost function in the form of Eq. (2.3.6) and an undirected graph. For instance, the graph shown in this figure represents the cost function $\|\mathbf{e}_0(\mathrm{T}_0,\mathrm{T}_1)\|^2 + \|\mathbf{e}_1(\mathrm{T}_0,\mathrm{T}_1)\|^2 + \|\mathbf{e}_2(\mathrm{T}_1,\mathrm{T}_2)\|^2 + \|\mathbf{e}_3(\mathrm{T}_0,\mathrm{T}_3)\|^2$.

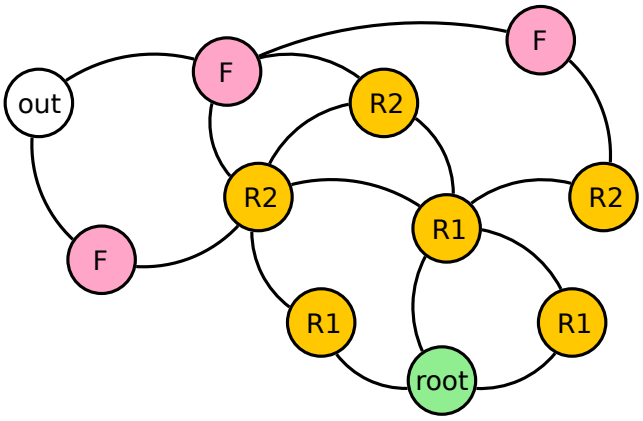


Figure 2.3.4: We devise a local optimization approach: starting from the root node (“root” - green), we include into the optimization problem its neighbors (“R1”, “R2” - orange) up to a certain ring. Then, a final ring of fixed vertexes (“F” - purple) is considered for global consistency. All other vertexes (“out” - white) does not contribute at all to the minimization problem.

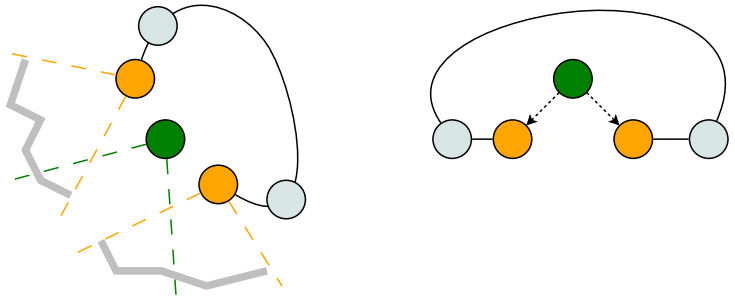


Figure 2.3.5: SlamDunk tracks features within an active window of neighboring keyframes. Thus, a loop can be implicitly closed (left image) by a camera frame (green) matching temporally distant keyframes (orange). Also, if these new links would significantly reduce the distant between the two ends of the loop (right image), a local optimization is triggered.

feature match constraints connected to the first ring of neighbors. Then, we iterate up to a certain ring R, *e.g.* $R = 3$ in Fig. 2.3.4, and set as *fixed* the last ring of vertexes. In this way, the edges connected to a *fixed* vertex contribute to the cost function, but the camera pose does not change, so that we keep the local map consistent with the whole camera path. Should the R^{th} ring be empty, we fix the root node in order to avoid gauge freedom.

2.3.4 Loop Closures

As discussed in Sec. 2.3.1, the keyframes which contribute to the *Feature Pool* are chosen for their spatial proximity, rather than their mutual distance on the pose graph (see Sec. 2.3.3). This way, during a loopy browsing of a small scene, a naturally dense connection of keyframes arises and enables fast and accurate camera localization. Also, small to medium size loop closures are implicitly handled by the system, while other approaches require a dedicated step in the pipeline [35, 44, 30]. However, while camera tracking benefits from old keyframes entering the active window, a local optimization of the camera path is triggered only when a new keyframe is spawn, *i.e.* when the overlapping between

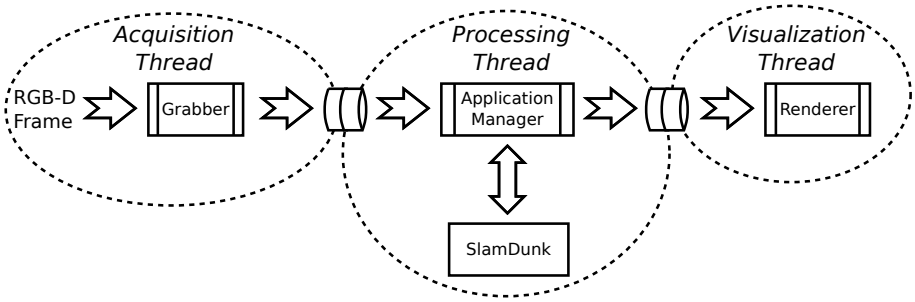


Figure 2.4.1: SlamDunk has been embedded in a mobile application for Android devices. Three main threads decouple image acquisition (left), actual SLAM application (center) and instant visualization of the 3D reconstruction (right).

the last tracked frame and the local map is low (see Sec. 2.3.2). Hence, drift errors are not corrected if such event does not happen. Therefore, we also explicitly detect loop closures by measuring the keyframe distance on the pose graph. More precisely, after a successful tracking, we consider the set of tracked features and collect their parent keyframes in the active window. Then, for each possible pair of keyframes, we compute their distance on the pose graph (see Fig. 2.3.5) and, if above a threshold, we create a new keyframe from the last tracked camera frame and perform a local optimization as discussed in Sec. 2.3.3. Being this process closely related to the size of the subgraph we optimize, we found a proper value for the keyframe distance threshold to be equal to the number of rings R introduced in Sec. 2.3.3.

2.4 SlamDunk For Mobile Devices

The SlamDunk system described in Sec. 2.3 exhibits low memory consumption and low computational complexity, due to the smart keyframe detection policy and the local mapping paradigm. Therefore, it represents a natural candidate for a novel mobile RGB-D SLAM application. We have chosen the popular Android operating system due to its widespread availability, but we plan to release an implementation

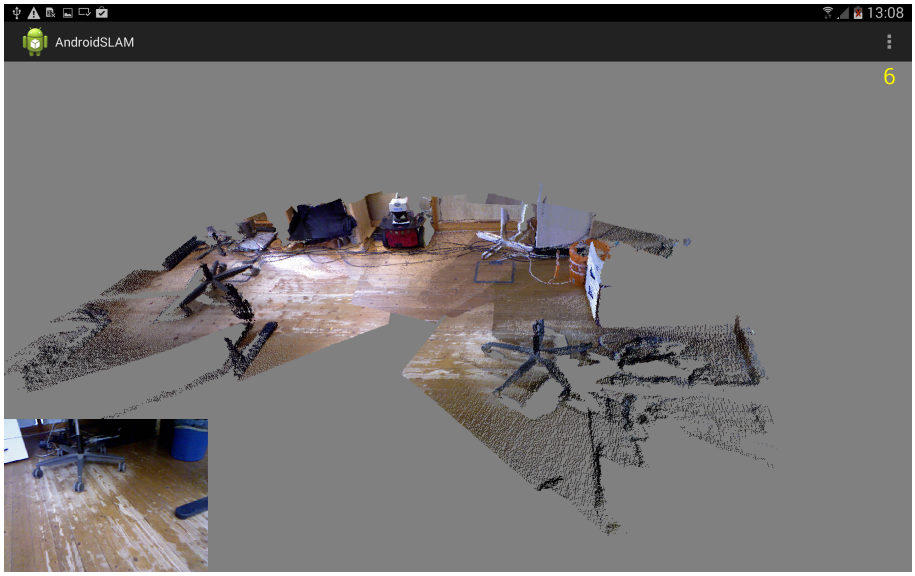


Figure 2.4.2: Application interface for the Android implementation of SlamDunk. The incremental reconstruction is rendered within a 3D window, while the current RGB image is displayed in the bottom left corner.

for iOS and Windows Phone operating systems with minimal changing in the near future. The software architecture is organized in three modules, running on concurrent threads, which are named in Fig. 2.4.1 as *Acquisition Thread*, *Processing Thread* and *Visualization Thread*. These modules asynchronously process and share data through double buffer interfaces. A screenshot of the running application is shown in Fig. 2.4.2.

The application is conceived for both testing and live data processing. Purposely, the *Acquisition Thread* is built around a *Grabber* entity which abstracts from the actual image source. Thus, RGB-D frames are gathered from a real sensor, *i.e.* a Microsoft Kinect or a Asus Xtion PRO Live, or read from a recorded sequence at a given frame rate and written in a concurrent double buffer. This buffer is always up-to-date with the latest available data, so that the *Processing Thread* simply skips frames if running at a lower rate, instead of queuing and lagging. Though a similar approach may harm SLAM systems based on pairwise tracking,

as long as the camera does not exit the active window SlamDunk is not affected by the amount of motion between two successive frames (see Sec. 2.3.2).

The *Processing Thread* hosts the *Application Manager* and it is the most resource consuming thread. It receives RGB-D image pairs through a shared buffer interface and runs the SlamDunk algorithm, which returns the estimated camera pose. If a local optimization has been run (see Sec. 2.3.3), it retrieves also a list of refined keyframe poses. Then, we iterate through the current depth map and, for each valid pixel measurement $D_i(u,v)$ we compute the point

$$\breve{\mathbf{p}} = \begin{pmatrix} u \cdot D_i(u,v) \\ v \cdot D_i(u,v) \\ D_i(u,v) \\ 1 \end{pmatrix}, \tag{2.4.1}$$

which is a 3D point in homogeneous coordinates. Let the corresponding estimated pose be the 4×4 transformation matrix T_i (cfr. Eq. (2.3.3)), then, recalling Eq. (2.3.1), the world coordinates for the point (u,v) would be

$$\mathbf{p} = \mathrm{T}_i \cdot \begin{pmatrix} \mathrm{K}^{-1} & \overline{\mathbf{0}} \\ \overline{\mathbf{0}} & 1 \end{pmatrix} \cdot \breve{\mathbf{p}} \tag{2.4.2}$$

expressed in 3D homogeneous coordinates. However, for efficiency, in the output buffer we only list points as in Eq. (2.4.1) together with the 4×4 matrix

$$\mathrm{T}_i \cdot \begin{pmatrix} \mathrm{K}^{-1} & \overline{\mathbf{0}} \\ \overline{\mathbf{0}} & 1 \end{pmatrix}, \tag{2.4.3}$$

leaving to the *Visualization Thread* the final projection and transformation. This module, then, updates the poses refined during the optimization step and shows the last tracked frame for instant feedback. In our implementation the rendering pipeline has been developed using the

OpenGL ES framework [79].

In the original SlamDunk proposal, described in Sec. 2.3, we have chosen to track SIFT [57] or SURF [7] features. However, both these algorithms do not suit a mobile platform for the following reasons:

- the extraction and description pipeline requires more computational power and cannot be run at interactive frame rate;
- the feature descriptor is long (up to 128 floating-point numbers), thus limiting the speed of the feature matching stage (see Sec. 2.3.2) due to the time consuming calculation of Euclidean distances.

Therefore, we considered many other keypoint detectors and feature descriptors proposed by the scientific community and compared efficiency and effectiveness on this specific use case. The most promising variants we found are the following:

- ORB as keypoint detector and feature descriptor [73];
- ORB as keypoint detector [73] and BRISK as feature descriptor [52];
- Upright-SURF (U-SURF) as keypoint detector [7] and BRISK as feature descriptor [52].

The SURF keypoint detector has been deployed with no computation of the feature orientation, *i.e.* the *upright* variant, and, also, exploiting the ARM NEON instruction set for fast computation. Finally, we point out that all the three variants produce binary vectors, which can be easily matched by computing the Hamming distance, rather than the slower Euclidean distance.

2.5 Experimental Results

The SlamDunk RGB-D SLAM system has been evaluated both quantitatively and qualitatively. In particular, in this section we will report the

Table 2.5.1: SlamDunk (desktop) compared to RGB-D SLAM. RMS of absolute trajectory error (meters) on four sequences from the RGB-D benchmark dataset [89] using a Microsoft Kinect camera.

Sequence	SD SIFT	SD SURF64	SD SURF128	RGB-D SLAM	RGB-D SLAM w/ EMM
fr1/xyz	0.017	**0.016**	**0.016**	0.021	<0.02
fr1/360	0.111	0.101	0.084	0.103	**<0.07**
fr1/desk	**0.022**	0.027	0.025	0.049	0.026
fr1/floor	0.044	0.052	**0.042**	0.055	<0.05
AVERAGE	0.048	0.049	**0.042**	0.057	

results obtained on various sequences from the widely adopted RGB-D benchmark dataset introduced by Sturm *et al*. [89], which provides color and depth image pairs acquired at full resolution from a Microsoft Kinect and an Asus Xtion PRO Live cameras, together with ground truth data estimated by a motion capture system tracking the camera movements. Though sometimes lacking accuracy, mainly due to the complexity of the system, still the available reference trajectory is useful to fairly compare different methods.

In Tab. 2.5.1 we report the RMS of the absolute trajectory error (ATE) obtained with our approach and the state-of-the-art method RGB-D SLAM [30] on four sequences from the RGB-D benchmark dataset. For a fair comparison, here we compare the accuracy of our desktop implementation when using different visual feature descriptors, namely SIFT (SD SIFT) [57], 64-element SURF (SD SURF64) and 128-element SURF (SD SURF128) [7], while for RGB-D SLAM we have considered the original results reported in [29] (RGB-D SLAM) and the improved version [30] (RGB-D SLAM w/ EMM), which makes use of an Environment Measurement Model to verify camera pose estimates. Both "RGB-D SLAM" and "RGB-D SLAM w/ EMM" deploy SIFT features computed on GPU [103], though, we achieve comparable or better results with all three variants of SlamDunk. Also, while RGB-D

Table 2.5.2: SlamDunk (desktop): additional results. RMS and median value of absolute trajectory error (meters) on four sequences from the RGB-D benchmark dataset [89] using a Asus Xtion PRO Live camera.

Sequence		SD SIFT	SD SURF64	SD SURF128
fr3/long office household	RMS	**0.023**	0.026	0.027
	median	**0.021**	0.024	0.022
fr3/structure texture near	RMS	**0.012**	0.015	0.016
	median	**0.007**	0.011	0.012
fr3/structure texture far	RMS	**0.024**	**0.024**	0.025
	median	0.022	0.019	**0.015**
fr3/teddy	RMS	**0.069**	0.095	0.089
	median	**0.033**	0.056	0.067
AVERAGE	RMS	**0.032**	0.040	0.039
	median	**0.021**	0.028	0.029

Figure 2.5.1: SlamDunk (desktop): robot navigation through an apartment.

Figure 2.5.2: SlamDunk (desktop): detailed reconstruction of a room.

SLAM runs at a frame rate between 6 and 15, SD SURF64 constantly reach 35/40 FPS, SD SURF128 20/25 FPS (with a maximum of 30 FPS) and SD SIFT 6/8 FPS. Indeed, as already mentioned, RGB-D SLAM achieves real-time operations only when using ORB features [73], which entail less accuracy, though. As for the rest of the SlamDunk's pipeline, on average, feature matching and RANSAC-based pose estimation takes about 2 ms, the update of the *Feature Pool*, which scales with the number of keyframes within the active window, requires 4.8/14.6 ms, while the local optimization step ranges from 3.6 and 37.3 ms. However, the optimization is run only upon keyframe spawning, so the impact on the overall performance may vary and it is usually limited. Tab. 2.5.2 reports additional results for the desktop implementation of SlamDunk, showing both the RMS and the median value of the ATE. Indeed, a high RMS error could be related to few large estimation errors, while the median reflects the average accuracy. The SD SIFT variant consistently achieve better results, though the frame rate does not allow for real-time operations. Therefore, SURF features can be considered a good trade-off between speed and reconstruction quality. Finally, Fig. 2.0.1, 2.5.1 and 2.5.2 show the effectiveness of our proposal in a various settings, including object reconstruction, indoor mapping and robot navigation.

Table 2.5.3: SlamDunk (Android): RMS of absolute trajectory error (meter) for selected sequences from the RGB-D benchmark dataset [89].

Sequence	SD ORB-ORB	SD ORB-BRISK	SD USURF-BRISK
fr1/floor	0.058	0.055	**0.051**
fr1/desk	0.049	0.052	**0.042**
fr1/room	0.270	0.278	**0.140**
fr3/structure texture near	0.092	0.047	**0.025**
fr3/structure texture far	0.052	0.045	**0.028**
fr3/no structure texture near with loop	0.046	0.057	**0.030**
fr3/no structure texture far	0.178	0.139	**0.083**
fr3/long office household	0.058	0.063	**0.041**
AVERAGE	0.100	0.092	**0.055**

Recordings are available on our website[1] showing the chair in Fig. 2.0.1 and the room in Fig. 2.5.2. Also, we demonstrate recovery after occlusion in a live reconstruction video. Indeed, as long as the incoming frame does not exit the current active window, SlamDunk is able to track the camera. These sequences have been recorded by freely moving an Asus Xtion PRO Live camera and processed by the SD SURF128 version of our algorithm.

As for our Android implementation, Tab. 2.5.3 reports the RMS of the ATE on eight sequences from the RGB-D benchmark dataset [89]. We have compared the three different combination of keypoint detector and feature descriptor discussed in Sec. 2.4: ORB as detector and descrip-

[1] http://vision.deis.unibo.it/research/104-nfioraio-thesis

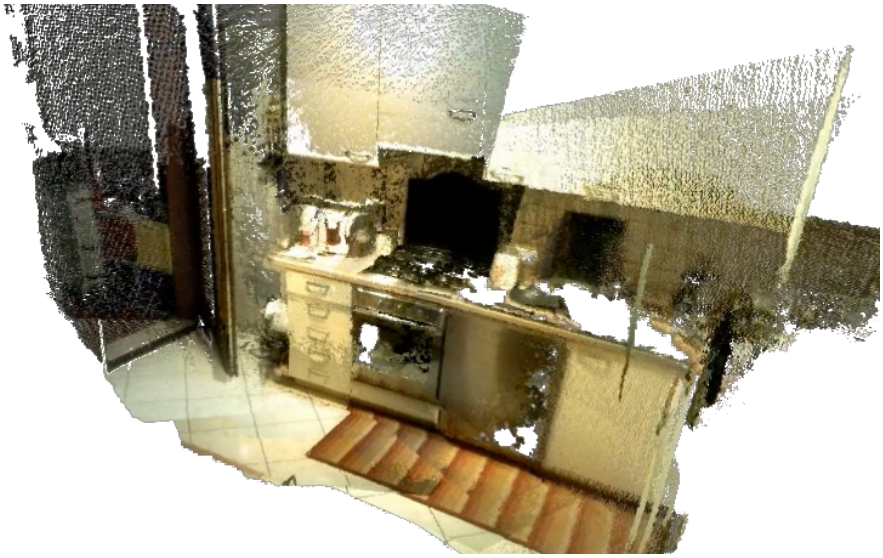
tor [73] (SD ORB-ORB), ORB as detector and BRISK as descriptor [52] (SD ORB-BRISK) and Upright-SURF [7] as detector and BRISK as descriptor. Clearly, the latter variant gives the best reconstruction accuracy, mainly due to the higher repeatability of the detected keypoints. Interestingly, it also represents a good choice for its timing performance. Indeed, it requires, on average, 84 ms for keypoint detection and description, whereas SD ORB-ORB takes 100 ms and SD ORB-BRISK only 35 ms, though it leads to much less accurate reconstructions. As for typical execution time of the other modules of the SlamDunk system, feature matching and RANSAC-based pose estimation require 1/10 ms, the update of the *Feature Pool* takes 20/40 ms and a local optimization could require a minimum of 20 ms up to seconds of processing time. Indeed, including all the feature matches as cost terms in the least-squares problem could seriously slow down the performance. Accordingly, we are currently investigating on alternative simplifications of the problem, *e.g.* by marginalization of the matches to pose-pose constraints. Also, including inertial data measurements, available on any mobile device, into the camera pose estimation step is another interesting research direction we are going to pursue.

Beside quantitative analyses, the accuracy can be visually inspected in Fig. 2.5.3, where we show the result obtained on the *fr1/floor* sequence from the RGB-D benchmark dataset [89] as well as the online reconstruction of a kitchen acquired by an Asus Xtion PRO Live camera connected to the tablet. The recording of the live reconstruction of the latter is provided on our website[2] together with the live reconstruction of a room.

Recently, we have replaced the external Asus camera with a more compact Structure sensor [68] attached to the device (see Fig. 2.5.4). Depth images are paired with color frames acquired from the on-board RGB

[2] http://vision.deis.unibo.it/research/104-nfioraio-thesis

(a) Reconstruction from the sequence *fr1/floor* of the RGB-D benchmark dataset [89].



(b) Online reconstruction of a kitchen on the tablet connected to an Asus Xtion PRO Live camera.

Figure 2.5.3: SlamDunk (Android): additional qualitative results.

Figure 2.5.4: A Structure depth sensor [68] has been attached to the tablet body and calibrated with the integrated RGB camera.

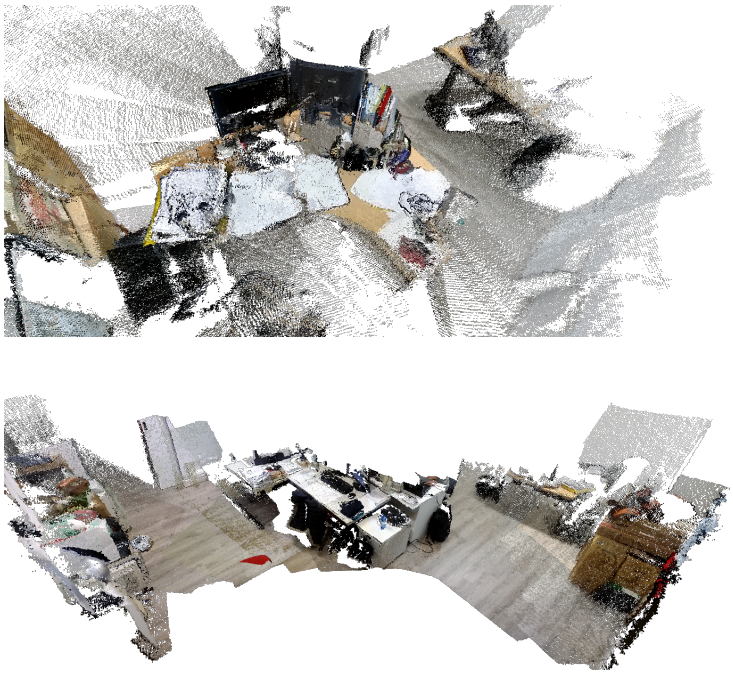


Figure 2.5.5: Preliminary results on a tablet device obtained combining a Structure sensor [68] with the on-board RGB camera.

camera, so that the SlamDunk algorithm can be run on a tablet with greater immersive user experience. Preliminary results using this setting are shown in Fig. 2.5.5. In the future, we plan to leverage on available inertial sensors, *e.g.* the gyroscope and the accelerometer, to further exploit the capability of a tablet device.

An important feature for any application developed for a mobile platform is its memory footprint, being the total amount of available memory usually very small compared to a desktop application. As for the SlamDunk framework, we will consider both the size required by the core algorithm and the visualization module. The map is represented as a collection of keyframes, each composed of a list of feature descriptors, whose length depends on the extraction method used, which are localized in camera space by 3 floating-point values. Also, we attach an integer index for fast retrieving of the parent frame during the feature matching step. Therefore, considering ORB and BRISK feature descriptors, which comprises, respectively, 32 and 64 byte element, the memory occupancy for 100 keyframes, each composed of an average number of 500 features, is

$$\begin{aligned} \mathfrak{M}_{\text{ORB}} &= (32\,\text{B} + 3 \cdot 4\,\text{B} + 4\,\text{B}) \cdot 100 \cdot 500 \\ &= 2400000\,\text{B} \approx 2.3\,\text{MB}, \end{aligned} \tag{2.5.1}$$

$$\begin{aligned} \mathfrak{M}_{\text{BRISK}} &= (64\,\text{B} + 3 \cdot 4\,\text{B} + 4\,\text{B}) \cdot 100 \cdot 500 \\ &= 4000000\,\text{B} \approx 3.8\,\text{MB}. \end{aligned} \tag{2.5.2}$$

Then, each keyframe has a corresponding node in the pose graph storing the estimated pose as a quaternion and a translation vector in double precision. Also, each node is uniquely identified by an integer index, leading to a memory occupancy for 100 keyframes of

$$\mathfrak{M}_{\text{poses}} = (7 \cdot 8\,\text{B} + 4\,\text{B}) \cdot 100 = 6000\,\text{B} \approx 6\,\text{KB}. \tag{2.5.3}$$

The local optimization problem maps each feature match to a cost term which stores the associated 3D points and the confidence weight in

double precision (cfr. Eq. (2.3.6) and (2.3.7)). Considering again 100 keyframes with 200 unique valid matches each, we get

$$\begin{aligned}\mathfrak{M}_{\text{cost terms}} &= (2 \cdot 3 \cdot 8\,\text{B} + 8\,\text{B}) \cdot 200 \cdot 100 \\ &= 1120000\,\text{B} \approx 1.1\,\text{MB}.\end{aligned} \tag{2.5.4}$$

The Android implementation of SlamDunk solves the least-squares problem in Eq. (2.3.8) using the same graph optimizer of the desktop version, *i.e*. G2O [49], which exploits the inherent sparsity of the problem by allocating space for vector **b** and matrix H in Eq. (2.3.14) only for non-zero blocks. Being the unknowns ΔT_i vectors in $\mathbb{R}^6$, G2O requires, for each keyframe, a 6×6 double precision block in H and a 6×1 double precision segment in **b** and, for each pair of matching keyframes, another 6×6 double precision block in H. Hence, if each keyframe is uniquely connected, on average, to 10 other keyframes, the memory occupancy is

$$\begin{aligned}\mathfrak{M}_{\text{solver}} &= (36 \cdot 8\,\text{B} \cdot (1 + 10) + 6 \cdot 8\,\text{B} + 6 \cdot 8\,\text{B}) \cdot 100 \\ &= 326400\,\text{B} \approx 319\,\text{KB},\end{aligned} \tag{2.5.5}$$

where we have also considered the 6 double precision values of every ΔT_i vector. Finally, the memory footprint of the core algorithm can be estimated, for ORB and BRISK features, as

$$\mathfrak{M}_{\text{SD ORB}} \approx 2.3\,\text{MB} + 6\,\text{KB} + 1.1\,\text{MB} + 319\,\text{KB} \approx 3.7\,\text{MB}, \tag{2.5.6}$$

$$\mathfrak{M}_{\text{SD BRISK}} \approx 3.8\,\text{MB} + 6\,\text{KB} + 1.1\,\text{MB} + 319\,\text{KB} \approx 5.2\,\text{MB}. \tag{2.5.7}$$

As for the visualization module, for each keyframe we save the 4×4 matrix in Eq. (2.4.3) and a list of 3D points as four floating point values as in Eq. (2.4.1). Though we may further reduce the required memory size by reducing the transformation matrix to only 3 rows and the point vectors to 3 elements, we prefer to aligned memory address to best exploit the graphics pipeline. Also, every point is paired with an

RGB color, again expressed as a 4 byte vector for alignment purposes, leading to a total occupied memory of

$$\begin{aligned}\mathfrak{M}_{\text{visualization}} &= (640 \cdot 480 \cdot (4 \cdot 4\,\text{B} + 4\,\text{B}) + 16 \cdot 4\,\text{B}) \cdot 100 \\ &= 614406400\,\text{B} \approx 585.9\,\text{MB},\end{aligned} \tag{2.5.8}$$

where we have considered 100 keyframes at VGA resolution. Though very high, this value can be extremely reduced by simply subsampling the source depth images, since the visualization is used only as feedback for the user and does not affect the core algorithm. Indeed, a sampling factor of two yields almost a 75% memory gain, that is $\mathfrak{M}'_{\text{visualization}} \approx 146.5\,\text{MB}$, giving a total memory footprint of

$$\mathfrak{M}_{\text{SD TOTAL ORB}} = \mathfrak{M}_{\text{SD ORB}} + \mathfrak{M}'_{\text{visualization}} \approx 150.2\,\text{MB}, \tag{2.5.9}$$

$$\mathfrak{M}_{\text{SD TOTAL BRISK}} = \mathfrak{M}_{\text{SD BRISK}} + \mathfrak{M}'_{\text{visualization}} \approx 151.7\,\text{MB}. \tag{2.5.10}$$

Clearly, the required space is dominated by the visualization block, thus vouching the low impact of the core algorithm on the available resources.

Chapter 3

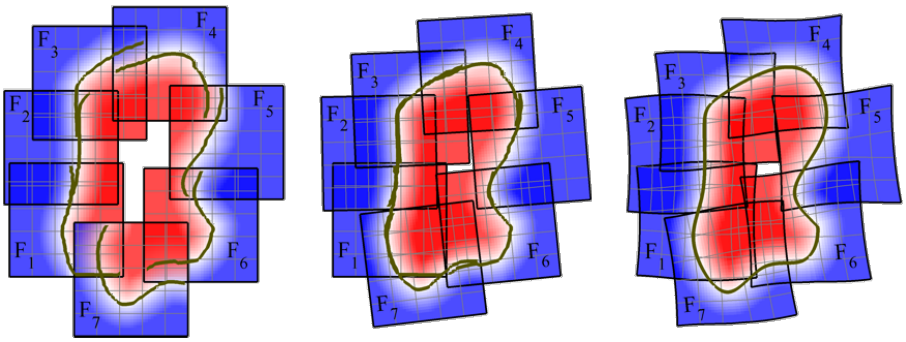
Large Scale Surface Reconstruction

In the previous chapter we have shown a RGB-D SLAM system, SlamDunk, which has been successfully implemented on a mobile platform. Purposely, SlamDunk has been designed for low memory consumption and low computational effort. Thereby, additional features which may have enhanced the user experience, *e.g*. realtime surface reconstruction, are not supported by the system. On the other hand, over the past few years desktop machines have witnessed considerable technological advances, so that today high-end processors coupled with GP-GPU hardware accelerators allow for addressing such challenges. Recently, KinectFusion and similar approaches [64, 38, 13, 72, 15, 66, 100] have shown impressive results in the field of real-time camera localization and surface reconstruction, even in complete dark environments. Key to these results is the use of a dense representation of the scene, *i.e*. the TSDF volume (see Sec. 3.2), that is a model built incrementally by fusing noisy depth measurements gathered from a freely moving Kinect-like camera sensor. This model implicitly encodes surfaces as zero-level sets of a *distance* function, so that a mesh can be extracted, *e.g.*, by ray casting or through the marching cubes algorithm [55].

However, most of the proposed approaches do not address the problem of error accumulation on the estimated camera path along large exploratory sequences. Indeed, though the camera tracking algorithm usually produces low-drift poses, small misalignments may show up as gross errors when closing a loop. To address these issues, Whelan *et al*. [101] extract and optimize a triangle mesh from the TSDF volume, while Zhou *et al*. [106, 107] process the data in multiple passes. However, we observe that the former approach implies loss of dense information in favor of a non-rigid mesh model, possibly leading to inconsistencies when coming back to an already mapped area, while the latter cannot run in real-time nor online and, moreover, it always requires all the input data. Also, both need RGB data and explicit loop closure, so that these methods cannot address challenging scenarios in low-textured environments and low-light conditions or even complete darkness, *e.g*. when playing AR games. In this chapter we will describe a different approach, outperforming existing solutions, which possesses all the features listed below:

- real-time camera tracking coupled with online model correction;
- no need to store any input data;
- no need of color data, *i.e*. only the depth image is used;
- no explicit loop closure detection;
- full global surface alignment, rather than non-rigid optimization of a subsampled mesh [101], taking minutes, instead of hours [106, 107], of processing.

We leverage the KinectFusion camera tracking system by subdividing the full map in smaller TSDF *subvolume* entities (see Sec. 3.3). Every subvolume is a low-drift local representation, while error accumulation can still be noted when considering the whole path, as shown in Fig. 3.0.1a. However, refining subvolumes' poses through global alignment leads to a consistent reconstruction, as in Fig. 3.0.1b. This pose



(a) (b) (c)

Figure 3.0.1: After a complete loop, drift errors may generate a discrepancy (a). Purposely, we consider smaller low-drift subvolumes (F_1-F_7) and refine their poses (b). However, beside *inter*-volume alignment, *intra*-volume surface deformation is still there and might be corrected by non-rigid pose estimation (c).

optimization exploits the dense TSDF data and can be performed online, rather than upon detection of a loop closure. Still, artifacts and deformations may be present in the final result, mainly due to unfiltered sensor noise. Non-rigid surface alignment might help to address such issues, as shown in Fig. 3.0.1c, though increasing the number of unknowns to estimate. Conversely, we propose to *blend* together the subvolumes according to their estimated 6-DOF poses (see Sec. 3.3.3). Indeed, this approach provides appealing results with less effort.

In the next section we are going to review the most important contributions so far to the problem stated. Then, the KinectFusion system is described in Sec. 3.2, being our reference algorithm for low-drift camera tracking. Surface reconstruction by subvolume modeling is presented in Sec. 3.3, while Sec. 3.4 reports experimental findings. This work has been carried out during an internship at Microsoft Research Cambridge (UK) and it has been recently accepted at the 2015 international conference on Computer Vision and Pattern Recognition (CVPR) [112].

3.1 Surface Reconstruction And Submapping

In the past few years, the availability of cheap RGB-D sensors capturing frames at 30 Hz, such as the Microsoft Kinect and the Asus Xtion PRO Live, has fostered the development of novel solutions to the SLAM problem. Beside sparse tracking approaches [35, 30], Newcombe *et al*. [64] introduces KinectFusion, a dense camera tracking technique capable of high quality real-time surface reconstruction by means of a *truncated signed distance function* (TSDF) representation of the scene. Soon, the ICP-based [8, 16] frame alignment was improved by Kubacki *et al*. [48] and Bylow *et al*. [13] by directly using that distance function as the cost to be minimized, outperforming the original KinectFusion proposal. Still, these contributions did not address the main limitation of the system, that is the inability of mapping outside a predefined volume, usually as large as a small room.

To overcome this issue, two different approaches have been investigated. First, Roth and Vona [72] and Whelan *et al*. [102] proposed to simply move the original volume according to camera movements. While [102] allows only translations of the volume to optimize TSDF data shifting, [72] places the volume always in front of the estimated camera position and implements a more complex interpolated sampling to support any kind of possible volume pose. Also, [102] extracts a point cloud from the voxel slices exiting the volume and triangulates such points, thus obtaining a reconstruction of the whole scene, though losing dense TSDF data as the volume moves. A different approach, then, exploits the sparsity of the TSDF volume, *i.e*. the number of valid voxels being usually small, to compress the data and extend the volume to large environments. To this end, Zeng *et al*. [104] have used octree spatial indexing, Chen *et al*. [15] a more adaptive multi-resolution scheme, while, recently, Niessner *et al*. [66] has leveraged hashing techniques. Also, to allow for reconstruction of even larger scenes, they all apply volume shifting, but, unlike [102], exiting voxels are streamed to the host memory, so that, when visiting known environments, voxels are

moved back on the GPU memory, thus simulating an unbounded TSDF volume.

Though all these approaches extend KinectFusion mapping capability, none is able to reduce the drift error accumulated over long trajectories. Indeed, if a TSDF volume is built while tracking, changing of camera poses after trajectory optimization would require to fuse again from scratch all the frames. Therefore other full-fledged SLAM systems either adopt a different model of the environment [42, 74] or perform camera trajectory optimization using different tracking and mapping methods [86, 85] which, unlike ours, usually need RGB data, deploying depth image fusion as just a final surface reconstruction step. Both Keller *et al*. [42] and Ruhnke *et al*. [74] propose a surfel-based representation [70] to jointly optimize camera poses and surface points for either real-time camera tracking [42] or trajectory refinement [74]. However, [42] does not counteract sensor drift, while [74] adjusts a given estimated trajectory by applying Generalized-ICP [78] to surfel models. Conversely, in this chapter we will present a real-time camera tracking algorithm decoupled from a global model refinement method which directly exploits TSDF measurement to compute dense surface correspondences, instead of the classical "normal-shooting" [74, 16].

Whelan *et al*. [101], suggest to use the volume for low-drift tracking only and they cast a global optimization over the extracted mesh. However, the optimization is triggered only when a loop closure is found by matching visual feature [32], thus requiring RGB data and forcing the user to induce such event. Moreover, they deploy non-rigid pose estimation over a deformation graph connected to a pose graph. Points on the mesh are then moved according to the estimated deformation. Instead, we do not change local appearance, rather we aim at finding the best position for each low-drift dense TSDF subvolume. Though the idea of splitting a global map into submap has been deeply studied [41, 10, 58], to the best of our knowledge this is the first work dealing directly with TSDF subvolumes.

Recently, Zhou *et al*. [106, 107] have shown impressive results outperforming any KinectFusion-based approach, achieving low-drift reconstruction on large scenes. In their first work [106] subgraphs are created around *point of interest* in the scene and the error is spread over such nodes. Then, this intuition is further expanded in [107] by considering general trajectory fragments, similar to our subvolumes, optimized in a non-rigid fashion. However, this method requires multiple passes over the data and hours of processing, so it cannot support online correction. Moreover, they initially estimate camera path by running the RGB-D SLAM algorithm [30], which leverages RGB data for camera tracking and loop closing. Therefore, [106, 107] cannot be run in untextured and low-light or dark settings.

Our approach extends the KinectFusion system and addresses all the highlighted issues. In the next section we will describe the tracking approach, which has been adapted from [13], while in Sec. 3.3 we will introduce our main contribution.

3.2 Depth Map Fusion

Recalling Sec. 2.3, we will assume to receive, at each time stamp i, a depth image D_i from a Kinect-like camera sensor. The internal parameters are known and we will refer to Eq. (2.3.1) and (2.3.2) when needed. Then, we aim at reconstructing the sensed surface while localizing the position of the camera with respect to the current model of the scene. If we are able to get a good estimate of the camera pose for each incoming frame, such model can be incrementally built by sequential fusion of the depth images into a Signed Distance Function (SDF). Following Curless and Levoy [20], we define a SDF as a function

$$\begin{aligned} \psi : \mathbb{R}^3 &\longrightarrow \mathbb{R}^2 \\ \mathbf{u} &\longmapsto (F(\mathbf{u}), W(\mathbf{u})) \end{aligned} \tag{3.2.1}$$

which assigns to each point $\mathbf{u}$ in the 3D space its distance from the nearest surface $F(\mathbf{u})$ and a confidence value $W(\mathbf{u})$. Also, such distance is signed, so that the surface can be extracted as the zero-level set of the SDF, *e.g.* through the Marching Cubes algorithm [55]. A key property of this representation is that a discretized version of such function can be built incrementally by independent update of each voxel every time new depth measurements become available. Following Newcombe *et al.* [64], let D_i be a depth image to be integrated and T_i the corresponding camera pose. First, we project each voxel $\mathbf{u} \in \mathbb{R}^3$ onto the image plane and retrieve the new surface measurement

$$d = D_i\left(\left\lfloor \pi_2\left(\mathrm{K}\pi_3\left(\mathrm{T}_i^{-1}\left[\mathbf{u}\right]\right)\right)\right\rfloor\right), \tag{3.2.2}$$

where $[\cdot] : \mathbb{R}^3 \to \mathbb{R}^4$ is the *homogeneous* operator, $\pi_i : \mathbb{R}^{i+1} \to \mathbb{R}^i$ is its inverse function, *i.e.* $\pi_2(x,y,z) := (x/z, y/z)$ and $\pi_3(x,y,z,w) := (x/w, y/w, z/w)$, and $\lfloor\cdot\rfloor : \mathbb{R}^2 \to \mathbb{Z}^2$ is the integer truncation. In the following, to improve readability, we will generally omit both the homogeneous operator and its inverse. Then, the estimated distance value is computed by the depth difference

$$F_{D_i}(\mathbf{u}) = \zeta\left(\mathrm{T}_i^{-1}\mathbf{u}\right) - d, \tag{3.2.3}$$

where ζ extracts the third coordinate, and the SDF is updated as

$$F(\mathbf{u}) := \frac{F(\mathbf{u})W(\mathbf{u}) + F_{D_i}(\mathbf{u})W_{D_i}(\mathbf{u})}{W(\mathbf{u}) + W_{D_i}(\mathbf{u})}, \tag{3.2.4}$$

$$W(\mathbf{u}) := W(\mathbf{u}) + W_{D_i}(\mathbf{u}). \tag{3.2.5}$$

Though the weight $W_{D_i}(\mathbf{u})$ can be related to sensor noise [20] or performance considerations [13], in practice a constant value is usually enough for camera tracking and surface reconstruction [13]. Moreover, to prevent a strong prior on early frames and allow a certain degree of robustness in dynamic scenes, the weight is truncated to some value. As for the distance value, Curless and Levoy [20] noted that surfaces sensed

on different sides of an object could interfere one to another, ruining the 3D model. Accordingly, they suggest to reduce to zero the weights $W_{D_i}(\mathbf{u})$ when $F_{D_i}(\mathbf{u}) > \mu$ and truncate the signed distance $F_{D_i}(\mathbf{u})$ to $-\mu$ before the surface, thus getting a *Truncated* Signed Distance Function (TSDF). In the following, we will refer to the value 2μ as the *truncation band* and to the voxels within it as *surface* or *non-truncated* voxels. In principle, μ should reflect sensor uncertainty and be, *e.g.*, inversely proportional to the measured depth. However, due to the limited range of a Kinect-like depth camera, usually ~0.5/4 m, we can safely set μ to a constant value, which also enables compact representations of the TSDF values. Indeed, if the truncation band is fixed, the distance function F can be scaled to the range $[-1, 1]$ and stored in 16-bit fixed point notation. Considering other 16 bits for the weight value, a single voxel requires only 32 bits of memory, leading to 512 MB for a typical volume of 512^3 voxels.

The updating step in Eq. (3.2.4) and (3.2.5) allows for fast parallel computation on a modern GPU [64]. Moreover, surface ray casting can be easily carried out by marching along the ray from the image plane until the detection of the zero level set. Such synthetic depth image is a surface prediction which is used in KinectFusion [64] for dense ICP alignment [8, 16] of each new depth image. Though the pose is iteratively refined by means of a multi-scale approach, ICP is known to be prone to local minima of the cost function, thus harming camera tracking in complex environments. Besides alternative techniques based on photometric error [84, 43, 98], which require a color image to be available, so that they do not suit low-light mapping scenarios, Bylow *et al.* [13] have proposed to minimize a cost function directly built onto the TSDF volume. More precisely, for each valid measurement $D_i(u, v)$ they consider the corresponding 3D point $\mathbf{p}$, as defined in Eq. (2.3.1), and note that, given the *true* camera pose T'_i and a noise-free acquisition, the distance value $F(\mathrm{T}'_i\mathbf{p})$ should be equal to zero. Assuming i.i.d. Gaussian noise on depth measurements, such transformation is found

by minimization of the following cost function:

$$E\left(\mathrm{T}_i\right) = \sum_j \left\| F\left(\mathrm{T}_i \mathbf{p}_j\right) \right\|^2, \tag{3.2.6}$$

where j runs over the whole depth image D_i. As already discussed (see Sec. 2.3.3), $\mathrm{T}_i \in \mathbb{SE}_3$ and, therefore, a minimal representation should be used to correctly handle the six degrees of freedom of the transformation. Accordingly, they rewrite Eq. (3.2.6) using the Lie algebra representation $\xi \in \mathfrak{se}_3 \subseteq \mathbb{R}^6$ as

$$E\left(\xi_i\right) = \sum_j \left\| F\left(\exp\left(\xi_i\right) \mathbf{p}_j\right) \right\|^2 = \sum_j \left\| F_j\left(\xi_i\right) \right\|^2, \tag{3.2.7}$$

where $F_j\left(\xi_i\right) = F\left(\exp\left(\xi_i\right) \mathbf{p}_j\right)$ and $\exp : \mathfrak{se}_3 \to \mathbb{SE}_3$ is the exponential mapping from the Lie algebra to the Special Euclidean Group of 4×4 matrices. Then, each cost term is linearized around an initial guess $\hat{\xi}_i$, *e.g.* $\log \mathrm{T}_{i-1}$, with $\log : \mathbb{SE}_3 \to \mathfrak{se}_3$, by its first order Taylor expansion

$$\begin{aligned} E\left(\xi_i\right) &\simeq \sum_j \left\| F_j\left(\hat{\xi}_i\right) + \nabla F_j\left(\hat{\xi}_i\right)\left(\xi_i - \hat{\xi}_i\right) \right\|^2 \\ &= \sum_j F_j\left(\hat{\xi}_i\right)^2 + 2F_j\left(\hat{\xi}_i\right)\nabla F_j\left(\hat{\xi}_i\right)\left(\xi_i - \hat{\xi}_i\right) \\ &\quad + \left(\xi_i - \hat{\xi}_i\right)^\top \nabla F_j\left(\hat{\xi}_i\right)^\top \nabla F_j\left(\hat{\xi}_i\right)\left(\xi_i - \hat{\xi}_i\right) \end{aligned} \tag{3.2.8}$$

and the solution is found by setting the derivative w.r.t. $\Delta\xi_i \triangleq \left(\xi_i - \hat{\xi}_i\right)$ to zero, *i.e.*

$$\sum_j F_j\left(\hat{\xi}_i\right)\nabla F_j\left(\hat{\xi}_i\right)^\top + \nabla F_j\left(\hat{\xi}_i\right)^\top \nabla F_j\left(\hat{\xi}_i\right)\Delta\xi_i = 0. \tag{3.2.9}$$

Deploying a classic parallel prefix-scan reduction on GPU they compute

$$\mathbf{b} = \sum_j F_j\left(\hat{\xi}_i\right)\nabla F_j\left(\hat{\xi}_i\right)^\top \in \mathbb{R}^6, \tag{3.2.10}$$

$$\mathrm{H} = \sum_j \nabla F_j \left(\hat{\xi}_i\right)^\top \nabla F_j \left(\hat{\xi}_i\right) \in \mathbb{R}^{6\times 6} \tag{3.2.11}$$

and, combining these with Eq. (3.2.9), solve, on host side, the problem

$$\mathrm{H}\Delta\xi_i = -\mathbf{b}. \tag{3.2.12}$$

Given the incremental solution $\Delta\xi_i^\star$, a new linearization point is found as $\xi_i^\star = \hat{\xi}_i + \Delta\xi_i^\star$ and the procedure iterated until convergence. This tracking approach, which does not need any synthetic surface prediction, has proved to be fast and reliable, outperforming the original KinectFusion proposal on a publicly available benchmark dataset [13].

3.3 Subvolume Reconstruction

The KinectFusion approach [64, 13] allows a highly detailed reconstruction of small workspaces. As we have seen in the previous sections, moving volume [72, 102] as well as data compression [104, 15, 66] techniques have been used for large scale mapping, with impressive results. However, the drift error inevitably corrupts the result, especially if the camera path involves one or more complex loops. To highlight such limitations, hereinafter we will refer to a reference moving volume algorithm exhibiting the following features:

- *camera tracking deploys the method introduced by Bylow* et al. [13] see Sec. 3.2;
- *the active volume is only translated* [102]
  indeed, rotating the active volume [72] would force a time consuming resampling of the function;
- *such translation occurs at multiples of the voxel size*
  this way, a rolling buffer technique [102] can be deployed to process only a small subset of voxels upon volume shifting;

- *voxels exiting the active volume are copied into a larger volume on host memory* [15, 66]
  so, we do not extract a mesh as in [102];

- *voxels entering the active volume are queried from the host volume and copied back to the GPU memory* [15, 66]
  this way, we generate an unbounded TSDF volume.

Two interesting failure cases of the reference moving volume algorithm are shown in Fig. 3.3.1. Upon loop closure, measurements integrated at the beginning of the sequence enter the active volume, though, they are misaligned w.r.t. the last pose estimates, so that camera tracking is performed against a corrupted model. This usually leads to a complete failure when the surface in the active volume falls before the previous one, *i.e*. within *empty* space, as in the top image of Fig. 3.3.1; otherwise, empty space is integrated with older non-truncated voxels, gradually deleting the previous surface, but leaving unpleasant inconsistencies, as in the bottom image of Fig. 3.3.1. Despite the catastrophic accumulation of error over long routes, we can still find locally the distinctive low-drift reconstruction of KinectFusion. Indeed, had we not copied voxels from host to active volume, the tracking task might have turned out successful, though leaving open the problem of global surface alignment upon voxel exiting the active volume.

In this chapter we will address large scale reconstruction within the KinectFusion framework by combining low-drift local modeling with global surface alignment. In Sec. 3.3.1 we will describe our robust camera tracking approach, which localizes the camera within a reliable local model of the scene, then in Sec. 3.3.2 we will show how a global model is kept updated by online registration of *subvolumes*, *i.e*. low-drift segments of the environment. Finally, being our final model a collection of subvolumes, in Sec. 3.3.3 we will investigate on retrieval of the final surface by blending together such subvolumes.

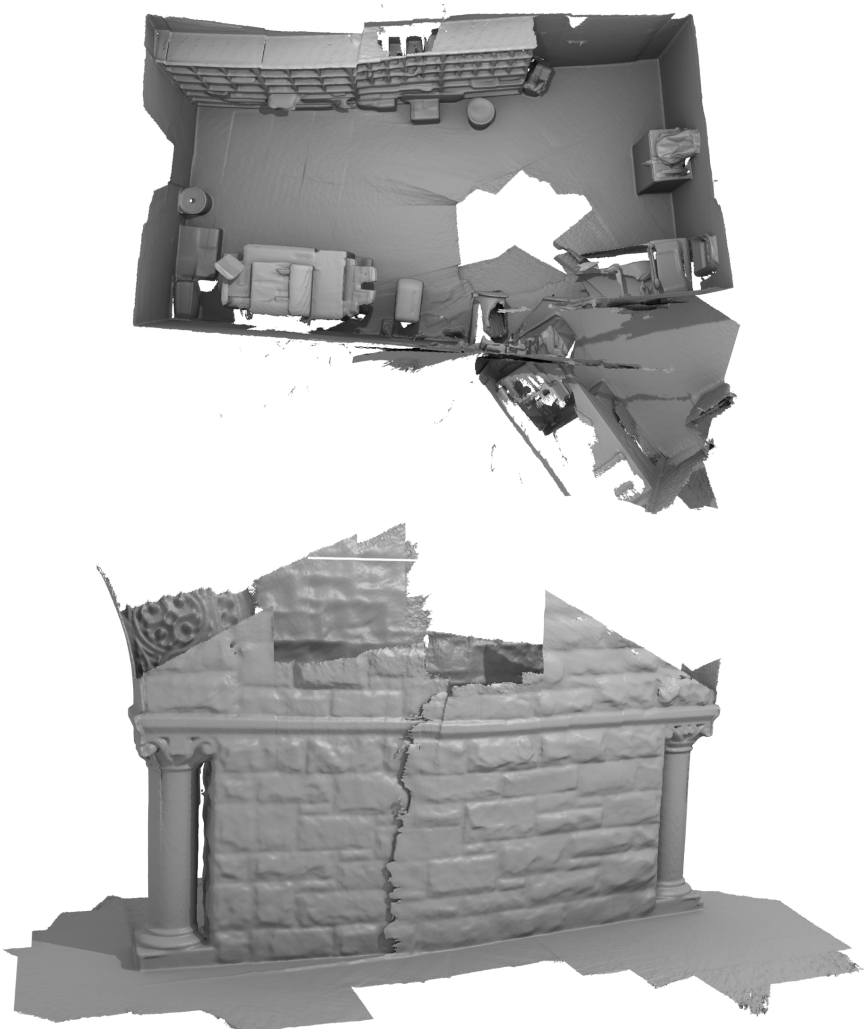


Figure 3.3.1: *Copyroom* (top) and *Stonewall* (bottom) sequences from [106]: the reference moving volume KinectFusion approach is hindered by the accumulated drift, leading to complete failures (top) or inconsistent reconstructions (bottom).

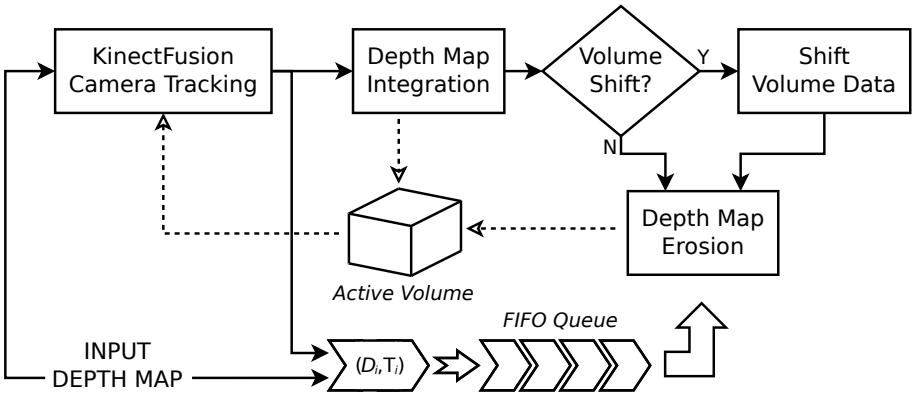


Figure 3.3.2: Wrokflow of the proposed system for low-drift camera tracking and surface reconstruction.

3.3.1 Low-drift Local Modeling

Following the general intuition, we define a large active volume on GPU main memory, *e.g.* ~5/6 m each side, and initially place the camera at its center. Then, when needed, we shift this volume by multiples of the voxel size to keep the camera within the central region of the cube. In this way, we can safely fix volume position when the camera rotates. However, common approaches stream data from GPU to host memory upon volume shifting, thus hindering the localization and mapping tasks, which finally leads to the issues previously discussed. Instead, in our system voxels exiting the active volume are purposely lost and no TSDF data is copied back from host to GPU memory. Indeed, we exploit the low-drift property of KinectFusion on short distances by building the active volume using only the last K depth frames, *e.g.* $K = 50$. We argue that such local reconstruction helps camera pose estimation by including only consistent data and, therefore, we expect to obtain a camera path exhibiting no failures and, locally, lower drifting errors. Clearly, this approach does not solve for global misalignment nor addresses overall surface registration. We will discuss these issues in Sec. 3.3.3.

Fig. 3.3.2 sketches the pipeline of our system. Depth images are aligned

to the current active volume according to [13] (see Sec. 3.2) and integrated by applying Eq. (3.2.4) and (3.2.5) to each voxel. Estimated camera translation from volume's center is compared to a threshold and, if larger, the volume is shifted. As already described, voxels which exit the volume as a consequence of the shifting procedure are simply discarded. To keep into the active volume the model created from the integration of the last K frames only, instead of building from scratch at every time stamp, we introduce the *erosion* process after every successful integration, by which the integration of the $(i-K)^{\text{th}}$ frame is reverted. This can be achieved by applying again the standard integration algorithm, but replacing Eq. (3.2.4) and (3.2.5) with, respectively,

$$F(\mathbf{u}) := \frac{F(\mathbf{u})W(\mathbf{u}) - F_{D_{i-K}}(\mathbf{u})W_{D_{i-K}}(\mathbf{u})}{W(\mathbf{u}) - W_{D_{i-K}}(\mathbf{u})}, \tag{3.3.1}$$

$$W(\mathbf{u}) := W(\mathbf{u}) - W_{D_{i-K}}(\mathbf{u}). \tag{3.3.2}$$

The erosion process forces to store previous depth images together with their estimated camera pose into a FIFO queue of K elements. Frames and poses are pushed into the queue when they are fused into the active volume and retrieved after K time instants for erosion. As the erosion process operates independently at each voxel, it exhibits the same constant complexity of depth integration. Therefore, the proposed approach is still constant time and allows for real-time camera tracking.

3.3.2 Online Subvolume Registration

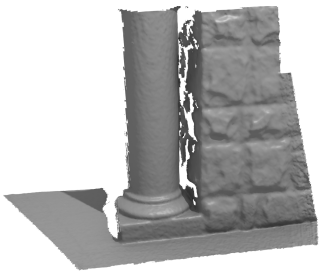
The erosion process described in the previous section enables real-time high-quality camera tracking. To deal with overall camera path alignment, classical approaches [47, 87, 30] rely on keyframe optimization, thus loosing the low-noise dense TSDF model. More recently, non-rigid mesh optimization [101] has been deployed, though this approach requires explicit loop closure detection based on feature descriptors extracted from RGB images. Conversely, we leverage on low-drift local

reconstructions to actively reduce drift and achieve global surface alignment. Interestingly, a similar intuition can be found in the work of Zhou *et al*. [107], where small subsets of depth frames are fused together in fragments, which are subsequently aligned together. However, unlike [107], our method creates *subvolumes*, *i.e*. fragments, in constant time alongside with camera tracking and optimizes their poses online, so that the overall reconstruction error is small even in absence of an explicit loop closure event.

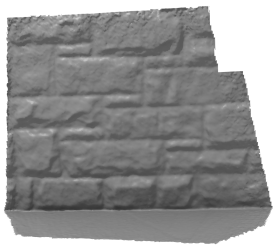
As explained in Sec. 3.3.1, depth images are pushed into a FIFO queue and extracted after K time instants for erosion. Initially, being the queue empty, frames are fused, but no erosion takes place. Once the queue is filled, we copy the whole active volume onto the host memory and tag the data as our first *subvolume*. Then, the erosion process starts and the first fused TSDF measurements are gradually removed from the volume. After K frames, the active volume no longer includes information from the first set of data, which shapes the first subvolume, and a new subvolume is spawned by copying the current active volume onto the host memory. Afterwards, we continue creating subvolumes every K integrations and erosions, thus mapping the environment as a collection of low-drift TSDF subvolumes. Fig. 3.3.3 shows an example of the surfaces extracted from typical subvolumes.

Though expensive, we notice that a full copy of the active volume involves only a single *reading* operation per voxel, so it is, in principle, a real-time procedure. However, current hardware constraints on GPU side as well as limited data bandwidth between host and device memory require a smart implementation to reduce the actual time needed. Purposely, we exploit the sparsity of the TSDF by considering, in parallel, blocks of 16^3 voxels:

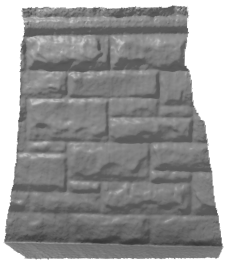
- blocks composed exclusively by voxels having zero weight are discarded;
- blocks consisting of truncated distance values only are marked as



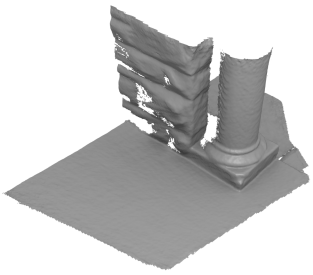
(a) Subvolume #0.



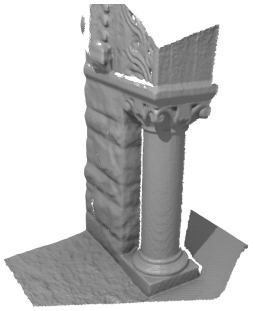
(b) Subvolume #5.



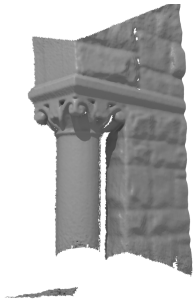
(c) Subvolume #10.



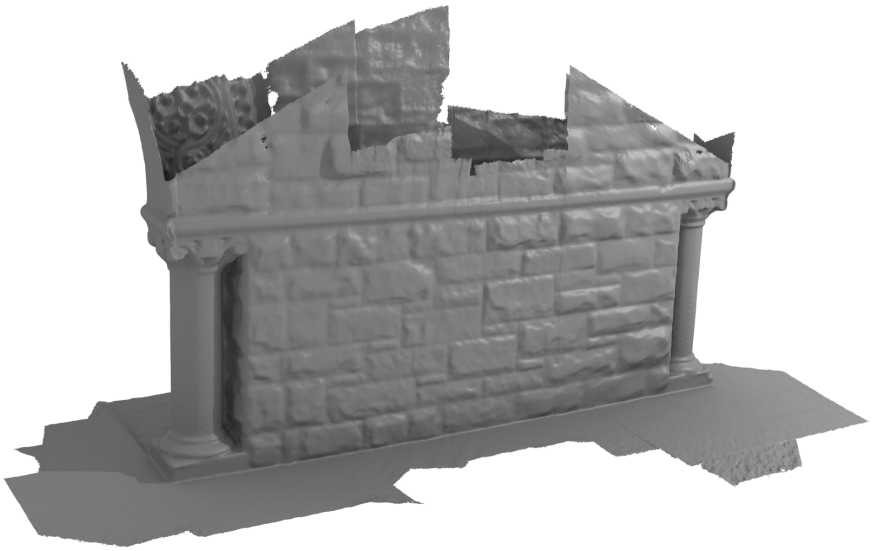
(d) Subvolume #20.



(e) Subvolume #40.



(f) Subvolume #45.



(g) Final reconstruction after global surface alignment.

Figure 3.3.3: Subvolumes are low-drift TSDF volumes built from K frames (here, $K = 50$). (a)-(f) show surfaces extracted as zero-level set from subvolumes, (g) shows the final reconstruction of the *Stonewall* sequence introduced in [106].

empty and just the maximum weight is saved;

- blocks including at least one surface voxel are fully copied.

Block data are streamed in an unstructured arrangement for higher efficiency. Then, on host side, they are gathered in a multi-resolution index by creating a new level of bricks, each comprising 4^3 blocks. Again, bricks formed by empty blocks only are folded and only the maximum weight is retained. In this way, we reduce the total memory footprint and enable the accumulation of a large number of subvolumes.

Every time a new subvolume is spawned, a pose V, *i.e.* the current position of the active volume, is attached to the TSDF data $\psi = (F, W)$ and global surface alignment is performed to reduce the overall drift by refinement of the subvolumes' poses. Though this is an online process, the complexity of the optimization routine increases with the number of subvolumes, so it cannot operate in real-time. Nevertheless, the camera tracking module does not need to wait for refined poses, and thus it can keep running in a separate thread while pushing new subvolumes into a shared buffer. The optimization aims at estimating a rigid-body pose for each subvolume by constraining points sampled on the zero-level set. To this aim, we developed an iterative algorithm inspired by the popular point-to-plane ICP method [16].

Given a subvolume $\psi_j = \left(F_j, W_j\right)$ with pose V_j, we extract a set of points $S_j = \left\{\mathbf{p}_i^j\right\}$ at the zero-level set of the distance function F_j and compute the respective normals as the normalized gradient at $\mathbf{p}_i^j$, *i.e.* $\mathbf{n}_i^j = \hat{\nabla} F_j\left(\mathbf{p}_i^j\right) \triangleq \frac{\nabla F_j\left(\mathbf{p}_i^j\right)}{\left\|\nabla F_j\left(\mathbf{p}_i^j\right)\right\|}$. Also, we find the minimum bounding box B_j aligned with ψ_j's reference frame. We note that, being the TSDF data fixed throughout the optimization, these requirements can be satisfied once for all upon subvolume creation. Then, for each subvolume ψ_j, we compare its bounding box B_j with all other bounding boxes to retrieve a candidate set $C_j = \left\{h \,\middle|\, B_j \cap B_h \neq \varnothing\right\}$ of overlapped subvolumes. To constrain each point $\mathbf{p}_i^j$ to these surfaces, we traverse the candidate set

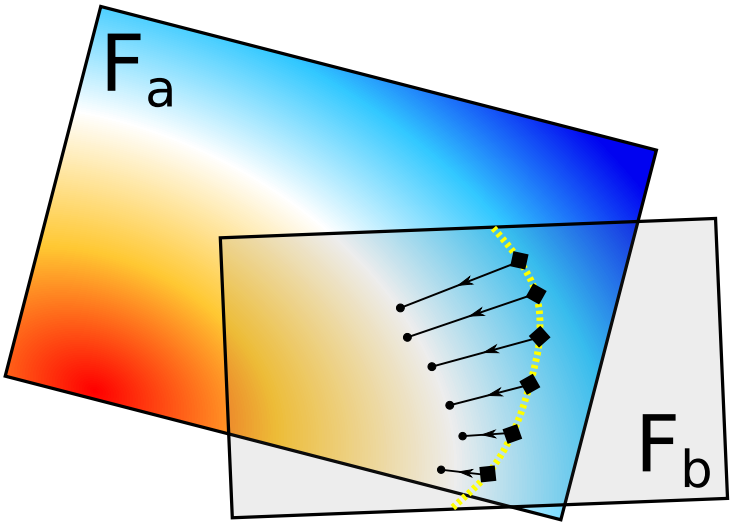


Figure 3.3.4: From each point $\mathbf{p}_i^b$ (black diamonds) sampled on the zero-level set of F_b (dashed yellow line) we move according to the distance function F_a and its gradient $\hat{\nabla}F_a$ (blue-white-red color gradient) to find a match (black circles).

and, for each $h \in C_j$, we define a corresponding point as

$$\mathbf{q}_i^{hj} = \mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{p}_i^j - F_h\left(\mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{p}_i^j\right)\hat{\nabla}F_h\left(\mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{p}_i^j\right), \tag{3.3.3}$$

where $F_h\left(\mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{p}_i^j\right)$ and $\hat{\nabla}F_h\left(\mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{p}_i^j\right)$ are estimated by trilinear interpolation. If F_h or its gradient are not defined at $\mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{p}_i^j$, the match is ignored. This method, illustrated in Fig. 3.3.4 by a graphical example, shares similar intuitions with the work of Kubacki *et al.* [48]. However, unlike [48], we aim here at surface reconstruction rather than camera tracking.

For each valid match so established, we build a cost term resembling the point-to-plane ICP error function [16], which computes the distance between one point and the tangent plane to the surface at the other point. To this end, the vector difference is projected along the direction of the normal, *i.e.*

$$e_i^{hj} = \left(\mathbf{p}_i^j - \mathrm{V}_j^{-1}\mathrm{V}_h\mathbf{q}_i^{hj}\right)\cdot\mathbf{n}_i^j, \tag{3.3.4}$$

where we have preferred $\mathbf{n}_i^j$ to $\hat{\nabla}F_h\left(\mathbf{q}_i^{hj}\right)$, the reason being twofold. On one hand, $\mathbf{n}_i^j$ can be computed once upon subvolume creation, while $\hat{\nabla}F_h\left(\mathbf{q}_i^{hj}\right)$ usually changes after every successful minimization, *i.e.* when Eq. (3.3.3) evaluates differently. On the other hand, we do not have any clue about the shape of F_h around $\mathbf{q}_i^{hj}$, hence the gradient may be undefined. Given the set of cost terms, defined as in Eq. (3.3.4), we estimate the poses for a number v of subvolumes by solving the following nonlinear least squares problem:

$$\begin{aligned}&\underset{\{\mathrm{V}_0,\dots,\mathrm{V}_{v-1}\}}{\arg\min} \sum_{j=0}^{v-1}\sum_{i\in I_j}\sum_{h\in C_j} e_i^{hj} \\ &\qquad = \underset{\{\mathrm{V}_0,\dots,\mathrm{V}_{v-1}\}}{\arg\min} \sum_{j=0}^{v-1}\sum_{i\in I_j}\sum_{h\in C_j} \gamma_{hij}\left(\mathbf{p}_i^j - \mathrm{V}_j^{-1}\mathrm{V}_h\mathbf{q}_i^{hj}\right)\cdot\mathbf{n}_i^j, \quad (3.3.5)\end{aligned}$$

where $I_j = \left\{ i \,\middle|\, \mathbf{p}_i^j \in S_j \right\}$ and

$$\gamma_{hij} = \begin{cases} 1 & \text{if } F_h \text{ and } \hat{\nabla}F_h \text{ are defined at } \mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{p}_i^j \\ 0 & \text{ow} \end{cases}. \quad (3.3.6)$$

We solve Eq. (3.3.5) using a Levenberg-Marquardt methods [53, 59] through the Ceres solver [2] (see Sec. 2.3.3 for further details).

In rare circumstances, the value γ_{hij} in Eq. (3.3.5) could be 0 for most or, possibly, all $h \in C_j$ and $i \in I_j$, *i.e.* that particular subvolume ψ_j has few or no connections with the rest of the map, leading to poor pose estimation or an underconstrained problem. In such cases, we reinforce the solution given by the camera tracker through the introduction of pose-pose constraints linking ψ_j to the previous subvolume, if any, and the following subvolume, if any. Let $\mathrm{Z}_{j-1,j}$ be a 4×4 matrix mapping points from the ψ_j reference frame to the ψ_{j-1} reference frame according to the tracker estimate. Then, the derived pose-pose error function is given by

$$e^{j-1,j} = \Phi_{j-1,j}\log\left(\mathrm{Z}_{j-1,j}\mathrm{V}_j^{-1}\mathrm{V}_{j-1}\right), \quad (3.3.7)$$

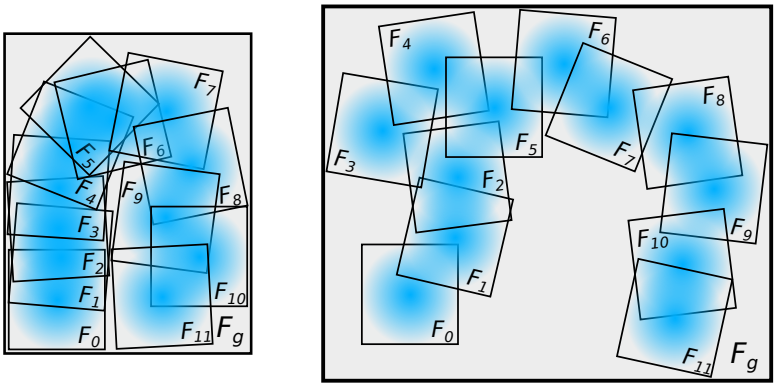


Figure 3.3.5: Building a single global volume ψ_g by subvolume averaging forces to traverse the whole collection even for zero-weight voxels. Therefore, computation time is strongly affected by relative positions of subvolumes, so that evaluation of the global volume shown on the left is faster than the one on right including the same subvolumes with the same extent but different poses.

where $\log : \mathbb{SE}_3 \rightarrow \mathfrak{se}_3$ and $\Phi_{j-1,j}$ is a 6×6 stiffness matrix, empirically set to the identity in our experiments.

Once a solution is found for Eq. (3.3.5) by iterative minimization, the candidate set C_j is rebuilt for each subvolume and correspondences are updated, leading to a new optimization problem. The whole process is iterated until convergence. Finally, to remove gauge freedom and ensure consistencies between the final estimated poses and the tracker's reference frame, the last subvolume's pose is always kept fixed in Eq. (3.3.5). Accordingly, the next spawned subvolume will be placed beside the previous one with consistent relative alignment.

3.3.3 Surface Reconstruction By Subvolume Blending

In the previous sections we have described our approach for low-drift camera tracking and global surface alignment. However, as already discussed, noisy depth measurements generate surface deformations which appear as unpleasant artifacts when the zero-level set is extracted from the subvolumes. This is especially enhanced within overlapped regions,

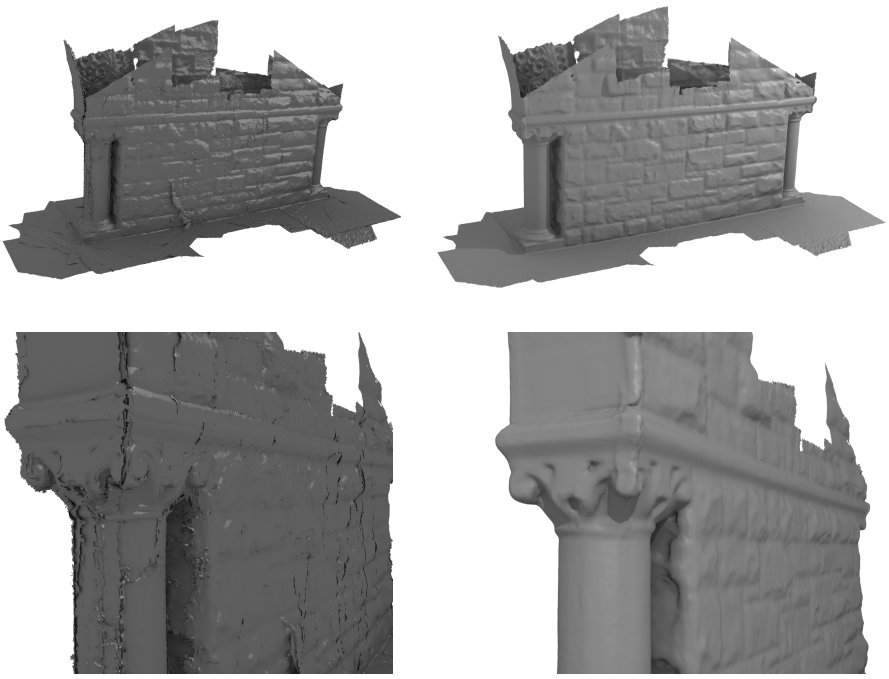


Figure 3.3.6: *Stonewall* sequence from [106]: Optimized subvolumes still exhibit surface deformations (left), while our blending approach (right) overcomes these issues.

i.e. where fewer frames have been fused, so that noise filtering is limited. To overcome these issues, a non-rigid pose estimation could be deployed in Eq. (3.3.5), though increasing the complexity of the problem. An alternative approach, however, would be the fusion of all the subvolumes into a global volume ψ_g. Indeed, as shown below, this is equivalent to the integration of the whole depth frame sequence according to the refined subvolumes' poses. Recalling Eq. (3.2.4) and (3.2.5), we define $\psi_g(\mathbf{u})$ for each valid $\mathbf{u} \in \mathbb{R}^3$ as follows

$$F_g(\mathbf{u}) := \begin{cases} \frac{\sum_j F_j\left(\mathrm{V}_j^{-1}\mathbf{u}\right) W_j\left(\mathrm{V}_j^{-1}\mathbf{u}\right)}{\sum_j W_j\left(\mathrm{V}_j^{-1}\mathbf{u}\right)} & \text{if } \sum_j W_j\left(\mathrm{V}_j^{-1}\mathbf{u}\right) > 0 \\ 0 & \text{ow} \end{cases}, \tag{3.3.8}$$

$$W_g(\mathbf{u}) := \sum_j W_j\left(\mathrm{V}_j^{-1}\mathbf{u}\right), \tag{3.3.9}$$

where F_j and W_j returns a trilinearly interpolated value. Such global volume should extend over all the subvolumes and every voxel is computed by traversing the whole collection. Indeed, even invalid regions are detected either because $\sum_j W_j\left(\mathrm{V}_j^{-1}\mathbf{u}\right) = 0$ in Eq. (3.3.8) and (3.3.9) or the voxel is outside all subvolumes' bounding boxes. In both cases, the presence of a high number of zero-weight voxels and poor overlapping can hinder performances. As shown in Fig. 3.3.5 this issue is strongly affected by the estimated poses. In order to make the required time more predictable and, in general, to reduce the computational effort, we instead address the final surface reconstruction by *blending* each subvolume together its neighbors. Given a subvolume ψ_j with overlapping subvolumes' indexes C_j, we assign to each voxel $\mathbf{u}$ inside ψ_j the values

$$F_j(\mathbf{u}) := \frac{F_j(\mathbf{u}) W_j(\mathbf{u}) + \sum_{h \in C_j} F_h\left(\mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{u}\right) W_h\left(\mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{u}\right)}{W_j(\mathbf{u}) + \sum_{h \in C_j} W_h\left(\mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{u}\right)}, \tag{3.3.10}$$

$$W_j(\mathbf{u}) := W_j(\mathbf{u}) + \sum_{h \in C_j} W_h\left(\mathrm{V}_h^{-1}\mathrm{V}_j\mathbf{u}\right). \tag{3.3.11}$$

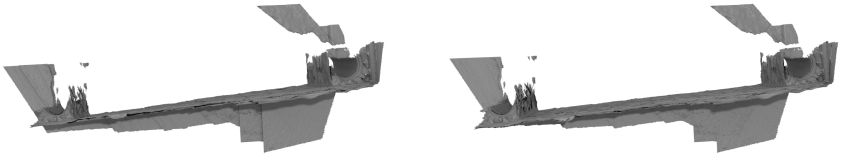
Unlike the estimation of a single global volume, now we can skip the computation if $\psi(\mathbf{u})$ has zero weight or a truncated distance value, *i.e*. it represents empty space. Though the same TSDF value may be estimated in different subvolumes at overlapping regions, in practice we have found that this approach is roughly five to ten times faster than global volume resampling. A qualitative comparison between the surfaces before and after blending is shown in Fig. 3.3.6.

3.4 Results

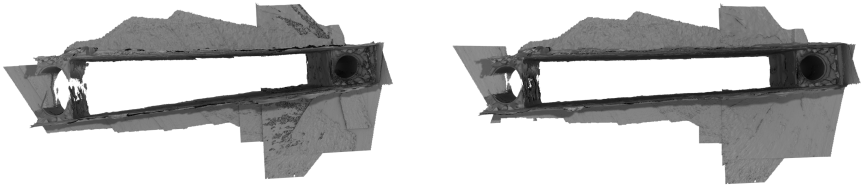
In the previous sections we have described our large scale surface reconstruction method based on subvolume optimization. We already showed in Fig. 3.3.3 the benefit of the erosion process for creating low-drift subvolumes covering a relatively large area. We wish to highlight here how our online surface alignment drastically reduces the drift error over large trajectories. Fig. 3.4.1, 3.4.2 and 3.4.3 compare the incremental solution provided by our method to the simple accumulation of subvolumes yielded by the camera tracker from, respectively, top and left viewpoints. Purposely, we rendered the scenes with a strong directional light source to enhance surface deformations and small inconsistencies, so that the superior mapping accuracy of our approach it is evident after a complete loop as well as after a small number of subvolumes.

As for evaluating the whole SLAM system, we ran several qualitative experiments on four sequences from the dataset introduced in [106], namely *stonewall*, *copyroom*, *lounge* and *burghers*, and three sequences acquired using our own Asus Xtion depth camera: *dark room*, *bookshop 1*, *bookshop 2*. Our subvolume-based approach has been compared to the reference moving volume algorithm described in Sec. 3.3 and to the results provided by Zhou and Koltun[106]. Although Zhou *et al*. [107] is a more recent work following [106], we note that [107] performs non-rigid surface deformation over a control lattice, while in this work we estimate only 6-DOF poses and address mesh artifacts through the

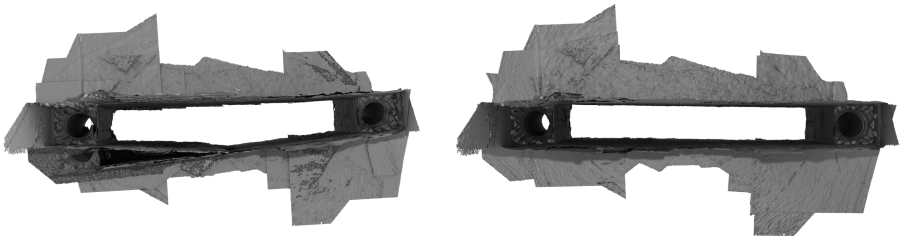
(a) After 5 subvolumes.



(b) After 20 subvolumes.

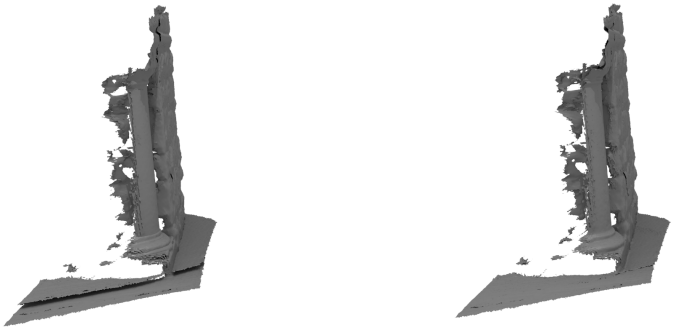


(c) After 36 subvolumes.

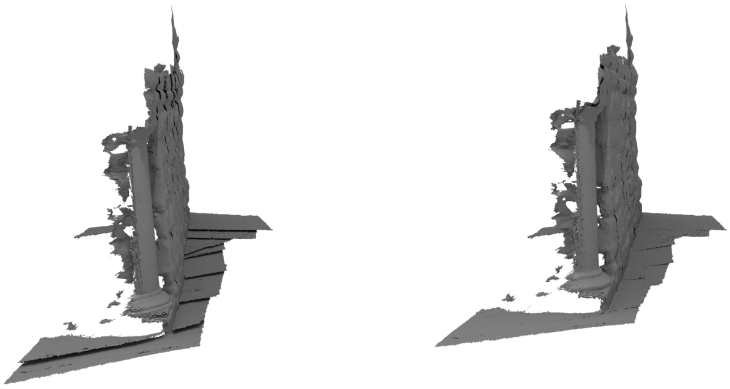


(d) After 55 subvolumes.

Figure 3.4.1: *Stonewall* sequence from [106], top view: online optimization of subvolumes' poses counteracts drift error. Left: without optimization. Right: with optimization. No volume blending applied.

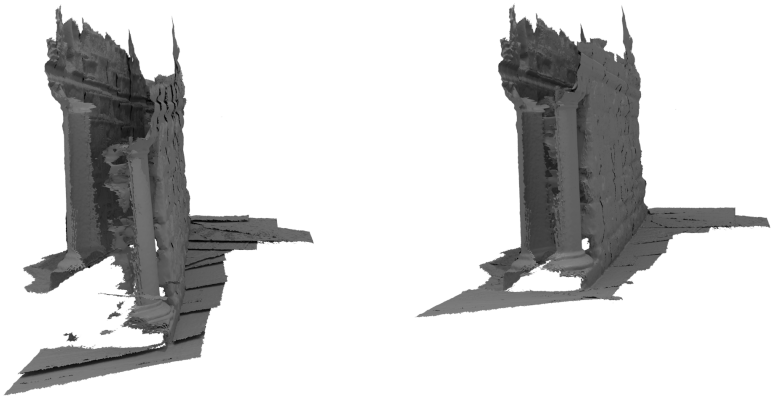


(a) After 5 subvolumes.

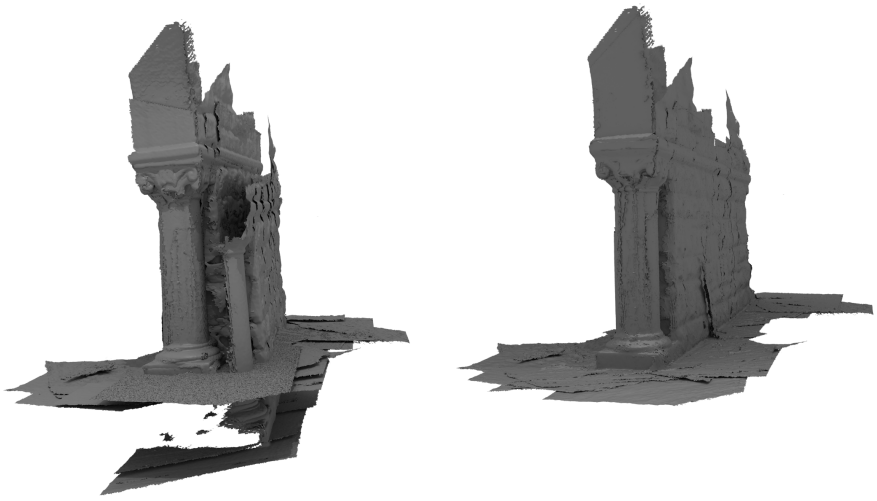


(b) After 20 subvolumes.

Figure 3.4.2: *Stonewall* sequence from [106], left-most column: online optimization of subvolumes' poses counteracts drift error. Left: without optimization. Right: with optimization. No volume blending applied.



(a) After 36 subvolumes.



(b) After 55 subvolumes.

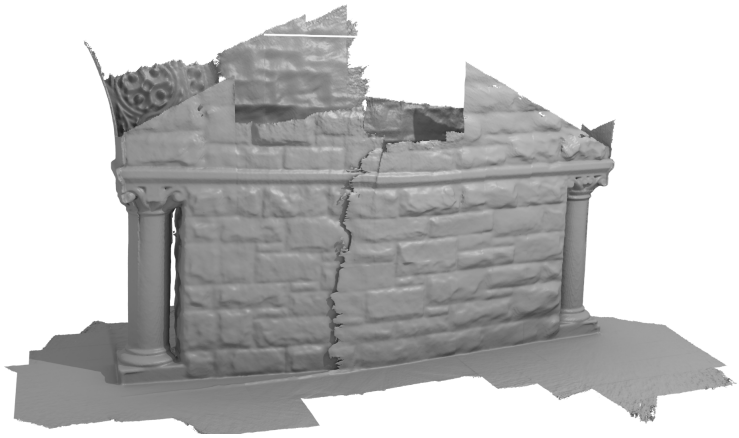
Figure 3.4.3: *Continued from Fig. 3.4.2*

Table 3.4.1: Number of frames and subvolumes spawned by our approach for the dataset used in our experiments.

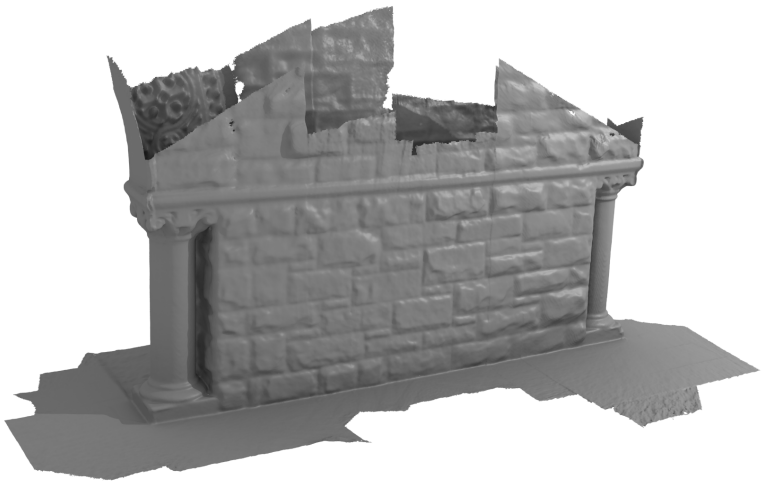
Sequence	Number of frames	Number of subvolumes
stonewall	2700	55
copyroom	5490	110
lounge	3000	61
burghers	11230	225
dark room	4841	97
bookshop 1	5000	101
bookshop 2	5700	115

final volume fusion, as in [106]. Number of frames as well as spawn subvolumes for each sequence are reported in Tab. 3.4.1.

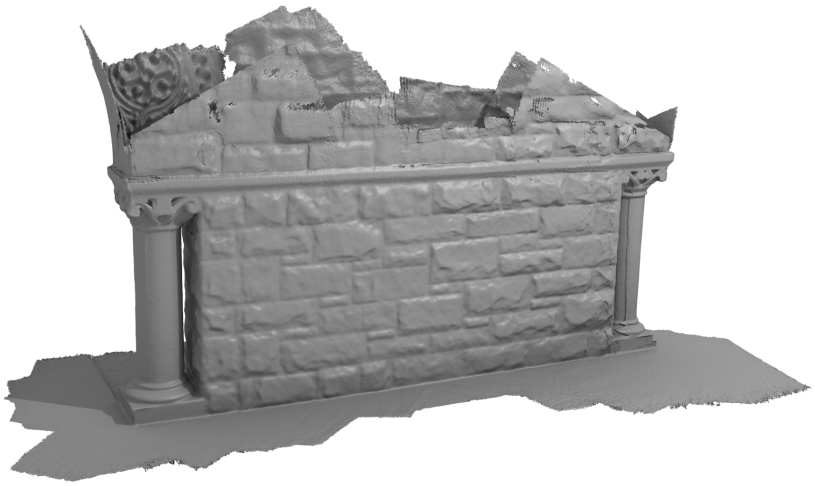
In Fig. 3.4.4 – 3.4.12 we compare the final reconstructions obtained through the reference moving volume method, our approach and Zhou and Koltun [106]. We used a TSDF volume on the GPU main memory with 512^3 voxels and a resolution of $0.96\,{}^{\text{cm}}/_{\text{voxel}}$ for moving volume and ours as well, while we have inferred a resolution of about $0.6\,{}^{\text{cm}}/_{\text{voxel}}$ from the meshes provided by the authors in [106]. Overall, the moving volume method gives the worst performance, while we usually get similar or better results than [106] with much less computational effort. In the *stonewall* sequence (Fig. 3.4.4, 3.4.5, 3.4.6) we have already pointed out the inconsistencies produced by the moving volume method, while [106] correctly aligns the surfaces. However, high frequency noise appears at loop closure (see Fig. 3.4.6) and even on the right-most column in Fig. 3.4.4, far from the ends of the loop. Conversely, our reconstruction smoothly removes artifacts. As for the *copyroom* sequence (Fig. 3.4.7, 3.4.8, 3.4.9), [106] is able to capture fine details, such as the cables on the copying machine in Fig. 3.4.8, but it fails at the opposite corner (Fig. 3.4.9). Although the *lounge* sequence (Fig. 3.4.10) is less challenging, the moving volume approach still returns an incomplete reconstruction, while [106] introduces hollows and rugged surfaces on



(a) Moving volume approach.



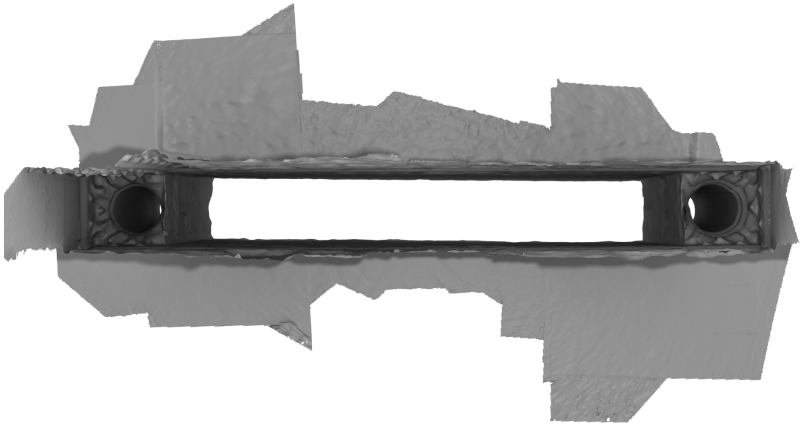
(b) Our approach.



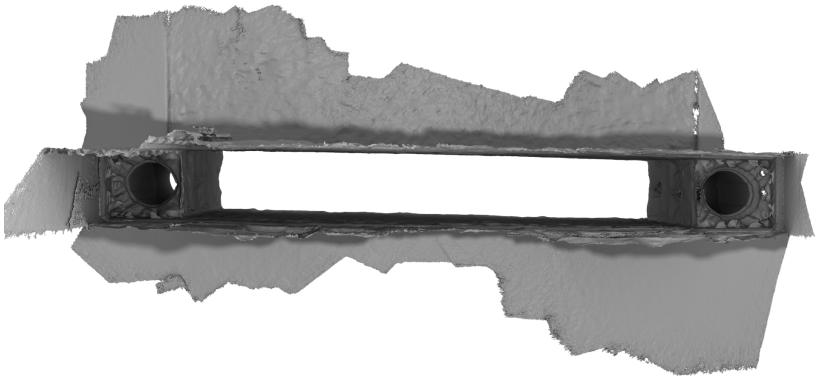
(c) Zhou and Koltun[106].

Figure 3.4.4: Final reconstruction of the *stonewall* sequence from [106], front view.

(a) Moving volume approach.

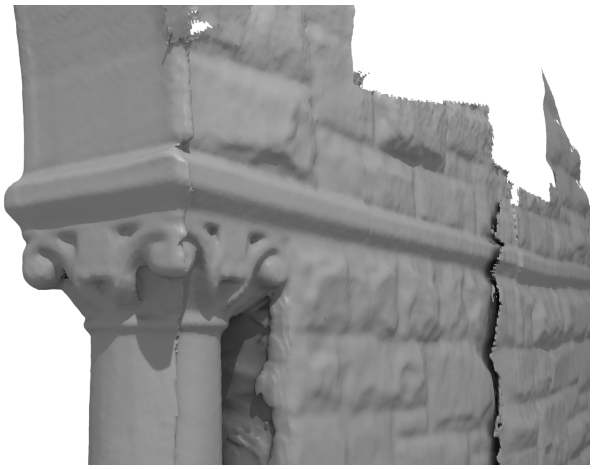


(b) Our approach.



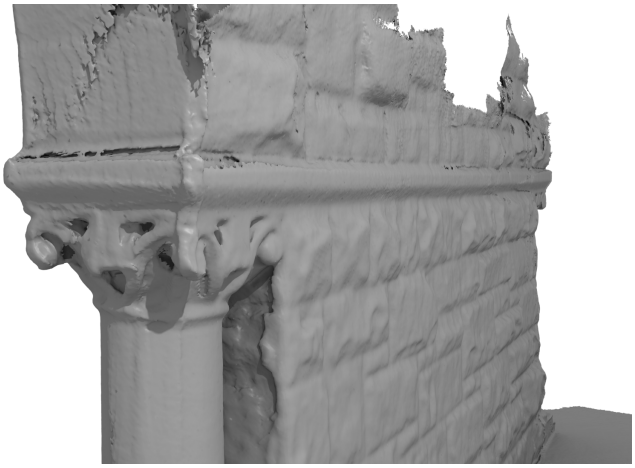
(c) Zhou and Koltun[106].

Figure 3.4.5: Final reconstruction of the *stonewall* sequence from [106], top view.



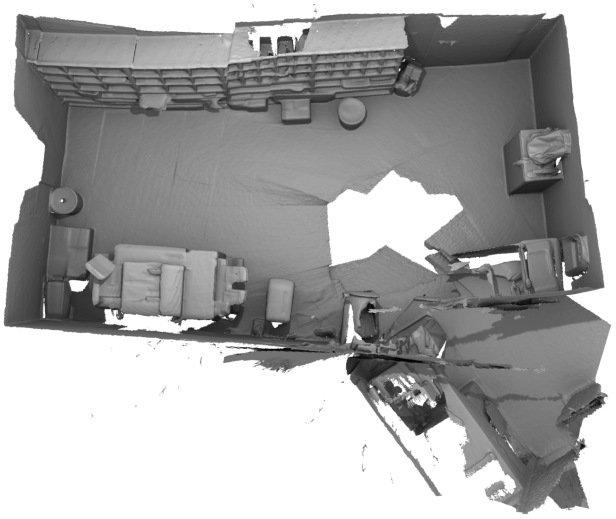
(a) Moving volume approach.

(b) Our approach.

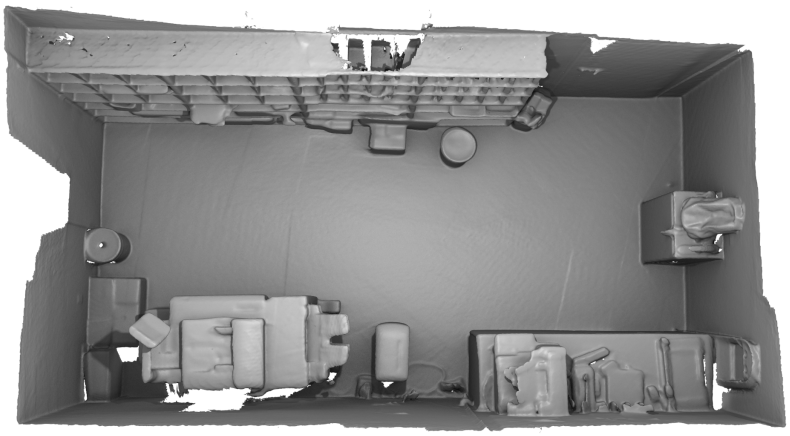


(c) Zhou and Koltun[106].

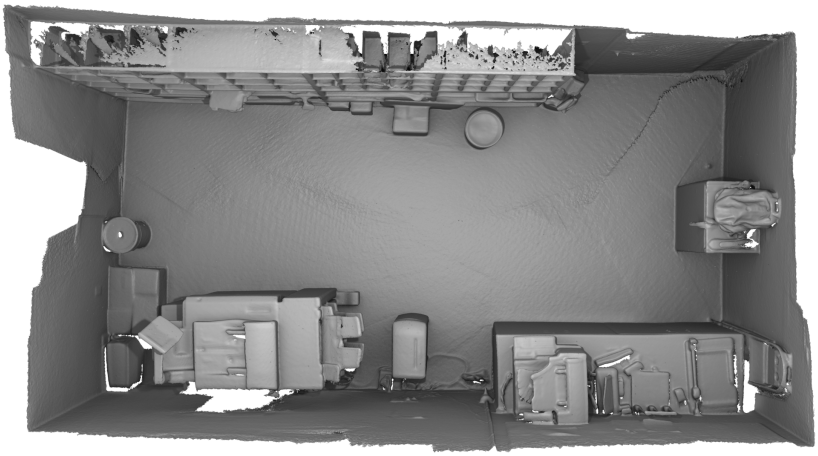
Figure 3.4.6: Final reconstruction of the *stonewall* sequence from [106], top view.



(a) Moving volume approach.



(b) Our approach.



(c) Zhou and Koltun[106].

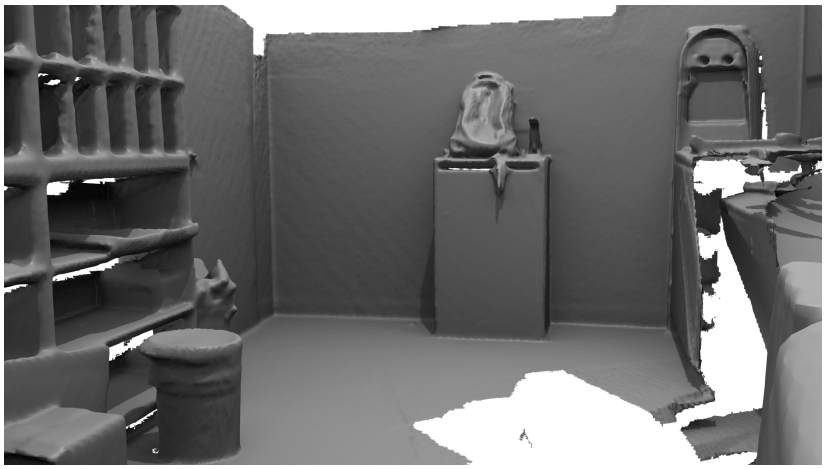
Figure 3.4.7: Final reconstruction of the *copyroom* sequence from [106], top view.

(a) Moving volume approach.

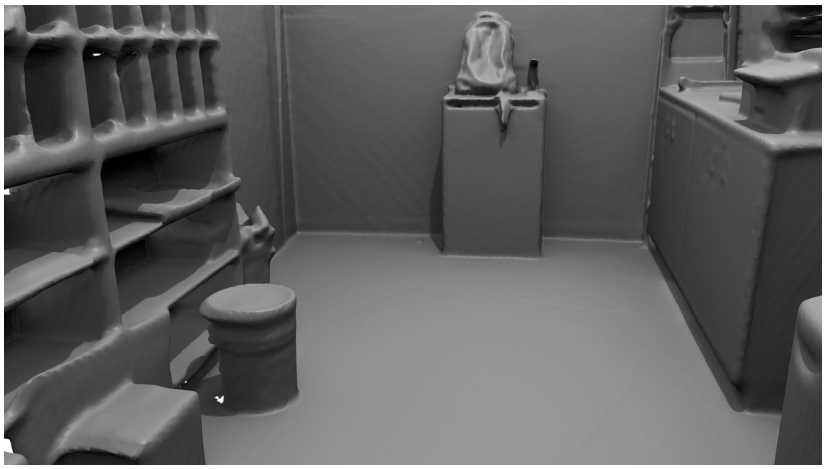
(b) Our approach.

(c) Zhou and Koltun[106].

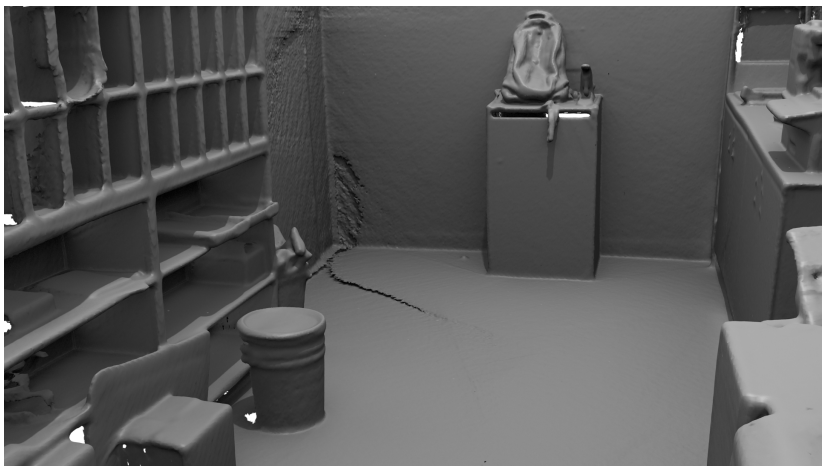
Figure 3.4.8: Final reconstruction of the *copyroom* sequence from [106], detail of the copying machine.



(a) Moving volume approach.

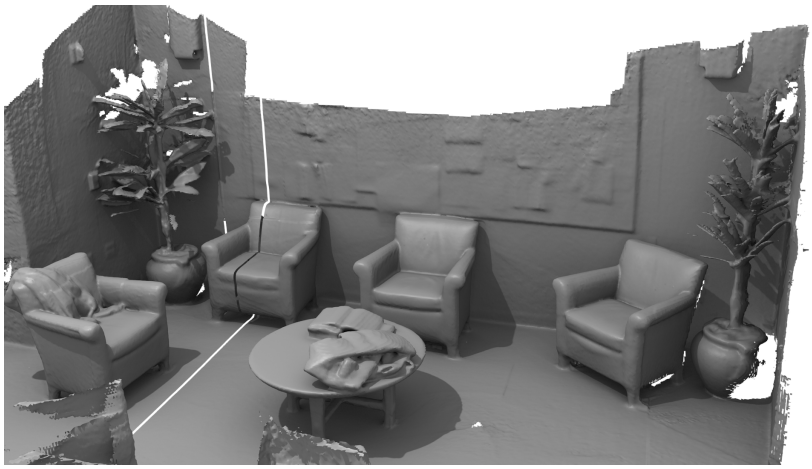


(b) Our approach.

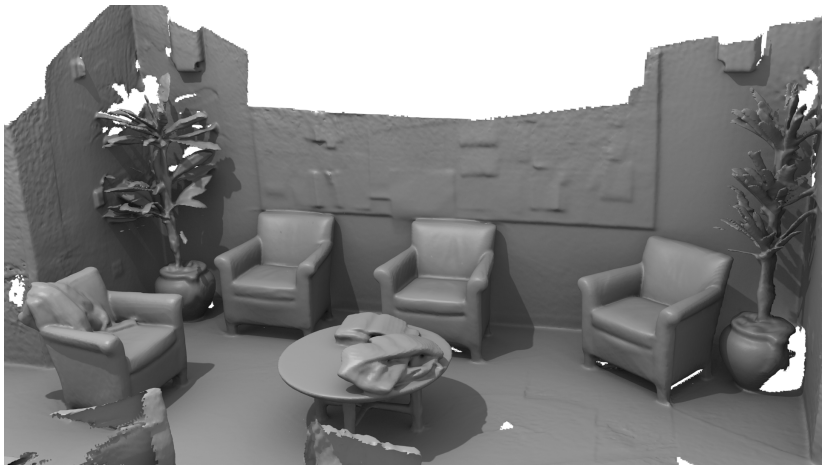


(c) Zhou and Koltun[106].

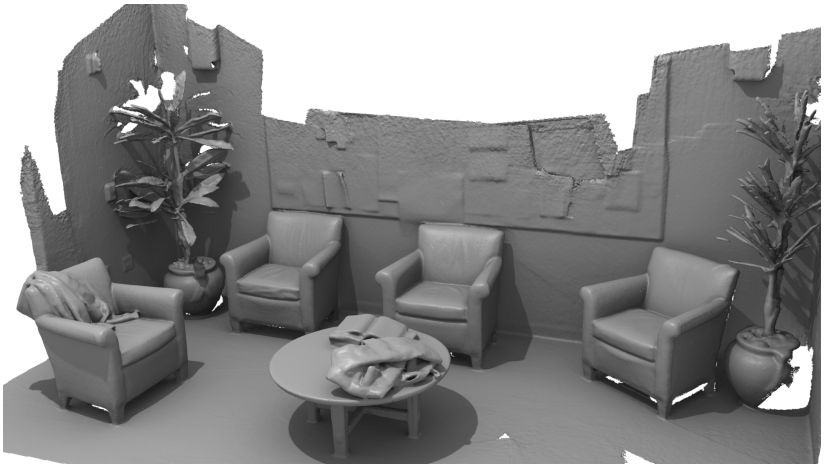
Figure 3.4.9: Final reconstruction of the *copyroom* sequence from [106], detail of the corner.



(a) Moving volume approach.

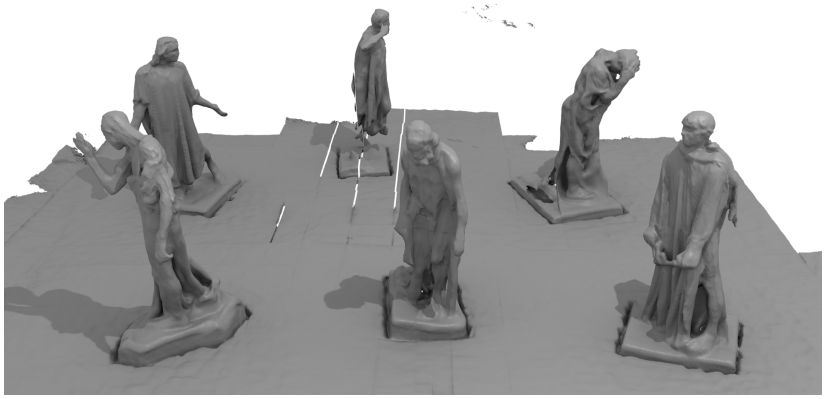


(b) Our approach.

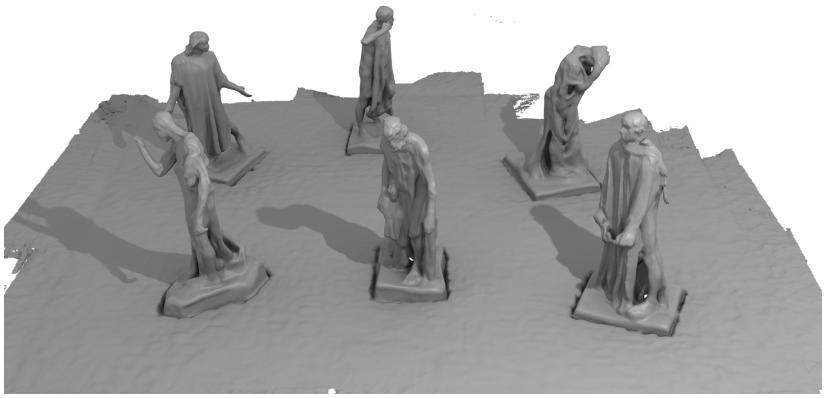


(c) Zhou and Koltun[106].

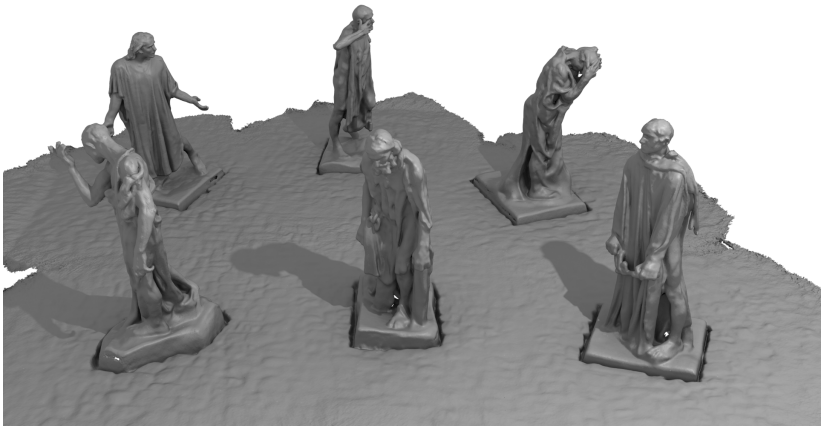
Figure 3.4.10: Final reconstruction of the *lounge* sequence from [106].



(a) Moving volume approach.

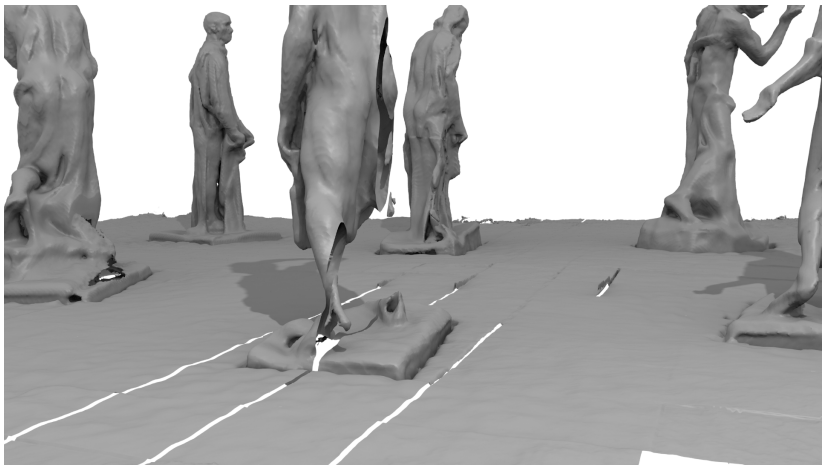


(b) Our approach.

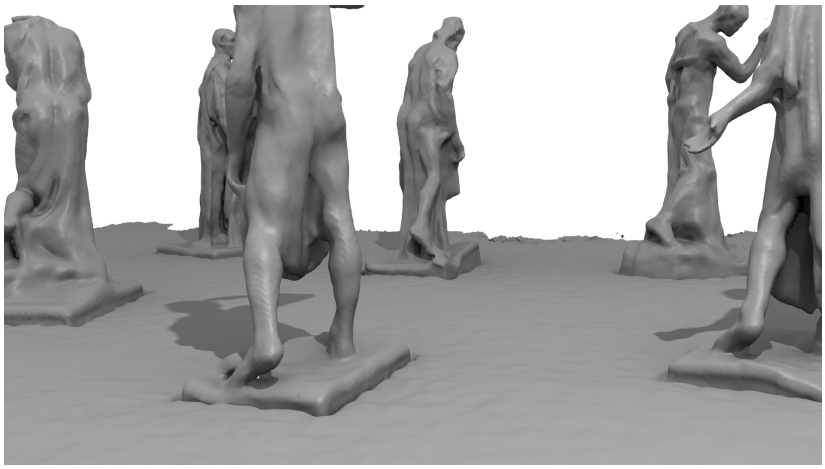


(c) Zhou and Koltun[106].

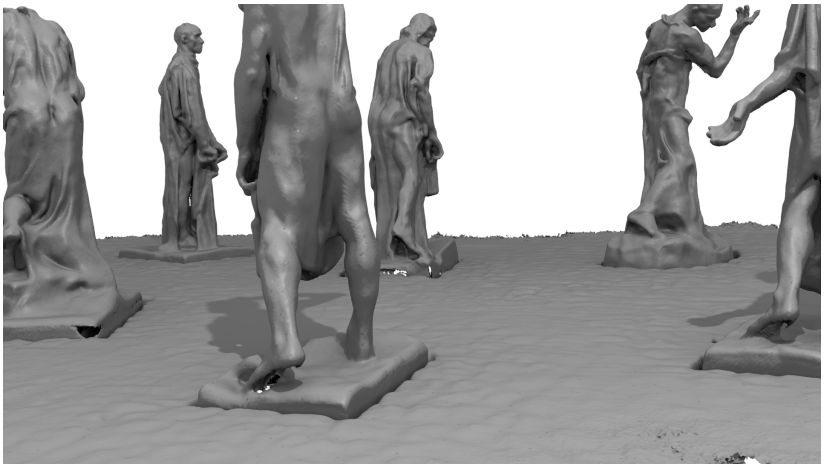
Figure 3.4.11: Final reconstruction of the *burghers* sequence from [106], front view.



(a) Moving volume approach.



(b) Our approach.



(c) Zhou and Koltun[106].

Figure 3.4.12: Final reconstruction of the *burghers* sequence from [106], rear view.

Table 3.4.2: Timing performance of our approach.

Sequence	Subvolume Optimization			Volume Blending
	Min	Max	Average	
Stonewall	0.154 s	30.136 s	5.388 s	3 min 3 s
Copyroom	0.085 s	35.297 s	11.238 s	10 min 30 s
Lounge	0.084 s	15.149 s	5.830 s	4 min 3 s
Burghers	0.047 s	5 min 4 s	1 min 30 s	22 min 27 s
Dark room	0.064 s	45.882 s	12.416 s	7 min 17 s

Table 3.4.3: Timing performance of [106]: *POI* is the detection of points of interest, *Reg*. is the two-pass registration, *Opt*. is the global registration, *Final Fusion* is the final model reconstruction by integration of all the depth frames. Note that *dark room* cannot be processed due to the lack of RGB data.

Sequence	Pose Optimization				Final Fusion
	POI	Reg.	Opt.	Total	
Stonewall	1 min	17 min	1 h 54 min	2 h 12 min	21 min
Copyroom	1 min	14 min	52 min	1 h 7 min	47 min
Lounge	1 min	12 min	16 min	29 min	40 min
Burghers	5 min	40 min	10 min	55 min	2 h 1 min
Dark room	—	—	—	—	—

the wall. Instead, on the *burghers* sequence (Fig. 3.4.11, 3.4.12) [106] gives the best reconstruction result by enhancing details on faces and bodies, which are filtered out by our larger voxels. Again, the moving volume method fails due to fusion of drifted apart depth images.

Additional results provided by our approach are shown in Fig. 3.4.13 and 3.4.14, the former showing two large explorations, the latter a reconstruction in complete darkness. Indeed, nowadays full-fledged RGB-D SLAM systems [106, 107, 100, 30, 87] mandate color data to adjust camera path and globally align surfaces, while ours seamlessly reduces drift error using only depth measurements and TSDF subvolumes.

Finally, we have investigated on the timing performance of our opti-

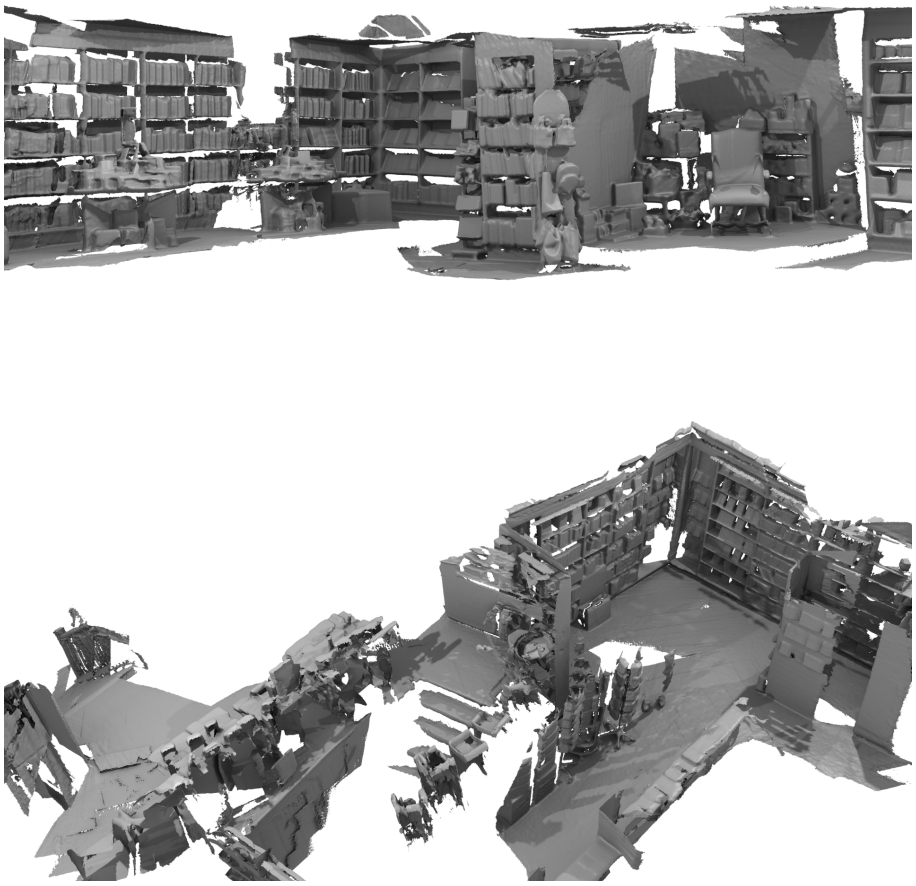


Figure 3.4.13: Final reconstruction of the *bookshop 1* (top) and *bookshop 2* (bottom) sequences.

Figure 3.4.14: Final reconstruction of the *dark room* sequence. Due to the lack of RGB-D data, existing RGB-D SLAM system, including [106, 107, 30], would have failed.

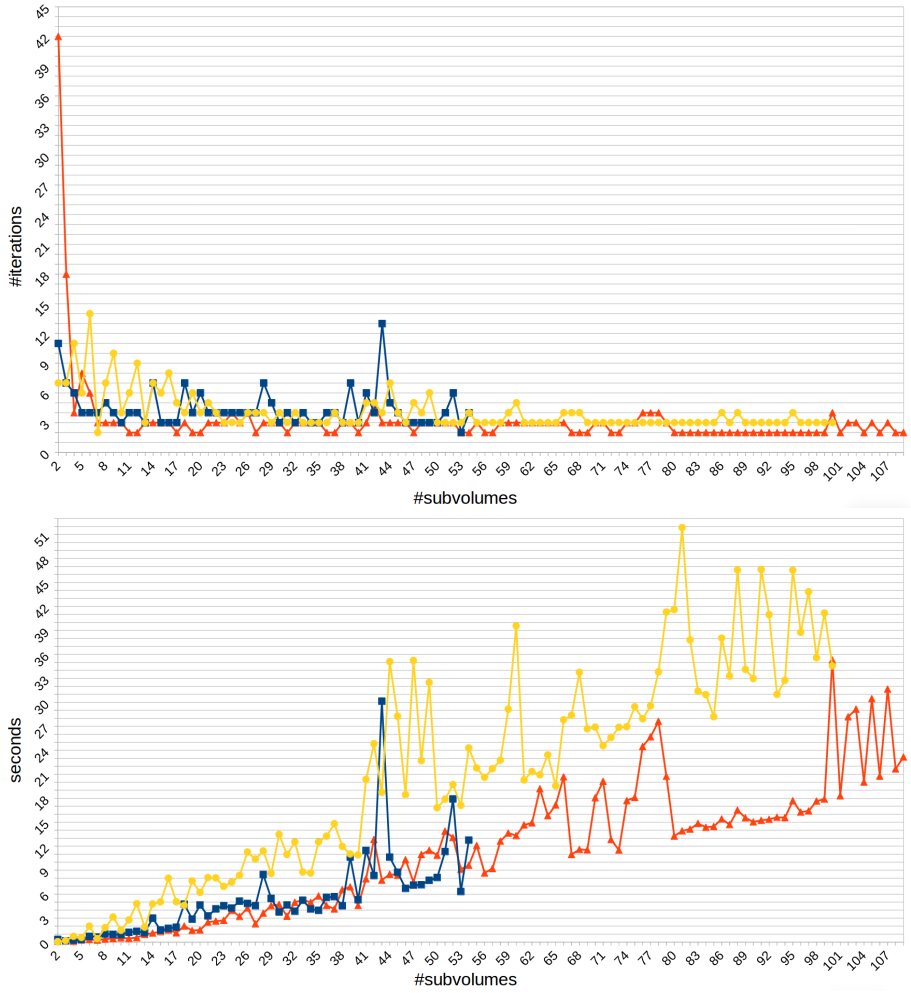


Figure 3.4.15: Number of iterations (top) and time (bottom) spent by subvolume optimization in *stonewall* (blue squares), *copyroom* (red triangles) and *bookshop 1* (yellow circles) sequences for increasing number of subvolumes.

mization and volume blending steps. Fig. 3.4.15 shows the number of iterations and the time spent after every new subvolume spawning in 3 different typical scenarios: the camera moving around a large object (*stonewall*), a loop in a medium-size room (*copyroom*), an exploratory sequence (*bookshop 1*). Since we actively reduce drifting, the subvolumes are usually placed near the global optimum, so that the number of iterations is usually low. Conversely, the time required by the subvolume optimization increases somehow linearly with the number of subvolumes, due to the presence of unknowns and constraints in the minimization problem. Nevertheless, as reported in Tab. 3.4.2, the subvolume optimization step usually requires much less than a minute, while [106] may need hours of processing (see Tab. 3.4.3). Also, Tab. 3.4.2 and 3.4.3 show how our proposed volume blending runs from 4 to 7 times faster than the standard depth fusion algorithm deployed by [106].

Chapter 4

Semantic Bundle Adjustment

In the previous chapters we have described two different approaches for solving the RGB-D SLAM problem on mobile platforms (see Chap. 2) and on powerful high-end desktop machines as well (see Chap. 3). Though they both address the drift error by leveraging pose graph optimization, no information beyond geometric modeling is inferred from the scene and introduced into the problem statement, such as, *e.g.*, the kind of objects present in the scene as well as their configuration.

Indeed, recognizing object instances and repeated structures across multiple frames may introduce useful constraints into the pose graph, while a, possibly partial, reconstruction of the environment might help the object detection task. This intuition is illustrated in Fig. 4.0.1, where a car is seen by a moving camera. The top image represents a typical pose graph, where unknown camera poses T_0, T_1 and T_2 are linked by means of cost terms, *e.g.* pose-pose error functions such as Eq. (3.3.7), that ignore any semantic information. However, a set of calibrated views might help identifying object instances by matching features from different viewpoints to a learned model, as depicted in the middle image in Fig. 4.0.1 where we repeatedly collect cues from different frames to infer the presence of the car. On one hand, we can estimate the pose of the detected object by, *e.g.*, minimizing the distance between matching feature points. On the other hand, we may constrain the camera path

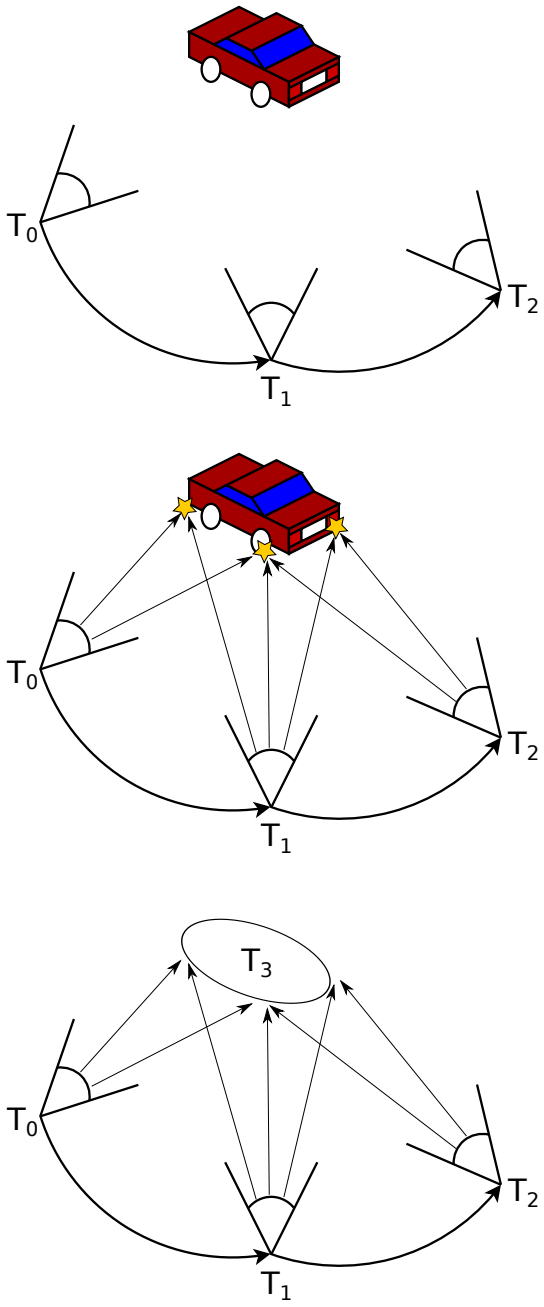


Figure 4.0.1: Classical approaches to the SLAM problem constrain camera poses without any assumption about the semantic of the scene under exploration (top). Using calibrated views may improve object feature matching across multiple frames (middle), which provides additional information that we exploit by jointly estimating camera and object poses (bottom).

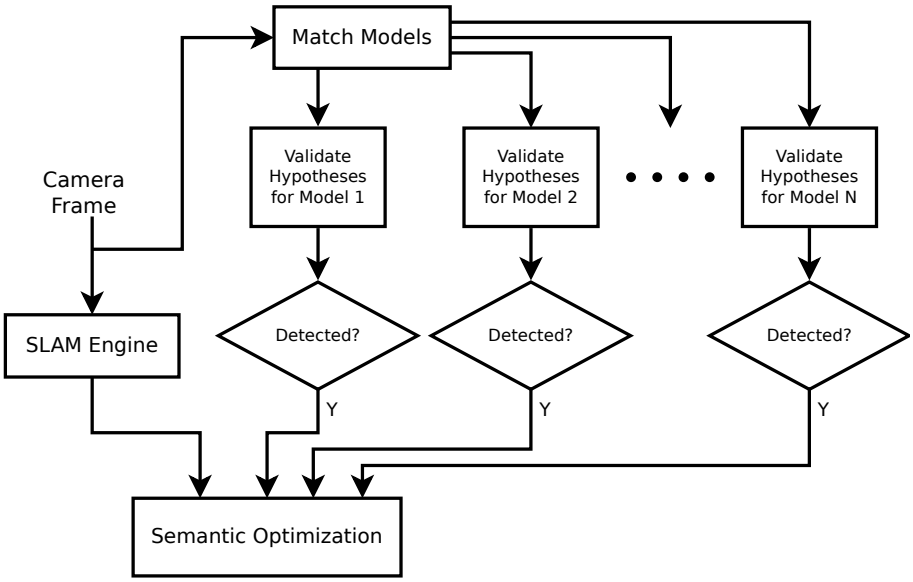


Figure 4.0.2: A schematic view of our joint detection, tracking and mapping approach.

to be consistent with those matches by refining estimated sensor poses accordingly. Hence, our proposal is to introduce the feature matches into the SLAM problem and jointly optimize camera and object poses (bottom image in Fig. 4.0.1).

The work-flow of our framework is depicted in Fig. 4.0.2. Visual features are extracted from camera frames and matched to a model database. As we will see, our system can work with both 2D [57, 7, 73] and 3D features [39, 93, 94] and any visual sensing system, such as monocular cameras, stereo rigs or RGB-D sensors. Then, new cues are separately validated for each matched objects by checking for consistencies with the current reconstruction. Previous matches contribute too and a careful cleaning procedure is deployed to remove unreliable connections. When enough hypotheses have been gathered, the object is detected and added to the SLAM problem as a 6-DOF unknown pose. Finally, accurate object localization and consistent camera path refinement is carried out by a global semantic optimization.

The chapter is organized as follows. The next section briefly discusses

similar work and state-of-art approaches for joint object detection and SLAM. Then, in Sec. 4.2 we will describe our approach [110] and, in Sec. 4.3, a possible integration with the KinectFusion framework [109]. Results are reported in Sec. 4.4.

4.1 Scene Understanding And Mapping

Integrating SLAM and object detection to achieve semantic reconstruction has been investigated in many different scenarios. However, most works do not jointly solve for the two facets of the problem, but rather improve only one of the two tasks. Thus, while we enforce consistency across multiple views, [28, 36, 54] perform single-view object detection. Moreover, [28] is more concerned with room-level description and does not accurately localize objects. Conversely, Meger *et al*. [60] identify objects within a map built with FastSLAM [62], but the two processes are separated. Cornelis *et al*. [19] first introduce the notion of *cognitive loop* indicating the tight relationship between mapping and object detection. However, they restrict to the specific case of urban environment reconstruction by means of Structure-from-Motion (SfM) [34] and the detection is limited to cars and pedestrians. Also, strong assumptions are made, *e.g*. by constraining cars onto the known ground surface, and, more importantly, unlike our approach, detected objects are not used to improve reconstruction results.

A step in the direction of a synergistic integration is made by the work of Castle *et al*. [14] where a monocular camera is tracked by an Extended Kalman Filter (EKF) [23], while SIFT features [57] are extracted from acquired images to detect planar objects. Then, matched points are inserted into the SLAM map by augmenting the EKF state. Civera *et al*. [17] extend this approach to non-planar objects, though the detection pipeline still deploys single-view feature matching and, therefore, is not fully integrated into the SLAM framework.

Bao *et al*. [6, 5, 4] introduced the Semantic Structure-from-Motion paradigm to address joint object recognition and camera pose estimation. However, it differs from our work in several aspects. First, they deal with the complete image sequence at once, rather than incremental reconstruction, so that all data are always available, while we tackle the problem of adjusting a previous solution as soon as new information arrives. Also, they cast hypotheses by means of an independent detection algorithm, while we work with raw feature matching measurements and develop an integrated pipeline to establish upon object presence. Finally, we aim at estimating full 6-DOF pose for each object instance, rather than category level recognition and image plane localization.

Incremental adjustment and full 3D localization is performed, instead, by Salas-Moreno *et al*. [76] in their SLAM++ framework. They adopted the object detection approach proposed by Drost *et al*. [25] and, once an object has been found, check for consistency with the current frame only by ICP alignment between the learned model and the acquired depth map. Conversely, we will show how to exploit the SLAM graph for object detection purpose. Also, though they jointly optimize camera and object poses as we do, they link graph vertexes by means of pose-pose error constraints, while we never marginalize feature matches, thereby casting a novel *semantic bundle adjustment* problem.

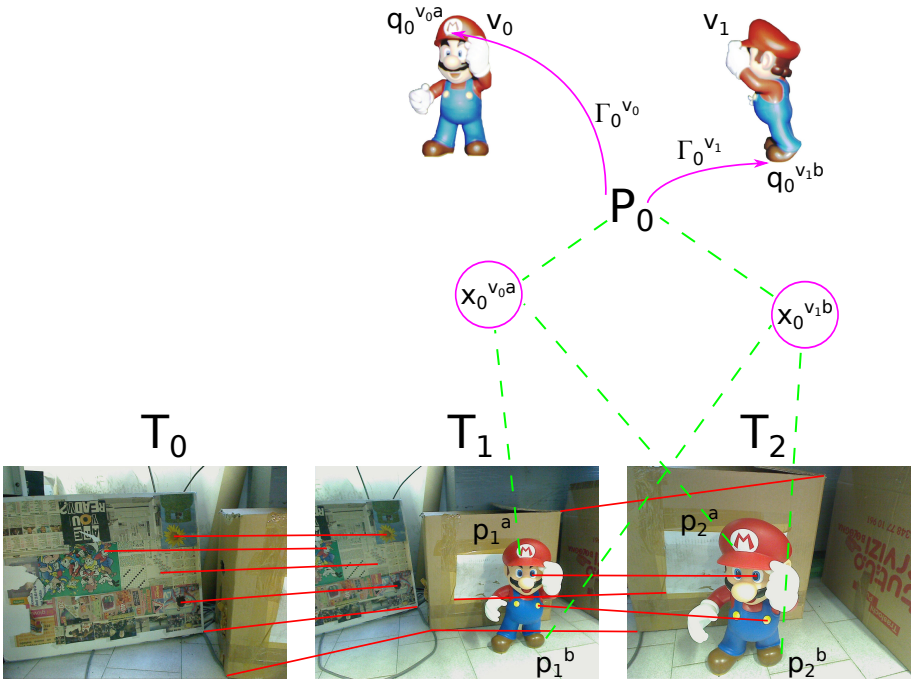
4.2 Joint Detection, Tracking And Mapping

In this section we will describe a unified approach for object detection and mapping. Throughout the section we will make use of the notation introduced in Sec. 2.3 and we will generally refer to an unspecified input camera frame, comprising, *e.g*., one color image and, possibly, a depth or disparity map. Indeed, our method applies to both monocular camera settings and 3D sensors. Also, our mathematical framework does not force to adopt any particular SLAM engine, so we will simply assume to have frame-to-frame constraints available, *e.g*. feature matches such

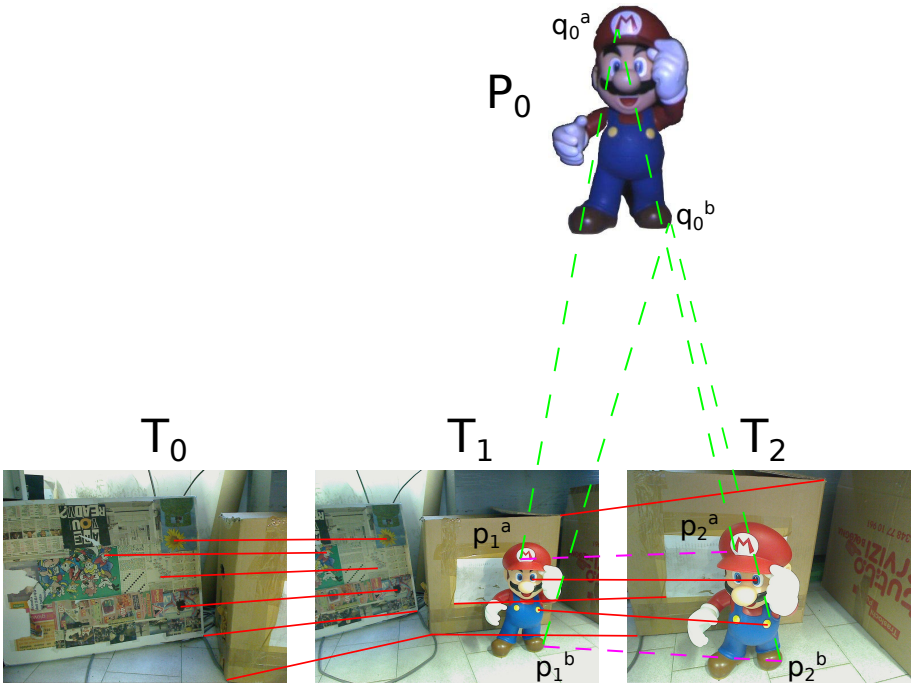
as in Eq. (2.3.7). We will denote with T_i the unknown 6-DOF pose of the i^{th} camera frame and with P_o the unknown 3D transformation which maps the o^{th} object model into the scene reference frame. The model database is built as a collection of features extracted either from full 3D reconstructions or sets of calibrated images. In the former representation, 3D keypoint detectors, *e.g*. [105], and feature descriptors, *e.g*. [39, 93, 94], are extracted, while, in the latter, equivalent 2D algorithms [57, 71, 7, 73] are deployed. Then, descriptors are indexed, *e.g*. using a single KD-Tree structure as described in Sec. 2.3.2, and keypoints are saved, either as points on the full 3D model or pixel locations on an image of the calibrated set. This way, given a match between a camera frame and an object point feature, we are able to constrain the corresponding camera and object poses by minimizing either the 3D point distance or the reprojection error in the image plane. In the next sections we will show how this intuition leads to robust object detection and semantic optimization.

4.2.1 The Validation Graph

Fig. 4.2.1 visualizes a toy example for the 2D and 3D case. While the SLAM engine tracks points across the three frames (red lines), feature descriptors extracted from camera images are matched to the model database. At this stage, high recall is preferable, since we will filter out outliers afterward. As shown in Fig. 4.0.2, matches are grouped by object instances and each set is separately validated for consistency with the current reconstruction of the environment. Purposely, a *validation* graph is deployed including both last and all previous feature matches. As for the 2D embodiment, we introduce a new 6-DOF unknown for the object pose and a 3D landmark location for each 2D object feature. In Fig. 4.2.1a, P_0 is the object pose and $\mathbf{x}_0^{ij}$ is the 3D location of the j^{th} feature on the i^{th} view of object #0, while T_k is the k^{th} camera pose and $\mathbf{p}_k^j$ is its j^{th} feature point (for simplicity, we have used the same j value for corresponding features). Therefore, considering the landmark



(a)



(b)

Figure 4.2.1: A toy example illustrating the validation graph for 2D (a) and 3D (b) SLAM problems. See Sec. 4.2 for details.

$\mathbf{x}_0^{v_1 b}$ in Fig. 4.2.1a, the associated cost would be (green dashed lines in Fig. 4.2.1a):

$$\left\|\mathbf{q}_0^{v_1 b} - \Gamma_0^{v_1} \mathrm{P}_0^{-1} \mathbf{x}_0^{v_1 b}\right\|^2 + s_1^b \left\|\mathbf{p}_1^b - \mathrm{K}\mathrm{T}_1^{-1} \mathbf{x}_0^{v_1 b}\right\|^2 + s_2^b \left\|\mathbf{p}_2^b - \mathrm{K}\mathrm{T}_2^{-1} \mathbf{x}_0^{v_1 b}\right\|^2, \quad (4.2.1)$$

where $\mathbf{q}_0^{v_1 b}$ is the b^{th} 2D feature point on view v_1 of object 0 matching $\mathbf{p}_1^b$ and $\mathbf{p}_2^b$ with probability s_1^b and s_2^b respectively, $\Gamma_0^{v_1}$ projects 3D points in object #0's reference frame onto view v_1's image plane (magenta arrows in Fig. 4.2.1a), K is the camera projection matrix (see Eq. (2.3.2)). Homogeneous operators and their inverses have been omitted for better readability. Conversely, when 3D measurements are associated to each feature point, no landmark is needed and point-point constraints can be used (green dashed lines in Fig. 4.2.1b, cfr. Eq. (2.3.7)). Also, different frames matching the same object features can be further constrained considering induced *virtual* matches (dashed magenta lines in Fig. 4.2.1b). These stem from the intuition that if we know that "*feature j matches feature i*" (m_{ij}) with probability s_{ij} and "*feature k matches feature i*" (m_{ik}) with probability s_{ik}, then we wish to know the probability of the event "*feature k matches feature j*" (m_{jk}). Accordingly, if m_{ij} and m_{ik} are independent and

$$\Pr\left(m_{jk} \middle| m_{ij}, m_{ik}\right) = \begin{cases} 1 & \text{if } m_{ij} = \text{TRUE} \wedge m_{ik} = \text{TRUE} \\ 0 & \text{otherwise} \end{cases}, \quad (4.2.2)$$

then $\Pr\left(m_{jk}\right) = s_{ij} s_{ik}$. Therefore, considering the 3D feature point $\mathbf{q}_0^a$ in Fig. 4.2.1b, the associated cost would be:

$$s_1^a \left\|\mathbf{p}_1^a - \mathrm{T}_1^{-1} \mathrm{P}_0 \mathbf{q}_0^a\right\|^2 + s_2^a \left\|\mathbf{p}_2^a - \mathrm{T}_2^{-1} \mathrm{P}_0 \mathbf{q}_0^a\right\|^2 + s_1^a s_2^a \left\|\mathbf{p}_2^a - \mathrm{T}_2^{-1} \mathrm{T}_1 \mathbf{p}_1^a\right\|^2. \quad (4.2.3)$$

Finally, we add to the validation graph any relevant frame-to-frame constraints from the SLAM engine and the first frame matching the object is linked to the previous one for increased robustness.

4.2.2 Object Detection

The validation graph is updated and the associated cost function optimized every time new feature matches are found. The minimization follows the Levenberg-Marquardt [53, 59, 56] method (see Sec. 2.3.3) provided by the G2O library [49]. Then, we compare the result with the last global weighted mean residual $\bar{\rho}$ (see Sec. 4.2.3) to decide upon reliability of every single match. More precisely, we go through all the semantic cost terms related to visual features from the last camera frame and compare their final error with $\bar{\rho}$. For the 2D case, we consider all the frame-to-landmark edges and remove the term from the validation graph if

$$\left\| \mathbf{p}_i^a - \mathrm{K}\mathrm{T}_i^{-1}\mathbf{x}_j^{v_u a} \right\|^2 \geq \alpha\bar{\rho}, \tag{4.2.4}$$

where $\alpha \in \mathbb{R}^3$ is a given parameter. This operation may leave a landmark attached only to its object pose. In that case we completely remove the landmark unknown from the problem. Similarly, in the 3D formulation we take into account all the frame-to-object edges and remove the cost term if

$$\left\| \mathbf{p}_i^a - \mathrm{T}_i^{-1}\mathrm{P}_j\mathbf{q}_j^a \right\|^2 \geq \alpha\bar{\rho}. \tag{4.2.5}$$

The inferred virtual edges are then removed if one of the two related matches is deleted.

This first cleaning procedure, driven by the expected SLAM reconstruction error, is applied to the cost term from the last object matching stage. If the remaining edges are below a threshold, such as, *e.g.*, 3, we treat them as noise and ignore the result. Otherwise, the validation graph is optimized again to refine the estimates and a new cleaning process is performed on the whole edge set. Then, we count the number of se-

mantic edges E_s and decide upon object detection by comparing E_s to two given thresholds ε_f, ε_t with $\varepsilon_f < \varepsilon_t$:

$E_s < \varepsilon_f$ the detection is not confirmed; the validation graph is destroyed and the object is removed from the SLAM problem, if present (see Sec. 4.2.3);

$\varepsilon_f \leq E_s < \varepsilon_t$ the result is unclear; we save the validation graph waiting for more cues, but the object is removed from the SLAM problem, if present (see Sec. 4.2.3);

$E_s \geq \varepsilon_t$ the object has been detected; the whole validation graph is included into the SLAM problem (see Sec. 4.2.3).

Since the previous cleaning procedures leave only those edges that turn out consistent with the current reconstruction, the two threshold ε_f and ε_t are not critical and we set them to, respectively, 3 and 10 in our experiments. Indeed, higher values for ε_t could hinder the detection of objects in very cluttered scenes, while the difference $\Delta_\varepsilon = \varepsilon_t - \varepsilon_f$ is related to the robustness of the detection pipeline: the larger Δ_ε is, the more feature matches, possibly from many different point of views, are needed to validate a detection, thus reducing the propagation of false positives to the SLAM graph (see Sec. 4.2.3). Finally, we note how our two-step cleaning process is able to seamlessly recognize and delete false matches introduced by previous frames as soon as correct information are fed to the pipeline. We will show an interesting example of this behavior in Sec. 4.4.

4.2.3 Semantic Optimization

In the previous section we have described the object detection algorithm, which is separately deployed for every possible instance. To this end, we have introduced a validation graph which comprises the subset of the whole SLAM graph linked to the object vertex through the

matched visual features. In order to correctly inject into the whole map the information about detected objects, we perform a global semantic optimization including all the validation graphs of such objects and all the constraints from the SLAM engine. Accordingly, we achieve joint estimation of camera and object poses.

Once the error has been spread across the graph, we compute the global weighted mean residual as

$$\bar{\rho} = \frac{\sum_i w_i \left\| \mathbf{e}_i \right\|^2}{\sum_i w_i}, \tag{4.2.6}$$

where w_i weights the confidence of the constraint $\mathbf{e}_i$, *e.g.* the match reliability s_1^b and s_2^b in Eq. (4.2.1) or the weight w_i in Eq. (2.3.6). This value can be interpreted as the expected reconstruction error and, therefore, is our reference for validating a possible insertion of an object in the map. Indeed, under a probabilistic interpretation, the SLAM problem in Eq. (2.3.8) can be written as a Maximum-A-Posteriori (MAP) estimation assuming a Gaussian measurement model:

$$\begin{aligned} Y = \arg\max_Y P(Y|Z) &= \arg\max_Y \prod_i P(Y_i|Z_i) \\ &= \arg\max_Y \prod_i \exp^{-\frac{1}{2} w_i \|\mathbf{e}_i\|^2}, \end{aligned} \tag{4.2.7}$$

where Y is the set of unknowns, *i.e.* poses and, possibly, landmarks, while Z is the set of measurements, *e.g.* feature matches. It is easy to see that Eq. (2.3.8) is derived from Eq. (4.2.7) by extracting the negative logarithm, which is a monotonic function, and converting to a minimization problem. Also, by means of Eq. (2.3.8) we can interpret the validation graph as the estimation of the most probable configuration assuming a correct detection of the object. This way, we compare different possible solutions without explicitly running an expensive and costly full multi-modal estimation as proposed by Bao *et al.* [6].

4.3 Semantic KinectFusion

In the previous section we have presented our semantic bundle adjustment framework [110] for joint detection, tracking and mapping. The approach can be easily integrated into many existing SLAM system, therefore we purposely assumed only to work with general frame-to-frame constraints. On one hand, using a naïve SLAM engine we can better highlights the improvements yielded by our enhanced semantic optimization, as we will show in Sec. 4.4. On the other hand, however, we are also interested at the mutual benefit achievable by combining the integrated object detection pipeline with state-of-art SLAM algorithms. Accordingly, in this section we will investigate how to extend the popular KinectFusion system [64, 38] so as to support our SLAM-driven object detection method, thus paving the way towards a possible future semantic KinectFusion framework [109].

We already described the KinectFusion camera tracking algorithm in Sec. 3.2. However, this method is conceived for surface reconstruction and has no notion of pose graph nor frame-to-frame matches as required by semantic bundle adjustment. Conversely, other approaches, such as SlamDunk (see Chap. 2), suggest a straightforward extension by looking for objects in keyframe images, whose poses are estimated within a pose graph representation. Unfortunately, KinectFusion-based works [72, 104, 15, 66, 100] address different issues, *e.g.* data compression or unbounded reconstruction, so that we cannot simply push semantic optimization on top. The large scale approach presented in Chap. 3 might be a more favorable environment, where, *e.g.*, 3D features could be extracted from subvolumes' meshes and matched to the object database, while subvolume and object poses are jointly estimated in a unified semantic graph. Nevertheless, here we prefer to pursue a different strategy which tries to introduce keyframe-based optimization into a KinectFusion system. In this way, we are able to seamlessly add the semantic framework and, also, to investigate a different approach to large scale surface reconstruction.

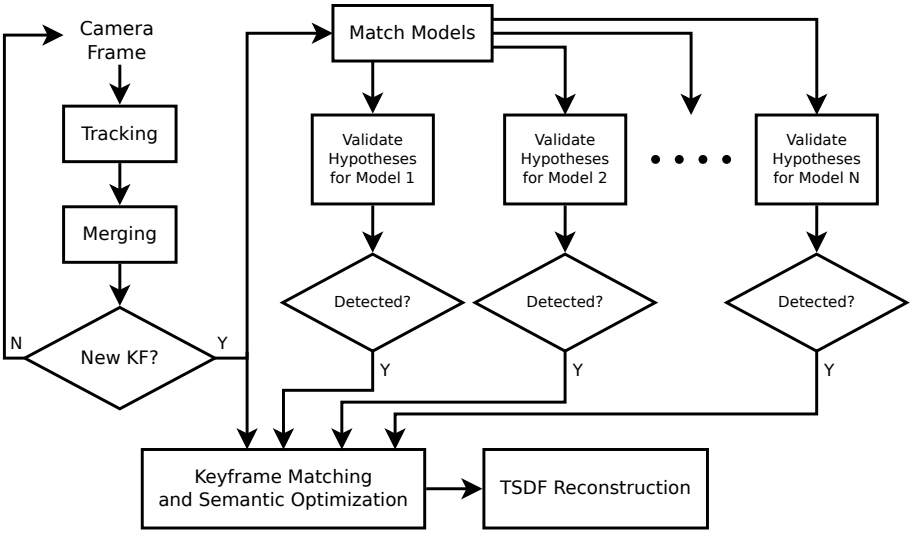


Figure 4.3.1: The generic "SLAM Engine" and "Semantic Optimization" blocks in Fig. 4.0.2 have been adapted so as to integrate the KinectFusion camera tracking system.

In Fig. 4.3.1 we have expanded the flowchart in Fig. 4.0.2 to our semantic KinectFusion pipeline. The "SLAM Engine" block has been replaced by a KinectFusion camera tracker (see Sec. 3.2) with a new check for keyframe detection after the fusion of each new depth image. Then, visual features are extracted and matched from RGB-D keyframes following the algorithm described in Sec. 4.2 and a semantic optimization is run over the pose graph. Finally, the TSDF volume is shifted and reconstructed from the optimized keyframes.

Keyframes are selected by spatial sampling of the camera path, that is a frame is saved as keyframe when the overlap with the current map is below a threshold. To this end, we extended TSDF voxel data with a list of keyframe indexes. When a keyframe is merged, all the modified voxels within the truncation band, *i.e.* the surface voxels (see Sec. 3.2), add its index to the list, so that we will be able to know which subset of the map can actually see that point. On the other hand, a surface voxel with an empty index list is definitely outside the area mapped by the keyframe set. Therefore, when the TSDF volume is built from the

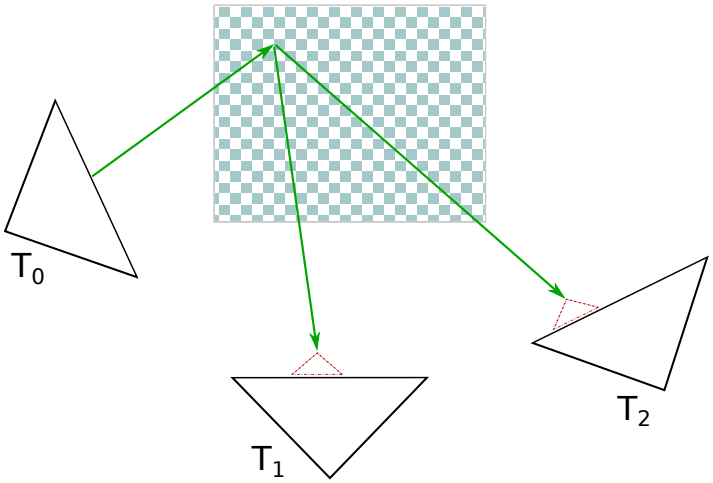


Figure 4.3.2: Our proposed matching strategy exploits information stored into the TSDF volume to find possible matches. In this example, we move from T_0 camera frame to T_1 and T_2, then we perform a local search on the image plane for the best corresponding point.

keyframe map, we also save the total number of surface voxels v_{map}. Then, upon depth image fusion, we count the number of such surface voxels having an empty keyframe index list v_{out}. When the ratio v_{out}/v_{map} rises above a threshold, the area outside the current map is large enough to justify the creation of a new keyframe.

Every time a new keyframe is spawn, we carry out a semantic optimization. As in SlamDunk (see Chap. 2), we apply the locality principle and, beside the last detected keyframe, we consider only the keyframes whose indexes have been written in the TSDF volume. We go through all the depth images and build a set of correspondences by means of the TSDF itself. As sketched in Fig. 4.3.2, for each valid depth measurements we search for matches as follows:

1. the pixel is reprojected according to Eq. (2.3.1) and transformed by the current camera pose;

2. the TSDF volume is queried at that point and the keyframe list retrieved;

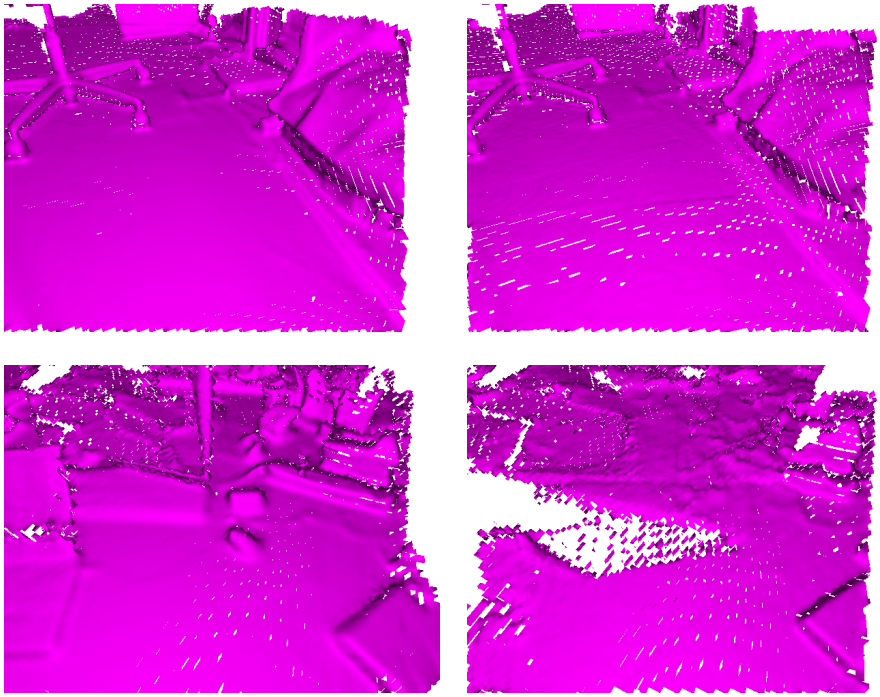


Figure 4.3.3: After a successful optimization, the TSDF volume has to be reconstructed from keyframes' depth maps. However, the loss of data, *i.e.* all the frames which are not keyframes, generates holes and noise in the distance function. Left: a surface, extracted as the zero-level set, before keyframe optimization. Right: the same surface after the reconstruction from keyframes only.

3. for each keyframe in the list, the voxel is projected onto the corresponding image plane by applying Eq. (3.2.2);

4. the nearest 3D point is found within a squared window.

Also, to increase robustness, we require normal coherence for each matching pair, *i.e.* the angle between the two normals must be less than a threshold. From this set of 3D correspondences we define a cost function as the sum of Euclidean distances (cfr. Eq. (2.3.6) and (2.3.7)) and add all the detected objects with their feature matches (see Sec. 4.2).

The results of the semantic optimization, carried out by the G2O library [49], is a new pose estimate for every objects and keyframes included in the problem. Therefore, the TSDF volume will no longer represent the

actual scene and must be rebuilt from scratch by fusion of the nearest keyframes. On one hand, by centering the volume on the last keyframe pose we implicitly allow for unbounded reconstruction and naturally follow camera movements. On the other hand, since we can only merge keyframes, in practice too few data are fused and the distance function appears noisy and incomplete. Typical defects caused by this issue are shown in Fig. 4.3.3. A flawed surface reconstructed from a small set of measurements, besides being less graphically appealing, may also harm camera tracking due to its holes and roughness. Therefore, in the future it will be worth considering how to employ subvolumes, introduced in Sec. 3.3, instead of a keyframe set of raw depth images.

4.4 Results

To validate the semantic SLAM framework, we carefully devised a set of experiments deploying semantic bundle adjustment in its 3D embodiment. First, we chose full 3D reconstructions of seven objects from our lab, with the meshes shown in Fig. 4.4.1, and we extracted 3D features by detecting keypoints at three different scales with Intrinsic Shape Signature [105] and describing their neighborhood with Spin Images [39]. We created a separate index at each scale including the descriptors extracted from all the seven models at that scale. Then, in the following experiments we match to these three data structures the Spin Image features extracted described at the same scales from the meshes associated with incoming depth frames. For each Spin Image descriptor we retrieve both the nearest match and the next-closest one belonging to a different object model, so that we accept the correspondence only if the ratio between the two distances is below some threshold. Also, this ratio can be used to weight the cost terms in Eq. (4.2.3). Nonetheless, this matching strategy returns a very large number of outliers, so a filtering step is required before entering our validation pipeline. Therefore, we deploy a RANSAC-based outlier rejection scheme similar to

(a) Doll. Size: $18 \times 39 \times 15$ cm.

(b) Duck. Size: $20 \times 28 \times 58$ cm.

(c) Frog. Size: $34 \times 32 \times 33$ cm.

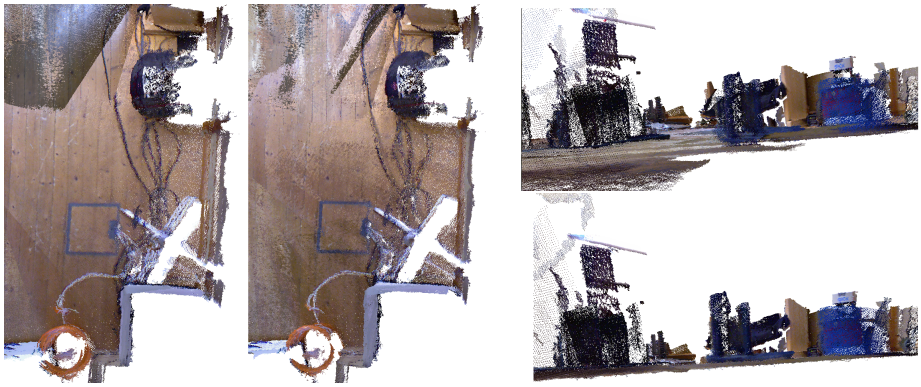
(d) Mario. Size: $19 \times 32 \times 22$ cm.

(e) Rabbit. Size: $20 \times 33 \times 20$ cm.

(f) Squirrel. Size: $15 \times 17 \times 20$ cm.

(g) Tortoise. Size: $20 \times 14 \times 20$ cm.

Figure 4.4.1: The full 3D meshes of the seven objects used throughout our experiments.



(a) (b)

Figure 4.4.2: We performed ICP-like refinement on the *fr1/floor* sequence of the RGB-D benchmark dataset [89] to improve ground truth poses. (a) Left: original frames. Right: optimized poses. Compare the right wall. (b) Top: original frames. Bottom: optimized poses. Compare the floor and the blue robot.

Alg. 2.1 using a threshold $\tau = 0.05\,\text{m}$, which is roughly 20 times the mesh resolution. Setting such high threshold still leaves incorrect correspondences, but our validation step is then able to cope with them.

To quantitatively evaluate our framework we purposely created two sequences with ground truth information on both the camera path and the 6-DOF object poses. We sampled RGB-D frames at 2 Hz from the *fr1/floor* sequence of the RGB-D benchmark dataset [89] and performed ICP-like [31] refinement to improve ground truth accuracy. Indeed, the camera path provided with the RGB-D benchmark dataset, though estimated through a motion capture system, does not always provide accurate frame alignment, as clearly shown in Fig. 4.4.2. Then, we placed in the 3D scene objects from our model database and generated the *4-objects* and *7-objects* sequences by raycasting, respectively, 4 and 7 objects on the sampled frames.

As for the SLAM engine, in these two experiments we deployed a 2D feature-based camera tracking algorithm inspired by existing SLAM frameworks [35, 30]. SIFT features [57] are extracted from the cur-

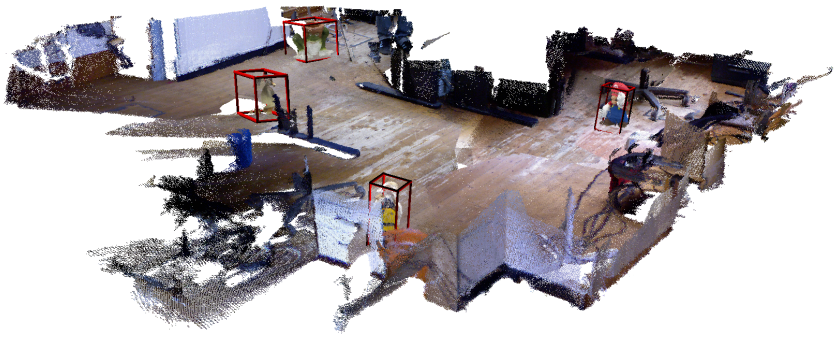


Figure 4.4.3: Final semantic reconstruction for the *4-objects* sequence. Bounding boxes aligned according to estimated poses are shown around detected objects.

rent RGB frame and matched to the previous one, then reprojected into the 3D space by means of the associated depth measurements and, as usual, filtered through a RANSAC-based procedure (cfr. Alg. 2.1). Although the adopted SLAM engine is quite basic, by including semantic constraints we can achieve global alignment and object localization, as shown in Fig. 4.4.3 for the *4-objects* sequence.

Since loop closure events are not handled, the adopted basic SLAM engine inevitably accumulates error, eventually leading to a poor reconstruction (see top image in Fig. 4.4.4). Conversely, our semantic approach adds links between distant images and may implicitly close loops through object instances (see bottom image in Fig. 4.4.4). This behavior can be qualitatively appreciated also in Fig. 4.4.5, where we compare the plain SLAM approach to our semantic bundle adjustment framework on a *real* sequence acquired in our lab by a Kinect sensor. Moreover, since the global semantic optimization includes all camera poses, the whole path estimation improves and the drift error shrinks, as vouched by Fig. 4.4.6 and 4.4.7.

In Sec. 4.2.2 we have described our object detection pipeline and we have shown how matching hypotheses are constantly refined as soon as new cues are found. This behavior is illustrated in Tab. 4.4.1, where

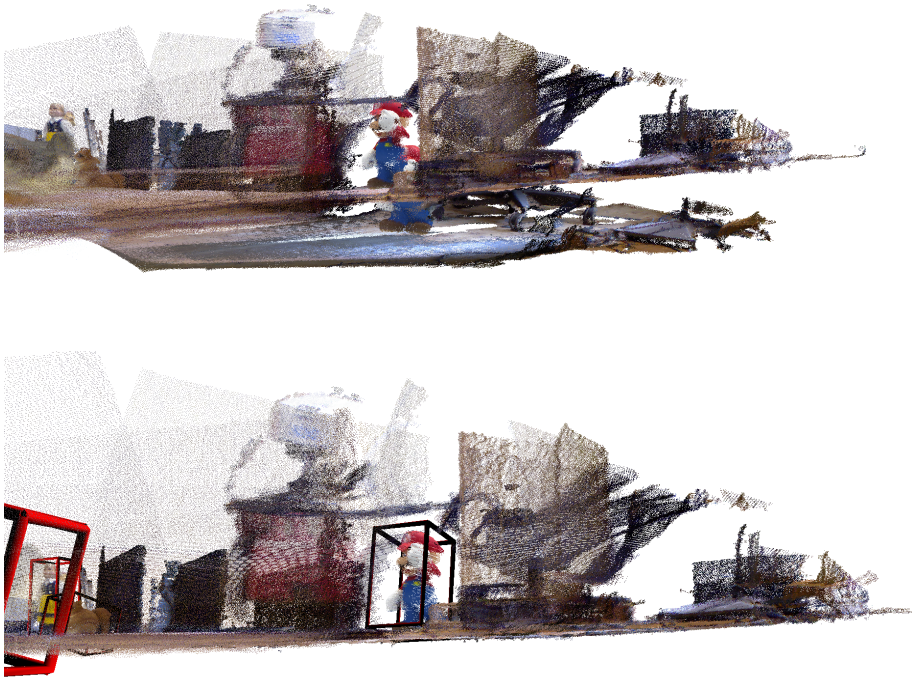
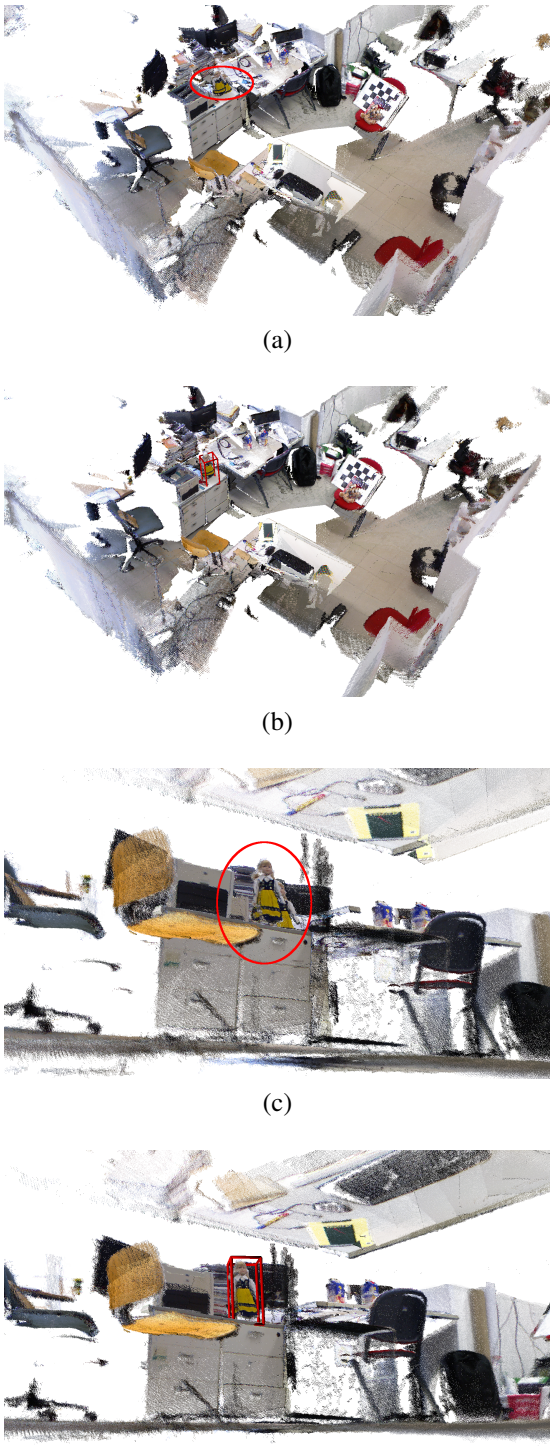


Figure 4.4.4: Detecting the same object instance in multiple views helps counteract the drift error, especially at loop closures. Top: a detail from the *7-objects* sequence reconstructed by the basic SLAM engine without deployment of semantic information about the objects. Bottom: the same sequence reconstructed by our semantic bundle adjustment approach. Bounding boxes aligned according to estimated poses are shown around detected objects.



(a)

(b)

(c)

(d)

Figure 4.4.5: We performed a complete loop with a Kinect camera capturing the object *Doll* at the beginning and at the end of the sequence. While a basic SLAM engine, (a) and (c), accumulates drift, our semantic approach, (b) and (d), implicitly closes the loop by detecting the object. In this experiment we have deployed the Color-SHOT descriptor [94] rather than Spin Images so to rely on more distinctive features.

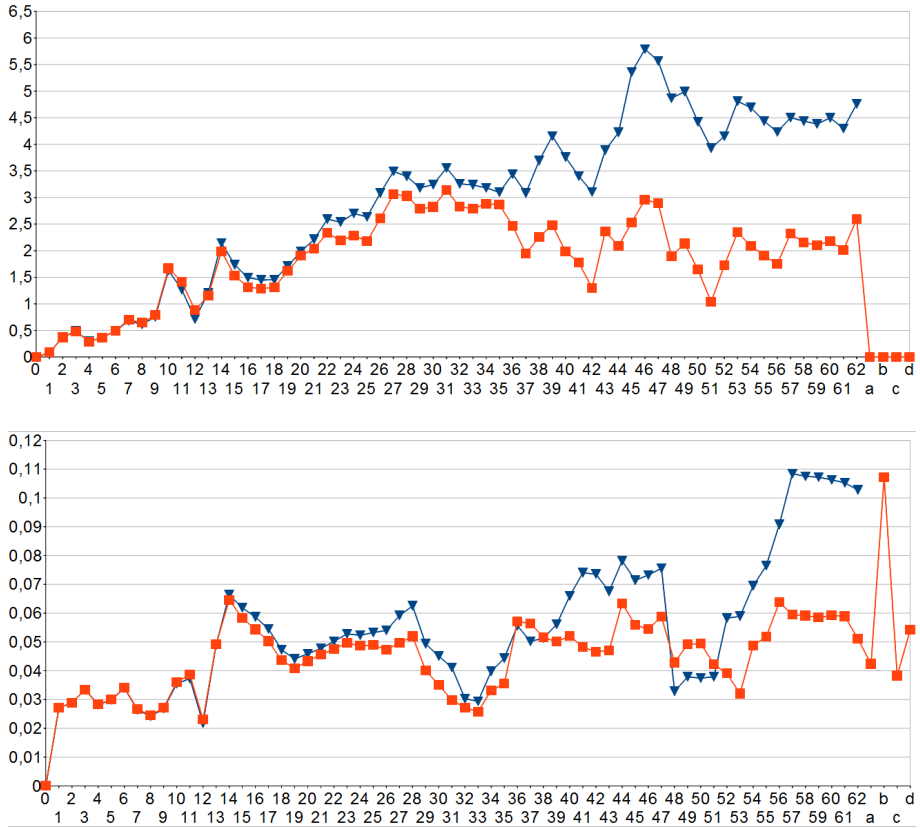


Figure 4.4.6: *4-objects* sequence: (top) rotation error, in degrees, and (bottom) translation error, in meters, for every frame and detected objects. Blue triangles: plain SLAM. Red squares: semantic bundle adjustment. The numbers denote the frame indexes while the letters the four objects (cfr. Fig. 4.4.1).

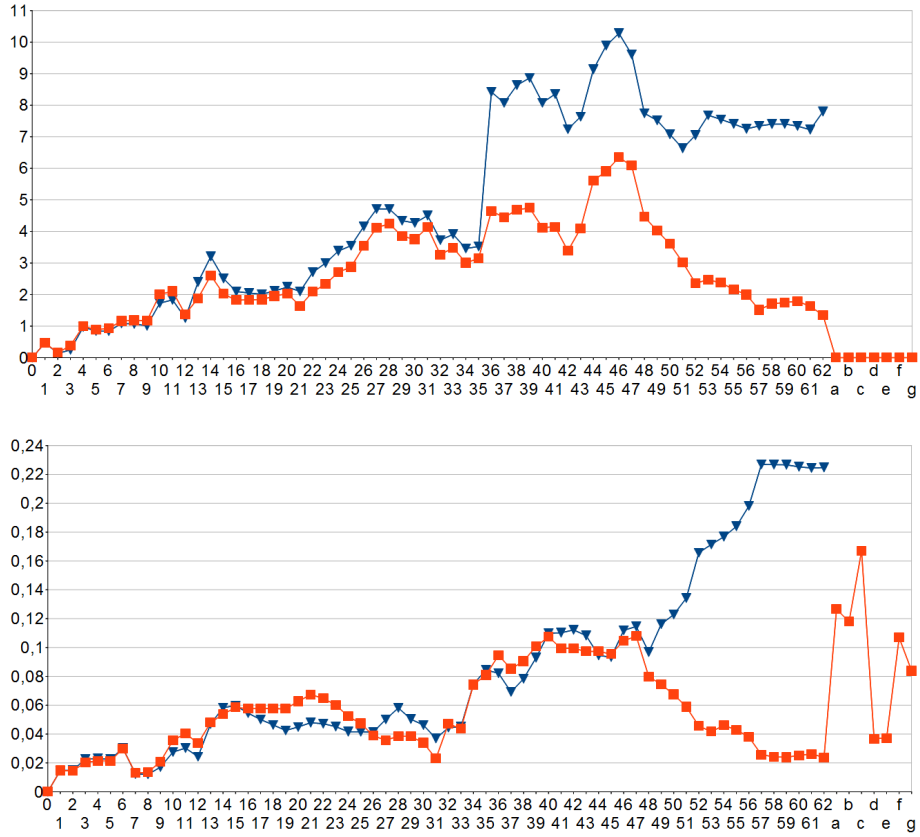


Figure 4.4.7: *7-objects* sequence: (top) rotation error, in degrees, and (bottom) translation error, in meters, for every frame and detected objects. Blue triangles: plain SLAM. Red squares: semantic bundle adjustment. The numbers denote the frame indexes while the letters the seven objects (cfr. Fig. 4.4.1).

Table 4.4.1: An excerpt of the validation graph for object *Frog* in the *7-objects* sequence. We report the number of frame-to-object constraints attached to the specified camera poses at the end of the validation phase for the features extracted from the frame in first column. Also, matches before the first cleaning procedure are shown in brackets. The pose estimation error for object *Frog* in the global SLAM graph is reported in the last column.

Frame	T_{16}	T_{34}	T_{35}	T_{36}	T_{37}	T_{38}	T_{39}	Pose Error (rot. and transl.)
#16	3 (11)	—	—	—	—	—	—	—
#34	3	31 (38)	—	—	—	—	—	114.6° 1.053 m
#35	3	31	55 (55)	—	—	—	—	88.5° 1.190 m
#36	3	31	55	122 (122)	—	—	—	86.3° 1.237 m
#37	3	31	55	122	127 (127)	—	—	86.8° 1.273 m
#38	0	31	55	122	127	123 (123)	—	0° 0.180 m
#39	0	31	55	122	127	123	0 (47)	0° 0.200 m

Frame #16

Frame #36

Frame #37

Frame #38

we report the number of frame-to-object edges in the validation graph of the object *Frog* for some camera frames of the *7-objects* sequence. In this case, 11 matches to the model are found in frame 16, although *Frog* is not present, and only 8 are cleaned during the validation phase. When in frame 34 the object appears, many more feature correspondences are found, but the pose estimate is harmed by the previous 3 outliers. However, after a few frames the validation pipeline is able to detect and remove those false matches and correctly localize the object.

Finally, in Fig. 4.4.8 we demonstrate object-aware augmented reality with occlusion handling. Indeed, SLAM techniques typically lack semantic information and allow only for augmenting the whole scene [45, 64], while our framework peculiarly detects and updates object poses as the camera moves and, therefore, even when the object is occluded, context specific data can be effectively rendered. Also, this sequence as well as the loop shown in Fig. 4.4.5 demonstrate the effective recognition of a *real* object, while the quantitative experiments above suffer from the *simulated* insertion of the object models, which may bias the results.

The final reconstructions for the *4-objects* as well as the *7-objects* sequences can be visually inspected by watching the video provided on our website[1]. The recording includes also the qualitative sequence acquired in our lab and the object-aware AR demonstration.

In Sec. 4.3 we have discussed how to extend KinectFusion to support our semantic bundle adjustment framework. We now compare this new SLAM system to SlamDunk and RGB-D SLAM [29, 30] (see Chap. 2). Purposely, we considered the *4-objects* sequence at full frame rate and, also, we augmented two more sequences from the RGB-D benchmark dataset [89] with objects *Mario* and *Doll*: *fr1/360* and *fr1/desk*. Then, we apply again the 3D embodiment of our framework by extracting

[1] http://vision.deis.unibo.it/research/104-nfioraio-thesis

Figure 4.4.8: Estimating object poses in the 3D environment allows for object-aware augmentation. In this example a red umbrella is rendered, with occlusion handling, near the *Doll*, even when the object is not visible. Top: 3D reconstruction. Bottom: two augmented frames. In this experiment we have deployed the Color-SHOT descriptor [94] for matching object feature.

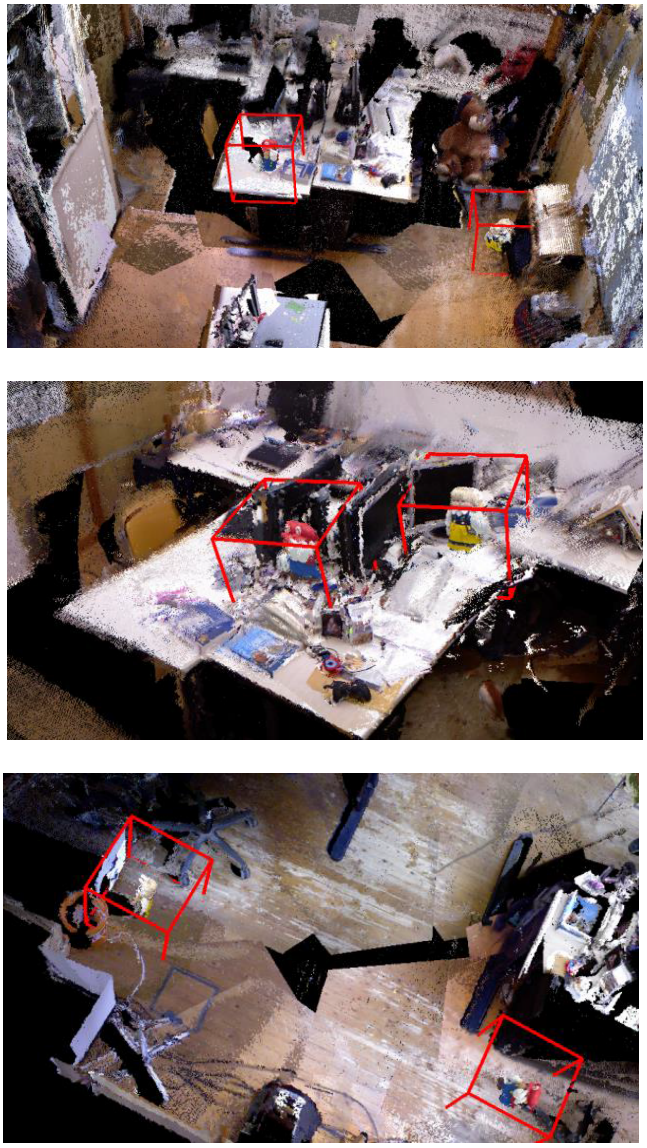


Figure 4.4.9: Results on augmented sequences from the RGB-D benchmark dataset [89] for our semantic extension to KinectFusion. Left: *fr1/360*. Middle: *fr1/desk*. Right: *fr1/floor*.

Table 4.4.2: We have compared our extension to KinectFusion, with (Semantic Ext KF) and without (Ext KF) semantic information, to SlamDunk and RGB-D SLAM. Each row reports RMS of absolute trajectory error (meters) on a sequence from the RGB-D benchmark dataset [89]. Note: *fr1/floor* is the *4-objects* sequence.

Sequence	Ext KF	Semantic Ext KF	SD SURF128	RGB-D SLAM	RGB-D SLAM w/ EMM
fr1/360	0.091	0.073	0.084	0.103	**<0.07**
fr1/desk	0.047	0.048	**0.025**	0.049	0.026
fr1/floor	0.062	0.059	**0.042**	0.055	<0.05
AVERAGE	0.067	0.060	**0.050**	0.069	

SIFT3D keypoints [57, 75] and Color-SHOT features [94]. Snapshot from the reconstruction are shown in Fig. 4.4.9, while Tab. 4.4.2 quantitatively compares the different methods. Clearly, simply extending KinectFusion with keyframe optimization (Ext KF) does not outperform existing solutions, but it usually achieves similar results. Indeed, as already discussed in Sec. 4.3, building a TSDF volume from a limited set of depth maps harms camera tracking (cfr. Fig. 4.3.3). However, introducing joint object detection and localization improves the final reconstruction, while estimating a 6-DOF pose for each detected object.

Chapter 5

Conclusion

In this thesis we have addressed the large scale mapping problem and the semantic understanding of the environment under exploration. Various techniques have been described and compared to state-of-art approaches. Following the general trend towards the use of 3D measurements from RGB-D data, in Chap. 2 we developed a novel SLAM system, SlamDunk, which combines standard 2D feature matching with multi-view camera tracking by reprojecting keypoints from the image plane into the 3D space. We devised a scalable framework based on local keyframe optimization and implicit loop closing, avoiding global alignment for efficiency reasons. Nevertheless, SlamDunk performs well w.r.t. the state-of-the-art on publicly available benchmark datasets. We have shown how this SLAM system can be effectively implemented on a mobile device and how to cope with the limited resources available on such architectures. We carried out performance analysis in terms of computational time and descriptiveness of several keypoint detector and feature extractor pairs, in order to find the best trade-off between speed and accuracy. Experimental results show that the system is capable of running at interactive frame rate with no significant loss in the overall reconstruction quality. Though tests have been performed by plugging an external Asus Xtion PRO Live sensor to a tablet device, we are currently working on the integration of a Structure depth camera to

enhance user experience. Then, the next step may concern the fusion of measurements coming from on-board sensors, such as, *e.g.*, the gyroscope and the accelerometer. We expect to improve the robustness and accuracy of the system, especially on low-textured areas of the environment, where our camera tracking approach is likely to fail. Finally, we wish to reduce the time required by the local optimization step, *e.g.* by marginalizing point-point constraints to pose-pose cost terms.

In Chap. 3 we have moved to a different architecture to address high-quality surface reconstruction at large scale using depth images only. We have described a camera tracking method based on KinectFusion which erodes old frames from the TSDF volume after every successful depth map integration, so as to maintain a low-drift model for tracking and avoid integration of misaligned TSDF data. Also, the erosion process has a complexity equal to the integration algorithm, so that the system can still operate in real-time, and, moreover, is key for extracting low-drift subvolumes from the scene under exploration. Then, global consistency is ensured by a novel online subvolume optimization, which directly exploits TSDF measurements, while a final blending step efficiently merges together aligned data. We have shown outstanding results in diverse settings, outperforming existing approaches, which either fail, require additional measurements, *e.g.* RGB images, cannot work incrementally and online or need much more time to complete. In the future, we are going to work on several aspects to further improve the framework. First, the number of frames fused in the active volume is currently fixed, while it could be possibly related to camera movements instead. Indeed, this number is also the same for the subvolumes, while it could be worth investigating how to decouple these two tasks and, possibly, sharing frames between subvolumes, since currently every subvolume is built by merging a different subset of camera frames. Then, the optimization step, though operating online, has a complexity which increases with the number of subvolumes, because a global alignment is performed every time a new subvolume is spawned.

However, valuable alternatives may be adopted from the submapping and relative bundle adjustment literature, so as to leverage on local optimization problems. Finally, the proposed volume blending algorithm has been devised as a final offline post-processing step, though we could imagine to create a feedback loop by injecting reliable blended TSDF measurements back into the active volume for further reducing drift error in camera motion estimation.

Semantic understanding has been addressed in Chap. 4 as an object instance detection task fully integrated into the SLAM pipeline. Unlike previous approaches, we have exploited the incremental reconstruction to validate hypotheses about object presence and we have devised a novel Semantic Bundle Adjustment framework to jointly estimate both camera and object 6-DOF poses. Such general formulation has been implemented in a 3D scenario by deploying RGB-D data and promising results have been shown on benchmark sequences. Moreover, we developed a Semantic KinectFusion system which extends the original KinectFusion proposal by introducing keyframe spawning and object detection. However, we have shown that a TSDF volume built from only a subset of frames, *i.e*. the keyframes, presents holes and noise which, in turn, hinder camera tracking. Applying instead the subvolume principle could be a valuable alternative to overcome such issue. As for object detection, we wish to pursue several directions. First, features could be extracted directly from the TSDF volume, being a low-noise model of the scene and, therefore, usually holding more reliable data fused from different point of views than a single RGB-D frame. Then, we aim at supporting the detection of multiple instances of the same object by, *e.g*., replacing RANSAC with a Hough-based outlier rejection and pose estimation step. Finally, we wish to extend the current framework to deal also with category-level recognition.

Bibliography

[1] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M. Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, October 2011.

[2] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[3] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *Pattern Analysis and Machine Intelligence (PAMI), IEEE Trans. on*, 9(5):698–700, September 1987.

[4] Sid Yingze Bao, Mohit Bagra, Yu-Wei Chao, and Silvio Savarese. Semantic structure from motion with points, regions, and objects. In *Computer Vision and Pattern Recognition (CVPR), IEEE Int'l Conf. on*, Providence (RI), USA, June 2012.

[5] Sid Yingze Bao, Mohit Bagra, and Silvio Savarese. Semantic structure from motion with object and point interactions. In *Challenges and Opportunities in Robot Perception, IEEE workshop at the International Conference on Computer Vision (ICCV)*, Barcelona, Spain, November 2011.

[6] Sid Yingze Bao and Silvio Savarese. Semantic structure from motion. In *Computer Vision and Pattern Recognition (CVPR), IEEE Int'l Conf. on*, Colorado Springs (CO), USA, June 2011.

[7] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346 – 359, September, 10 2008.

[8] P. J. Besl and H. D. McKay. A method for registration of 3-d shapes. *Pattern Analysis and Machine Intelligence (PAMI), IEEE Trans. on*, 14(2):239–256, 1992.

[9] S. Betge-Brezetz, P. Hebert, R. Chatila, and M. Devy. Uncertain map making in natural environments. In *International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1048–1053, April 1996.

[10] J.-L. Blanco, J. Gonzalez-Jimenez, and J.-A. Fernandez-Madrigal. Sparser relative bundle adjustment (srba): Constant-time maintenance and local optimization of arbitrarily large maps. In *International Conference on Robotics and Automation (ICRA)*, pages 70–77, May 2013.

[11] Dorit Borrmann, Jan Elseberg, Kai Lingemann, Andreas NÃŒchter, and Joachim Hertzberg. Globally consistent 3d mapping with scan matching. *Journal of Robotics and Autonomous Systems*, 56:130–142, February 2008.

[12] Rodney A. Brooks. Symbolic error analysis and robot planning. *International Journal of Robotics Research*, 1(4):29–68, 1982.

[13] E Bylow, Jurgen Sturm, Christian Kerl, Fredrik Kahl, and Daniel Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. In *Robotics: Science and Systems (RSS)*, Berlin, Germany, June 2013.

[14] R. O. Castle, G. Klein, and D. W. Murray. Combining monoslam with object recognition for scene augmentation using a wearable camera. *Journal of Image and Vision Computing*, 28(11):1548–1556, 2010.

[15] Jiawen Chen, Dennis Bautembach, and Shahram Izadi. Scalable real-time volumetric surface reconstruction. *ACM Transaction on Graphics (TOG)*, 32(4), July 2013.

[16] Yang Chen and Gerard Medioni. Object modelling by registration of multiple range images. *Journal of Image and Vision Computing*, 10:145–155, April 1992.

[17] Javier Civera, Dorian Gálvez-López, Luis Riazuelo, Juan D. Tardós, and J. M. M. Montiel. Towards semantic SLAM using a monocular camera. In *Intelligent Robots and Systems (IROS), IEEE/RSJ Int'l Conf. on*, pages 1277–1284, San Francisco (CA), USA, September 2011.

[18] Javier Civera, Oscar G. Grasa, Andrew J. Davison, and J. M. M. Montiel. 1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry. *Journal of Field Robotics*, 27(5):609–631, September 2010.

[19] N. Cornelis, B. Leibe, K. Cornelis, and L. Van Gool. 3d urban scene modeling integrating recognition and reconstruction. *International Journal of Computer Vision (IJCV)*, 78(2-3):121–141, July 2008.

[20] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH Int'l Conf.*, New Orleans (LA), USA, 1996.

[21] A. J. Davison. *Mobile Robot Navigation Using Active Vision*. PhD thesis, Oxford, UK, 1998.

[22] Andrew Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *Pattern Analysis and Machine Intelligence (PAMI), IEEE Trans. on*, 29(6):1052–1067, June 2007.

[23] Andrew J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *IEEE International Conference on Computer Vision (ICCV)*, page 1403, Washington (DC), USA, 2003.

[24] F. Dellaert and M. Kaess. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12):1181–1204, December 2006.

[25] B. Drost, Markus Ulrich, N. Navab, and S. Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *Computer Vision and Pattern Recognition (CVPR), IEEE Int'l Conf. on*, pages 998–1005, June 2010.

[26] I. Dryanovski, R.G. Valenti, and Jizhong Xiao. Fast visual odometry and mapping from rgb-d data. In *International Conference on Robotics and Automation (ICRA)*, pages 2305–2310, May 2013.

[27] E. Eade and T. Drummond. Monocular SLAM as a graph of coalesced observations. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, Rio de Janeiro, Brasil, October 2007.

[28] S. Ekvall, P. Jensfelt, and D. Kragic. Integrating active mobile robot object recognition and slam in natural environments. In *Intelligent Robots and Systems (IROS), IEEE/RSJ Int'l Conf. on*, Beijing, China, October 2006.

[29] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB-D SLAM system. In *International Conference on Robotics and Automation (ICRA)*, St. Paul (MA), USA, May 2012.

[30] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3d mapping with an RGB-D camera. *IEEE Transactions on Robotics (T-RO)*, 30(1):177–187, 2013.

[31] Nicola Fioraio and Kurt Konolige. Realtime visual and point cloud SLAM. In *RGB-D: Advanced Reasoning with Depth Cameras, workshop at Robotics: Science and System (RSS)*, Los Angeles (CA), USA, June 27 2011.

[32] D. Galvez-Lopez and J.D. Tardos. Real-time loop detection with bags of binary words. In *Intelligent Robots and Systems (IROS), IEEE/RSJ Int'l Conf. on*, pages 51–58, September 2011.

[33] Google Inc. Project Tango. `https://www.google.com/atap/projecttango/`, 2014.

[34] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[35] P. Henry, Michael Krainin, E. Herbst, X. Ren, and Dieter Fox. RGB-D mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *International Journal of Robotics Research*, 31(5):647–663, February 2012.

[36] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Putting objects in perspective. *International Journal of Computer Vision (IJCV)*, 80(1):3–15, October 2008.

[37] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[38] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve

Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *ACM Symposium on User Interface Software and Technology*, Santa Barbara (CA), USA, October 2011.

[39] A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *Pattern Analysis and Machine Intelligence (PAMI), IEEE Trans. on*, 21(5):433–449, 1999.

[40] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics (T-RO)*, 24(6):1365–1378, December 2008.

[41] Gerald Schweighofer Katrin Pirker, Matthias Rüther and Horst Bischof. Gpslam: Marrying sparse geometric and dense probabilistic visual mapping. In *British Machine Vision Conference (BMVC)*, pages 115.1–115.12, Dundee, UK, September 2011.

[42] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *3D Vision (3DV), Int'l Conf. on*, pages 1–8, June 2013.

[43] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for rgb-d cameras. In *International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013.

[44] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Dense visual SLAM for RGB-D cameras. In *Intelligent Robots and Systems (IROS), IEEE/RSJ Int'l Conf. on*, Tokyo, Japan, November 2013.

[45] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 225–234, Nara, Japan, November 2007.

[46] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 83–86, Orlando (FL), USA, October 2009.

[47] Kurt Konolige and Motilal Agrawal. FrameSLAM: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics (T-RO)*, 24(5):1066–1077, October 2008.

[48] Daniel B. Kubacki, Huy Q. Bui, S. Derin Babacan, and Minh N. Do. Registration and integration of multiple depth images using signed distance function. *SPIE, Computational Imaging*, 8296:22, February, 9 2012.

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

[50] Wonwoo Lee, Kiyoung Kim, and Woontack Woo. Mobile phone-based 3D modeling framework for instant interaction. In *Computer Vision Workshops at the IEEE International Conference on Computer Vision (ICCV)*, Kyoto, Japan, September 2009.

[51] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligence for Mechanical Systems, workshop at the IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 1442–1447, November 1991.

[52] S. Leutenegger, M. Chli, and R.Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *IEEE International Conference on Computer Vision (ICCV)*, November, 6 2011.

[53] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathematics*, II(2):164–168, 1944.

[54] Li-Jia Li, Richard Socher, and Li Fei-Fei. Towards total scene understanding: Classification, annotation and segmentation in an automatic framework. In *Computer Vision and Pattern Recognition (CVPR), IEEE Int'l Conf. on*, Miami (FL), USA, June 2009.

[55] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH Int'l Conf.*, volume 21, pages 163–170, July 1987.

[56] Manolis I. A. Lourakis and Antonis A. Argyros. Sba: a software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software*, 36(1):1–30, 2009.

[57] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–119, January, 5 2004.

[58] R. Maier, J. Sturm, and D. Cremers. Submap-based bundle adjustment for 3d reconstruction from rgb-d data. In *German Conference on Pattern Recognition (GCPR)*, Münster, Germany, September 2014.

[59] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, 1963.

[60] David Meger, Per-Erik Forssén, Kevin Lai, Scott Helmer, Sancho McCann, Tristram Southey, Matthew Baumann, James J. Little, and David G. Lowe. Curious george: An attentive semantic robot. *Journal of Robotics and Autonomous Systems*, 56(6):503–511, June 2008.

[61] M. Montemerlo and S. Thrun. Large-scale robotic 3-d mapping of urban structures. In *International Symposium on Experimental Robotics (ISER)*, Singapore, June 2004.

[62] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *National Conference on Artificial Intelligence*, pages 593–598, Edmonton, Alberta, Canada, 2002.

[63] R.A. Newcombe and A.J. Davison. Live dense reconstruction with a single moving camera. In *Computer Vision and Pattern Recognition (CVPR), IEEE Int'l Conf. on*, pages 1498–1505, June 2010.

[64] Richard Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, Basel, CH, October 2011.

[65] Richard Newcombe, Steven Lovegrove, and Andrew Davison. DTAM: Dense tracking and mapping in real-time. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2320–2327, 2011.

[66] Matthias Niessner, Michael Zollhofer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transaction on Graphics (TOG)*, 32(6), November 2013.

[67] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Computer Vision and Pattern Recognition (CVPR), IEEE Int'l Conf. on*, volume 1, pages 652–659, June 2004.

[68] Occipital Inc. The Structure sensor. `http://structure.io/`, 2014.

[69] Qi Pan, Clemens Arth, Edward Rosten, Gerhard Reitmayr, and Tom Drummond. Rapid scene reconstruction on mobile phones

from panoramic images. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, Basel, CH, October 2011.

[70] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Computer Graphics and Interactive Techniques*, SIGGRAPH, pages 335–342, New York (NY), USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[71] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1508–1511, October 2005.

[72] Henry Roth and Marsette Vona. Moving volume KinectFusion. In *British Machine Vision Conference (BMVC)*, Guildford, UK, September 2012.

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

[74] M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Highly accurate 3d surface models by sparse surface adjustment. In *International Conference on Robotics and Automation (ICRA)*, St. Paul (MN), USA, May 2012.

[75] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point cloud library (PCL). In *International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[76] Renato F. Salas-Moreno, Richard A. Newcombe, Hauke Strasdat, Paul H. J. Kelly, and Andrew J. Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Computer Vision and Pattern Recognition (CVPR), IEEE Int'l Conf. on*, Portland (OR), USA, June 23-28 2013.

[77] S.A. Scherer and A. Zell. Efficient onbard rgbd-slam for autonomous mavs. In *Intelligent Robots and Systems (IROS), IEEE/RSJ Int'l Conf. on*, pages 1062–1068, November 2013.

[78] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Robotics: Science and Systems (RSS)*, Seattle (WA), USA, June 2009.

[79] M. Segal and K Akeley. The opengl graphics system: A specification. `http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf`, October, 22 2004.

[80] Gabe Sibley, Larry Matthies, and Gaurav Sukhatme. A sliding window filter for incremental slam. In Danica Kragic and Ville Kyrki, editors, *Unifying Perspectives in Computational and Robot Vision*, volume 8 of *Lecture Notes in Electrical Engineering*, pages 103–112. Springer US, 2008.

[81] Gabe Sibley, Christopher Mei, Ian Reid, and Paul Newman. Adaptive relative bundle adjustment. In *Robotics: Science and Systems (RSS)*, Seattle (WA), USA, June 2009.

[82] R. Smith, M. Self, and P. Cheeseman. Autonomous robot vehicles. chapter Estimating Uncertain Spatial Relationships in Robotics, pages 167–193. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[83] Randall Smith, Matthew Self, and Peter Cheeseman. A stochastic map for uncertain spatial relationships. In *International Symposium on Robotics Research*, pages 467–474, Cambridge, MA, USA, 1988. MIT Press.

[84] F. Steinbruecker, J. Sturm, and D. Cremers. Real-time visual odometry from dense rgb-d images. In *Live Dense Reconstruction with Moving Cameras, IEEE workshop at International Conference on Computer Vision (ICCV)*, Barcelona, Spain, November 2011.

[85] F. Steinbruecker, J. Sturm, and D. Cremers. Volumetric 3d mapping in real-time on a cpu. In *International Conference on Robotics and Automation (ICRA)*, Hongkong, China, May 2014.

[86] Frank Steinbruecker, Christian Kerl, Jurgen Sturm, and Daniel Cremers. Large-scale multi-resolution surface reconstruction from RGB-D sequences. In *IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia, December 2013.

[87] Hauke Strasdat, Andrew J. Davison, J.M.M. Montiel, and Kurt Konolige. Double window optimisation for constant time visual SLAM. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2352–2359, Los Alamitos (CA) USA, November 2011.

[88] Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. Visual slam: Why filter? *Image and Vision Computing*, 30(2):65–77, February 2012.

[89] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Intelligent Robots and Systems (IROS), IEEE/RSJ Int'l Conf. on*, Vilamoura (Algarve), Portugal, October 2012.

[90] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys. Live metric 3D reconstruction on mobile phones. In *IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia, December, 13 2013.

[91] Russell Highsmith Taylor. *The Synthesis of Manipulator Control Programs from Task-level Specifications*. PhD thesis, Stanford, CA, USA, 1976.

[92] S. Thrun, Y. Liu, D. Koller, A.Y. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *International Journal of Robotics Research*, 23(7-8):693–716, August 2004.

[93] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *IEEE European Conference on Computer Vision (ECCV)*, Heraklion, Greece, September, 5-11 2010.

[94] Federico Tombari, Samuele Salti, and Luigi Di Stefano. A combined texture-shape descriptor for enhanced 3D feature matching. In *IEEE International Conference on Image Processing (ICIP)*, pages 809–812, Brussels, Belgium, September 2011.

[95] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment – a modern synthesis. 1883:298–372, 2000.

[96] Kartik Venkataraman, Dan Lelescu, Jacques Duparré, Andrew McMahon, Gabriel Molina, Priyam Chatterjee, Robert Mullis, and Shree Nayar. Picam: An ultra-thin high performance monolithic camera array. *ACM Transaction on Graphics (TOG)*, 32(6), November 2013.

[97] A. Wendel, M. Maurer, G. Graber, T. Pock, and H. Bischof. Dense reconstruction on-the-fly. In *Computer Vision and Pattern Recognition (CVPR), IEEE Int'l Conf. on*, pages 1450–1457, June 2012.

[98] T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, and J. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In *International Conference on Robotics and Automation (ICRA)*, pages 5724–5731, May 2013.

[99] T. Whelan, M. Kaess, R. Finman, M.F. Fallon, H. Johannsson, J.J. Leonard, and J. McDonald. 3D mapping, localisation and object retrieval using low cost robotic platforms: A robotic search engine for the real-world. In *RGB-D: Advanced Reasoning with Depth Cameras, workshop at Robotics: Science and System (RSS)*, Berkeley (CA), USA, July 2014.

[100] T. Whelan, M. Kaess, H. Johannsson, M.F. Fallon, J.J. Leonard, and J.B. McDonald. Real-time large scale dense RGB-D SLAM with volumetric fusion. *International Journal of Robotics Research*, 2014.

[101] T. Whelan, M. Kaess, J.J. Leonard, and J.B McDonald. Deformation-based loop closure for large scale dense RGB-D SLAM. In *Intelligent Robots and Systems (IROS), IEEE/RSJ Int'l Conf. on*, Tokyo, Japan, November 2013.

[102] Thomas Whelan, John Mcdonald, Michael Kaess, Maurice Fallon, Hordur Johannsson, and John Leonard. Kintinuous: Spatially extended KinectFusion. In *RGB-D: Advanced Reasoning with Depth Cameras, workshop at Robotics: Science and System (RSS)*, Sydney, Australia, July 2012.

[103] Changchang Wu. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). `http://cs.unc.edu/~ccwu/siftgpu`, 2007.

[104] Ming Zeng, Fukai Zhao, Jiaxiang Zheng, and Xinguo Liu. A memory-efficient kinectfusion using octree. In *Computational Visual Media (CVM), Int'l Conf. on*, pages 234–241, Beijing, China, November 2012.

[105] Yu Zhong. Intrinsic shape signatures: A shape descriptor for 3d object recognition. In *Computer Vision Workshops at the IEEE International Conference on Computer Vision (ICCV)*, pages 689–696, Kyoto, Japan, October 2009.

[106] Qian-Yi Zhou and Vladlen Koltun. Dense scene reconstruction with points of interest. *ACM Transaction on Graphics (TOG)*, 32(4), July 2013.

[107] Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. Elastic fragments for dense scene reconstruction. In *IEEE International*

Conference on Computer Vision (ICCV), pages 473–480, Sydney (NSW), Australia, December, 1-8 2013.

Author's Publications During The PhD Course

[108] Nicholas Brunetto, Nicola Fioraio, and Luigi Di Stefano. Interactive RGB-D SLAM on mobile devices. In *Intelligent Mobile and Egocentric Vision, IEEE workshop at Asian Conference on Computer Vision (ACCV)*, Singapore, November 2 2014.

[109] Nicola Fioraio, Gregorio Cerri, and Luigi Di Stefano. Towards semantic KinectFusion. In *International Conference on Image Analysis and Processing (ICIAP)*, Naples, Italy, September 9-13 2013.

[110] Nicola Fioraio and Luigi Di Stefano. Joint detection, tracking and mapping by semantic bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), IEEE Int'l Conf. on*, Portland (OR), USA, 2013.

[111] Nicola Fioraio and Luigi Di Stefano. SlamDunk: Affordable real-time RGB-D SLAM. In *Consumer Depth Cameras For Computer Vision, IEEE workshop at European Conference on Computer Vision (ECCV)*, Zurich, Switzerland, September 6 2014.

[112] Nicola Fioraio, Jonathan Taylor, Andrew Fitzgibbon, Luigi Di Stefano, and Shahram Izadi. Large-scale and drift-free surface reconstruction using online subvolume registration. In *Com-*

puter Vision and Pattern Recognition (CVPR), IEEE Int'l Conf. on, Boston (MA), USA, 2015.

[113] Samuele Salti, Alioscia Petrelli, Federico Tombari, Nicola Fioraio, and Luigi Di Stefano. A traffic sign detection pipeline based on interest regions extraction. In *International Joint Conference on Neural Networks (IJCNN)*, Dallas (TX), USA, August 4-9 2013.

[114] Samuele Salti, Alioscia Petrelli, Federico Tombari, Nicola Fioraio, and Luigi Di Stefano. Traffic sign detection via interest region extraction. *Pattern Recognition*, June 9 2014.

[115] Federico Tombari, Nicola Fioraio, Tommaso Cavallari, Samuele Salti, Alioscia Petrelli, and Luigi Di Stefano. Automatic detection of pole-like structures in 3d urban environments. In *Intelligent Robots and Systems (IROS), IEEE/RSJ Int'l Conf. on*, Chicago (IL), USA, September 14-18 2014.