Real-time Dense Simultaneous Localisation and Mapping over Large Scale Environments

Thomas J. Whelan

A dissertation submitted for the degree of

Doctor of Philosophy

Department of Computer Science

Faculty of Science and Engineering

National University of Ireland Maynooth

August 2014

Head of Department: Dr. Adam Winstanley

Supervisor: Dr. John B. McDonald

In memory of my grandfathers and namesakes, Thomas Whelan and John Newell.

Acknowledgements

Firstly I would like to express my enormous gratitude to my supervisor John McDonald for his insight, advice and support throughout this work. I could not have asked for a better supervisor who showed inspirational unwavering encouragement and enthusiasm over the past few years. I would also like to acknowledge the invaluable support and knowledge of my "grand supervisor" John Leonard. Between my time spent at CSAIL and the subsequent long term collaboration with his group he has played a key role in the success of my work and shaping my academic career as a whole.

I would also like to thank my examining committee, Prof. Ronan Reilly and Prof. Wolfram Burgard for their helpful comments and feedback on this thesis.

I was fortunate enough to become acquainted with a number of extremely talented individuals during my time spent at CSAIL. I would like to express my special gratitude to Michael Kaess for his consistent expert advice on a plethora of different topics throughout this research. In addition to this I would like to thank Hordur Johannsson and David Rosen for their insightful discussions on various different technical aspects of research. Special thanks also go to Ross Finman, Maurice Fallon and Mark VanMiddlesworth for their eagerness to collaborate and proficiency in doing so. I must also extend a warm thank you to the members of the VCA group at TU/e, in particular Lingni Ma, Egor Bondarev and Peter H. N.

de With for their patience and professionalism in what has been a truly fruitful collaboration for all parties involved.

Back in Maynooth there are a number of people in my own department I would like to thank for their support over the years including Adam Winstanley, Tom Lysaght and Guillaume Gales. Special thanks to Donagh Hatton for general computer science and life discussion over many well deserved breaks.

Research presented in this thesis was funded by a Strategic Research Cluster grant (07/SRC/I1168) by Science Foundation Ireland under the Irish National Development Plan and the Embark Initiative of the Irish Research Council. This funding is gratefully acknowledged.

Finally I would like to thank my parents for their unfaltering love and support throughout this whole endeavour. Their unmatched wisdom has guided me towards achieving everything I have so far in my life and ultimately realising my dreams. It was them who taught me the most valuable lessons I have ever learned, not in any lab or lecture theatre but in the loving environment of our family home.

Contents

List of Figures

List of Tables

Abstract

The ability for a robot to create a map of an unknown environment and localise within that map is of critical importance in intelligent autonomous operation. This problem is referred to as *Simultaneous Localisation and Mapping* (or *SLAM*) and has been one of the major focusses of robotics research over the past 25 years. Although the initial focus was on 2D laser scan SLAM, more recently full 3D SLAM has become the dominant paradigm.

The recent expansion in popularity of full, dense 3D SLAM is arguably a result of the release of the Microsoft Kinect commodity RGB-D sensor, which provides high quality depth sensing capabilities for a little over one hundred US dollars. Before the advent of the Kinect, 3D SLAM methods required either time of flight (TOF) sensors, 3D lidar scanners or stereo vision, which were typically either quite expensive or not suitable for fully mobile real-time operation if dense reconstruction was desired. Another recent technology which is often coupled with dense methods is General-Purpose computing on Graphics Processing Units (GPGPU) which exploits the massive parallelism available in GPU hardware to perform high speed and often real-time processing on entire images every frame. Being an affordable commodity technology, GPU-based programming is arguably another large enabler in recent dense SLAM research.

Many visual SLAM systems and 3D reconstruction systems (both offline and online) have been published in recent times that rely purely on RGB-D sensing capabilities because of the Kinect's low price and accuracy; [43, 26, 113, 86]. However given the density of the data available, many existing systems have one or many limitations imposed by the challenges of processing such large amounts of information. These include a limitation in operating area, the inability to function in real-time over large scales, or not producing a globally consistent reconstruction of the explored environment or a map representation which is meaningful for robotic operations. In this thesis we address these issues through the development of a system which allows real-time globally consistent dense mapping over large scales, while providing a map representation which is useful for both autonomous robot navigation and higher level functionality such as object detection.

The development of this system involves solving a number of critical issues including efficient real-time dense mapping over large scales, robust real-time camera pose estimation, a scalable means of correcting dense reconstructions for global consistency and representing the map in a format suitable for robotic operations. We address these issues respectively by 1) employing an efficient rolling cyclical buffer representation for mapping in the local frame; 2) estimating a dense photometric camera pose constraint in conjunction with a dense geometric constraint and jointly optimising for a camera pose estimate; 3) optimising the dense map by means of a non-rigid space deformation parameterised by a loop closure constraint; and, 4) intelligently simplifying the dense map reconstruction to a planar representation.

As part of this the system is implemented as a set of hierarchical multi-threaded components which are capable of operating in real-time. The architecture facilitates the creation and integration of new modules with minimal impact on the performance of the overall system. This yields an adaptable and easily extendable system which is easily combined with other software systems designed for related operations.

We provide a comprehensive quantitative and qualitative evaluation of all aspects of the system's performance, demonstrating real-time dense SLAM over large scales. Our evaluation includes comparisons to other approaches on standard benchmarks in terms of computational performance, trajectory estimation and surface reconstruction quality.

CHAPTER 1

Introduction

1.1 Motivation

In recent years visual SLAM has reached a significant level of maturity with a number of robust real-time solutions being reported in the literature [65, 25, 17, 111]. Although these techniques permit the construction of an accurate map of an environment, the fact that they are feature-based means that they result in sparse point cloud maps that either cannot be used directly or have limited utility in many robotic tasks (e.g. obstacle avoidance, path planning, manipulation, etc.). This issue has motivated the development of dense mapping approaches that aim to use information from every pixel from the input video frames to create 3D surface models of the environment [114, 85]. The emergence of RGB-D cameras, and in particular the Microsoft Kinect, has seen this work being taken a step further.

The motivation for the work in this thesis is to take local small scale dense visual mapping to larger scales. There are numerous advantages to dense scene reconstructions over sparse classical point-based reconstructions. Substantially more information about the scene is captured which is extremely useful in robotics applications. This information includes the occlusion relationships between surfaces in the scene, something that is invaluable when any kind of physical interaction with the scene is desired. In particular large textureless regions of an environment (such as walls and floors) are difficult to reconstruct using visual feature-based approaches and are inherently useful for any robotic platform to be aware of for path planning. A full 3D scene model provides a physically predictive reconstruction that can immediately be used for motion planning and manipulation. This removes the need to rely on 2D images or other sensors to infer additional information about free space and potential interface points between objects in the world and end effectors.

Along with the obvious benefits achieved with dense over sparse representations, real-time operation is another strong motivation for the work in this thesis. While methods exist to compute very high quality maps from the same RGB-D data used in our work [133, 134], their inability to function in real-time or over very large scales restricts their potential applications. In order for dense representations to be useful in the context of closed-loop robotics scenarios (and indeed other settings where perhaps a human is involved in the loop) an approach which is capable of processing captured data in real-time online without any batch steps is necessary. An example of this is provided in Chapter 8. Other such scenarios beyond autonomous control which would necessitate a real-time capable approach include augmented/mixed and virtual reality frameworks.

The quantity of the data which needs to be processed to achieve high fidelity dense mapping in real-time introduces a challenge in terms of algorithm scalability and computational performance. With GPUs becoming more and more common place in consumer electronic devices, initially only in desktop PCs but more recently in phones and tablets, we can expect in the coming years that sufficient highly parallel processing power will be available in all kinds of platforms. There is a well matched coupling between the information processing needs of dense SLAM and GPU processing capabilities. However GPGPU brings with it a need to adapt and develop techniques that can specifically exploit parallel processing methods. While older methods for visual SLAM focussed on using distinct visual features, which typically only occur in relatively small numbers, there is a new paradigm shift with regards to dense visual SLAM that tries to process and reconstruct all available information, using every pixel in the image if possible.

Typical processes in SLAM such as map correction after loop closure detection are no longer as simple as rigidly transforming a sparse set of visual feature points after pose graph optimisation. Solving such issues is a major part of the motivation for the work in this thesis which has many implications for a number of different fields. By equipping autonomous devices such as mobile robotic platforms with the capability to densely reconstruct large scale environments in real-time, immediately their cognitive capabilities are enormously enhanced. With rich, high quality 3D environment data available more intelligent autonomous reasoning can be carried out in processes such as navigation and planning. Higher level processing such as object recognition and change detection is also made easier by the abundance of scene data available.

Aside from the robotics applications made possible, there are a number of different tasks which can benefit from RGB-D based real-time large scale dense 3D reconstruction that previously would have required expensive specialist equipment such as lidar scanners. One of the most immediately apparent of these is the surveying of indoor structures. Classic lidar approaches require the scanner to be placed at a number of different fixed locations and the resulting set of scans to be aligned to achieve a global point-based model. A dense visual SLAM approach using a handheld sensor provides a human operator with much more freedom when reconstructing an environment by allowing free continuous motion throughout the area being scanned. Parts of a scene which would otherwise be difficult to cover with a lidar device are much more easily captured with a handheld freely movable RGB-D camera.

The ability to quickly capture dense models of environments has immediate application in many areas such as modelling for renovation planning, set capture for visual effects in the film industry, digital archaeology and real estate. Indeed, real-time low cost methods for dense reconstruction are much more tangible for non-experts to use and understand [66]. This is in contrast to previous work in visual SLAM which was mainly of interest to roboticists. With the advent of Google's new Project Tango[1] for mobile devices visual SLAM is arguably becoming more and more mainstream and may be in the pocket of anyone with a smartphone in a very short period of time. The widescale presence of SLAM-capable sensors in everyone's hands sets a need for algorithms capable of exploiting these sensors to their extent, where dense methods can provide the richest and most informative representations.

In addition to the motivation to take dense reconstruction to larger scales, there is the step beyond producing such maps that exploits the extra information present in these models. The object segmentation work of Finman *et al.* discussed in Chapter 8 is one such example of this. In the context of real-time mobile robot operation high vertex-count dense maps are not necessarily the most computationally efficient format for planning and navigation. This motivates the development of a method to exploit the dense information contained within these maps to derive a representation which is rich in information required for planning and navigation while also being in a format that is suitable for real-time computation over during closed-loop autonomous control.

[1]`https://www.google.com/atap/projecttango`

1.2 Background

To date there has been numerous solutions to the SLAM problem with a wide variety of forms and functionality. When posed as the task of estimating a map of an unknown environment whilst simultaneously localising within that environment the actual outputs of a SLAM solution in a more technical sense can be described as an estimate of the trajectory that the sensor took while exploring the environment as well as an estimated map of the structure sensed along that trajectory in some coordinate frame. Typically in visual SLAM (where a camera is the primary input sensor) the former would take the form of a sequence of camera poses in the appropriate space (e.g. two translation components and a rotation component for SLAM in a 2D environment, otherwise known as $\mathbb{SE}_2$, or three translation components and a three dimensional rotation representation in a 3D environment, often chosen to be $\mathbb{SE}_3$). There does exist a number of SLAM systems which attempt to fit a continuous time function to the sensor's trajectory such as that of Furgale *et al.* [35], however the work in this thesis is only concerned with the more common approach of estimating a sequence of distinct camera poses.

When discussing map representations there is much more variety in the kinds of outputs SLAM systems produce, where the context of the contributions of this thesis become more apparent. Early 2D laser scan SLAM would typically produce quite sparse point-based maps of explored environments that resembled building floor plans in appearance, shown in Figure 1.1. These kinds of maps are quite useful for basic 2D robot localisation in indoor environments [20], but do not yield very much information in terms of scene understanding beyond building layout. Later work using 3D laser scanners produces similar maps in term of point density and appearance (although in three dimensions), but relies on even more costly full 3D laser scanning sensors which are not widely available or as portable as some handheld sensors, while also sometimes being inapplicable in real-time.

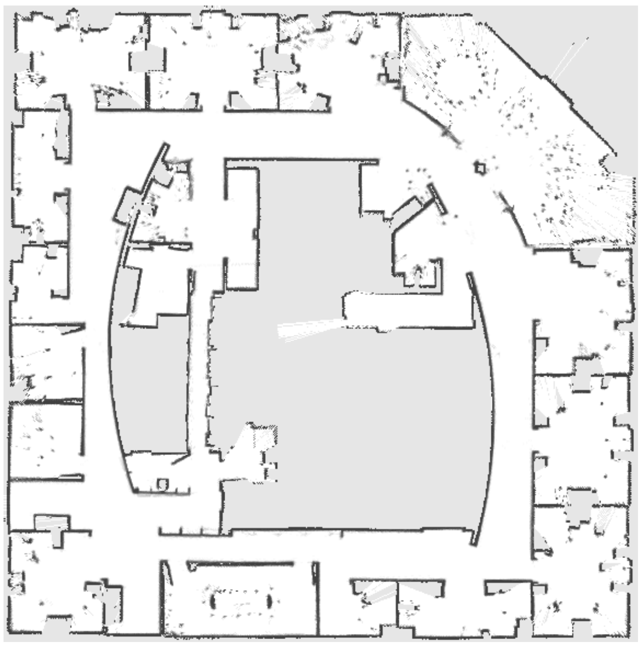


Figure 1.1: The Intel Research Lab dataset, one of the most well known 2D laser scan SLAM sequences in the community [48].

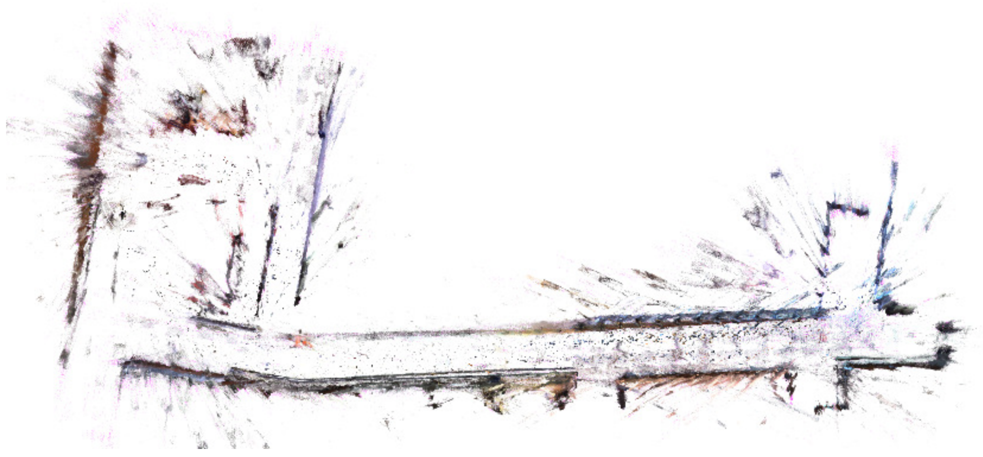


Figure 1.2: Plan view of a stereo visual SLAM keypoint-based map [80]. The sparse structure of the map only made up of image keypoints is apparent.

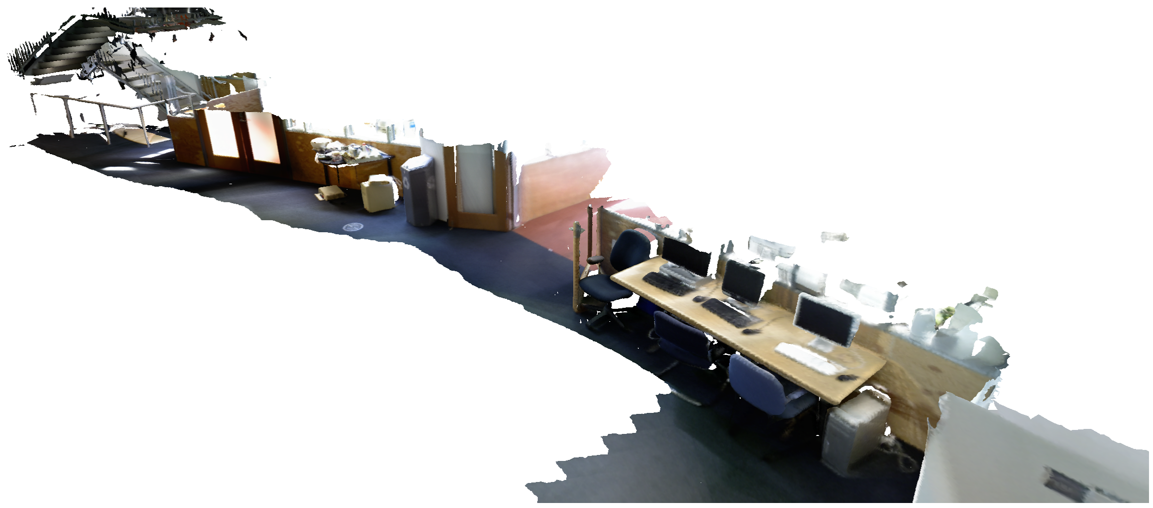


Figure 1.3: Dense mesh-based map representation captured by the SLAM system described in this thesis.

There are a number of map representations producible with visual SLAM systems with the most common being keypoint-based sparse 3D reconstructions. Both monocular SLAM systems (using a single camera, e.g. [17]) and stereo SLAM systems (using a pair of cameras to estimate depth, e.g. [80]) can be used to produce these kinds of maps in real-time, with low cost standard camera technologies. An example of these kinds of maps is shown in Figure 1.2 where the map is only made up of a sparse set of image keypoints used in estimating the camera's trajectory. Again while useful for localisation and coarse-grained building mapping, this kind of map representation is less useful for general scene understanding and reasoning about free space and other processes that require knowledge of complete surfaces. Other approaches which use standard passive RGB cameras can produce complete surface maps using dense whole image-based techniques (such as that of Newcombe *et al.* [87]), however these SLAM systems are very much restricted in the area over which they can operate, typically only in desk sized scenes.

The map representation used by the system presented in this thesis is what is typically referred to as dense, where each pixel in the input data is used to reconstruct the final map. This is partially made possible by the use of a depth sensor as the primary input. Typically referred to as RGB-D sensing (Red, Green, Blue, Depth), every image captured by the camera provides an estimated depth to each pixel as well as color. This can be seen as "shortcutting" the depth estimation problem normally tackled in stereo vision setups, or the *structure from motion* problem involved in monocular SLAM (later discussed in Chapter 2). Newcombe *et al.*'s work on KinectFusion exploited this type of data to create dense volumetric surface reconstructions of desk sized scenes [86], similar in appearance to those which could be created with the DTAM monocular SLAM system [87]. The work in this thesis uses the same volumetric representation as KinectFusion in the frontend but converts it to a more scalable dense mesh-based representation in the final backend map representation, allowing massively scalable dense mapping over large scales. This kind of map representation is shown in Figure 1.3. It can be seen that in contrast to other SLAM systems the reconstructions obtained with our approach are useful in many contexts including building layout analysis, 3D scene understanding and motion planning due to their global consistency, high fidelity and full surface reconstruction characteristics.

1.3 Scope

The goal of this thesis is to create a real-time large scale dense visual SLAM system based on RGB-D data that provides high quality surface reconstructions useful for higher level autonomous robotic tasks. Newcombe *et al.*'s seminal work on real-time dense RGB-D based fused surface reconstruction [86] is a key foundation on which the goals of this thesis are realised. Our work goes beyond previously presented dense RGB-D based visual SLAM systems and is the first of its kind to (i) provide a high quality fused surface reconstruction over extended scales in real-time, (ii) use both dense geometric and photometric information in camera pose estimation, (iii)

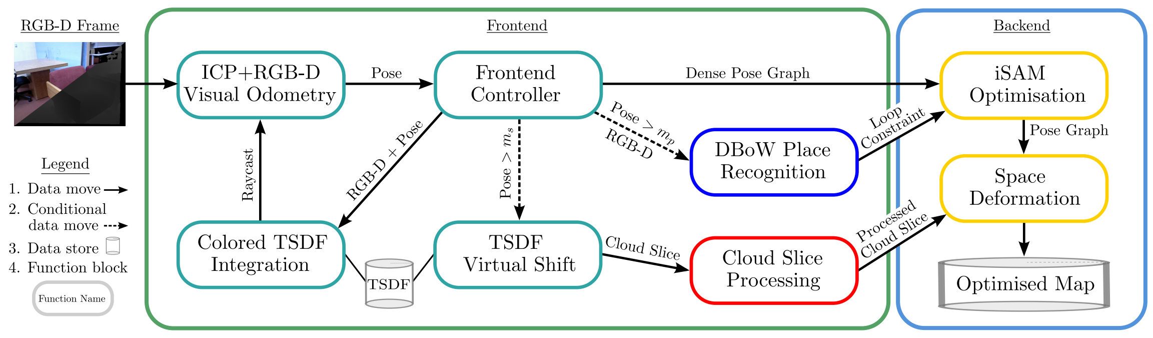


Figure 1.4: System architecture diagram. The major components of the system include; (i) The TSDF Virtual Shift, (ii) ICP+RGB-D Visual Odometry, (iii) Map Space Deformation. Not shown is the planar simplification component, which runs directly off the system output.

provide an up-to-date globally consistent representation of the map in real-time given detected loop closures, and (iv) provide a simplified planar representation of the mapped environment in addition to a dense mesh-based reconstruction.

Our system architecture for accomplishing these goals is shown in Figure 1.4, which follows the approach of a typical SLAM system comprised of a frontend and a backend. The frontend is concerned with camera pose estimation, local fused surface reconstruction and place recognition while the backend performs pose graph optimisation along with dense map optimisation to ensure global reconstruction consistency. Being highly modular in design the framework is also amenable to the addition of new components and substitution of existing components.

As part of this thesis we introduce efficient GPU-based methods for both extended scale fused surface reconstruction and camera pose estimation inclusive of photometric information. Processing full frame RGB-D data at camera framerate of 30Hz requires computation over 45MB of raw data per second, which necessitates highly parallel GPU-based methods to maintain reliable online operation. The system's hierarchical multi-threaded architecture facilitates the creation and integration of new modules with minimal impact on the performance of the overall system.

Incorporating large updates to expansive dense 3D mesh-based maps in an efficient manner is a non-trivial task that is necessary when not relying on raw point cloud reprojection for scene reconstruction. In our work we employ a novel parameterisation of a surface-wide non-rigid space deformation exploiting the information gained through pose graph optimisation. This enables the capture of smooth, locally and globally consistent dense mesh-based maps in real-time over large trajectories.

We provide a comprehensive qualitative and quantitative analysis of many aspects of the system's performance, demonstrating large scale real-time globally consistent fused surface reconstruction and high quality planar simplification which yields high level information about the area reconstructed. The experimental datasets include both real-world and synthetic data, as well as a standard RGB-D benchmark and numerous other datasets captured in both a handheld fashion and onboard a wheeled robotic platform.

1.4 Publications

The work described in this thesis has appeared in the following publications:

Real-time Large Scale Dense RGB-D SLAM with Volumetric Fusion

T. Whelan, M. Kaess, H. Johannsson, M.F. Fallon, J. J. Leonard and J.B. McDonald. *International Journal of Robotics Research Special Issue on Robot Vision, 2014* [129]

Incremental and Batch Planar Simplification of Dense Point Cloud Maps

T. Whelan, L. Ma, E. Bondarev, P. H. N. de With, and J.B. McDonald. *Robotics and Autonomous Systems ECMR '13 Special Issue, 2014* [130]

3D Mapping, Localisation and Object Retrieval using Low Cost Robotic Platforms: A Robotic Search Engine for the Real-world

T. Whelan, M. Kaess, R. Finman, M.F. Fallon, H. Johannsson, J.J. Leonard and J.B. McDonald. *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras, (Berkeley, USA), July 2014* [128]

Deformation-based Loop Closure for Large Scale Dense RGB-D SLAM

T. Whelan, M. Kaess, J.J. Leonard, and J.B. McDonald. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS, (Tokyo, Japan), November 2013* [127]

Planar Simplification and Texturing of Dense Point Cloud Maps

L. Ma, T. Whelan, E. Bondarev, P. H. N. de With, and J.B. McDonald. *European Conference on Mobile Robotics, ECMR, (Barcelona, Spain), September 2013* [76]

Robust Real-Time Visual Odometry for Dense RGB-D Mapping

T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, and J.B. McDonald. *IEEE Intl. Conf. on Robotics and Automation, ICRA, (Karlsruhe, Germany), May 2013* [126]

Kintinuous: Spatially Extended KinectFusion

T. Whelan, M. Kaess, M.F. Fallon, H. Johannsson, J.J. Leonard and J.B. McDonald. *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras, (Sydney, Australia), July 2012* [125]

1.5 Structure

The remainder of this thesis is structured as follows. Chapter 2 provides an overview of the SLAM problem and an introduction to visual SLAM. Following on from this the chapter will provide a discussion on dense methods for 3D reconstruction and mapping. The chapter will end with an in-depth review of the related work relevant to the research presented in this thesis.

Our first contribution, achieving extended scale fused dense real-time RGB-D mapping, is fully described in Chapter 3. An introductory background on volumetric fusion is provided in this chapter, along with our method for extended scale mapping, TSDF parameterisation, surface reconstruction and color estimation.

Chapter 4 describes our second contribution, robust dense visual odometry for volumetric fusion-based mapping. In this chapter we formulate methods for both geometry-based and photometrics-based camera pose estimation. Following on from this we provide our method for combining geometric and photometric pose estimation into a unified estimate. This chapter also provides a detailed description of how GPU-based parallelisation of the photometric component of the technique was implemented.

Chapter 5 describes our approach to combining the techniques outlined in Chapters 3 and 4 with a novel method for performing online dense loop closure to complete our large scale dense visual SLAM system. This is the third contribution of the thesis. In this chapter we provide details of our SLAM framework and each component involved, specifically place recognition, space deformation and map optimisation.

Chapter 6 provides a description of our fourth contribution, a method for performing planar simplification and triangulation of dense point cloud maps (such as those produced by our system outlined in Chapter 5). We outline two processing frameworks in this chapter, one for batch processing and one for online incremental processing. The method we use for planar segmentation is provided, as well as descriptions of our methods for segment decimation, triangulation and texturing.

In Chapter 7 we provide extensive experimental evaluation of many different aspects of the presented systems. These include qualitative and quantitative evaluations of camera pose estimation, surface reconstruction quality, segmentation performance, triangulation quality and computational performance.

Chapter 8 describes two case studies where the contributions of this thesis have been applied. The first of these is work on point cloud segmentation carried out by Finman *et al.* [30, 31]. The second is a fully closed-loop autonomous robotic system which marries techniques presented in this thesis with existing related work to solve a simple robotic task.

Finally, Chapter 9 reviews the contributions of this thesis and discusses future research directions. In particular we focus on real-time robotics applications and the future of dense methods for autonomous platforms.

CHAPTER 2

Visual Simultaneous Localisation and Mapping

The SLAM problem poses the task of estimating a map of an unknown environment whilst simultaneously localising the sensing platform within that map. A wide variety of sensors have been applied to the SLAM problem where visual SLAM refers to the instances where one or more cameras are used as the primary sensor input. In this chapter we will provide an overview of the SLAM problem including its history and the most widely applied solutions to date. We will also specifically review visual SLAM before moving onto a detailed overview of dense mapping and work specifically related to RGB-D based localisation and mapping.

2.1 The SLAM Problem

Although historically embedded in the robotics community, the problem phrased as SLAM is strongly related to earlier research in communities such as photogrammetry where it is referred to as *bundle adjustment* and the computer vision community where it is known as *structure from motion* [120]. Although the same problem in principle, the main differentiator between SLAM and other formulations is the focus on incremental solutions and real-time robotics applications. A situation where SLAM would typically be applied involves a mobile robotic platform which requires an estimate of its position within a newly explored environment in real-time in order to make high level decisions for navigation and other purposes. For this reason many of the work on SLAM within the robotics community has focussed on computationally efficient and scalable solutions to the problem.

Smith and Cheeseman are typically credited with carrying out the seminal work on the SLAM problem within the robotics community [106]. Their work set the initial paradigm for approaching the problem by providing a means of estimating the uncertainty and correlations between both the sensor location and the environment structure through use of Approximate Transforms (ATs) estimated with the *extended Kalman filter* (EKF) [107]. The EKF approach quickly became a key part of subsequent real-time SLAM systems [17, 8, 88], but has a number of limitations. The most significant of these is the poor scalability of the filtering-based approach. Although the state vector only grows linearly with the size of the map, the covariance grows quadratically (and hence the space and computational complexity as well). This characteristic of the EKF translates into a strict bound on the size of the map which can be computationally feasibly estimated in real-time.

This limitation resulted in a large amount of subsequent research on filter-based SLAM including local submap filtering by Williams [131], sequential map joining by Tardos *et al.* [118], and the compressed filter algorithm by Guivant and Nebot [40]. However these approaches failed to cancel out the quadratic complexity of the process, only reducing the computation required by a constant factor. One notable exception is the Hierarchical SLAM algorithm of Estrada *et al.* which managed to achieve linear complexity [28]. Another issue with the EKF approach is the assumed unimodal distribution, which meant there was an inherent inability for multiple hypotheses to be naturally represented. This again limits the applicability of the filter in scenarios where perceptual aliasing is encountered or global localisation is required in spite of ambiguity. This manifests itself in the classic *kidnapped robot* problem [119].

The advent of the application of Monte-Carlo methods to robot localisation problems brought with it a much more scalable and computationally favorable means of state representation [20]. Otherwise known as particle filters, these techniques provided a solution to the scalability issues associated with EKF-based SLAM systems. The most notable of these is the FastSLAM system of Montemerlo *et al.* [83]. One of the key intuitions of their system was their Rao-Blackwellization of the problem which represented the posterior distribution as a particle filter over the robot's path while allocating an independent Kalman filter for each landmark associated with each particle. Central to the computational tractability of this approach is the combination of conditional independence with Rao-Blackwellization coupled with the use of a tree-based data structure for map management. The FastSLAM approach provides a solution to global localisation and ambiguity problems by naturally supporting the approximation of multimodal distributions.

Although proven to be successful in many applications, Julier and Uhlmann showed that over extended timescales the state estimates provided by filter-based approaches are guaranteed to become inconsistent [54]. This motion against filter-based SLAM was later further argued on a different point by Strasdat *et al.* concluding that *bundle adjustment* is more efficient than filtering for visual SLAM [111]. An alternative SLAM paradigm to EKFs and particle filters is sometimes referred to as *smoothing* or pose graph SLAM. This paradigm has become extremely popular in recent years showing mapping capabilities at scales far beyond what is capable with filter-based approaches. In pose graph SLAM the problem is represented as a set of nodes and edges in a graph, where the nodes represent the latent variables and the edges represent the measurement constraints between the variables.

Lu and Milios first introduced this representation but chose to only include the sensor poses in the graph and rely on lidar scan projection for map reconstruction [74]. Since then a large amount of research on graph-based solutions to the SLAM problem have been published [91, 71, 69, 41, 19]. One of the most significant qualities of graph-based approaches over filter-based approaches is the fact that the optimisation to solve the SLAM problem maintains all information about all poses that have been recorded so far. This is in contrast to filter-based approaches which marginalise out all previous poses over time. In addition to this the graphical formulation has direct parallels to sparse linear algebra matrix optimisation techniques which have been successfully exploited to achieve highly scalable solutions [56, 57]. In particular in this thesis we make use of the incremental smoothing and mapping (iSAM) approach of Kaess *et al.* as an efficient means of solving the SLAM problem over the pose graph [56].

iSAM formulates the SLAM problem in the form of a factor graph which consists of variables nodes and factor nodes. Variables nodes are connected to factor nodes by edges in the graph, where a factor node represents a function over all of the variables to which it is connected. As discussed this formulation of the pose graph results in a sparse linear algebra factorisation problem which iSAM exploits to solve the system efficiently. In addition to this, one of the key contributions of iSAM is the efficient and exact method for incrementally updating the factorisation of the system by only recomputing elements of the matrix which actually change [55]. This is of enormous value in real-time settings where new poses are added to the graph incrementally online by the SLAM frontend and there is no longer a need to completely solve the entire system each time this occurs. More recent work on graph-based SLAM looks at robust and efficient means for dealing with outliers in the graph [92, 1, 117, 72]. There has also been focus in recent times on reducing the complexity of the SLAM problem as it scales over time [53] and as graph complexity grows [75].

2.2 Visual SLAM

While the majority of early SLAM systems used sensors such as radar, sonar and lidar to perceive the environment, it wasn't long before systems based on visual information came to the forefront. One of the first was an active stereo vision-based SLAM system that was developed by Davison in his thesis which could capture a consistent map of sparse landmark features [18]. Following on from this was the breakthrough work on visual SLAM using a single monocular camera, also by Davison, in his MonoSLAM system [17]. This approach applied the aforementioned EKF method to SLAM in the domain of monocular vision. MonoSLAM was capable of estimating the pose of the camera in real-time while simultaneously estimating the locations of a sparse set of visual feature points, maintaining full covariance over the entire state vector. One of the key contributions of this work was the introduction of an inverse depth parameterisation for visual landmark features [84].

Many visual SLAM systems have a high degree of focus on real-time operation. The PTAM system of Klein and Murray is one of the key milestones in visual SLAM, where real-time operation was essential to accomplish convincing augmented reality tasks in arbitrary small scale workspaces [65]. Again using only monocular visual information, the PTAM system separated the processes of estimating the camera's pose and reconstructing the map into two independent parallel threads. This design choice was motivated by the proposition that only tracking needs to be carried out at framerate, while the estimation of the sparse feature map can be carried out in a lower priority intermittent batch thread. Their system avoided the propagation of linearisation errors commonly associated with EKF-based approaches by applying bundle adjustment across a set of wide-baseline keyframes.

Extensions were made to PTAM to extend it to larger scales, such as Castle *et al.* who adopted a submapping approach [9]. Though successful in mapping building scale environments, the submaps were only topologically connected in the keyframe space and as a result a global estimate of the environment was unavailable. Other more specifically tailored solutions to large scale visual SLAM include those of Konolige *et al.* [67, 68, 70] who presented the FrameSLAM system for large scale mapping, those of Sibley *et al.* [103, 104] who combined appearance-based place recognition with relative bundle adjustment and that of McDonald *et al.* [80] which exploits the use of anchor nodes to achieve multi-session visual SLAM over large scales. Up to this point in visual SLAM sparse visual feature maps were the standard. However with an increase in computational power available more recent visual SLAM approaches have looked beyond sparse features towards richer, denser mapping techniques which we discuss in the following sections.

2.3 Dense Mapping

One of the earliest methods for real-time handheld vision-based dense 3D reconstruction was presented by Newcombe and Davison [85]. They leverage the monocular SLAM system of Klein and Murray [65] to bootstrap a dense depth map reconstruction based on view-predictive optical flow and constrained scene flow. Soon after Newcombe *et al.* published the DTAM system [87], capable of live dense 3D reconstruction of small scale scenes again with only a monocular camera. Their work made use of an inverse depth cost volume with GPU accelerated inverse depth map regularisation. Closely related to this system is the work of Stühmer *et al.* which adopts an arguably simpler method that directly estimates a depth map using a variational approach without a cost volume [114]. A notable recent approach to real-time monocular reconstruction is the semi-dense method of Engel *et al.* [27], which estimates the inverse depth of all pixels with a non-negligible image gradient with a Gaussian probability distribution.

When the Microsoft Kinect RGB-D sensor became available to the research community it was quickly adopted across a number of different fields due to a number of desirable characteristics. It was the first sensor of its kind to provide high resolution (640×480) depth information as well as color information at a 30Hz framerate at such a low cost. This made it immediately applicable to a number of problems in robotic perception and in particular to visual SLAM. One of the earliest SLAM systems built using the Kinect was that of Henry *et al.* [43]. Later, KinectFusion brought with it a strong focus on real-time dense 3D volumetric reconstruction at camera frame rate [86]. KinectFusion married the increasing popularity in GPGPU with the high quality real-time full-frame depth maps provided by the Kinect sensor to produce a system capable of camera frame rate dense 3D reconstruction. KinectFusion was the first system able to produce full volumetric 3D reconstructions of desk sized scenes at subcentimetre resolution in real-time, using only a low cost widely available sensor.

Central to the functionality of KinectFusion is the use of a volumetric data structure for representing the reconstruction. The truncated signed distance function (TSDF), first introduced by Curless and Levoy [14], provides a convenient scene representation which allows computationally efficient means of fusing depth map information while essentially providing a moving average of the scene being reconstructed, resulting in a much smoother model. The TSDF as a data structure is very amenable to parallel processing and hence perfect for use with GPGPU for enhanced computational performance. This is one of the keys to KinectFusion's real-time functionality.

2.4 Related Work

In contrast to the previous section, this section will provide a detailed overview of the related work specifically using RGB-D devices over other sensors for visual mapping and localisation. A large number of papers have been published over the last few years specifically using RGB-D data for camera pose estimation, dense mapping and full SLAM pipelines. While many visual SLAM systems existed prior to the advent of active RGB-D sensors (e.g. Comport *et al.* [12]), we will focus mainly on the literature which makes specific use of active RGB-D platforms. One of the earliest RGB-D tracking and mapping systems used FAST feature correspondences between frames for visual odometry and offloaded dense point cloud map building to a post-processing step utilising sparse bundle adjustment (SBA) for global consistency by minimising feature reprojection error [50]. One of the first real-time dense RGB-D tracking and mapping systems estimates an image warping function with both geometric and photometric information to compute a camera pose estimate, however only relied on rigid reprojection for point cloud map reconstruction without using a method for global consistency [3]. Similar work on dense RGB-D camera tracking was done by Steinbrücker *et al.*, also estimating an image warping function based on geometric and photometric information [108]. Recent work by Kerl *et al.* presents a more robust dense photometrics-based RGB-D visual odometry system that proposes a t-distribution-based error model which more accurately matches the residual error between RGB-D frames in scenes that are not entirely static [62].

Henry *et al.* presented one of the first full SLAM systems based entirely upon RGB-D data, using visual feature matching with Generalised Iterative Closest Point (GICP) to build up a pose graph and following that an optimised surfel map of the area explored [43]. The use of pose graph optimisation versus SBA is evaluated, minimising feature reprojection error in an offline rigid transformation framework. Visual feature correspondences are used in conjunction with pose graph optimisation in the RGB-D SLAM system of Endres *et al.* [26]. An octree-based volumetric representation is used to store the map, created by reprojecting all point measurements into the global frame. This map representation is provided by the OctoMap framework of Hornung *et al.*, which includes the ability to take measurement uncertainties into account and implicitly represent free and occupied space while being space efficient [47].

An explicit voxel volumetric occupancy representation is used by Pirker *et al.* in their GPSlam system which uses sparse visual feature correspondences for camera pose estimation [95]. They make use of visual place recognition and sliding window bundle adjustment in a pose graph optimisation framework. To achieve global consistency the occupancy grid is "morphed" by a weighted average of the log-odds perceptions of each camera for each voxel. Stückler *et al.* register surfel maps together for camera pose estimation and store a multi-resolution surfel map in an octree, using pose graph optimisation for global consistency [113]. After pose graph optimisation is complete a globally consistent map is created by fusing key views together. In recent work Hu *et al.* proposed a system that uses bundle adjustment in order to make use of pixels for which no valid depth exists [49], and Lee *et al.* presented a system which exploits GPU processing power for real-time camera tracking [73]. Both systems produce an optimised map as a final step in the process. Kerl *et al.* use their previously discussed dense keyframe-based visual odometry system in a pose graph optimisation framework to build a dense visual SLAM system that produces keyframe-based globally consistent maps after a final post-processing optimisation step [63].

A substantial number of derived works have been published recently after the advent of the KinectFusion system of Newcombe *et al.* [86], mostly focused on extending the range of operation, with other related work on object recognition and motion planning [59, 124]. Recent work by Bylow *et al.* and Canelhas *et al.* directly tracks the camera pose against the accumulated volumetric model by exploiting the fact that the TSDF representation used by KinectFusion stores the signed distance to the closest surface at voxels near the surface [6, 7]. This avoids the need to raycast a vertex map for each frame to perform camera pose estimation, which potentially discards information about the surface reconstruction.

Roth and Vona extend the operational range of KinectFusion by using a double buffering mechanism to map between volumetric models upon camera translation and rotation, using a voxel interpolation for the latter [97]. However no method for recovering the map is provided. Zeng *et al.* replace the explicit voxel representation used by KinectFusion with an octree representation which allows mapping of areas up to 8m×8m×8m in size [132]. However this method does increase the chance for drift within the map and provides no means of loop closure or map correction. Steinbrücker *et al.* make use of a multi-scale octree to represent the signed distance function, allowing full color reconstructions of scenes as large as an entire corridor including nine rooms spanning a total area of 45m×12m×3.4m [109]. After an RGB-D sequence has been processed, a globally consistent camera trajectory is resolved and the model is reconstructed. Keller *et al.* present an extended fusion system made space efficient by using a point-based surfel representation, although with no method for drift correction or loop closure detection [61]. Chen *et al.* present a novel hierarchical data structure that enables extremely space efficient volumetric fusion, using a streaming framework allowing effectively unbounded mapping range, limited only by available memory [10]. However the system lacks any method for mitigating drift or enforcing global consistency. Nießner *et al.* present an alternative space efficient method for large scale dense fusion that uses an intelligent voxel hashing function to minimise the amount of memory required for reconstruction, but again without a means of correcting for drift [89].

An alternative approach to the modern SLAM problem (making use of higher level semantic information) is introduced by Salas-Moreno *et al.*, whereby known objects are detected, tracked and mapped in real-time in a dense RGB-D framework known as SLAM++ [99]. Pose graph optimisation is used to ensure global consistency on the level of camera poses and detected object positions. This does allow loop closure, however less influence is placed on a full scene reconstruction with only point cloud reprojections being used for mapped loop closure. Recent work by Henry *et al.* uses multiple smaller "patch volumes" to segment the mapped space into a set of discrete TSDFs, each with a 6-degrees-of-freedom (6-DOF) pose which is rigidly optimised upon loop closure detection [45]. This approach can be seen as similar to the SLAM++ approach of Salas-Moreno *et al.* whereby the patch volumes are analogous to objects. While achieving global consistency between each volume, there is no clear solution presented for correcting the surface within any one given volume or stitching surfaces which are split between volumes, leaving local surfaces disconnected.

Zhou *et al.* present an impressive method for reconstructing 3D scenes that specifically targets the high-frequency noise and low-frequency distortion effects often encountered with RGB-D data [134]. By reconstructing fragments of the scene which are then aligned and deformed very high quality reconstructions can be obtained, however in what is a strictly offline framework. Similar work also by Zhou and Koltun presents a method which detects points of interest in a scene and specifically optimises the camera trajectory to preserve detailed geometry around these points, within an offline framework [133].

An number of approaches that rely on keyframes have been developed to tackle the problem of RGB-D mapping and SLAM. Tykkälä *et al.* present a system which uses real-time dense photometric keyframe-based camera tracking to determine a camera trajectory around an indoor environment [121]. Individual RGB-D frames are also fused into existing keyframes to improve reconstruction quality. An optional bundle adjustment step can then be taken to optimise the camera poses before a watertight Poisson mesh reconstruction is computed as a post-processing step. Meilland and Comport propose a model that unifies the benefits of a dense voxel-based representation with a keyframe representation allowing high quality dense mapping over large-scales, although without detecting large loop closures or correcting for drift [81]. An intelligent forward composition approach is proposed which enables efficient combination of reference images to create a single predicted frame without repeated redundant image warps. In our work we chose to avoid a keyframe approach in spite of the resulting higher memory requirement. A fully 3D voxel-based method, such as the one presented in this thesis, removes the need to implement specific schemes to overcome the problems associated with reconstructing complex non-concave objects and non-convex scenes.

On the topic of map representation and planar simplification, triangular meshing of 3D point clouds is a well-studied problem with many existing solutions. One class of triangulation algorithms computes a mathematical model prior to triangulation to ensure a smooth mesh while being robust to noise [60, 52]. This type of algorithm assumes surfaces are continuous without holes, which is usually not the case in open scene scans or maps acquired with typical robotic sensors. Another class of algorithms connects points directly, mostly being optimised for high-quality point clouds with low noise and uniform density. While these algorithms retain fine details in objects [5, 101], they are again less applicable to noisy datasets captured with an RGB-D or lidar sensor, where occlusions create large discontinuities.

With real-world environment triangulation in mind, the Greedy Projection Triangulation (GPT) algorithm has been developed [39, 79]. The algorithm creates triangles in an incremental mesh-growing approach, yielding fast and accurate triangulations. However, the GPT algorithm keeps all available points to preserve geometry, which is not always necessary for point clouds containing surfaces that are easily characterised by geometric primitives. To solve this problem a hybrid triangulation method was developed by Ma *et al.*, where point clouds are segmented into planar and non-planar regions for separate triangulation [77]. The QuadTree-Based (QTB) algorithm was developed to decimate planar segments prior to triangulation. The QTB algorithm significantly reduces the amount of redundant points, although a number of limitations degrade its performance. For example, the algorithm does not guarantee that final planar points will lie inside the original planar region, which can lead to noticeable shape distortion. The algorithm also produces duplicate vertices, overlapping triangles and artificial holes along the boundary.

There is a vast amount of literature on planar segmentation available. Oehler *et al.* adopt a multi-resolution approach that relies on using a Hough transform over co-planar clusters of surfels, ultimately relying on RANSAC for the plane fitting component [90]. Deschaud and Goulette use a region growing approach made robust to noise by growing in a voxel space over the input data rather than the raw points themselves [21]. Some algorithms extend 2D graph cut theory towards 3D point cloud data [38, 37, 112]. These algorithms are designed for general object segmentation and their complexity is in general too high for plane detection, unlike the low-complexity algorithm proposed by Rabbani *et al.* [96], which imposes a smoothness constraint on segmentation. However, like in previous work these methods are all concerned with batch processing scenarios rather than the incremental segmentation growing scenario [77]. Related also is the work of Ruhnke *et al.* which looks at simplifying dense point cloud maps by learning local surface attributes via sparse coding, rather than explicitly extracting geometric primitives [98].

As discussed there exists a large number of systems utilising RGB-D data for SLAM and related problems. However, most are either unable to (i) operate in real-time, (ii) provide an up-to-date optimised representation of the map at runtime or any time it is requested or (iii) efficiently incorporate large non-rigid updates to the map. Non-rigid surface correction is of great interest specifically in the realm of volumetric fusion as typically reconstructions are locally highly accurate but drift slowly over large scales over time, where a smooth continuous deformation of the surface is most suitable for correction. Additionally, while there is a significant amount of existing work on planar simplification and triangulation, none of the methods discussed above are optimal for the type of data we wish to process in a potentially real-time framework. In the following chapters we will fully describe our approach to RGB-D SLAM with volumetric fusion which is capable of functioning in real-time over large scale trajectories, while efficiently applying non-rigid updates to the dense map upon loop closure to ensure global consistency. We will also describe our method for efficient planar simplification and triangulation of dense point cloud maps, in both batch and incremental online settings.

CHAPTER 3

Extended Scale Dense Mapping

In this chapter we will provide some background on the usage of volumetric fusion for dense RGB-D based tracking and mapping and describe our extension to KinectFusion, the most widely cited system that employs this approach, to allow spatially extended mapping.

3.1 Background

Real-time volumetric fusion with RGB-D cameras was brought to the forefront by Newcombe *et al.* with the KinectFusion system [86]. A significant component of the system is the cyclical pipeline used for camera tracking and scene mapping, whereby full depth maps are fused into a volumetric data structure, which is then raycast to produce a predicted surface that the subsequently captured depth map is matched against using ICP. We review the method for integrating new depth maps into the volumetric data structure in Section 3.2, while our method for tracking the camera

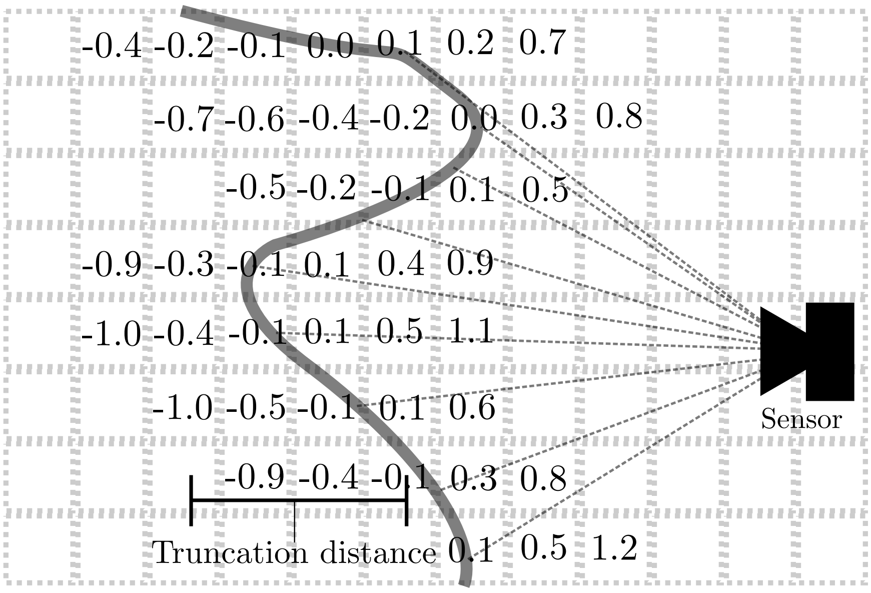


Figure 3.1: Two dimensional example of the structure of the truncated signed distance function representation of an implicit surface. Shown are example signed distance values stored at voxels within the truncation distance of the observed surface, with rays cast from the observing sensor.

motion against the predicted surface model is described in Chapter 4. The truncated signed distance function (TSDF) is the volumetric data structure used that encodes implicit surfaces by storing the signed distance to the closest surface at each voxel up to a given truncation distance from the actual surface position. Points at which the sign of the distance value changes are known as zero crossings, which represent the actual position of the surface, shown in Figure 3.1. Each voxel also stores a weight for the distance measurement at that point, effectively providing a moving average of the surface position. In the case of KinectFusion, the TSDF is stored as a three dimensional voxel grid in GPU memory where dense depth map integration is accomplished by sweeping through the volume and updating distance measurements accordingly, while surface raycasting is carried out by simply projecting rays from the current camera pose and returning the depth and surface normals at the first zero crossings encountered. Surface normals are easily computed by taking the finite difference around a given position within the TSDF, as exploited by Bylow *et al.* [6]

and Canelhas *et al.* [7]. The entire process is very amenable to parallelisation and greatly benefits in execution time from being implemented on a GPU. A point to note is that the TSDF representation has a minimal surface thickness limitation imposed by the selected truncation distance. This manifests itself in the fact that two surfaces closer than the selected truncation distance will effectively “overwrite” each other when observed. This problem was highlighted and explored by Henry *et al.* in their work on multiple fusion volumes [44].

3.2 Volume Representation

Defining the voxel space domain as $\Psi \subset \mathbb{N}^3$ the TSDF volume S at some location $\mathbf{s} \in \Psi$ has the mapping $S(\mathbf{s}) : \Psi \to \mathbb{R} \times \mathbb{N} \times \mathbb{N}^3$. Within GPU memory the TSDF is represented as a 3D array of voxels. Each voxel contains a signed distance value $S(\mathbf{s})^D$, a weight value $S(\mathbf{s})^W$ and a value for each color component R, G and B ($S(\mathbf{s})^R, S(\mathbf{s})^G, S(\mathbf{s})^B$). The integration of new surface measurements (i.e. depth map values projected into the TSDF from the current camera pose) is carried out in the same fashion as Newcombe *et al.*, when integrating a new signed distance function measurement $S(\mathbf{s})^D_i$ during the fusion of a new depth map, each voxel $\mathbf{s} \in \Psi$ at time i is updated with:

$$S(\mathbf{s})^{D'}_i = \frac{S(\mathbf{s})^W_{i-1} S(\mathbf{s})^D_{i-1} + S(\mathbf{s})^W_i S(\mathbf{s})^D_i}{S(\mathbf{s})^W_{i-1} + S(\mathbf{s})^W_i} \tag{3.1}$$

$$S(\mathbf{s})^{W'}_i = \min(S(\mathbf{s})^W_{i-1} + S(\mathbf{s})^W_i, max\_weight) \tag{3.2}$$

As is the case with previous approaches, we take $S(\mathbf{s})^W_i = 1$ to provide a simple moving average. Bylow *et al.* have experimented with different weighting schemes [6], however we have found the original value of 1 used by Newcombe *et al.* to provide good performance [86]. Using only a cubic volume, we parameterise the TSDF by the

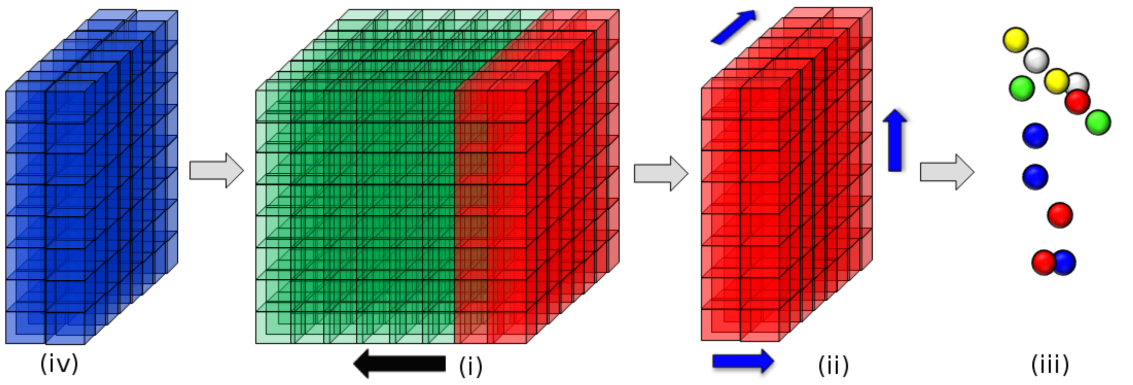


Figure 3.2: Visualisation of the volume shifting process for spatially extended mapping; (i) The camera motion exceeds the movement threshold m_s (direction of camera motion shown by the black arrow); (ii) Volume slice leaving the volume (red) is raycast along all three axes to extract surface points and reset to free space; (iii) The raycast surface is extracted as a point cloud and fed into the Greedy Projection Triangulation (GPT) algorithm of Marton *et al.* [79]; (iv) New region of space (blue) enters the volume and is integrated using new modulo addressing of the volume.

side length in voxels, v_s, and the dimension in metres, v_d. Both of these parameters control the resolution of the reconstruction along with the size of the immediate "active area" of reconstruction. In all experiments we set $v_s = 512$ for total GPU memory usage of 768MB, where each signed distance value is a truncated float16 and each weight and color component value is an unsigned int8. The 6-DOF camera pose within the TSDF at time i is denoted as P_i^T, composed of a rotation $\mathbf{R}_i^T \in \mathbb{SO}_3$ and a translation $\mathbf{t}_i^T \in \mathbb{R}^3$. The origin of the TSDF coordinate system is positioned at the center of the volume with basis vectors aligned with the axes of the TSDF. Initially $\mathbf{R}_0^T = \mathbf{I}$ and $\mathbf{t}_0^T = (0, 0, 0)^\top$. The position of the TSDF volume in voxel units in the global frame is defined by $\mathbf{g}_i$ and initialised to be $\mathbf{g}_0 = (0, 0, 0)^\top$. Note that the superscript ^{T} refers to a pose within the TSDF coordinate frame and not the transpose $^\top$ operator.

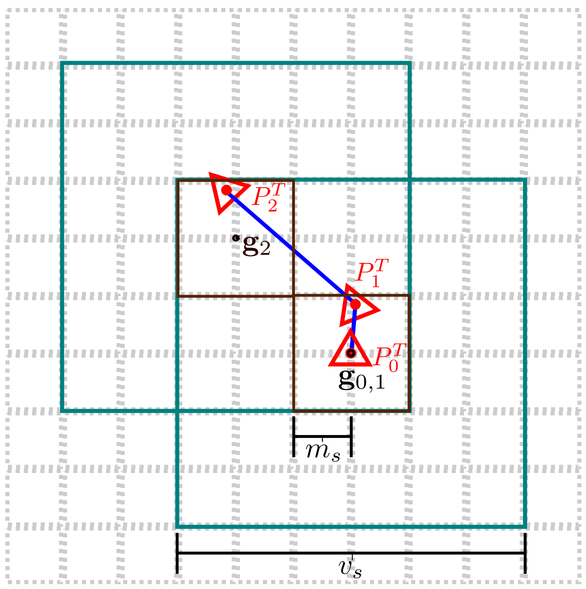


Figure 3.3: Visualisation of the interaction between the movement threshold m_s and the shifting process. Between frames 0 and 1 the camera does not cross the movement boundary (dark brown) and no shift occurs. At frame 2, the pose crosses the boundary and causes a volume shift, recentering the volume (teal) around P_2^T and updating $\mathbf{g}_2$. The underlying voxel grid quantisation is shown in light dashed lines.

3.3 Volume Shifting

Unlike the original work of Newcombe *et al.*, camera pose estimation and surface reconstruction is not restricted to only the region around which the TSDF was initialised. By employing modulo arithmetic in how the TSDF volume is addressed in GPU memory we can treat the structure like a cyclical buffer which virtually translates as the camera moves through an environment. Figure 3.2 provides a visual example and description of the shifting process. It is parameterised by an integer movement threshold m_s, defining the cubic movement boundary (in voxels) around $\mathbf{g}_i$ which upon crossing, causes a volume shift, shown in Figure 3.3. Discussion on the choice of value for m_s is provided in Chapter 7. Each dimension is treated independently during a shift. When a shift is triggered, the TSDF is virtually translated about the camera pose (in voxel units) to bring the camera's position to within one voxel of $\mathbf{g}_{i+1}$. The new pose of the camera P_{i+1}^T has no change in rotation, while the shift corrected camera translation $\mathbf{t}_{i+1}^{T'}$ is calculated from $\mathbf{t}_{i+1}^{T}$ by first computing the number of voxel units crossed;

$$\mathbf{u} = \left\lfloor \frac{v_s \mathbf{t}_{i+1}^{T}}{v_d} \right\rfloor \tag{3.3}$$

and then shifting the pose while updating the global position of the TSDF;

$$\mathbf{t}_{i+1}^{T'} = \mathbf{t}_{i+1}^{T} - \frac{v_d \mathbf{u}}{v_s} \tag{3.4}$$

$$\mathbf{g}_{i+1} = \mathbf{g}_i + \mathbf{u} \tag{3.5}$$

3.3.1 Implementation

There are two parts of volumetric fusion that require indexed access to the TSDF volume; 1) Volume Integration and 2) Volume Raycasting. Referring again to Figure 3.2, the new surface measurements shown in blue can be integrated into the memory previously used for the old surface contained within the red region of the TSDF by ensuring all element look ups in the 3D block of GPU memory reflect the virtual voxel translation computed in Equation 3.5. Assuming row major memory ordering, an element in the unshifted cubic 3D voxel grid can be found at the 1D memory location a given by:

$$a = (x + y v_s + z v_s^2) \tag{3.6}$$

The volume's translation can be reflected in how the TSDF is addressed for integration and raycasting by substituting the indices in Equation 3.6 with values that are offset by the current global position of the TSDF and bound within the dimensions of the voxel grid using the modulus operator:

$$x' = (x + \mathbf{g}_{ix}) \mod v_s \tag{3.7}$$

$$y' = (y + \mathbf{g}_{iy}) \mod v_s \tag{3.8}$$

$$z' = (z + \mathbf{g}_{iz}) \mod v_s \tag{3.9}$$

$$a = (x' + y'v_s + z'v_s^2) \tag{3.10}$$

The original KinectFusion work benefits greatly from memory caching and pipelining functionality within GPU memory to achieve high computational performance within the integration step [86]. In our implementation we have found that use of a cyclical addressing method has no significant effect on real-time performance. An explanation for the lack of a drastic performance decrease is that even after buffer cycling there still exists continuous blocks of memory which at least partially maintain pipelining.

3.3.2 Surface Extraction

In order to recover the surface from the TSDF that moves out of the region of space encompassed by the volume, the **u** value computed in Equation 3.3 is used with $\mathbf{g}_i$ to index a three dimensional slice of the volume to extract surface points from. These points are extracted by three orthogonal raycasts aligned with the axes of the TSDF through the slice, extracting zero crossings as individual surface vertices. We filter out noisy measurements at this point by only extracting points that are above a minimum voxel weight. The same 3D slice of the volume is then reset to free space to allow integration of new surface measurements. The extracted vertices are transferred to main system memory where further processing takes place. The orthogonal raycast can result in duplicate vertices if the TSDF is obliquely aligned to the surface being reconstructed. A voxel grid filter is used to remove these points by overlaying a voxel grid (with the same voxel size as the TSDF) on the extracted point

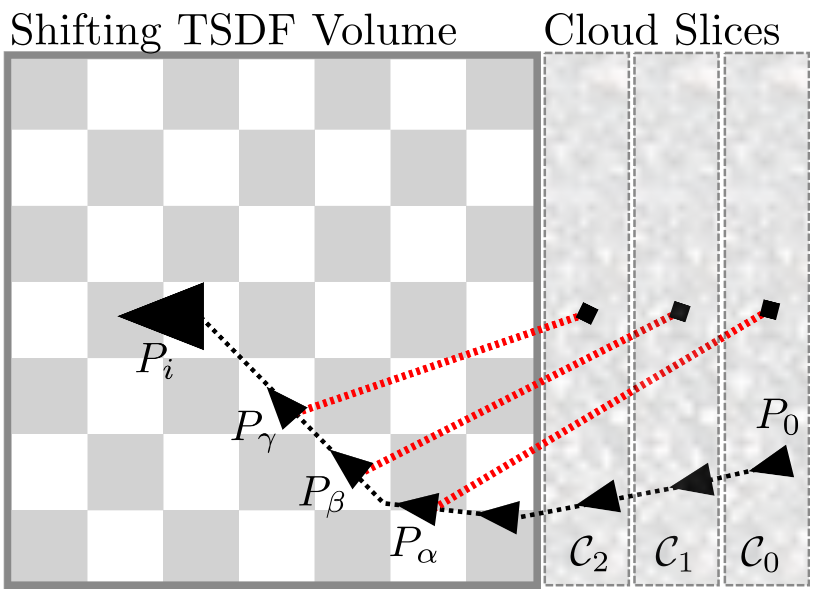


Figure 3.4: Two dimensional visualisation of the association between extracted cloud slices, the camera poses and the TSDF volume. Note that the camera poses here are in global coordinates rather than internal TSDF coordinates. A red dashed line links camera poses with extracted slices of the TSDF volume (P_γ, P_β and P_α with $\mathcal{C}_2, \mathcal{C}_1$ and $\mathcal{C}_0$ respectively). The large triangles represent camera poses that caused volume shifts while the small black squares represent those that didn't.

cloud and returning a new point cloud with a point for each voxel that represents the centroid of all points that fell inside that voxel. Each set of vertices extracted from the TSDF in this fashion is known as a "cloud slice". From here, we rebuild the surface by incrementally triangulating successive cloud slices using an incremental mesh growing variant of the GPT algorithm to ensure surface connectivity between slices [79].

We choose not to perform marching cubes because this would lock the TSDF data structure in GPU memory and delay the reset of the extracted volume slice, impacting volume shifting performance overall. Axis-aligned orthgonal raycasting is extremely fast and allows us to offload the data from the GPU and unlock the TSDF volume as quickly as possible. This way the GPU-based tracking and integration components of the system can continue with minimal interruption while the extracted cloud slice is triangulated on the CPU asynchronously.

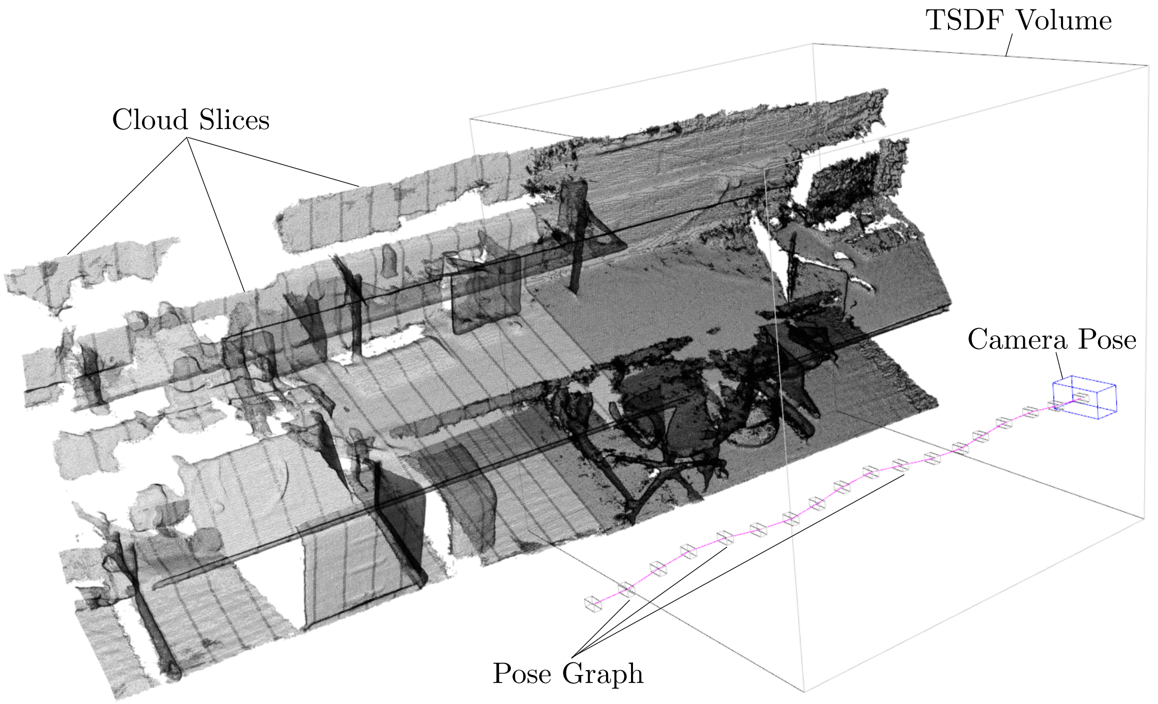


Figure 3.5: Visualisation of a shifted TSDF volume with extracted cloud slices and pose graph highlighted, using dynamic cube positioning discussed in Section 3.4. The pose graph is drawn in pink, while small cuboids are drawn for camera poses that have cloud slices associated with them. Note that the apparent striping of the boundaries between slices has been added for visualisation purposes only.

We associate with each cloud slice the pose of the camera at the time of the slice's extraction. This is visualised in Figure 3.4. At this point we introduce camera poses in the global coordinate frame outside of the TSDF volume P_i, composed of a rotation $\mathbf{R}_i \in \mathbb{SO}_3$ and a translation $\mathbf{t}_i \in \mathbb{R}^3$. The global pose P_i of a camera from the TSDF at time i is made up of:

$$\mathbf{R}_i = \mathbf{R}_i^T \tag{3.11}$$

$$\mathbf{t}_i = \mathbf{t}_i^T + \frac{v_d \mathbf{g}_i}{v_s} \tag{3.12}$$

We construct a pose graph incrementally using each global camera pose P_i, that is, we associate a camera pose to every frame, whereas only some poses are attached to cloud slices. The full shifting and surface extraction process is shown in Figure 3.5, where only the poses with associated cloud slices are drawn.

3.4 Dynamic Cube Positioning

As mentioned in Section 3.2, we position the camera in the center of the TSDF volume and roughly maintain this position inside the TSDF at all times. This parameterisation of the camera position relative to the volume is wasteful as most of the volume is unused (i.e. behind the camera) and there is little overlap between the camera frustum and the volume, shown in Figure 3.6. By dynamically changing the position of the volume relative to the camera depending on the camera's orientation we can achieve greater frustum-volume overlap and make better use of the entire TSDF volume. In a typical SLAM setting a circular parameterisation is sufficient.

Defining β_i to be the rotation around the vertical axis of the camera pose at time i, we can compute the new position of the center of the TSDF volume relative to the camera as:

$$\mathbf{r}^T = \left(\frac{v_d}{2} \cdot \cos\left(\beta_i + \frac{\pi}{2}\right), 0, \frac{v_d}{2} \cdot \sin\left(\beta_i - \frac{\pi}{2}\right)\right)^\top \tag{3.13}$$

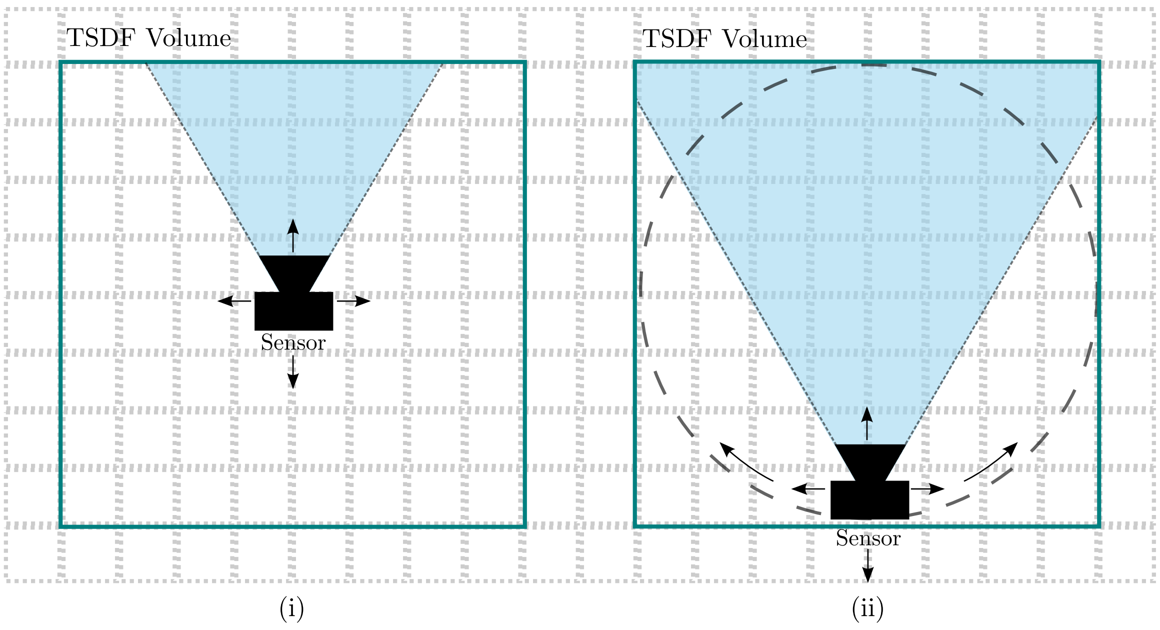


Figure 3.6: Visualisation of frustum-volume overlap for regular and dynamic cube positioning, from left to right; (i) By keeping the camera centered in the TSDF, there is poor overlap between the camera's field of view and the volume; (ii) By using a circular (or spherical) parameterisation of the volume's position relative to the camera, greater overlap with and usage of the TSDF can be achieved.

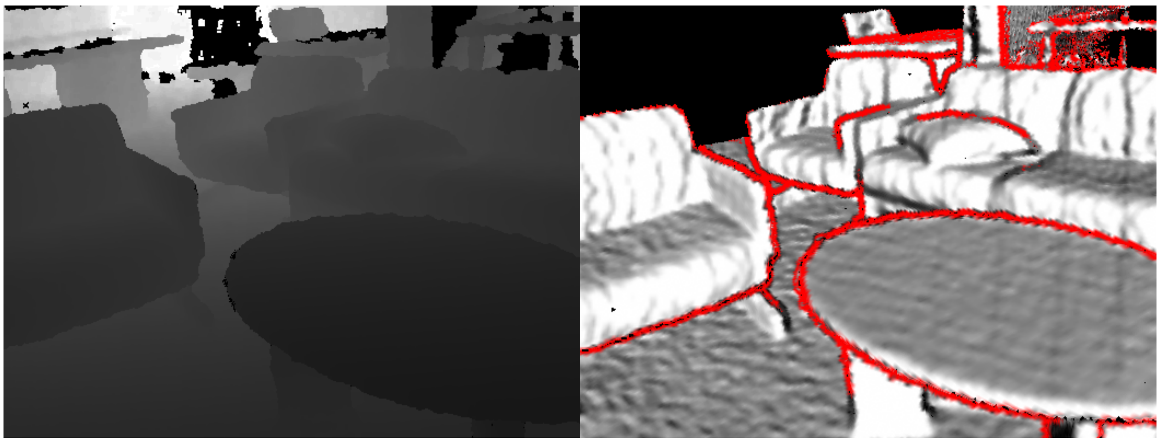


Figure 3.7: From left to right; (i) Input depth map registered to RGB channel; (ii) Color measurements from pixels highlighted in red are rejected for being on depth discontinuities. Lighter surfaces are weighted higher in color integration due to being well aligned with the camera sensor.

This dynamic parameterisation enables more intelligent use of the volume and maintains a larger active reconstruction area in front of the camera at all times.

3.5 Color Estimation

As well as estimating the surface itself in the reconstruction process, we also estimate the color of the surface. Color is integrated into the TSDF in a similar manner to depth measurements including value truncation and averaging. The only distinction is that the predicted surface color values obtained from the volume raycast are not used in camera pose estimation. The motivation for this decision is discussed in Section 4.2. Color fusion has similar advantages to depth map fusion in that sensor noise and other optical phenomena are averaged out from the final reconstruction over time.

3.5.1 Artifact Reduction

The estimated surface color is sometimes inaccurate around the edges of closed objects in a scene due to poor calibration between the RGB and depth cameras or light diffraction around objects. We have observed that there typically exists stark discontinuities in the depth channel around such edges which can in turn cause the background to blend with the foreground surface or vice-versa. To address this issue we opt to reject the integration of color measurements close to or on strong boundaries in the depth image. A color measurement is deemed to be on a boundary if some of its neighbours are more than a given distance away from it in depth. We consider a pixel neighbourhood window of 7×7 pixels around each RGB value to be integrated. Figure 3.7 shows a source depth image and rejected measurements on the TSDF surface model. In addition to this it is ideal to weight color measurements on surfaces well aligned with the sensor higher than those at extreme angles. We weight each color measurement update by the normal angle on the surface with respect to the sensor, visualised in Figure 3.7. The more parallel the surface is to the image plane, the higher the weight on the color measurement.

Defining the image space domain as $\Omega \subset \mathbb{N}^2$, an RGB-D frame I_i is composed of an RGB image $\mathbf{rgb}_i : \Omega \to \mathbb{N}^3$, a depth image $\mathbf{d}_i : \Omega \to \mathbb{R}$ and a timestamp i. We also define a normal map computed for $\mathbf{d}_i$ as $\mathbf{n}_i : \Omega \to \mathbb{R}^3$. We list the algorithm for color integration in Algorithm 3.1. Note that we define the z-axis to point outward from the sensor and in all experiments use an RGB-D frame resolution of 640×480. An example reconstruction is shown in Figure 3.8 comparing surface coloring with and without the described measures. As can be seen incorporating these measures greatly reduces visual artifacts in the captured model.

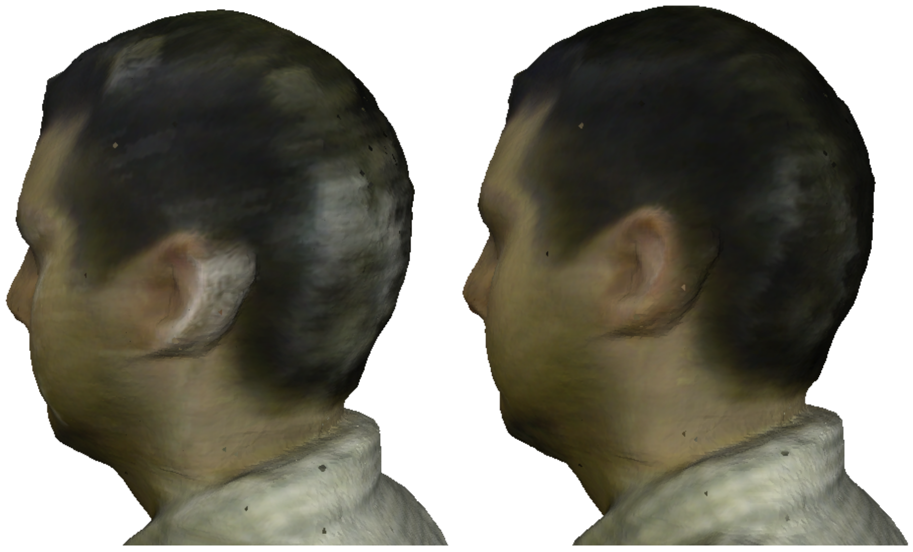

(i) (ii)

Figure 3.8: From left to right; (i) Light diffraction behind a foreground surface has caused incorrect color integration (ii) Incorporating a discontinuity check with surface angle weighting greatly reduces the visual artifacts captured.

Algorithm 3.1: Color Integration

```
Input: $\mathbf{rgb}_i$ Current RGB image
       $\mathbf{d}_i$ Current depth map
       $\mathbf{n}_i$ Current normal map
       $S(\mathbf{s})_i$ Current TSDF volume
       $\mathbf{s} \in \Psi$ Current voxel
       $\mathbf{p} \in \Omega$ Current pixel
do
    $c \leftarrow 0$
    for each $\mathbf{p}_k$ in $7 \times 7$ area around $\mathbf{p}$ do
        if $|\mathbf{d}_i(\mathbf{p}_k) - \mathbf{d}_i(\mathbf{p})| >$ depth threshold or $\mathbf{d}_i(\mathbf{p}_k) = 0$ then
            $c \leftarrow c + 1$
    if $c <$ count threshold then
        $w_c = \min(1.0, \mathbf{n}_i(\mathbf{p})_z / max\_weight)$
        $S(\mathbf{s})_i^{R'} = (S(\mathbf{s})_{i-1}^W S(\mathbf{s})_{i-1}^R + w_c \mathbf{rgb}_i(\mathbf{p})^R)/(S(\mathbf{s})_{i-1}^W + w_c)$
        $S(\mathbf{s})_i^{G'} = (S(\mathbf{s})_{i-1}^W S(\mathbf{s})_{i-1}^G + w_c \mathbf{rgb}_i(\mathbf{p})^G)/(S(\mathbf{s})_{i-1}^W + w_c)$
        $S(\mathbf{s})_i^{B'} = (S(\mathbf{s})_{i-1}^W S(\mathbf{s})_{i-1}^B + w_c \mathbf{rgb}_i(\mathbf{p})^B)/(S(\mathbf{s})_{i-1}^W + w_c)$
end
```

3.6 Summary

This chapter has described our method for dense volumetric fusion-based mapping which, unlike previous work, is capable of functioning over extended scale environments. This is the first key component in taking high quality real-time dense fused mapping (first proposed by Newcombe *et al.* [86]) to large scale environments and developing a fully fledged dense SLAM system. The contributions of this chapter alone are significant in how datasets shown in Whelan *et al.* [125] and some datasets later evaluated in Chapter 7 (such as datasets 2 and 4 in Section 7.2) can be captured with such a system. The contribution of advanced methods for fusing appearance information also comes to the forefront in our evaluation of surface reconstruction quality in Chapter 7, specifically in Section 7.1.2.2 when compared to alternative map representations.

CHAPTER 4

Robust Dense Visual Odometry

In this chapter we describe the geometric and photometric components of the camera pose estimation pipeline and our method for combining them to form a single joint pose constraint. A number of volumetric fusion systems use only depth information for camera pose estimation [86, 10, 6, 61, 97, 132, 7]. A reliance on geometric information alone for camera pose estimation has a number of well understood problems, such as the inability to function in corridor-like environments and other scenes with few 3D features. To avoid these problems, in a similar manner to Henry *et al.*, we combine dense geometric camera pose constraints with dense photometric constraints to achieve a more robust pose estimate in more challenging scenes [45]. We base our approach on the dense photometric image warping method of Steinbrücker *et al.* and Audras *et al.*, performing dense RGB-D alignment every frame in real-time [108, 3]. As with other components of the pipeline we utilise a GPU implementation of the algorithm.

4.1 Geometric Camera Pose Estimation

Many of the previous works on volumetric fusion estimate the pose of the camera each frame relative to the TSDF by aligning the current depth map with the TSDF, either by raycasting the volume to retrieve a vertex and normal map of the predicted surface (as done originally by Newcombe *et al.* [86]) and performing iterative closest point (ICP) or by directly minimising the distance to the surface in the TSDF [6, 7]. We perform the former in order to avoid expensive global memory accesses in the TSDF volume in GPU memory.

We aim to find the motion parameters ξ that minimise the cost over the point-to-plane error between vertices in the current depth frame and the predicted raycast surface:

$$E_{icp} = \sum_k \left\| \left(\mathbf{v}^k - \exp(\hat{\xi}) \mathbf{T} \mathbf{v}_n^k \right) \cdot \mathbf{n}^k \right\|_2^2, \tag{4.1}$$

where $\mathbf{v}_n^k$ is the k-th vertex in frame n, $\mathbf{v}^k, \mathbf{n}^k$ are the corresponding vertex and normal in the model, $\mathbf{T}$ is the current estimate of the transformation from the current frame to the model frame and $\exp(\hat{\xi})$ is the matrix exponential that maps a member of the Lie algebra $\mathfrak{se}_3$ (the minimal parameterisation ξ of a pose update) to a member of the corresponding Lie group $\mathbb{SE}_3$ (a homogeneous transformation matrix $\mathbf{T}$). Initially we set the estimated camera transformation matrix $\mathbf{T}$ to the identity, where

$$\mathbf{T} = \begin{bmatrix} & \mathbf{R} & & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{SE}_3 \tag{4.2}$$

with a rotation $\mathbf{R} \in \mathbb{SO}_3$ and translation $\mathbf{t} \in \mathbb{R}^3$. For simplicity of notation we omit conversions between 3-vectors (as needed for dot and cross products) and their corresponding homogeneous 4-vectors (as needed for multiplications with $\mathbf{T}$). We utilise projective data association as originally proposed by Newcombe *et al.* for fast point correspondence between the vertex maps by projecting the vertices from the depth map $\mathbf{v}_n$ onto the predicted surface vertices $\mathbf{v}$. Linearising the transformation around the identity we get:

$$\begin{aligned}
E_{icp} &\approx \sum_k \left\| \left(\mathbf{v}^k - (\mathbf{I} + \hat{\xi})\mathbf{T}\mathbf{v}_n^k \right) \cdot \mathbf{n}^k \right\|_2^2 && (4.3)\\
&= \sum_k \left\| \left(\mathbf{v}^k - \mathbf{T}\mathbf{v}_n^k \right) \cdot \mathbf{n}^k - \hat{\xi}\mathbf{T}\mathbf{v}_n^k \cdot \mathbf{n}^k \right\|_2^2 && (4.4)\\
&= \sum_k \left\| \begin{bmatrix} -\mathbf{T}\mathbf{v}_n^k \times \mathbf{n}^k \\ -\mathbf{n}^k \end{bmatrix}^\top \xi + \left(\mathbf{v}^k - \mathbf{v}_n^k \right) \cdot \mathbf{n}^k \right\|_2^2 && (4.5)\\
&= \| \mathbf{J}_{icp}\xi + \mathbf{r}_{icp} \|_2^2 && (4.6)
\end{aligned}$$

At this point we must compute the least-squares solution

$$\arg\min_{\xi} \ \| \mathbf{J}_{icp}\xi + \mathbf{r}_{icp} \|_2^2 \tag{4.7}$$

to compute an improved camera transformation estimate

$$\begin{aligned}
\mathbf{T}' &= \exp(\hat{\xi})\mathbf{T} && (4.8)\\
\hat{\xi} &= \begin{bmatrix} \multicolumn{3}{c}{[\boldsymbol{\omega}]_\times} & \mathbf{x} \\ 0 & 0 & 0 & 0 \end{bmatrix} && (4.9)
\end{aligned}$$

with $\xi = [\boldsymbol{\omega}^\top \mathbf{x}^\top]^\top$, $\boldsymbol{\omega} \in \mathbb{R}^3$ and $\mathbf{x} \in \mathbb{R}^3$.

Blocks of the measurement Jacobian and residual can be populated in tandem and solved with a highly parallel tree reduction on the GPU to produce a 6×6 system of normal equations which are then transferred to the CPU and solved with Cholesky decomposition to yield ξ. As in related work [86] we compute the alignment iteratively with a three level coarse-to-fine depth map pyramid scheme.

4.2 Photometric Camera Pose Estimation

As mentioned previously, for photometric camera pose estimation we choose to match between consecutive RGB-D frames instead of matching to the texture predicted from the surface reconstruction. This is motivated by the fact that depending on the configuration of the TSDF there may be poor overlap between the camera frustum and the volume, which limits the amount of photometric information which can be used, where distant photometric features are desirable to constrain camera rotation. Furthermore, the resolution of the TSDF in terms of voxels may produce a raycast image with a much lower resolution than the image produced by the RGB sensor. By default the Microsoft Kinect and Asus Xtion Pro Live, two of the most popular RGB-D sensors, have automatic exposure and white balance enabled, which can cause unusual coloring of the surface reconstruction over time, again hindering model-based photometric tracking. While these functions of the camera can be disabled we have found that it is sometimes desirable to keep them enabled as in general scene illumination can vary to a certain degree.

Given two consecutive RGB-D frames $[\mathbf{rgb}_{n-1}, \mathbf{d}_{n-1}]$ and $[\mathbf{rgb}_n, \mathbf{d}_n]$ we compute a rigid camera transformation between the two that maximises photoconsistency. Defining $V : \Omega \to \mathbb{R}^3$ to be the back-projection of a point $\mathbf{p}$, dependent on a metric depth map $M : \Omega \to \mathbb{R}$ and camera intrinsics matrix $\mathbf{K}$ made up of the principal points c_x and c_y and the focal lengths f_x and f_y:

$$V(\mathbf{p}) = \left(\frac{(\mathbf{p}_x - c_x)M(\mathbf{p})}{f_x}, \frac{(\mathbf{p}_y - c_y)M(\mathbf{p})}{f_y}, M(\mathbf{p})\right)^\top \tag{4.10}$$

We also define perspective projection of a 3D point $\mathbf{v} = (x, y, z)^\top$ including dehomogenisation by $\Pi(\mathbf{v}) = (x/z, y/z)^\top$. The cost we wish to minimise depends on the difference in intensity values between two images $I_{n-1}, I_n : \Omega \to \mathbb{N}$:

$$E_{rgbd} = \sum_{\mathbf{p} \in \mathcal{L}} \left\| I_n(\mathbf{p}) - I_{n-1}\left(\Pi_{n-1}(\exp(\hat{\xi})\mathbf{T}V_n(\mathbf{p}))\right) \right\|_2^2 \tag{4.11}$$

Where $\mathcal{L}$ is the list of valid interest points populated in Algorithm 4.1 and $\mathbf{T}$ is the current estimate of the transformation from I_n to I_{n-1}. Similar to the geometric pose estimation method we solve for this transformation iteratively with a three level image pyramid.

4.2.1 Preprocessing

For both pairs we perform preprocessing on the RGB image and depth map. For each depth map we convert raw sensor values to a metric depth map $M : \Omega \to \mathbb{R}$ and we compute an intensity image $I = (\mathbf{rgb}^R * 0.299 + \mathbf{rgb}^G * 0.587 + \mathbf{rgb}^B * 0.114)$ with $I : \Omega \to \mathbb{N}$. Following this a three level intensity and depth pyramid is constructed using a 5×5 Gaussian kernel for downsampling. We compute the partial derivatives $\frac{\partial I_n}{\partial x}$ and $\frac{\partial I_n}{\partial y}$ using a 3×3 Sobel operator coupled with a 3×3 Gaussian blur with $\sigma = 0.8$. Each of these steps is carried out on the GPU acting in parallel with one GPU thread per pixel.

4.2.2 Precomputation

As with the ICP method described in Section 4.1, we use projective data association between frames to populate the point correspondences. For the sake of speed we only include point correspondences with a minimum gradient in the intensity image, with the motivation that other low gradient points will not have a significant effect on the final transformation. We implement this optimisation by using a list of interest points, which involves a much larger set of points than a point feature extractor could provide. Compiling this list of points as a parallel operation is done using a basic parallel reduction exploiting shared memory in each CUDA thread block as inspired by a similar operation by van den Braak *et al.* [123]. Algorithm 4.1 lists the operation as it would operate for each level of the pyramid.

Algorithm 4.1: Interest Point Accumulation

```
Input: $\frac{\partial I_n}{\partial x}$ and $\frac{\partial I_n}{\partial y}$ intensity image derivatives
       $s$ minimum gradient scale for pyramid level
Output: $\mathcal{L}$ list of interest points
        $k_{\mathcal{L}}$ global point count
Data: $\alpha$ thread block x-dimension
      $\beta$ thread block y-dimension
      $\gamma$ pixels per thread
      $\iota$ shared memory local list
      $\kappa$ shared memory local index
      $blockIdx$ CUDA block index
      $threadIdx$ CUDA thread index
in parallel do
    $i \leftarrow \beta * blockIdx.y + threadIdx.y$            // compute starting pixel
    $j \leftarrow \alpha * \gamma * blockIdx.x + \gamma * threadIdx.x$
    if $threadIdx.x = 0$ and $threadIdx.y = 0$ then
        $\kappa \leftarrow 0$
    syncthreads()
    for $l \leftarrow 0$ to $\gamma$ do                        // for each pixel in this thread
        $\mathbf{p} \leftarrow (i, j + l)$
        $g^2 = \frac{\partial I_n}{\partial x}(\mathbf{p})^2 + \frac{\partial I_n}{\partial y}(\mathbf{p})^2$
        if $g^2 \geq s$ then                                   // add pixel if gradient high enough
            $idx \leftarrow$ atomicInc($\kappa$)
            $\iota_{idx} \leftarrow \mathbf{p}$
    syncthreads()
    $b \leftarrow \alpha * \gamma * threadIdx.y + \gamma * threadIdx.x$
    for $l \leftarrow 0$ to $\gamma$ do                        // reduce local list into global list
        $a \leftarrow b + l$
        if $a < \kappa$ then
            $idx \leftarrow$ atomicInc($k_{\mathcal{L}}$)
            $\mathcal{L}_{idx} \leftarrow \iota_a$
end
```

In the computation of the Jacobian matrix the projection of each point in M_{n-1} is required. For each pyramid level the 3D projection $V_{n-1}(\mathbf{p})$ of each point $\mathbf{p}$ in the depth map is computed prior to beginning iteration. Only projecting certain points based on a condition results in performance hindering branching and a reduction in pipelining. Empirically it was found to be faster to simply project the entire depth map rather than only project points required in correspondences.

4.2.3 Iterative Transformation Estimation

Our iterative estimation process takes two main steps; (i) populating a list of valid correspondences from the precomputed list of interest points and (ii) solving the linear system for an incremental transformation and concatenating these transformations. The first step involves a reduction similar to the one in Algorithm 4.1, but rather than reducing from a 2D array to a 1D array it reduces from a 1D array to another 1D array; a distinction which results in a notable difference in implementation. Before each iteration we compute the projection of the current estimated transformation $\mathbf{T}$ into the image (given the camera intrinsics matrix $\mathbf{K}$) before uploading to the GPU as

$$\mathbf{R}^I = \mathbf{K}\mathbf{R}\mathbf{K}^{-1}, \qquad \mathbf{t}^I = \mathbf{K}\mathbf{t}. \tag{4.12}$$

Algorithm 4.2 lists the process of populating a list of point correspondences from the list of interest points which can then be used to construct the Jacobian.

Similar to the previous section, with a list of valid correspondences we need only compute the least-squares solution

$$\arg\min_{\xi} \; \|\mathbf{J}_{rgbd}\xi + \mathbf{r}_{rgbd}\|_2^2 \tag{4.13}$$

to compute the motion update vector ξ. We first normalise the intensity difference sum σ computed in Algorithm 4.2 to enable a weighted optimisation $\sigma' = \sqrt{\sigma / k_{\mathcal{C}}}$. Computation of the σ value in parallel is in fact an optimisation exploiting the atomic arithmetic functions available in the CUDA API. From here $\mathbf{J}_{rgbd}$ and $\mathbf{r}_{rgbd}$ can be

Algorithm 4.2: Correspondence Accumulation

```
Input: $\mathcal{L}$ list of interest points
       $d_\delta$ maximum change in point depth
       $[I_{n-1}, M_{n-1}]$ previous intensity depth pair
       $[I_n, M_n]$ current intensity depth pair
       $\mathbf{R}^I$ camera rotation in image
       $\mathbf{t}^I$ camera translation in image
Output: $\mathcal{C}$ correspondence list of the form $(\mathbf{p}, \mathbf{p}', \Delta)$
        $k_\mathcal{C}$ global point count
        $\sigma$ global intensity difference sum
Data: $\alpha$ thread block x-dimension
      $\gamma$ pixels per thread
      $\iota$ shared memory local list
      $\kappa$ shared memory local index
      $blockIdx$ CUDA block index
      $threadIdx$ CUDA thread index
in parallel do
    $i \leftarrow \alpha * \gamma * blockIdx.x + \gamma * threadIdx.x$        // compute starting point
    if $threadIdx.x = 0$ then
        $\kappa \leftarrow 0$
    syncthreads()
    for $l \leftarrow 0$ to $\gamma$ do                          // for each pixel in this thread
        $\mathbf{p} \leftarrow \mathcal{L}_{i+l}$
        $z \leftarrow M_n(\mathbf{p})$
        if isValid($z$) then                                  // test if depth valid
            $(x', y', z')^\top \leftarrow z(\mathbf{R}^I(\mathbf{p}, 1)^\top) + \mathbf{t}^I$        // transform with estimate
            $\mathbf{p}' \leftarrow (\frac{x'}{z'}, \frac{y'}{z'})^\top$                       // project into previous image
            if isInImage($\mathbf{p}'$) then
                $d \leftarrow M_{n-1}(\mathbf{p}')$
                if isValid($d$) and $|z' - d| \leq d_\delta$ then        // depth delta test
                    $idx \leftarrow$ atomicInc($\kappa$)
                    $\iota_{idx} \leftarrow (\mathbf{p}, \mathbf{p}', I_n(\mathbf{p}) - I_{n-1}(\mathbf{p}'))$// add valid correspondence
    syncthreads()
    $b \leftarrow \gamma * threadIdx.x$
    for $l \leftarrow 0$ to $\gamma$ do                   // reduce local list into global list
        $a \leftarrow b + l$
        if $a < \kappa$ then
            atomicAdd($\sigma, \iota_{a_\Delta}^2$)                          // sum total residual
            $idx \leftarrow$ atomicInc($k_\mathcal{C}$)
            $\mathcal{C}_{idx} \leftarrow \iota_a$
end
```

populated including usage of σ' for weighting. Equation 4.13 is then solved using a tree reduction on the GPU followed by Cholesky factorisation of the linear system on the CPU.

4.3 Combined Camera Pose Estimate

Reliance on only depth information for camera pose estimation limits the kinds of scenes in which the camera pose will be properly constrained. Similarly, only relying on photometric information in what are typically considered quite noisy RGB images does not exploit the rich geometric information contained within the scene (and any captured model) to its full extent. As a result, we adopt a combined approach which reaps the benefits of both geometric-based frame-to-model tracking and photometric-based frame-to-frame tracking. We combine the cost functions of both the geometric and photometric estimates in a weighted sum. The sum of the RGB-D and ICP cost is defined as

$$E = E_{icp} + w_{rgbd} E_{rgbd} \tag{4.14}$$

where w_{rgbd} is the weight and was set empirically to 0.1 to reflect the difference in metrics used for ICP and RGB-D costs. A key distinction between our approach and that of Tykkälä *et al.* [122] is that we are combining two cost functions between a frame-to-model registration (for the ICP component) and a frame-to-frame registration (for the RGB-D component). For each step we minimise the linear least-squares problem by solving the normal equations

$$\begin{bmatrix} \mathbf{J}_{icp} \\ v\mathbf{J}_{rgbd} \end{bmatrix}^\top \begin{bmatrix} \mathbf{J}_{icp} \\ v\mathbf{J}_{rgbd} \end{bmatrix} \xi = \begin{bmatrix} \mathbf{J}_{icp} \\ v\mathbf{J}_{rgbd} \end{bmatrix}^\top \begin{bmatrix} \mathbf{r}_{icp} \\ \mathbf{r}_{rgbd} \end{bmatrix} \tag{4.15}$$

$$(\mathbf{J}_{icp}^\top \mathbf{J}_{icp} + w_{rgbd} \mathbf{J}_{rgbd}^\top \mathbf{J}_{rgbd}) \xi = \mathbf{J}_{icp}^\top \mathbf{r}_{icp} + v \mathbf{J}_{rgbd}^\top \mathbf{r}_{rgbd} \tag{4.16}$$

where $v = \sqrt{w_{rgbd}}$. The products $\mathbf{J}^\top\mathbf{J}$ and $\mathbf{J}^\top\mathbf{r}$ are computed on the GPU using a tree reduction. The normal equations are then solved on the CPU using Cholesky factorisation. The final estimate returns a locally optimal (in the least-squares sense) camera pose which jointly minimises the photometric error between the current and previous RGB-D frames and the geometric error between the current depth map and the TSDF surface reconstruction. This combined method provides a very accurate and stable trajectory estimate as well as surface reconstruction, which we expand upon in Chapter 7.

It should be noted that although there are a number of atomic operations in Algorithms 4.1 and 4.2, these are primarily operating on values contained in shared thread block memory, minimising impact on execution performance and overall degradation to serial execution. Our computational performance results in Chapter 7 demonstrate that the use of such atomic operations (in the standard reduction setting they are used in here) does not hinder real-time performance.

4.4 Summary

This chapter has detailed our method for jointly utilising dense geometric and photometric information between frames to robustly estimate the camera pose in challenging environments. This is another key step towards developing a robust dense visual SLAM system which can function in a wide variety of scenes. In contrast to other dense reconstruction systems which mostly focus on using geometric information for pose estimation [86, 10, 6, 61, 97, 132, 7], the contribution described in this chapter enables reliable camera tracking in either geometrically sparse or poorly textured areas. As well as being shown in Whelan *et al.* [126], this is also demonstrated in a number of datasets evaluated in Chapter 7 in particular in comparison to the DVO tracking system of Kerl *et al.* [62] given in Section 7.1.2.2.

CHAPTER 5

Large Scale Dense Visual SLAM

In this chapter we describe our method for performing real-time large scale loop closure. Using the techniques from Chapters 3 and 4 permits the reconstruction of large scale dense 3D mesh-based maps in real-time, however like all egomotion estimation systems drift will accumulate over space and time, warranting a method to correct the map to achieve global consistency when possible. A simple approach to this problem would be to associate each vertex in the mesh with the nearest camera pose, optimise the pose graph and reflect the camera pose transformations in the mesh vertices. This would however cause sharp discontinuities at points on the surface where the association between camera poses changes and ignores other important properties of the surface. For this reason we have chosen a non-rigid method of correcting the map. We now frame the system as a more traditional SLAM setup with a frontend (for camera tracking and surface extraction) and a backend (for pose graph optimisation and map optimisation). A detailed system architecture diagram is shown in Figure 5.1.

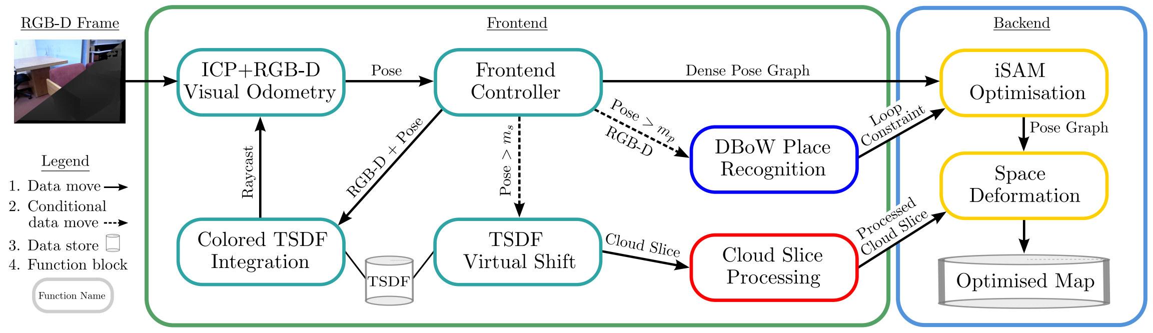


Figure 5.1: System architecture diagram. Differently colored function blocks execute in separate CPU threads. The m_s quantity denotes the volume shifting threshold and m_p denotes the place recognition movement threshold.

The frontend is made up of the extended scale volumetric fusion method described in Chapter 3 coupled with the combined geometric and photometric camera pose estimation method described in Chapter 4. The final component of the frontend not yet described is a visual place recognition module that relies on the DBoW place recognition system [36] which we describe in Section 5.2.

The backend provides a means of performing deformation-based dense map correction making use of incremental pose graph optimisation coupled with a non-rigid map optimisation. We use iSAM [56] to optimise the camera pose graph according to loop closure constraints provided by our place recognition module. The optimised trajectory is then used in conjunction with matched visual features to constrain a non-rigid space deformation of the map. We adapt the embedded deformation technique of Sumner *et al.* [116] to apply it to large scale dense maps captured with a pose graph backend and utilise efficient incremental methods to prepare the map for deformation.

We apply the SLAM principal to our framework by building constraints between multiple regions of the surface through frames anchored to the map via the place recognition system. These frames (and associated global camera poses) are connected to the pose graph, which upon optimisation propagates back to the surface through the deformation. Following we provide a detailed description of each component involved in the global consistency pipeline including pose graph representation, place recognition and loop closure, deformation graph construction and map optimisation.

5.1 Pose Graph

All camera poses added to the pose graph are given in global coordinates, as described in Section 3.3.2. A camera pose P_i is estimated for every processed frame. We evaluate the trade offs of using every pose versus a subset of poses in Chapter 7. As discussed in Section 3.3.2 some camera poses also have an associated cloud slice as shown in Figure 5.2, where the relationship between pose P_γ and cloud slice $\mathcal{C}_j$ is shown. This provides a useful association between camera poses and the extracted surface, capturing both temporal and spatial proximity. In order to model the uncertainty of inter-pose constraints derived from dense visual odometry we can approximate the constraint uncertainty with the Hessian as $\Sigma = (\mathbf{J}^\top\mathbf{J})^{-1}$, where $\mathbf{J}$ is the combined measurement Jacobian computed in Equation 4.16.

5.2 Place Recognition

We use Speeded Up Robust Feature (SURF) descriptors with the bag-of-words-based DBoW loop detector for place recognition [36]. Adding every RGB-D frame to the place recognition system is non-optimal, therefore we utilise a movement metric sensitive to both rotation and translation which indicates when to add a new frame to the place recognition system. Defining $\mathbf{r}(\mathbf{R}) : \mathbb{SO}(3) \rightarrow \mathbb{R}^3$ to provide the rotation vector form of some rotation matrix $\mathbf{R}$, we compute a movement metric between two poses a and b that compounds both translation and rotation into a single quantity as:

$$m_{ab} = \left\|\mathbf{r}(\mathbf{R}_a^{-1}\mathbf{R}_b)\right\|_2 + \left\|\mathbf{t}_a - \mathbf{t}_b\right\|_2 \tag{5.1}$$

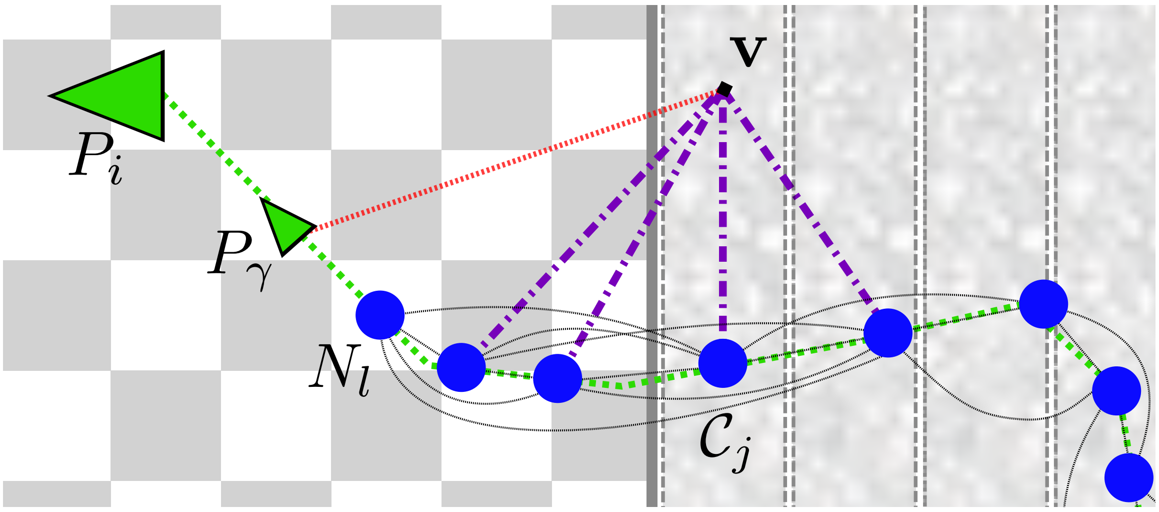


Figure 5.2: Two-dimensional example showing the current position of the TSDF shifting volume as a checkerboard pattern and the previously extracted cloud slices as textured columns. Also shown is the pose graph as small green points as well as a pose P_γ which caused a volume shift. The association between P_γ and the extracted cloud slice is shown with a dotted red line. A $k = 4$ connected sequential deformation graph is also shown, demonstrating the back-traversal vertex association algorithm on a random vertex $\mathbf{v}$.

For each frame we evaluate the movement distance between the current frame pose and the pose of the last frame added to the place recognition system according to Equation 5.1. If this metric is above some threshold m_p, a new frame is added. Empirically we found $m_p = 0.3$ provides good performance. Alternatively the two quantities can be separately thresholded such that motion is acknowledged when either $\|\mathbf{r}(\mathbf{R}_a^{-1}\mathbf{R}_b)\|_2$ goes above a specified angle θ_t threshold or $\|\mathbf{t}_a - \mathbf{t}_b\|_2$ goes above a distance m_t threshold. We have not found place recognition rates to vary significantly between schemes.

Upon receiving a new RGB-D frame $[\mathbf{rgb}_i, \mathbf{d}_i]$ the place recognition module first computes a set of SURF keypoints and associated descriptors $U_i \in \Omega \times \mathbb{R}^{64}$ for that frame. These features are cached in memory for future queries. The depth image $\mathbf{d}_i$ is also cached, however to ensure low memory usage it is compressed in real-time using lossless compression [22]. Following this, the existing bag-of-words descriptor database is queried. If a match is found the SURF keypoints and descriptors U_m and depth data $\mathbf{d}_m$ for the matched image are retrieved for constraint computation. A number of validation steps are performed to minimise the chance of false positives. They are detailed in the following three subsections.

5.2.1 SURF Correspondence Threshold

Given U_i and U_m we find correspondences by a k-nearest neighbour search in the SURF descriptor space. We use the Fast Library for Approximate Nearest Neighbors (FLANN) to perform this search and populate a set of valid correspondences $G \in \Omega \times \Omega$, thresholding matches using an L_2-norm between descriptors in $\mathbb{R}^{64}$. We discard the loop closure candidate if $|G|$ is less than some threshold; a value of 35 has been found to provide adequate performance in our experiments.

5.2.2 RANSAC Transformation Estimation

Given G and $\mathbf{d}_m$, we first attempt to approximate a 6-DOF relative transformation between the camera poses of frames i and m using a RANSAC-based 3-point algorithm [34]. Each matching keypoint in G is back-projected from image m to a 3D point, transformed according to the current RANSAC model and reprojected into the image plane of frame i (using standard perspective projection onto an image plane) where the reprojection error quantified by the L_2-norm in $\mathbb{R}^2$ is used for outlier detection. Empirically we chose a maximum reprojection error of 2.0 for inliers. If the percentage of inliers for the RANSAC estimation is below 25% the loop closure is discarded. Otherwise, we refine the estimated transformation by minimising all inlier feature reprojection errors in a Levenberg-Marquardt optimisation.

5.2.3 Point Cloud ICP

At this point only candidate loop closures with strong geometrically consistent visual feature correspondences remain. As a final step we perform a non-linear ICP step between $\mathbf{d}_i$ and $\mathbf{d}_m$. Firstly we back-project each point in both depth images to produce two point clouds. In order to speed up the computation, we carry out a uniform downsampling of each point cloud in $\mathbb{R}^3$ using a voxel grid filter. Finally, using the RANSAC approximate transformation estimate as an initial guess, we iteratively minimise nearest neighbour correspondence distances between the two point clouds using a Levenberg-Marquardt optimisation. We accept the final refined transformation if the mean L_2^2-norm of all correspondence errors is below a threshold. Typically we found a threshold of 0.01 to provide good results.

Once a loop closure candidate has passed all of the described tests, the relative transformation constraint between the two camera poses is added to the pose graph maintained by the iSAM module. Section 5.4 describes how this constraint is used to update the map.

5.3 Space Deformation

Our approach to non-rigid space deformation of the map is based on the embedded deformation approach of Sumner *et al.* [116]. Their system allows deformation of open triangular meshes and point clouds; no connectivity information is required as is the case with many deformation algorithms [58, 51]. Exploiting this characteristic, Chen *et al.* applied embedded deformation to automatic skeletonised rigging and real-time animation of arbitrary objects in their KinÊtre system [11]. Next we describe our adaptation of embedded deformation to apply it to large scale dense maps with a focus on automatic incremental deformation graph construction.

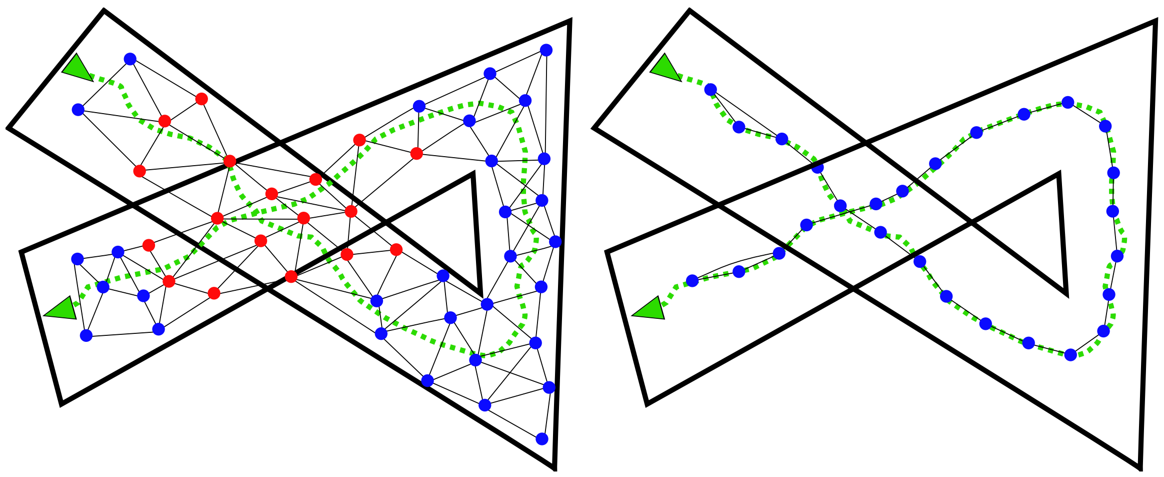


Figure 5.3: Two-dimensional example of deformation graph construction. On the left a spatially-constrained graph is constructed over a pre-loop closure map suffering from significant drift. The nodes highlighted in red are connected to nodes which belong in potentially unrelated areas of the map. On the right our incremental sampling and connectivity strategy is shown (two-nearest neighbours for simplicity) which samples and connects nodes along the pose graph, preventing unrelated areas of the map being connected by the deformation graph.

5.3.1 Deformation Graph

Sumner *et al.* propose the use of a deformation graph to facilitate space deformation of a set of vertices. A deformation graph is composed of nodes and edges spread across the surface to be deformed. Each node N_l has an associated position $N_l^{\mathbf{g}} \in \mathbb{R}^3$ and set of neighbouring nodes $\mathcal{N}(N_l)$. The neighbours of each node are what make up the edges of the graph. Each node also stores an affine transformation in the form of a 3×3 matrix $N_l^{\mathbf{R}}$ and a 3×1 vector $N_l^{\mathbf{t}}$, initialised by default to the identity and $(0, 0, 0)^\top$ respectively. The effect of this affine transformation on any vertex which that node influences is centered at the node's position $N_l^{\mathbf{g}}$.

5.3.2 Incremental Graph Construction

The original approach to embedded deformation relies on a uniform sampling of the vertices in $\mathbb{R}^3$ to construct the deformation graph. Chen *et al.* substitute this with a method that uses a 5D orientation-aware sampling strategy based on the Mahalanobis distance between surface points in order to prevent links in the graph between physically unrelated areas of the model [11]. Neither strategy is appropriate in a dense mapping context as drift in odometry estimation before loop detection may cause unrelated areas of the map to completely overlap in space. This issue also arises in determining connectivity of the graph. Applying sampling and connectivity strategies that are only spatially aware can result in links between completely unrelated points in the map, as shown in Figure 5.3. The effects of applying a nearest neighbour strategy are visualised in Figure 5.4. For this reason we derive a sampling and connectivity strategy that exploits the camera pose graph for deformation graph construction and connection. The method is computationally efficient and incremental, enabling real-time execution. Our sampling strategy is listed in Algorithm 5.1.

Algorithm 5.1: Incremental Deformation Node Sampling

```
Input: $P$ camera pose graph made up of $\mathbf{R}_i$ and $\mathbf{t}_i$
       $i$ pose id of last added node
       $d_p$ pose sampling rate
Output: $N$ set of deformation graph nodes
do
    $l \leftarrow |N|$
    if $l = 0$ then                                  // init new graph with first pose
        $N_l^{\mathbf{g}} \leftarrow \mathbf{t}_0$
        $l \leftarrow l + 1$
        $i \leftarrow 0$
    $P_{last} \leftarrow P_i$
    for $i$ to $|P|$ do                              // from last added pose to newest pose
        if $\|\mathbf{t}_i - \mathbf{t}_{last}\|_2 > d_p$ then     // add node to deformation graph
            $N_l^{\mathbf{g}} \leftarrow \mathbf{t}_i$
            $l \leftarrow l + 1$
            $P_{last} \leftarrow P_i$
end
```

We connect deformation graph nodes returned by our sampling strategy in a sequential manner, following the temporal order of the pose graph itself. That is to say our set of graph nodes N is ordered. We sequentially connect nodes up to a value

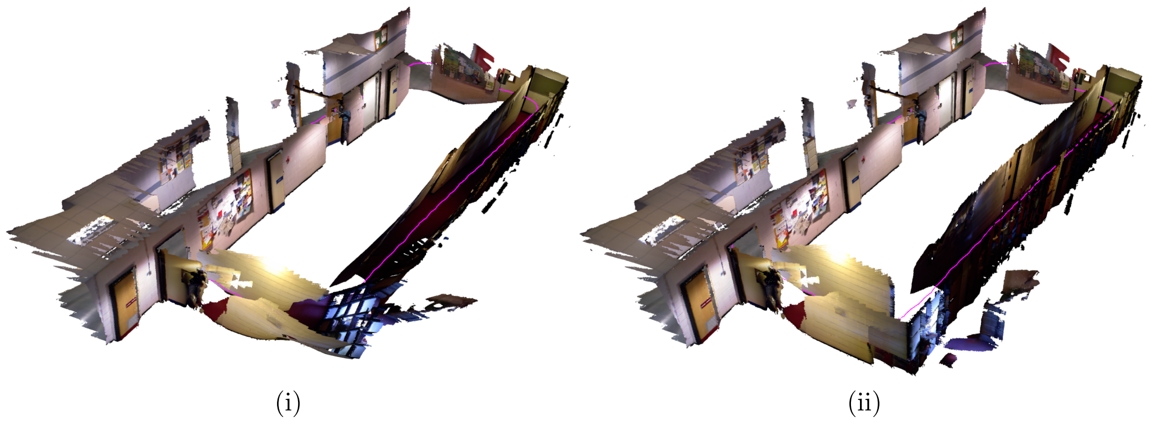


Figure 5.4: From left to right; (i) Highly distorted map produced when a naïve nearest neighbour sampling and connectivity strategy is used; In this example, parts of the floor close to the point of loop closure have been associated with the nearby window through the deformation graph. When optimised, these parts of the scene attempt to "stick together", drastically distorting the surrounding geometry. (ii) Non-distorted map loop closure using our proposed sampling and connectivity strategy. When the deformation graph is intelligently constructed across the map using our scheme, incorrect surface association problems as shown on the left are avoided.

k. We use $k = 4$ in all of our experiments. For example, a node l will be connected to nodes $(l \pm 1, l \pm 2)$. We show $k = 2$ connectivity in Figure 5.3. Note the connectivity of end nodes which maintains k-connectivity.

5.3.3 Incremental Vertex Weighting

Each vertex $\mathbf{v}$ has a set of influencing nodes in the deformation graph $\mathcal{N}(\mathbf{v})$. The deformed position of a vertex is given by:

$$\hat{\mathbf{v}} = \sum_{k \in \mathcal{N}(\mathbf{v})} w_k(\mathbf{v}) \left[N_k^{\mathbf{R}}(\mathbf{v} - N_k^{\mathbf{g}}) + N_k^{\mathbf{g}} + N_k^{\mathbf{t}} \right] \tag{5.2}$$

where $w_k(\mathbf{v})$ is defined as (all k summing to 1):

$$w_k(\mathbf{v}) = (1 - \|\mathbf{v} - N_k^{\mathbf{g}}\|_2 / d_{max})^2 \tag{5.3}$$

Here d_{max} is the Euclidean distance to the $k+1$-nearest node of $\mathbf{v}$. In previous work based on this technique the sets $\mathcal{N}(\mathbf{v})$ for each vertex are computed in batch using a k-nearest neighbour technique. Again, being based on spatial constraints alone this method fails in the example shown in Figure 5.3. To overcome this issue we derive an algorithm that assigns nearest neighbour nodes to each vertex using a greedy back-traversal of the sampled pose graph nodes.

Referring back to Figure 5.2 and Section 3.3.2, we recall that each pose that causes a volume shift has an associated set of vertices contained within a cloud slice. We can exploit the inverse mapping of this association to map each vertex onto a single pose in the pose graph. However, the associated pose is at least a distance of $\frac{v_d}{2}$ away from the vertex, which is not ideal for the deformation. In order to pick sampled pose graph nodes for each vertex that are spatially and temporally optimal, we use the closest sampled pose to the associated cloud slice pose as a starting point to traverse back through the sampled pose graph nodes to populate a set of candidate nodes. From these candidates the k-nearest neighbours of the vertex are chosen. We list the algorithm for this procedure in Algorithm 5.2 and provide a visual example in Figure 5.2.

The per-vertex node weights can be computed within the back-traversal algorithm, which itself can be carried out incrementally online while the frontend volume shifting component provides new cloud slices. The ability to avoid computationally expensive batch steps for deformation graph construction and per-vertex weighting by using incremental methods is the key to allowing low latency online map optimisation at any time.

Algorithm 5.2: Back-Traversal Vertex Association

```
Input: $C$ cloud slices
       $N$ set of deformation graph nodes
       $b_p$ number of poses to traverse back
       $P_{\mathcal{C}_j}$ pose associated with cloud slice $\mathcal{C}_j$
Output: $\mathcal{N}(\mathbf{v})$ for each $\mathbf{v}$
do
    foreach $C_j$ do
        foreach $\mathbf{v} \in C_j$ do        // for each vertex in each cloud slice
            $l \leftarrow$ binary_search_closest($P_{\mathcal{C}_j}, N$)     // get neighbouring nodes
            $N' \leftarrow \emptyset$
            $n \leftarrow 0$
            for $i \leftarrow 0$ to $b_p$ do    // traverse back and add each to a list
                $N'_n \leftarrow N_l$
                $n \leftarrow n + 1$
                $l \leftarrow l - 1$
            sort_by_distance($N', \mathbf{v}$)
            $\mathcal{N}(\mathbf{v}) \leftarrow N'_{1 \to k}$                // store k-nearest nodes
end
```

5.4 Optimisation

On acceptance of a loop closure constraint as described in Section 5.2 we perform two optimisation steps, firstly on the pose graph and secondly on the dense vertex map. The pose graph optimisation provides the measurement constraints for the dense map deformation optimisation in place of user specified constraints that were necessary in the original embedded deformation approach. Pose graph optimisation is carried out using the iSAM framework [56]. We benefit from the incremental sparse linear algebra representation used internally in iSAM, such that execution time is reasonable in terms of online operation.

5.4.1 Map Deformation

Sumner *et al.* define three cost functions over the deformation graph and user constraints to optimise the set of affine transformations over all graph nodes N. The first maximises rigidity in the deformation:

$$E_{rot} = \sum_l \left\| {N_l^{\mathbf{R}}}^\top N_l^{\mathbf{R}} - \mathbf{I} \right\|_F^2 \tag{5.4}$$

Where Equation 5.4 is the alternative Frobenius-norm form provided by Chen *et al.* [11]. The second is a regularisation term that ensures a smooth deformation across the graph:

$$E_{reg} = \sum_l \sum_{n \in \mathcal{N}(N_l)} \left\| N_l^{\mathbf{R}}(N_n^{\mathbf{g}} - N_l^{\mathbf{g}}) + N_l^{\mathbf{g}} + N_l^{\mathbf{t}} - (N_n^{\mathbf{g}} + N_n^{\mathbf{t}}) \right\|_2^2 \tag{5.5}$$

The third is a constraint term that minimises the error on a set of user specified vertex position constraints Q, where a given constraint $Q_p \in \mathbb{R}^3$ and $\phi(\mathbf{v})$ is the result of applying Equation 5.2 to $\mathbf{v}$:

$$E_{con} = \sum_p \|\phi(\mathbf{v}) - Q_p\|_2^2 \tag{5.6}$$

We link the optimised pose graph to the map deformation through the E_{con} cost function. With P being the pose graph (composed of rotations and translations $\mathbf{R}_i$ and $\mathbf{t}_i$) before loop constraint integration we set P' to be the optimised pose graph returned from iSAM. We then add each of the camera pose translations to the deformation cost as if they were user specified vertex constraints, redefining Equation 5.6 as:

$$E_{con_P} = \sum_i \|\phi(\mathbf{t}_i) - \mathbf{t}'_i\|_2^2 \tag{5.7}$$

A uniform constraint distribution across the surface obtained from this parameterisation aids in constraining both surface translation and orientation. However at some points the surface orientation may not be well constrained. In order to overcome this issue we add additional vertex constraints between the unoptimised and optimised 3D back-projections of each of the matched inlier SURF keypoints detected in Section 5.2, where P_i ($\mathbf{R}_i$ and $\mathbf{t}_i$) is the camera pose of the matched loop closure frame:

$$E_{surf} = \sum_q \|\phi((\mathbf{R}_i G_q) + \mathbf{t}_i) - ((\mathbf{R}'_i G_q) + \mathbf{t}'_i)\|_2^2 \tag{5.8}$$

The final total cost function is defined as:

$$w_{rot}E_{rot} + w_{reg}E_{reg} + w_{con_P}E_{con_P} + w_{surf}E_{surf} \tag{5.9}$$

With $w_{rot} = 1$, $w_{reg} = 10$, $w_{con_P} = 100$ and $w_{surf} = 100$, we minimise this cost function using the iterative Gauss-Newton algorithm choosing weighting values in line with those used by Sumner *et al.* [116]. The optimisation consistently converges to a satisfactory result with these weights, similar to the findings of Chen *et al.* [11]. As highlighted in previous work, the Jacobian matrix in this problem is sparse, enabling the use of sparse linear algebra libraries for efficient optimisation. We use the CHOLMOD library to perform sparse Cholesky factorisation and efficiently solve the system [16]. We then apply the optimised deformation graph N to all vertices over all cloud slices $\mathcal{C}$ in parallel across multiple CPU threads. As discussed in Section 3.3.2 we compute an incremental mesh surface representation of the cloud slices as they are produced by the frontend. The incremental mesh can be deformed by applying the deformation graph to its vertices. In our experience an incremental mesh typically contains more minuscule holes than a batch mesh, which in path planning is functionally almost identical but less visually appealing. In all results we show the batch mesh computed over the set of optimised vertices.

5.5 Summary

This chapter has described our method for achieving large scale dense visual SLAM using the frontend components described thus far combined with a pose graph optimation framework and a novel coupling between non-rigid space deformation and incremental dense mapping. The joining of pose graph optimisation with non-rigid space deformation optimisation in an efficient incremental online fashion is a key contribution of this thesis necessary to obtain globally consistent dense surface reconstructions over large scales in real-time, something which has not yet been demonstrated by other related systems. By adopting incremental deformation graph sampling and connectivity while anchoring the pose graph to the deformation constraint in an asynchronous manner the system is capable of closing multiple loops in very large trajectories online without any expensive batch or post-processing steps. This is demonstrated more clearly in Chapter 7, specifically in Section 7.1.2 where superior scalability performance and surface reconstruction quality is demonstrated in comparison to other related works.

CHAPTER 6

Map Simplification and Higher Level Representation

In the previous chapters we have presented a system for capturing large scale dense mesh-based maps in real-time. While this representation is extremely useful for a number of applications including visualisation and object recognition, in certain scenarios it is not computationally efficient to work with. This motivates the need for an alternative representation to one which contains millions of vertices that is more useful for real-time navigation and mobile robot localisation. In this chapter we give an overview of our system for simplifying dense point cloud maps into triangulated planar models. Individual sections detail batch planar segmentation, incremental planar segmentation, triangulation of planar segments and texture generation.

6.1 Building Blocks

Our batch system architecture is shown in Figure 6.1. It takes a point cloud as input and generates a triangular mesh as output. If the input is a colored point cloud, the

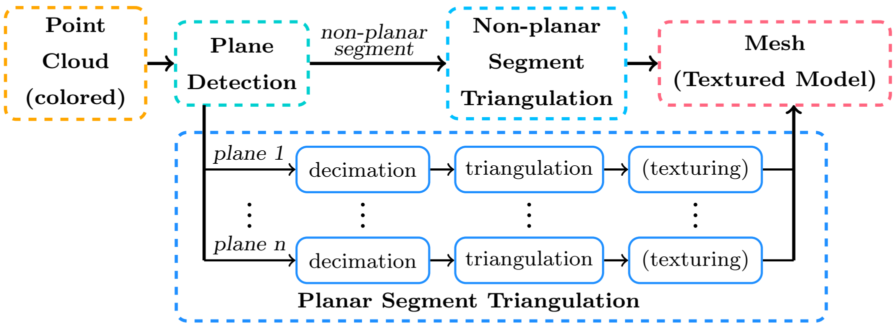


Figure 6.1: Parallel system architecture to process point clouds of large scale open scene scans or maps.

output can also be a textured 3D model. The processing pipeline consists of three main blocks.

Plane Detection segments the input point cloud into planar and non-planar regions to enable separate triangulation and parallel processing. This design is especially beneficial for real-world environments, where multiple independent planar surfaces occur frequently. In our system we apply a local curvature-based region growing algorithm for plane segmentation, which was shown to out perform RANSAC-based approaches [77]. Although not a contribution of this thesis, in the interest of completeness we include a description of this algorithm in Section 6.3. In the incremental scenario, the plane detection block continuously runs and only provides planar segments to be triangulated when they are marked as *finalised* (as detailed in Section 6.4).

Non-Planar Segment Triangulation generates a triangular mesh for non-planar segments using the GPT algorithm [79]. Given a colored point cloud, we preserve the color information for each vertex in the output mesh. Dense triangular meshes with colored vertices can be rendered (with Phong interpolation) to appear similar to textured models. Additionally, as opposed to using textures, maintaining color in vertices of non-planar segments provides easier access to appearance information for point cloud based object recognition systems.

Planar Segment Triangulation triangulates planar segments and textures the mesh afterwards, if given a colored point cloud. In our system we improve the decimation algorithm of Ma *et al.* [77] and further develop a more accurate and robust solution for triangulation. A detailed description of our algorithm is provided in Section 6.5. Our method for planar segment texture generation is described in Section 6.6.

6.2 Computationally Efficient Architecture

To improve computational performance, a multi-threaded architecture is adopted, exploiting the common availability of multi-core CPUs in modern hardware. We apply a coarse-grained parallelisation strategy, following the Single Program Multiple Data (SPMD) model [15]. Parallel triangulation of planar segments is easily accomplished by dividing the set of segments into subsets that are distributed across a pool of threads. For maximum throughput of the entire pipeline, segmentation and triangulation overlap in execution. With an n-core CPU, a single thread is used for segmentation and the remaining $n-1$ threads are used for triangulation, each with a queue of planar segments to be processed. Upon segmentation of a new planar region, the segmentation thread checks all triangulation threads and assigns the latest segment to the thread with the lowest number of points to be processed. This strategy ensures an even task distribution among all threads. When plane segmentation is finished, the segmentation thread begins the non-planar triangulation in parallel to the other triangulation threads.

6.3 Planar Segmentation

In this section we review the curvature-based algorithm used to segment multiple planes from a large 3D point cloud.

6.3.1 Curvature-Based Segmentation Algorithm

Planes are characterised by their perfect flatness and can be described as sets of points that have zero curvature. In practice, open scene point cloud data can be quite noisy and points belonging to planes do not have a curvature of exactly zero. However, the curvature of points lying on planes is still low enough to distinguish them from points belonging to non-planar surfaces. This observation motivates the functionality of this algorithm, which is partially developed from the work of Rabbani *et al.* [96].

The curvature-based algorithm consists of an iterative process. Firstly, the normal of the next plane to be segmented is chosen. This is done by finding the point with the lowest curvature from the set of remaining unsegmented points. From here a *region growing* process begins using the lowest curvature point as the first seed point. In each iteration the k-nearest neighbours of the current seed point are determined and their normals are compared to the estimated plane normal. A neighbouring point is added to the current segment if its normal does not deviate from the plane normal beyond an angle threshold. A qualified neighbour is also used as a new seed point for further region growing if its curvature is sufficiently small. When no more points can be added to the current segment, a plane is considered fully segmented. Afterwards, the whole process restarts with the remaining set of unsegmented points, until the entire cloud has been processed. The pseudocode of the algorithm is listed in Algorithm 6.1.

There are two major differences between this algorithm and the algorithm of Rabbani *et al.* [96]. The first difference is the integration of new points into the current segment. The original algorithm always updates the normal used for the integration of new points from the current seed point which may introduce points belonging to areas of moderate curvature in extracted segments. In this algorithm, the plane normal is fixed to the normal of the first seed point and only this normal is used throughout the segmentation of a single plane. Given that the first seed point

Algorithm 6.1: Curvature-Based Plane Segmentation.

```
Input: Point cloud made of points $\mathbf{p}_i \in \mathbb{R}^3$ with normals $\mathbf{n}_i$ and curvatures $c_i$
       $\theta_{th}$ angle threshold
       $c_{th}$ curvature threshold
Output: Set of planar segments
while points remain unsegmented or the queue is not empty do
    if the queue is empty then
        pick a seed point $p_s$ with the lowest curvature
        set the plane normal $\mathbf{n}_p$ to be the normal of $p_s$
    else
        pop out a seed point $p_s$ from the queue
    mark $p_s$ as segmented
    compute the k-nearest neighbours of $p_s$
    foreach unsegmented neighbour $p_i$ do
        if $\arccos(\mathbf{n}_p, \mathbf{n}_i) < \theta_{th}$ then
            add $p_i$ to the current segment, mark $p_i$ segmented if $c_i < c_{th}$ then
                add $p_i$ to the queue
    if the queue is empty then
        output the current segment as a plane
```

has the lowest curvature available, its normal can be assumed to be a good estimation for the normal of the entire planar segment. With this modification the detection of smoothly-connected shapes, such as spheres and cylinder-like structures is avoided.

The second modification is concerned with the estimation of point curvature. The algorithm of Rabbani *et al.* uses the residual of a least-squares plane fit as a substitute for curvature. In this algorithm the curvature is directly estimated using the original points. A necessary preprocessing step for this algorithm is normal estimation for point clouds. This is accomplished by local Principal Component Analysis (PCA) [82]. The PCA method for normal estimation also provides the curvature quantity using the following equation:

$$c = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}, \ \lambda_0 \leq \lambda_1 \leq \lambda_2, \tag{6.1}$$

where λ_0, λ_1 and λ_2 are the eigenvalues from the PCA process [94]. These eigenvalues

λ_0, λ_1 and λ_2 indicate the smallest, medium and largest variation along the directions specified by their corresponding eigenvectors. For points belonging to an ideal plane, we have $\lambda_0 = 0$ and hence $c = 0$. In the presence of noise, variable c becomes larger than zero.

This curvature-based algorithm works with two parameters. The first parameter θ_{th} specifies the maximum angle between the estimated plane normal and the normal of a potential point on the plane. Typically a 10° angle works well for noisy point clouds. The second parameter c_{th} is the curvature threshold, which is used to verify whether a point should be designated as a seed point for region growing. Empirically this threshold is set to a value below 0.1.

6.4 Incremental Planar Segmentation

In this section we describe our method for incrementally segmenting planes from a point cloud map which is being incrementally produced in real-time by a dense mapping system, e.g. as described in the previous chapters. Our method involves maintaining a pool of unsegmented points which are either segmented as new planes, added to existing planes or deemed to not belong to any planar segment. Firstly we define a distance-based plane merging method that determines whether or not to merge two planar segments based on the distance between the points in each segment. We list this as Algorithm 6.2 and henceforth refer to it as the *mergePlanes* method.

6.4.1 Segment Growing

Assuming the input to our system is a small part of a larger point cloud map that is being built up over time we must define a method for growing existing planar segments that were found in our map in the previous timestep of data acquisition. We maintain a persistent pool of unsegmented points $\mathcal{M}$ where each point $\mathcal{M}_i \in \mathbb{R}^3$

Algorithm 6.2: Method for merging two planar segments.

```
Input: $\mathcal{A}$ planar segment with normal $\mathcal{A}_{\mathbf{n}}$, point cloud $\mathcal{A}_{\mathcal{C}}$ and timestep $\mathcal{A}_t$
       $\mathcal{B}$ planar segment with normal $\mathcal{B}_{\mathbf{n}}$ and point cloud $\mathcal{B}_{\mathcal{C}}$
       $\mathcal{B}_{\mathcal{H}}$ concave hull of $\mathcal{B}$
       $d_{th}$ distance threshold
Output: True or False if segments were merged or not
foreach point $\mathbf{h}_i$ in $\mathcal{B}_{\mathcal{H}}$ do
    if $\exists \mathcal{A}_{\mathcal{C}_k}$ s.t. $\|\mathbf{h}_i - \mathcal{A}_{\mathcal{C}_k}\|_2 < d_{th}$ then
        $\mathcal{A}_{\mathbf{n}} \leftarrow (\mathcal{A}_{\mathbf{n}}|\mathcal{A}_{\mathcal{C}}| + \mathcal{B}_{\mathbf{n}}|\mathcal{B}_{\mathcal{C}}|)/(|\mathcal{A}_{\mathcal{C}}| + |\mathcal{B}_{\mathcal{C}}|)$
        $\mathcal{A}_t \leftarrow 0$
        append $\mathcal{B}_{\mathcal{C}}$ to $\mathcal{A}_{\mathcal{C}}$
        compute $k$d-tree of $\mathcal{A}_{\mathcal{C}}$
        return True
return False
```

and also contains a timestep value $\mathcal{M}_{i_t}$, initially set to zero. When a new set of points are added to the map, they are added to the set $\mathcal{M}$, which is then sorted by the curvature of each point. A batch segmentation of $\mathcal{M}$ is then performed (as described in Section 6.3), producing a set of newly segmented planes $\mathcal{N}$. From here we perform Algorithm 6.3, which will grow any existing segments and also populate the set $\mathcal{S}$, that maintains a list of pairs of planes which are similar in orientation but not close in space. Algorithm 6.4 lists the method for merging similar planes that eventually grow close enough in space to be merged together.

Each time new data is added to the map Algorithms 6.3 and 6.4 are run, after which the timesteps values of all remaining unsegmented points in $\mathcal{M}$ are incremented by 1. Points with a timestep value above a specified threshold are removed from the point pool and marked as non-planar. Algorithm 6.3 will add new segments to the map, grow recently changed segments and ensure that similar planes which have the potential to grow into each other are kept track of. Algorithm 6.4 merges segments which may not have initially been close together in space but have grown near to each other over time. Figure 6.2 shows an example of the incremental planar segmentation process in action.

Algorithm 6.3: Method for growing planar segments.

```
Input: $\mathcal{N}$ set of new planar segments with normals $\mathcal{N}_{i_\mathbf{n}}$, point clouds $\mathcal{N}_{i_\mathcal{C}}$ and
       timesteps $\mathcal{N}_{i_t}$
       $\mathcal{Q}$ set of existing planar segments with normals $\mathcal{Q}_{i_\mathbf{n}}$, point clouds $\mathcal{Q}_{i_\mathcal{C}}$
       and timesteps $\mathcal{Q}_{i_t}$
       $\mathcal{C}_t$ current position of sensor producing the map
       $n_{th}$ normal merge threshold
       $t_{th}$ timestep threshold
       $td_{th}$ timestep distance threshold
Output: $\mathcal{S}$ set of pairs of similar but non-merged segments
foreach newly segmented plane $\mathcal{N}_i$ do
    $\mathcal{R} \leftarrow \varnothing$
    $gotPlane \leftarrow$ False
    foreach existing plane $\mathcal{Q}_i$ do
        if $!\mathcal{Q}_{i_{finalised}}$ and $\arccos(\mathcal{N}_{i_\mathbf{n}}, \mathcal{Q}_{i_\mathbf{n}}) < n_{th}$ then
            compute concave hull $\mathcal{H}$ of $\mathcal{N}_i$
            $gotPlane \leftarrow mergePlanes(\mathcal{Q}_i, \mathcal{N}_i, \mathcal{H})$
            if $gotPlane$ then
                break
            else
                add $\mathcal{Q}_i$ to $\mathcal{R}$
    if $!gotPlane$ then
        compute $k$d-tree of $\mathcal{N}_{i_\mathcal{C}}$
        $\mathcal{N}_{i_t} \leftarrow 0$
        add $\mathcal{N}_i$ to $\mathcal{Q}$
        foreach similar plane $\mathcal{R}_i$ do
            add ($\mathcal{N}_i$, $\mathcal{R}_i$) tuple to $\mathcal{S}$
    remove all points $\mathcal{N}_{i_\mathcal{C}}$ from $\mathcal{M}$
foreach existing plane $\mathcal{Q}_i$ do
    $\mathcal{Q}_{i_t} \leftarrow \mathcal{Q}_{i_t} + 1$
    if $\mathcal{Q}_{i_t} > t_{th}$ and $\forall \mathbf{q} \in \mathcal{Q}_{i_\mathcal{C}}, \|\mathcal{C}_t - \mathbf{q}\|_2 > td_{th}$ then
        $\mathcal{Q}_{i_{finalised}} \leftarrow True$
foreach pair of similar planes $\mathcal{S}_i$ do
    if $\mathcal{S}_{i_1 finalised}$ or $\mathcal{S}_{i_2 finalised}$ then
        delete $\mathcal{S}_i$
```

Algorithm 6.4: Merging segments that have grown closer.

```
Input: $\mathcal{S}$ set of pairs of similar but non-merged segments with point clouds $\mathcal{S}_{i_\mathcal{C}}$
       and alpha values $\mathcal{S}_{i_\alpha}$
       $\mathcal{Q}$ set of existing planar segments
foreach pair of similar planes $\mathcal{S}_i$ do
    $gotPlane \leftarrow$ False
    if $|\mathcal{S}_{i_{1C}}| > |\mathcal{S}_{i_{2C}}|$ then
        swap $\mathcal{S}_{i_1} and \mathcal{S}_{i_2}$
    if $\mathcal{S}_{i_{1\alpha}} ! = |\mathcal{S}_{i_{1C}}|$ then
        compute concave hull $\mathcal{S}_{i_{1\mathcal{H}}}$ of $\mathcal{S}_{i_1}$
        $\mathcal{S}_{i_{1\alpha}} \leftarrow |\mathcal{S}_{i_{1C}}|$
    $gotPlane \leftarrow mergePlanes(\mathcal{S}_{i_2}, \mathcal{S}_{i_1}, \mathcal{S}_{i_{1\mathcal{H}}})$
    if $gotPlane$ then
        foreach existing plane $\mathcal{Q}_i$ do
            if $\mathcal{Q}_i == \mathcal{S}_{i_1}$ then
                delete $\mathcal{Q}_i$
                break
        foreach pair of similar planes $\mathcal{S}_j$ do
            if $i == j$ then
                continue
            if $\mathcal{S}_{j_1} == \mathcal{S}_{i_1}$ then
                $\mathcal{S}_{j_1} \leftarrow \mathcal{S}_{i_2}$
            else if $\mathcal{S}_{j_2} == \mathcal{S}_{i_1}$ then
                $\mathcal{S}_{j_2} \leftarrow \mathcal{S}_{i_2}$
            if $\mathcal{S}_{j_1} == \mathcal{S}_{j_2}$ then
                delete $\mathcal{S}_j$
        delete $\mathcal{S}_i$
```

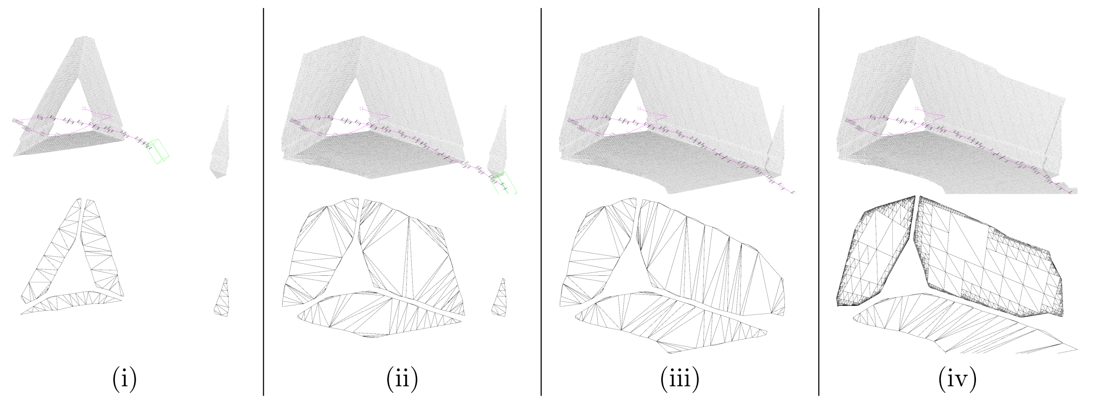


Figure 6.2: Incremental planar segmentation shown with point cloud and camera trajectory (in pink) shown above and resulting planar segments below. From left to right: (i) Initially there are four segments extracted from the point cloud; (ii) As the camera moves and more points are provided, the three segments on the left are grown (as described in Algorithm 6.3); (iii) The upper-right most segment grows large enough to be merged with the small segment on the right (as in Algorithm 6.4); (iv) Once the camera has moved far enough away from the two upper segments they are finalised.

6.5 Triangulation of Planar Segments

In this section, our algorithm for planar segment decimation and triangulation is described. A simplified mesh of a planar segment is generated by removing redundant points that fall within the boundary of the segment. In the following text the input planar segment is denoted as $\mathcal{P}$, made up of points $\mathbf{p} \in \mathbb{R}^3$. With colored point clouds, each point $\mathbf{p}$ also contains (R, G, B) color components.

6.5.1 QuadTree-Based Decimation

Planar segments have a simple shape which can be well described by points on the boundary of the segment. Interior points only add redundancy to the surface representation and complicate the triangulation results. Figure 6.3 shows an example of this where the planar segment is over-represented with thousands of triangles generated with the GPT algorithm using all planar points. However, a naïve solution that removes all interior points and triangulates only with boundary points normally leads

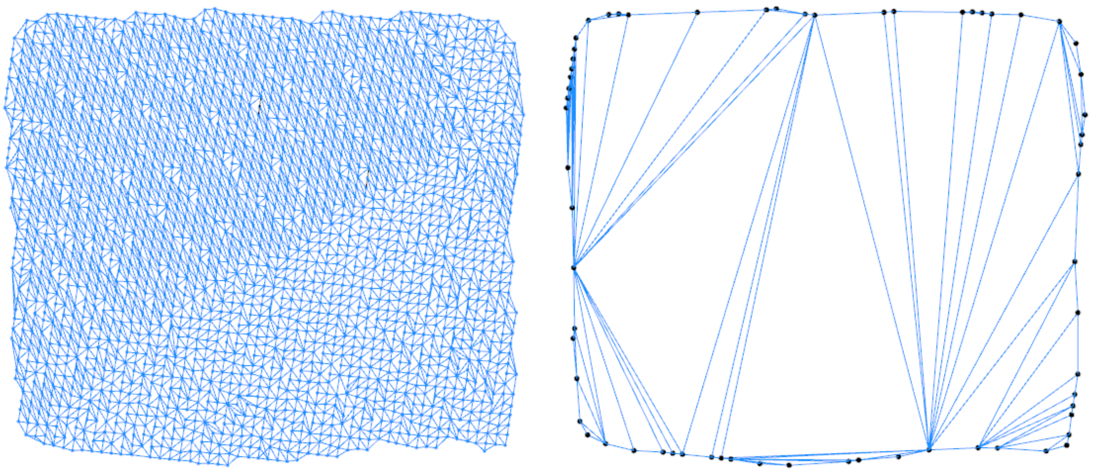


Figure 6.3: Undesirable planar triangulation: the left GPT mesh over-represents the shape while the right boundary-based Delaunay triangulation produces unnatural skinny triangles.

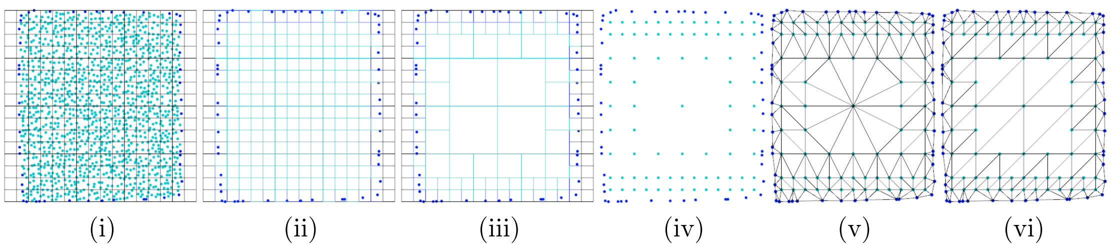


Figure 6.4: Planar decimation and triangulation (boundary and interior points are dark blue and teal, respectively), from left to right: (i) Initialise by subdividing the quadtree bounding box; (ii) Classify nodes into interior (teal), boundary (dark blue) and exterior (black); (iii) Merge interior nodes; (iv) Generate vertices; (v) Point-based triangulation; (vi) Polygon-based triangulation.

to skinny triangles, again shown in Figure 6.3. With these observations in mind, the quadtree proves to be a useful structure to decimate the interior points of a segment while preserving all boundary points for shape recovery [77].

6.5.1.1 Preprocessing

To prepare a planar segment for decimation it is first denoised and aligned to the x-y axes. We employ Principal Component Analysis (PCA) over the planar segment to compute a least-squares plane fit as well as an affine transformation T for x-y axes alignment, after which all points belonging to the segment are orthogonally projected onto the axis-aligned best fit plane. The aligned planar segment is denoted as $\mathcal{P}_t$. Afterwards, the boundary points of $\mathcal{P}_t$ are extracted as an α-shape [24, 93]. We denote the boundary as a concave hull $\mathcal{H}$ of the planar segment, which is an ordered list of vertices describing a polygon for which $\forall \mathbf{p} \in \mathcal{P}_t$ and $\mathbf{p} \notin \mathcal{H}$, $\mathbf{p}$ is inside the polygon.

6.5.1.2 Decimation

Planar segment point decimation consists of four steps as shown in Figure 6.4. Firstly, a quadtree is constructed by subdividing the bounding box of $\mathcal{P}_t$ into a uniform grid of small cells. Typically the 2D bounding box is non-square, in which case the smallest side is extended to equalise the width and height. The resulting bounding box $\mathbf{b}$ is composed of a minimum point $\mathbf{b}_{min}$ and a maximum point $\mathbf{b}_{max}$, with a dimension $\mathbf{s} = \mathbf{b}_{max} - \mathbf{b}_{min}$. Secondly, the quadtree nodes are classified as either interior, boundary or exterior. An *interior* node is fully contained within the polygon $\mathcal{H}$, while an *exterior* node is fully outside. All others are *boundary* nodes, which intersect $\mathcal{H}$. Thirdly, the interior nodes of the quadtree are merged to create nodes of variable size, typically largest around the center and becoming increasingly fine-grained when approaching the boundary. When a parent node contains only interior children, the four child nodes are merged into one. The merged node is then also classified as interior, allowing further recursive merging with its siblings. Finally, the corner points of the remaining interior nodes are extracted as the new internal vertices $\mathcal{I}$ of $\mathcal{P}_t$, while all boundary points $\mathcal{H}$ are preserved.

6.5.2 Triangulation

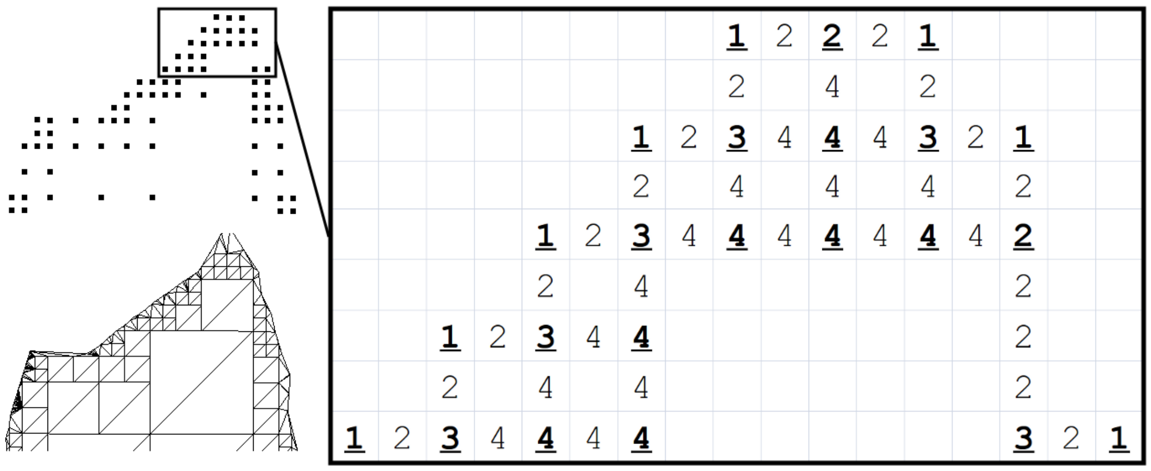
We provide two methods for triangulation of a simplified planar segment: 1) a low-complexity Point-Based Triangulation and 2) an alternative Polygon-Based Triangulation. Both methods make use of the Constrained Delaunay Triangulation (CDT) [23].

6.5.2.1 Point-Based Triangulation

The point-based approach is a low-complexity triangulation method, where CDT is directly applied to the decimated segment. The ordered boundary vertices $\mathcal{H}$ serve as constraining edges and the inner vertices $\mathcal{I}$ are used as input points. An example output is shown in Figure 6.4. Point-based triangulation has all of the advantages of Delaunay triangulation but does produce more triangles than the polygon-based approach described next.

6.5.2.2 Polygon-Based Triangulation

The regular grid pattern of the inner vertices $\mathcal{I}$ immediately lends itself to a simple triangulation strategy, where two right-angled triangles are created over each interior node of the merged quadtree. To complete the triangulation, the space between the interior right-angled triangles and the boundary points $\mathcal{H}$ is triangulated using CDT. Two sets of constraining edges are input to the CDT, one being $\mathcal{H}$ and the other being a rectilinear isothetic polygon that bounds interior triangles. This two-step triangulation is similar to the QTB algorithm of Ma *et al.* [77]. However, a major



								1	2	**2**	2	**1**				
								2		4		2				
						1	2	**3**	4	**4**	4	**3**	2	**1**		
						2		4		4		4		2		
				1	2	**3**	4	**4**	4	**4**	4	**4**	4	**2**		
				2		4								2		
		1	2	**3**	4	**4**								2		
		2		4		4								2		
1	2	**3**	4	**4**	4	**4**								**3**	2	**1**

Figure 6.5: Degree grid of part of a planar segment (0-valued cells hidden). The underlined bold values are the degrees of the inner vertices $\mathcal{I}$.

difference lies in how the boundary points are connected. With our CDT-based approach, we avoid overlapping triangles and artificial holes that would normally be produced by the QTB algorithm.

Efficient computation of the polygon which exactly bounds the interior vertices $\mathcal{I}$ is non-trivial, since the interior nodes provide only sparse spatial information for geometric operations. We invoke a solution that maps the interior vertices onto a binary image, where the bounding polygon can be easily extracted using a greedy nearest-neighbour tracing algorithm normally used in image processing [78].

The binary image is represented by an $n \times n$ array, where $n = 2^{d+1} + 1$ and d is the quadtree depth. This provides a 2D grid large enough to represent the empty space between the two vertices of any edge. To project a vertex $\mathbf{v} \in \mathcal{I}$ onto the array, a mapping function $f : \mathbb{R}^3 \to \mathbb{N}^2$ is defined by

$$f(\mathbf{v}) = \frac{n(\mathbf{v} - \mathbf{b}_{min})}{\mathbf{s}}, \tag{6.2}$$

where $\mathbf{b}$ is the bounding box and $\mathbf{s}$ is its dimension. The division is performed on an element-by-element basis. Given that $\mathcal{I}$ is aligned to the x-y axes, function f effectively maps from $\mathbb{R}^2$ to $\mathbb{N}^2$. We associate two elements with each array cell: a reference to the mapped vertex (effectively implementing f^{-1}) and a degree value to quantify vertex connectivity in the underlying quadtree. Initially, the degree is zero for all cells. During the triangulation of $\mathcal{I}$, the degree grid is populated. When a vertex is extracted from the merged quadtree, the reference of the corresponding cell is updated and its degree is increased by 1. This policy alone cannot fully recover the degree of a given vertex, since only the two ends of an edge are obtained from quadtree vertices. To overcome this problem, all cells between the two ends of an edge also have their degree increased by 2. Figure 6.5 shows part of the degree grid of a planar segment. If we consider the interior triangulation to be a graph, the 2D degree grid resolves the degree of each vertex. All non-zero cells are treated as "1-valued" foreground pixels and the rest as "0-valued" background pixels in the binary image representation.

6.6 Texture Generation

In this section we present our texture generation algorithm for planar segments using dense colored point clouds. Due to the significant loss of colored vertices during decimation, the appearance of a simplified planar segment is greatly diminished. We therefore generate textures prior to decimation for the purpose of texture mapping the simplified planar mesh.

We generate textures by projecting the vertex colors of the dense planar segment onto a 2D RGB texture $\mathcal{E}(x, y) \in \mathbb{N}^3$. We define a texture resolution $\mathbf{d}$ as some resolution factor r times $\mathbf{s}$, where $\mathbf{s}$ assumes the size of the bounding box $\mathbf{b}$. In our experiments a value of $r = 100$ provides good quality textures. The resolution factor can also be automatically computed based on point cloud density. Each pixel $\mathbf{a} \in \mathcal{E}$

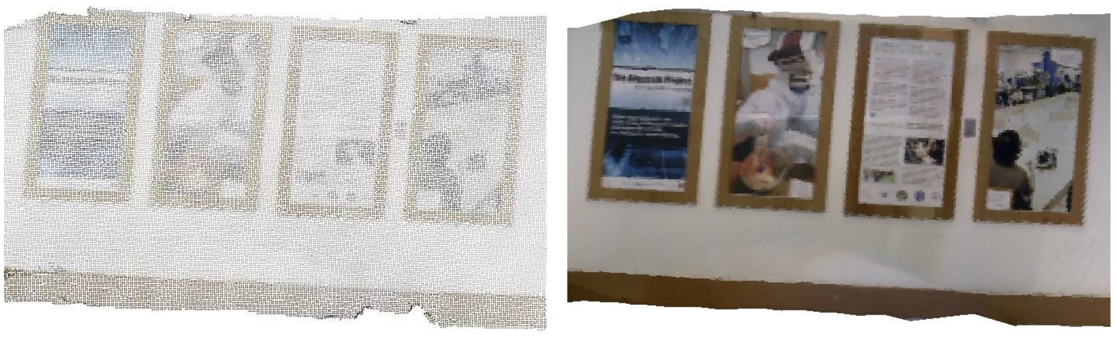


Figure 6.6: Texture generation, from left to right: (i) Plane segment from a colored point cloud; (ii) Generated texture.

is first mapped to a 3D point $\mathbf{v}$ by a mapping function $g : \mathbb{N}^2 \to \mathbb{R}^3$, defined as

$$g(\mathbf{a}) = \frac{\mathbf{as}}{\mathbf{d}} + \mathbf{b}_{min}, \tag{6.3}$$

with an element-by-element calculation. Since $\mathcal{P}_t$ is aligned to the x-y axes, the function g effectively maps to $\mathbb{R}^2$. A colored point corresponding to $\mathbf{v}$ in $\mathcal{P}_t$ is found by a nearest neighbour search using a kd-tree. We have chosen this approach as it produces good quality textures while being computationally inexpensive. However, it can be easily extended to produce even higher quality textures by averaging a number of k-nearest neighbours. Algorithm 6.5 describes the texture generation process. Figure 6.6 shows an input planar segment and the output texture.

Algorithm 6.5: Vertex color to texture.

```
Input: $\mathcal{P}_t$ set of transformed input vertices
       $\mathcal{H}$ concave hull of $\mathcal{P}_t$
Output: $\mathcal{E}$ 2D RGB texture
foreach pixel $\mathbf{p}$ in $\mathcal{E}$ do
    $\mathbf{v} \leftarrow g(\mathbf{p})$
    if $\mathbf{v}$ is inside $\mathcal{H}$ then
        $\mathbf{n} \leftarrow$ nearest-neighbour of $\mathbf{v}$ in $\mathcal{P}_t$
        $\mathbf{p} \leftarrow (\mathbf{n}_R, \mathbf{n}_G, \mathbf{n}_B)$
    else
        $\mathbf{p} \leftarrow (0, 0, 0)$
```

When texture mapping the final planar mesh, the *uv* texture coordinates $\mathcal{U}$ for the vertices $\mathcal{O}$ of each face are computed with the inverse function $g^{-1} : \mathbb{R}^3 \rightarrow \mathbb{N}^2$, derived from Equation (6.3) as

$$g^{-1}(\mathbf{v}) = \frac{\mathbf{d}(\mathbf{v} - \mathbf{b}_{min})}{\mathbf{s}}. \tag{6.4}$$

With *x*-*y* axes aligned points, g^{-1} is actually mapping from $\mathbb{R}^2$. Algorithm 6.6 describes the *uv*-coordinates computation. The list $\mathcal{U}$ guarantees a 1-to-1 mapping to the set $\mathcal{O}$.

Algorithm 6.6: *uv* texture coordinate calculation.

```
Input: $\mathcal{O}$ set of final face vertices
Output: $\mathcal{U}$ uv texture coordinates for $\mathcal{O}$
foreach vertex $\mathbf{v}$ in $\mathcal{O}$ do
    $\mathbf{a} \leftarrow g^{-1}(\mathbf{v})$
    $u \leftarrow \frac{\mathbf{a}_x}{\mathbf{d}_x}$
    $v \leftarrow 1.0 - \frac{\mathbf{a}_y}{\mathbf{d}_y}$
    Add $(u, v)$ to $\mathcal{U}$
```

Any objects lying on a planar segment are completely excluded from the texture and not projected onto the plane. In fact, the generated texture implicitly provides the Voronoi diagram of the face of the object lying on any plane, which in turn provides position and orientation information of any object lying on a segmented plane, as shown in Figure 6.7.

6.7 Summary

This chapter has detailed our method for simplifying dense point cloud maps into triangulated planar models in an accurate and computationally efficient manner. As we present in detail in Chapter 7, the approach we describe is superior to previous related methods on a number of both quantitative and qualitative measures, discussed spe-

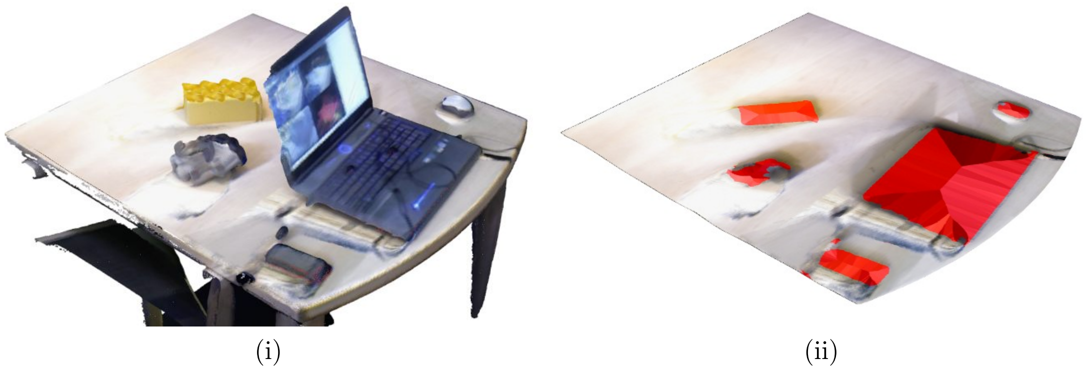


Figure 6.7: Implicit object information from texture generation, from left to right: (i) Input colored point cloud; (ii) Generated texture with implicit Voronoi diagrams and locations of objects resting on the plane highlighted.

cifically in Section 7.2. The contributions described in this chapter have immediate implications in the field of real-time robotics, with quick incremental online methods being of high importance. This is further justified in a real-world online robotics experiment which we describe in Chapter 8, demonstrating the utility and applicability of the map representation provided by the system described in this chapter (in particular when used in conjunction with systems such as the Kinect Monte Carlo Localisation (KMCL) system of Fallon *et al.* [29]).

CHAPTER 7

Experimental Results

In this chapter we present extensive experimental results on the techniques described in Chapters 3 through 6. Chapters 3 through 5 are concerned with accurate camera motion and scene structure estimation, which we evaluate different aspects of in Section 7.1. Chapter 6 is concerned with reducing the complexity of the captured scene models and is evaluated on a number of metrics in Section 7.2.

7.1 Dense Visual SLAM

We evaluate our SLAM system both quantitatively and qualitatively in terms of trajectory estimation, surface reconstruction and computational performance. We processed a combined total of over 79,000 unique RGB-D frames in our evaluation.

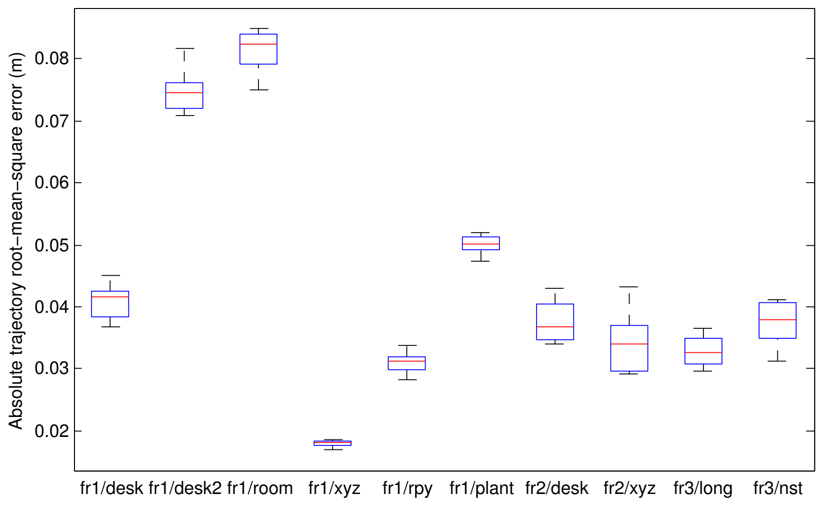


Figure 7.1: Boxplot of the ATE RMSE in metres per sequence evaluated. In each box the red central line is the median, the box edges the 25th and 75th percentiles and the whiskers extend to the minimum and maximum estimates. Each dataset was ran ten times to account for the randomness induced by the place recognition system in Section 5.2.

7.1.1 Trajectory Estimation

To evaluate the accuracy of our camera trajectory estimation we present results on the widely used RGB-D benchmark of Sturm *et al.* [115]. This benchmark provides synchronised ground truth poses for an RGB-D sensor moved through an environment, captured with a highly precise motion capture system. We evaluated multiple runs over ten datasets with quantitative results shown in Table 7.1 and a boxplot shown in Figure 7.1. We use the absolute trajectory (ATE) root-mean-square error metric (RMSE) to evaluate our system, which measures the root-mean-square of the Euclidean distances between all estimated camera poses and the ground truth poses associated by timestamp [115]. A brief description of each of the datasets captured and evaluated in this section is provided below (all datasets were captured by a human in a handheld manner);

1. fr1/desk: A 23 second trajectory over 9 metres containing several sweeps over four desks in a typical office environment.

2. fr1/desk2: A 24 second trajectory over 10 metres containing several sweeps over four desks in a typical office environment.

3. fr1/room: A 48 second trajectory over 16 metres containing a full 360 degree sweep of the inside of a small office room.

4. fr1/xyz: A 30 second trajectory over 7 metres where the sensor is pointed at an office environment and contains (mostly) only translatory motions along the sensor's principal axes.

5. fr1/rpy: A 27 second trajectory over 1 metre where the sensor is kept (mostly) fixed in translation and rotated around all three principal axes.

6. fr1/plant: A 41 second trajectory over 14 metres containing a full 360 degree scan of a small potted plant in an office environment.

7. fr2/desk: A 99 second trajectory over 19 metres containing a full 360 degree scan of a cluttered desk in a large open indoor environment.

8. fr2/xyz: A 122 second trajectory over 7 metres where the sensor is pointed at an office environment and contains (mostly) only translatory motions along the sensor's principal axes.

9. fr3/long: A 87 second trajectory over 21 metres where the sensor is carried around a full 360 degree scan of a large two sided desk setup in an open indoor environment.

10. fr3/nst: A 56 second trajectory over 13 metres containing a small loop around a highly textured ground plane with no strong geometric features.

Consistent performance is achieved on all sequences evaluated, with a notably higher error on the fr1/desk2 and fr1/room datasets (visually apparent by the large difference between the estimated trajectories and ground truth trajectories shown

Dataset	RMSE (m)	Median (m)	Max (m)	$\bar{\omega}$ ($^{\circ -1}$)
fr1/desk	0.0407	0.0352	0.0905	23.33
fr1/desk2	0.0747	0.0639	0.2309	29.31
fr1/room	0.0813	0.0739	0.2511	29.88
fr1/xyz	0.0180	0.0155	0.0392	8.92
fr1/rpy	0.0311	0.0213	0.0991	50.15
fr1/plant	0.0500	0.0425	0.1148	27.89
fr2/desk	0.0376	0.0315	0.0879	6.34
fr2/xyz	0.0341	0.0234	0.0979	1.72
fr3/long	0.0329	0.0297	0.0698	10.19
fr3/nst	0.0372	0.0335	0.0735	7.43

Table 7.1: Statistics on ATE on evaluated datasets. Trajectory values are in metres as the mean over ten runs of each dataset. The mean angular velocity is given as $\bar{\omega}$ in degrees per second, retrieved from the dataset specifications.

in Figure 7.2). This can be explained by the high average angular velocity on these sequences which causes motion blur, increases the effect of rolling shutter and violates the assumption of projective data association. From the results it can be seen that a higher RMSE is correlated with a high average angular velocity. Provided there is a low standard deviation in frame rate and good overlap between successive frames a strong trajectory estimate is achievable. Figure 7.2 shows two dimensional plots of the differences between the estimated trajectories and the ground truth trajectories. It should be noted that the subfigures within Figure 7.2 are rendered at varying scales to emphasise the variation within the datasets. Hence, for example, although the error for fr1/rpy is visually pronounced in the figure, as shown in Table 7.1 the translational error is at a similar order of magnitude to the other datasets. In all real world datasets evaluated the auto exposure and auto white balance features of the RGB-D camera were enabled.

7.1.1.1 Comparative Evaluation

We compare the trajectory estimation performance of our system to three recent state-of-the-art visual SLAM systems, DVO SLAM of Kerl *et al.* [63], RGB-D SLAM of Endres *et al.* [26] and multi-resolution surfel maps (MRS) of Stückler *et al.* [113].

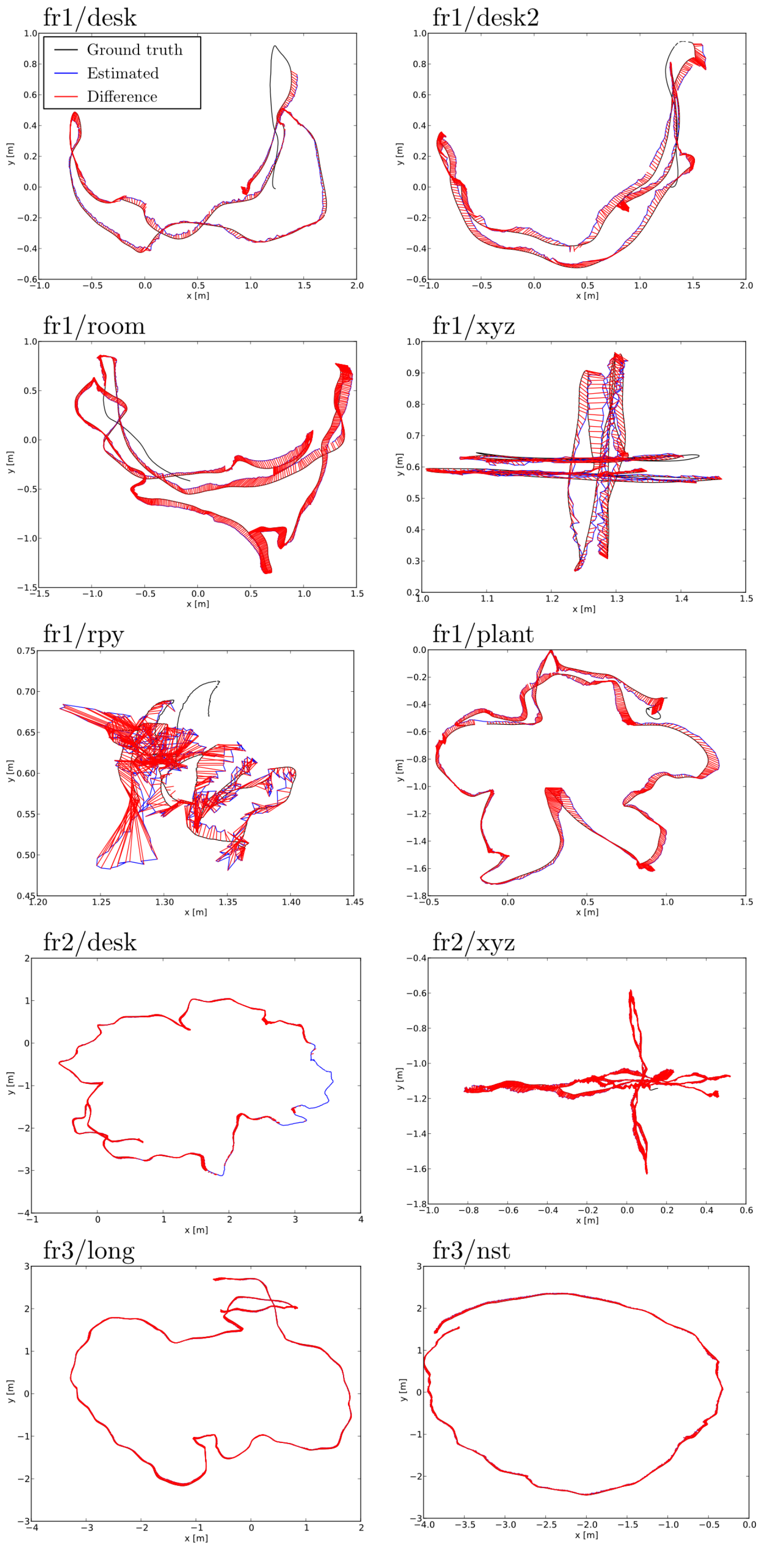


Figure 7.2: Two dimensional plot of estimated trajectories versus ground truth trajectories on evaluated sequences.

Table 7.2 summarises the results, where our values represent the best estimate over ten runs. From these we can see the performance of our system is comparable to other leading approaches, where performance of each algorithm is typically within no more than 3cm in total RMSE. We acknowledge the strong performance of the DVO SLAM system in trajectory estimation and perform a further comparison with their system in terms of reconstruction accuracy and larger trajectories in Section 7.1.2.2. We also provide a small comparison of results between our system and benchmark results provided by Meilland and Comport [81] from their unified keyframe SLAM system in Tables 7.3 and 7.4, again showing comparable performance (using their chosen metric of ATE Median and Max error, as opposed to RMSE).

In summary we can conclude from these trajectory estimation comparison results that the performance of the pose estimation component of the presented SLAM system is on par with other related state of the art systems. No one system scores the lowest error on all of the evaluated datasets but all systems are consistent in their performance, never more than 10cm away from the ground truth absolute trajectory. However, a large focus of this work has been on scalable dense surface reconstruction which cannot be evaluated by looking at trajectory estimates alone. In the following section we address the surface reconstruction quality and scalability aspect in more detail.

7.1.2 Surface Reconstruction

We present a number of quantitative and qualitative results on evaluating the surface reconstructions produced by our system. In our experience a high score on a camera trajectory benchmark does not always imply a high quality surface reconstruction due to the frame-to-model tracking component of the system. We found that although other methods for camera pose estimation may score better on benchmarks, the resulting reconstructions are not as accurate if frame-to-model tracking is not

Dataset	Ours (m)	DVO (m)	RGB-D (m)	MRS (m)
fr1/desk	0.037	0.021	0.023	0.043
fr1/desk2	0.071	0.046	0.043	0.049
fr1/room	0.075	0.053	0.084	0.069
fr1/xyz	0.017	0.011	0.014	0.013
fr1/rpy	0.028	0.020	0.026	0.027
fr1/plant	0.047	0.028	0.091	0.026
fr2/desk	0.034	0.017	0.057	0.052
fr2/xyz	0.029	0.018	0.008	0.020
fr3/long	0.030	0.035	0.032	0.042
fr3/nst	0.031	0.018	0.017	-

Table 7.2: Comparison of ATE RMSE on evaluated datasets and SLAM systems. All units given are in metres. MRS was unable to produce an estimate on the fr3/nst dataset.

Dataset	Ours (m)	Unified (m)
fr1/desk	0.031	0.018
fr2/desk	0.028	0.093
fr1/room	0.068	0.144
fr2/large_no_loop	0.256	0.187

Table 7.3: Comparison of ATE Median error on evaluated datasets and SLAM systems. All units given are in metres.

Dataset	Ours (m)	Unified (m)
fr1/desk	0.078	0.066
fr2/desk	0.079	0.116
fr1/room	0.231	0.339
fr2/large_no_loop	0.878	0.317

Table 7.4: Comparison of ATE Max error on evaluated datasets and SLAM systems. All units given are in metres.

being utilised. We evaluate seven different datasets captured in a handheld fashion across a wide range of environments, demonstrating the viability of our system for use over large scale trajectories both indoors and outdoors (within sensing limitations) and across multiple floors. It should be noted that it is technically possible for self-intersection to occur in the surface upon deformation. We have found this to be quite rare in practice as most deformations are quite smooth and do not deform the map in erratic ways. This aspect of the algorithm is one of the trade offs made in favor of computational performance. Following a brief description of each of the datasets captured and evaluated in this section is provided (each trajectory length is listed in Table 7.5);

1. Coffee: A human handheld dataset captured while walking around a single loop in a small coffee room with typical domestic structures present.

2. Indoors: A human handheld dataset captured while walking around a short single loop in a long corridor environment with low geometric complexity and varying degrees of visual texture.

3. Garden: A human handheld dataset captured outdoors at night with an LED array mounted on the sensor for illumination. The path follows a single loop with sweeping side to side pans of the camera in a cluttered semi-natural environment.

4. Outdoors: A human handheld dataset captured outdoors at night with an LED array mounted on the sensor for illumination. The path follows a very large single loop with long segments of data with only the ground plane visible (i.e., low geometric complexity).

5. Two floors: A human handheld dataset captured indoors over two floors of a large building closing a single large loop. The path takes the sensor through a wide variety of scenes including corridors, staircases and office environments.

6. In/outdoors: A human handheld dataset which spans both indoor and outdoor environments at night with an LED array mounted on the sensor for illumination. The path follows a complex trajectory which includes long stretches over areas of low geometric complexity (only a ground plane visible), transitioning into an indoor environment including travel up a staircase and down another, closing a first loop within the indoor environment before transitioning back outdoors to follow a new path back to the original starting position closing the second and final loop.

7. Apartment: A human handheld dataset captured indoors over two floors of an apartment. The path explored is very complex involving five loop closures at various points in the trajectory as three rooms, the landing and staircase are traversed. The trajectory is shown in pink in Figure 7.9.

7.1.2.1 Comparison to 2-pass Optimisation

In order to evaluate the accuracy of the deformation process we compare the resulting maps produced when a 2-pass approach is taken versus a single pass approach with a deformation for map correction. The 2-pass approach involves the following steps;

1. Build a pose graph with a camera pose for every frame.

2. Detect visual loop closures using the method described in Section 5.2.

3. At the end of the dataset, optimise the camera pose graph taking loop closure constraints into account.

4. Rerun the dataset using the optimised pose graph in place of the visual odometry frontend.

From here we can compare the two maps to determine a measure of similarity. This presents an interesting question as although the pose graphs for both the 2-pass and deformation-based maps are identical, the maps themselves may differ slightly due to the fact the 2-pass approach gives up frame-to-model registration on the second pass where the frustum-volume intersection may also slightly change. This means there will not be any reliable 1-to-1 point correspondences between the maps. For this reason we measure the map similarity by the residual error of a dense ICP-based registration of the maps. Given that both maps lie in the global coordinate frame we can iteratively minimise nearest neighbour point-wise correspondences between the two maps using standard point-to-plane ICP. This allows us to account for a small rigid transformation error between the two maps. We measure the remaining root-mean-square residual error between point correspondences as the residual similarity error between the two maps. Table 7.5 lists statistics on the seven evaluated datasets including the 2-pass residual registration error as well as the same error computed on maps deformed with a subsampled pose graph (2-pass fast), which we discuss in Section 7.1.3. It is clear that the deformation approach brings the map into strong alignment with the 2-pass output, with only a few millimetres in difference. This can be seen in Figure 7.3. Images of all datasets are provided in Figures 7.10-7.16. The Apartment dataset has a notably higher error than the other sequences, owing to the complexity of the trajectory and scene. However observing the reconstruction in Figure 7.16 it can be seen that a high quality map is still achieved.

7.1.2.2 Surface Reconstruction Comparison

In Figure 7.4 we present a comparison of the reconstructions produced by each of the systems evaluated in Section 7.1.1.1 on the fr1/xyz data. From this qualitative comparison it is evident that our approach benefits greatly from the use of a fused volumetic frontend, removing substantial noise from the reconstruction and producing a much cleaner model than other approaches. While the output from RGB-D SLAM, MRSMap and DVO SLAM can be fed through a signed distance fusion pipeline to

Dataset	Length(m)	v_d(m)	d_p(m)	Vertices	Volume(m^3)	2-pass(mm)	2-pass fast(mm)	Figure
Coffee	30.18	4.5	0.4	909,422	7,993	1.2	1.8	7.10
Indoors	49.57	7	0.4	1,603,116	21,918	2.7	4.9	7.11
Garden	71.49	6	0.8	2,418,331	28,340	2.1	2.5	7.12
Outdoors	152.05	6	0.8	2,961,966	34,711	2.3	8.8	7.13
Two floors	173.88	6	0.8	4,016,273	47,066	2.1	8.0	7.14
In/outdoors	317.95	6	0.8	5,985,669	70,145	2.8	7.5	7.15
Apartment	61.27	2.5	0.3	6,205,222	30,299	19.0	20.4	7.16

Table 7.5: Statistics on seven handheld datasets captured over a wide variety of environments using our approach.

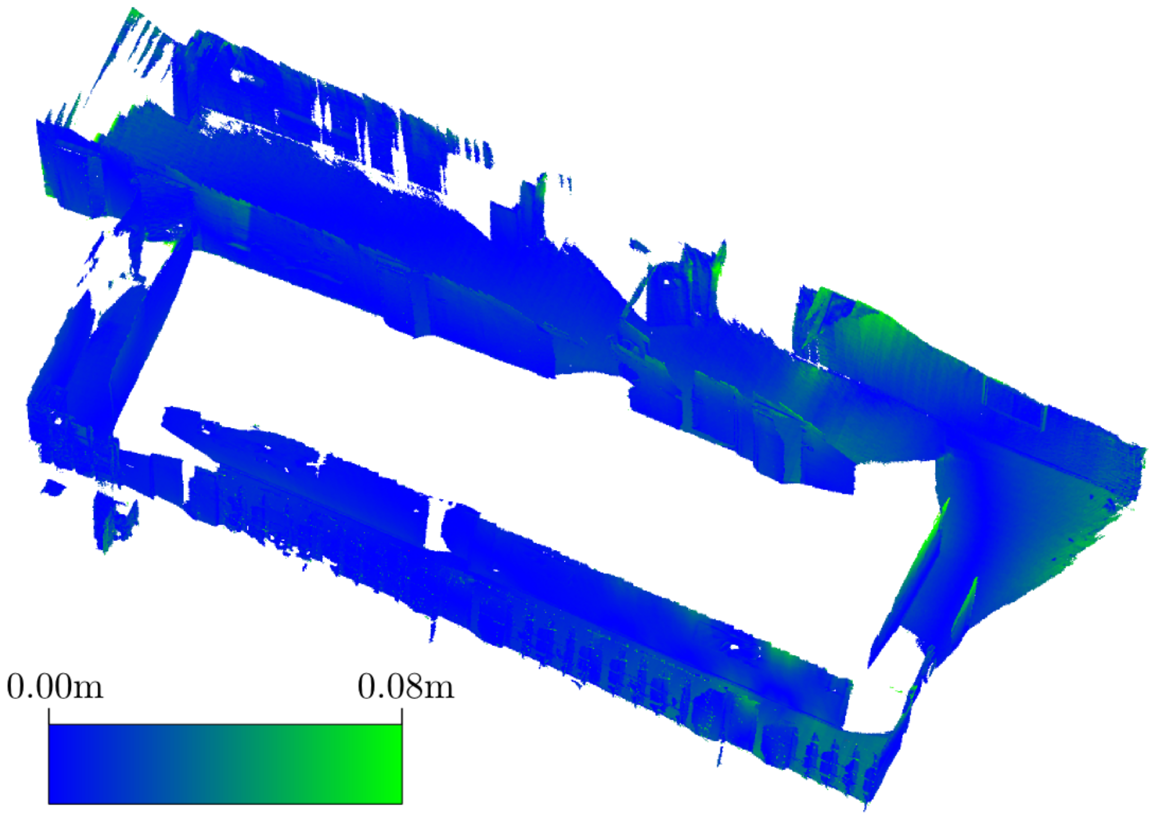


Figure 7.3: Heatmap showing the difference between the deformed reconstruction and 2-pass reconstruction of the Indoor dataset. Blue indicates no error and scales to pure green indicating an deviation of 0.08m.

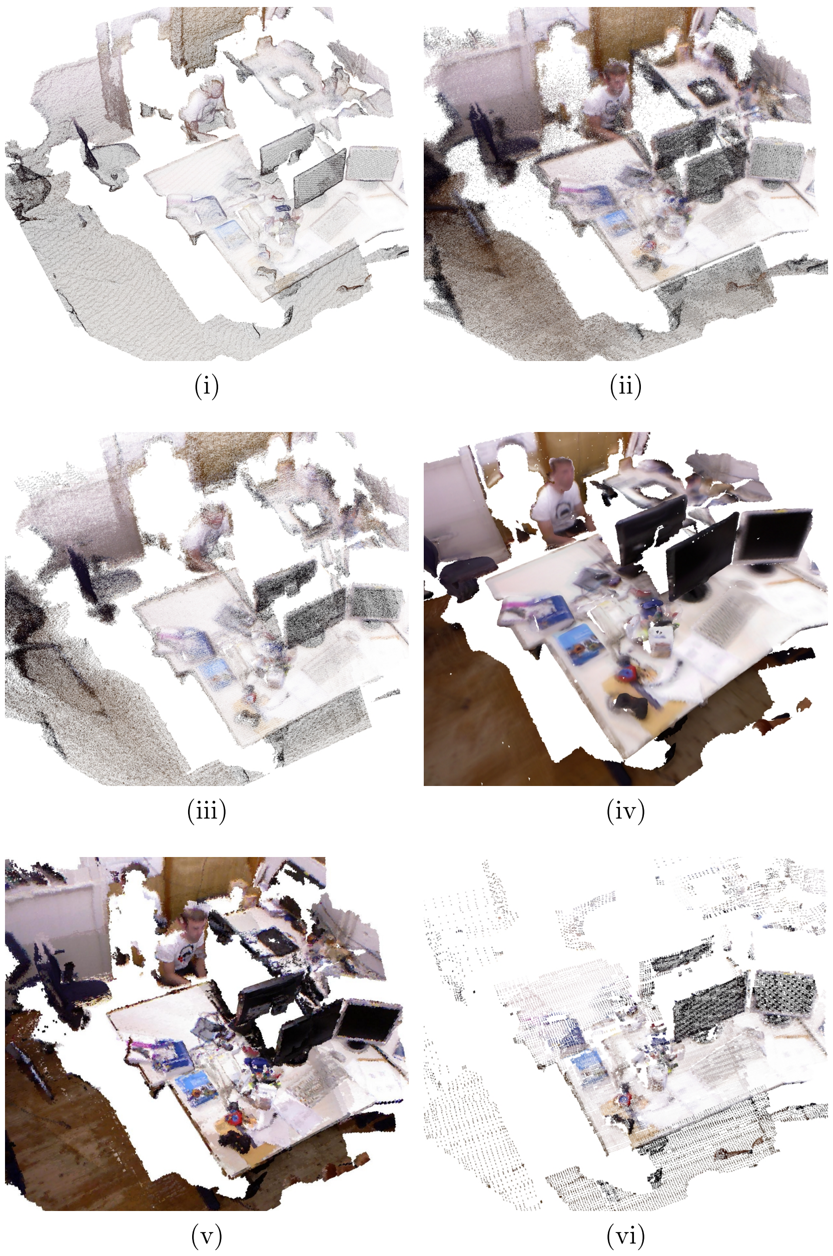


Figure 7.4: Comparison of reconstructions on the fr1/xyz dataset; (i) Point cloud reconstruction with our approach showing a smooth surface reconstruction. (ii) Reprojected keyframe reconstruction from RGB-D SLAM, showing a noisy surface with quantisation effects. (iii) Reprojected keyframe reconstruction from DVO SLAM again showing a noisy surface with quantisation effects. (iv) Triangular mesh reconstruction with our approach. (v) OctoMap [47] reconstruction from RGB-D SLAM, while in this form useful for motion planning, appears very jagged and is quantised to the nearest voxel. (vi) Point cloud sampled at highest resolution from surfel map with MRSMap showing an evident discretisation effect.

produce a similar output, this would strictly be a post-processing step that is not required by our own system to produce such reconstructions.

We also compare our reconstruction results on all of our seven evaluated datasets to the output produced by the open source DVO SLAM system of Kerl *et al.* [63], using the provided default configuration parameters. DVO SLAM performance statistics are listed in Table 7.6. In all datasets the DVO SLAM system frontend executed at 30Hz. The final pose graph optimisation and additional keyframe loop closure search time is listed in the "Post-processing" column. The DVO SLAM system uses no specific method for map reconstruction and must rely on point cloud reprojection of raw RGB-D keyframes to reconstruct the map after the pose graph has been optimised. This results in many redundant and repeated points in the map. To remedy this problem we apply a voxel grid downsampling filter (as mentioned in Section 3.3.2) with a resolution of 1cm to the output keyframe vertices to keep the map size tractable. These numbers are listed in the "Vertices" and "Vertices (filtered)" columns. As listed the system successfully reconstructs the Coffee and Indoors datasets, however in contrast to our approach post-processing time of between 7 and 31 seconds is required to optimise the final pose graph and resolve any additional keyframe loop closures (where our system does not require any post-processing or final batch steps). Failure to detect loop closures results in failed reconstructions on the Garden, Outdoors, Two floors, In/outdoors and Apartment datasets which could perhaps be remedied by using a bag-of-words visual features-based approach similar to ours or indeed, as suggested by Kerl *et al.*, the FAB-MAP algorithm [13]. Camera pose estimation failures were also encountered in the Outdoors, Two floors, In/outdoors and Apartment datasets, particularly in regions of the sequences which were mostly planar or had strong visual aliasing, such as staircases. From these results and those listed in Table 7.2 we observe that the method for detecting loop closures used by DVO SLAM is very strong in small sized environments but scales poorly as the explored area size grows, both

Dataset	Vertices	Vertices (filtered)	Post-processing (s)	Verdict	Figure
Coffee	34,813,313	3,925,307	7.41	Successful	7.5 (i)
Indoors	56,927,008	15,960,983	31.65	Successful	7.5 (iii)
Garden	145,054,728	15,385,263	159.52	Loop Closure Failure	N/A
Outdoors	104,948,878	9,106,848	190.76	Loop Closure & Tracking Failure	N/A
Two floors	237,308,674	26,724,533	841.93	Loop Closure & Tracking Failure	N/A
In/outdoors	275,096,207	18,836,984	5501.58	Loop Closure & Tracking Failure	N/A
Apartment	83,102,006	6,068,711	74.79	Loop Closure & Tracking Failure	N/A

Table 7.6: Statistics on seven handheld datasets captured over a wide variety of environments processed using the DVO SLAM system.

in terms of accuracy and computational performance (embodied in the consistently increasing post-processing time).

In Figure 7.5 we qualitatively compare the reconstruction quality of our approach versus the maps produced by DVO SLAM on the Coffee and Indoors datasets. For clarity we compare vertices only as DVO SLAM provides no method for mesh surface reconstruction. These results show that the reconstruction produced by our approach is much smoother and contains significantly less redundant vertices. Additionally, there are no raw RGB-D point cloud quantisation effects in our reconstructions. The reprojection principle taken to producing the map from DVO SLAM keyframes does result in entire frame back-projection which produces a more "fuller" looking map, however far away points in current generation RGB-D sensors are known to be extremely noisy and highly inaccurate [64].

To summarise our surface reconstruction comparison results, it can be observed that qualitatively the reconstructions produced by the SLAM system presented in this thesis are smoother and overall less noisy when compared to other approaches. This is down to the fact that our system actively fuses every depth map captured within the frontend to remove noise and improve the reconstruction, in contrast to other SLAM systems which mostly rely on the map for pose estimation alone (in fact most related work, with the exception of that of Meilland and Comport [81], focusses very strongly on trajectory estimation and not the actual map reconstruction quality). Additionally, looking at Table 7.6 it is shown that our approach scales to significantly larger environments and longer trajectories without sacrificing performance. To date no other dense RGB-D visual SLAM system has been demonstrated to function completely in real-time with no batch steps to produce a globally consistent surface reconstruction over the extended scale trajectories which we have shown.

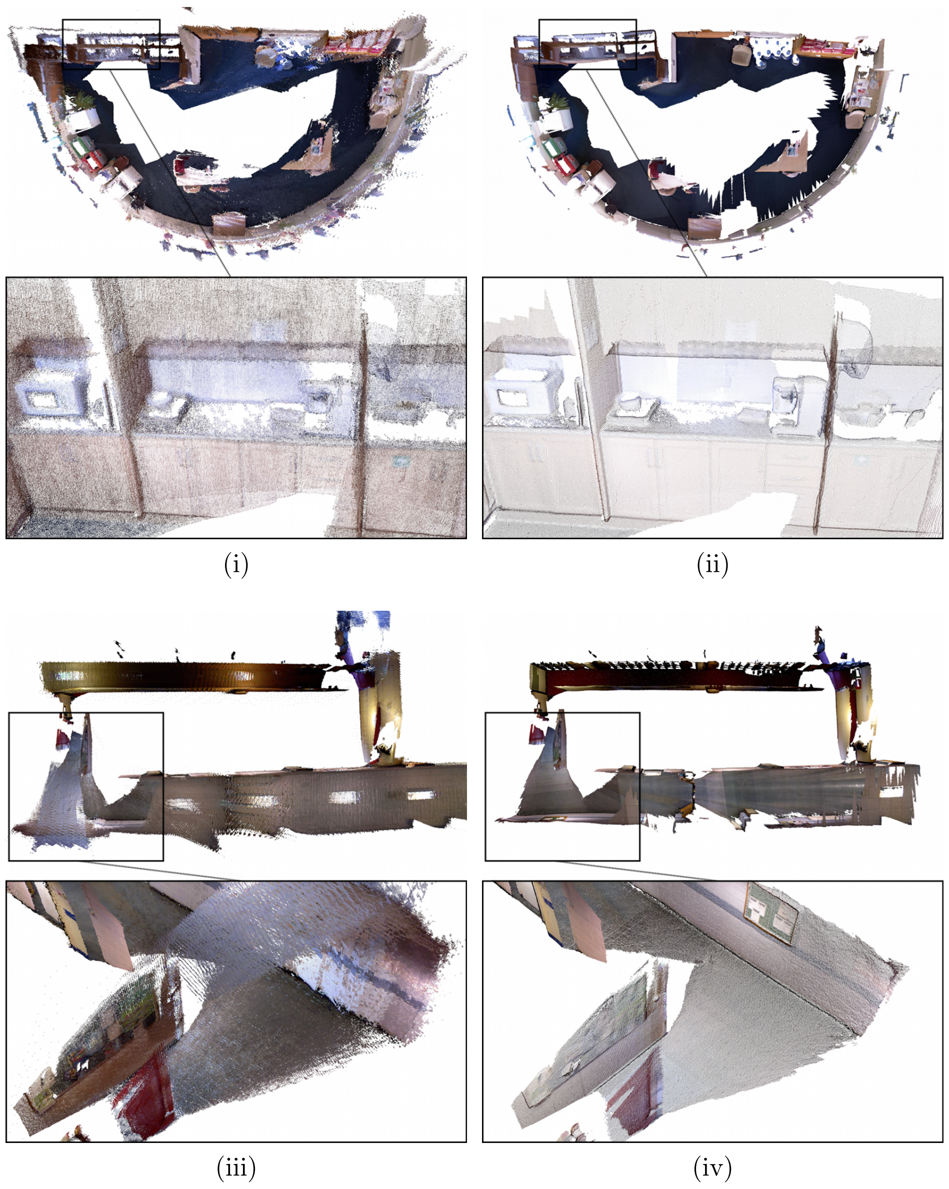


Figure 7.5: From top to bottom; (i) DVO SLAM keyframe reprojection of the Coffee dataset. The surfaces are notably noisy and quantisation effects are evident. (ii) Our reconstruction of the Coffee dataset, showing a smooth uniform reconstruction. (iii) DVO SLAM keyframe reprojection of the Indoors dataset. Again surfaces are very noisy and highly quantised. In contrast to our reconstruction, the ceiling has been mapped in most of the sequence, however being quite distant from the sensor suffers badly from discretisation effects. (iv) Our reconstruction of the Indoors dataset. The ceiling has not been reconstructed in this sequence since the configuration of the TSDF volume size caused it to fall outside of the area of reconstruction. This is however easily remedied by modifying the relative parameterisation of the volume with respect to the sensor, similar to the dynamic cube positioning technique we discussed in Section 3.4.

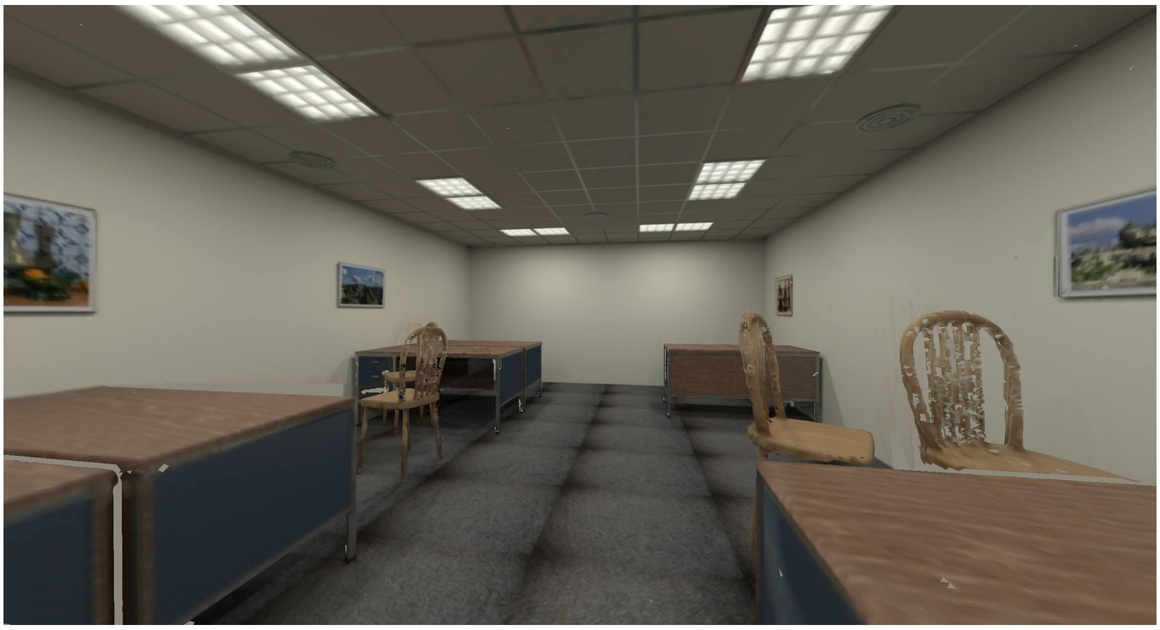


Figure 7.6: Mesh reconstruction of the first synthetic dataset. Note that the rough triangulation of parts of the chairs is due to a poor viewing angle throughout the sequence.

7.1.2.3 Surface Ground Truth

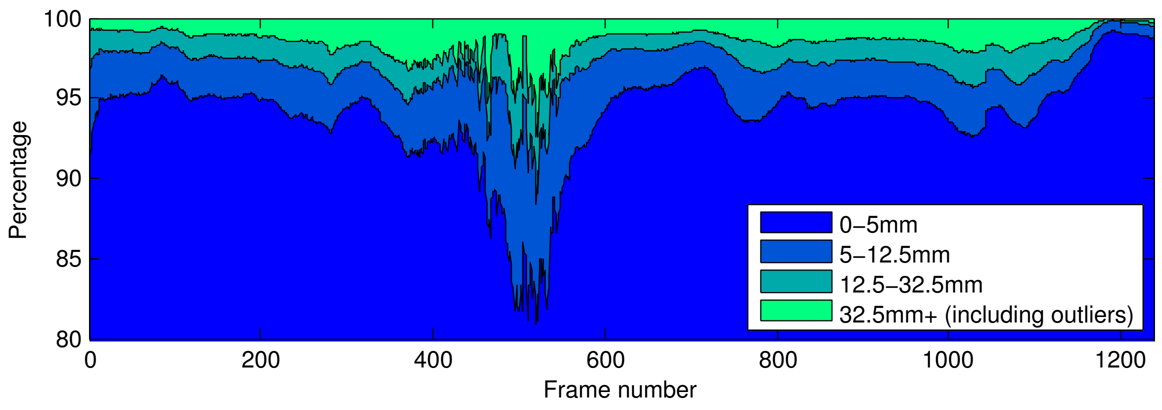
We evaluate the surface reconstruction quality of our approach quantitatively using synthetic data produced in an identical manner to the datasets created by Handa *et al.* [42]. Each dataset contains 30Hz RGB-D frames from a camera placed in a synthetic office environment. The camera trajectories were generated from real world data which was previously ran through our visual odometry frontend. Given that the datasets were produced using a procedural raytracing process (using POVRay), there is no actual surface to compare against. However, each RGB-D frame does have ground truth depth information which we compare against. For each frame in a dataset we compute a histogram of the per depth pixel L_1-norm error between the ground truth depth map and the predicted surface depth map raycast from the TSDF, normalising by the number of valid pixels before aligning all histograms into a two dimensional area plot. We evaluated two synthetic datasets of the same scene with different camera motions. The temporal error histograms are shown in Figure 7.7 while frames from each dataset are shown in Figure 7.8. Overall the synthetic surfaces are reconstructed very well, however occasional raycasting artifacts (particularly around the edges of objects and on nearby surfaces) can hinder the reconstruction quality score, as in the first dataset. These artifacts occur due to the fact that a fixed step size is used during ray casting and no special method is invoked to render smooth edges, both for performance reasons. Observing the final reconstruction in Figure 7.6 it is clear that the slight dip in accuracy did not effect the reconstruction quality by any significant amount. Typically around 95% of the estimated depth of the surface is within 5mm of ground truth.

7.1.3 Computational Performance

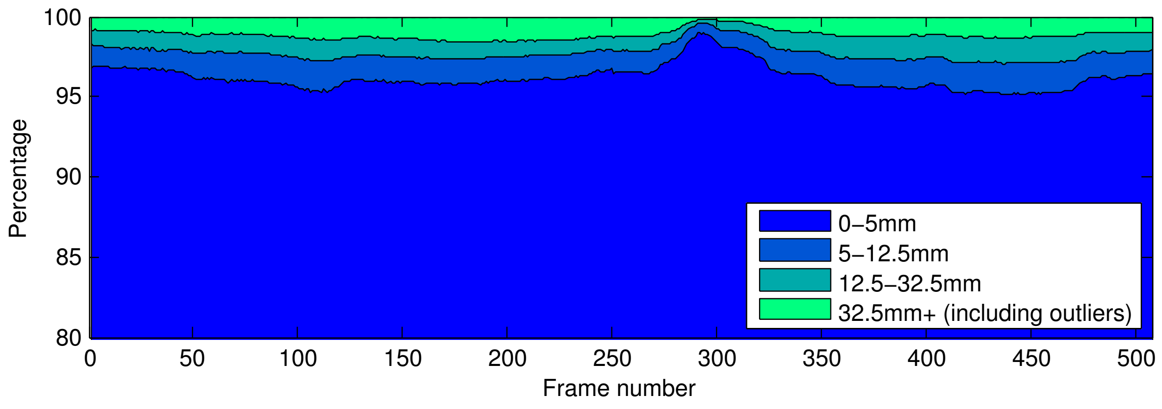
We evaluate the computational performance of both the frontend and backend of the system. The evaluation platform was a standard desktop PC running Ubuntu 12.04 with an Intel Core i7-3960X CPU at 3.30GHz, 16GB of RAM and an nVidia GeForce 680GTX GPU with 2GB of memory.

7.1.3.1 Frontend Performance

To evaluate the performance of the frontend (including volume integration, camera pose estimation, volume raycasting and volume shifting, essentially all teal colored function blocks in Figure 5.1) we provide frame processing timing results on the fr1/desk sequence comparing different choices of the m_s parameter discussed in Section 3.3. This parameter affects the frequency and size of each volume shift, which in turn affects frontend performance. Results are shown in Table 7.7. A shifting threshold of 16 voxels was found to be optimal, providing the best computational performance with an average frame rate comfortably above the frame rate of the sensor (30Hz) and with minimal spikes in execution time.



(i)

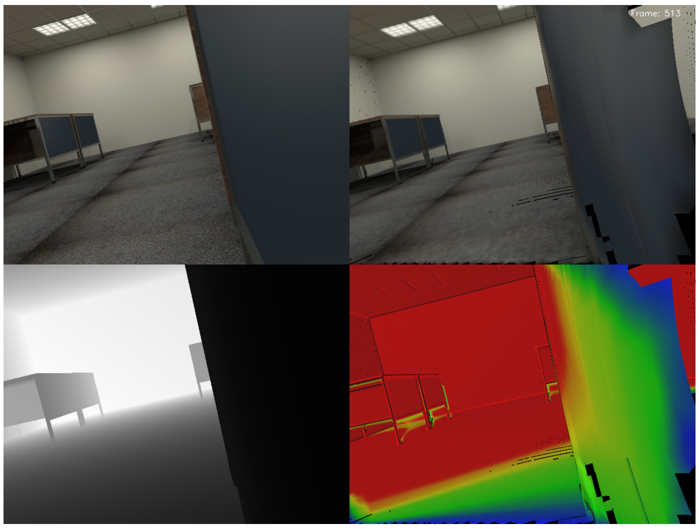


(ii)

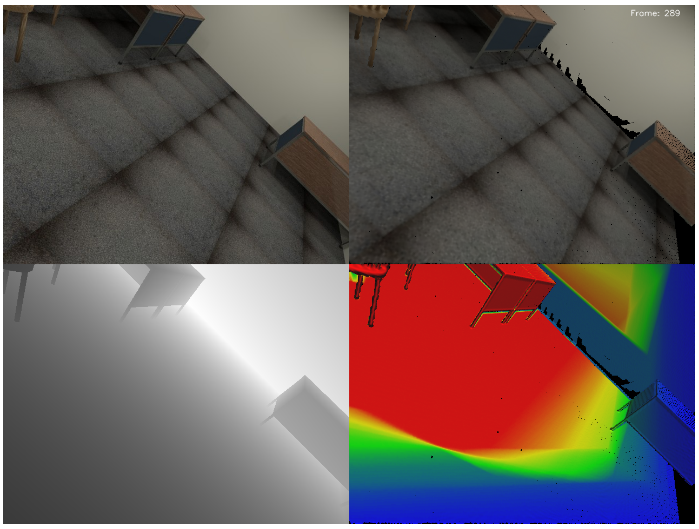
Figure 7.7: Temporal histograms of predicted depth versus ground truth depth on synthetic datasets. A frame from the dip in accuracy around the center of the first dataset is shown in Figure 7.8 (i) while a frame from the peak in accuracy in the center of the second dataset is shown in Figure 7.8 (ii).

m_s	Avg (ms)	Min (ms)	Max (ms)	StdDev (ms)
1	34.15	25.93	41.58	3.30
2	32.21	25.63	39.29	3.14
4	31.08	25.38	39.02	2.77
8	30.57	25.42	37.44	2.48
16	**29.94**	**24.97**	**37.25**	**2.26**
32	30.26	25.33	40.30	2.39
64	30.49	25.06	43.95	2.73

Table 7.7: Computational performance of the volumetric fusion thread on the fr1/desk dataset. The shifting threshold m_s is given in voxels while the frame processing timings are given in milliseconds. Highlighted is the optimal choice based on execution time.



(i)



(ii)

Figure 7.8: One frame from each surface ground truth evaluation dataset. Each shows in clockwise order the ground truth RGB, predicted RGB, predicted surface phong shaded by voxel weight and ground truth depth map. From top to bottom; (i) Here raycasting artifacts are visible in the predicted surface in the bottom right causing a high error in the evaluation; This is evident particularly in the top right-hand corner of the frame where the wall is visible through the side of the desk. (ii) Overall the surface is being well estimated and there are no raycasting artifacts.

7.1.3.2 Backend Performance

We quantify the computational performance of the backend in the context of an online real-time SLAM system by measuring the latency of the system. That is, how long is takes for 1) a loop closure to be recognised when one is encountered and 2) map correction to be completed. Table 7.8 shows execution time and latency statistics on our test platform for the first six datasets, while Table 7.10 shows performance statistics on the Apartment dataset. We also experimented with subsampling the pose graph used in the iSAM-based pose graph optimisation by the same sampling metric used in Algorithm 5.1. This affects the number of poses used in the final pose graph optimisation and the number of points available to constrain the map deformation in Equation 5.7.

Our results (shown in Tables 7.9 and 7.11) show that using a subsampled pose graph (akin to using only keyframes) instead of an every frame pose graph reduces execution time (and therefore latency) by up to almost an order of magnitude in some cases, while only mildly affecting map quality (quantified as "2-pass fast" in Table 7.5). As expected the appearance-based frontend scales very well over hundreds of metres while the backend is capable of correcting millions of vertices for global consistency in only 1-3 seconds. The results presented in Tables 7.10 and 7.11 demonstrate the capability of our approach to deal with complex trajectories with multiple loop closures. This is further highlighted by the plot of the camera trajectory on the Apartment dataset shown in Figure 7.9.

7.2 Map Simplification

In this section we evaluate our work on map simplification with a series of experiments. Four colored point clouds of real-world environments were used in the experiments, as shown in Figure 7.22i on page 122. These datasets encompass a wide variation in

Quantities	Datasets						
	Coffee	Indoors	Garden	Outdoors	Two floors	In/outdoors(1)	In/outdoors(2)
DBoW images	280	301	658	1171	1584	1662	2706
Poses	1544	2993	8634	5240	12952	17306	25586
Nodes	58	55	72	178	191	211	364
Vertices	932,056	1,352,919	2,256,475	2,805,083	3,896,281	3,560,994	5,867,125
Process	Timings (ms)						
Frontend	465	602	622	587	657	521	543
iSAM	257	510	1412	1299	3326	4386	6545
Deformation	266	390	1112	928	2197	3040	4473
Total latency	988	1502	3146	2814	6180	7947	11561

Table 7.8: Computational performance statistics on six datasets using an every frame pose graph. Quantities shown are at the moment of loop closure. The In/outdoors dataset contains two looping points which are both listed.

Quantities	Datasets						
	Coffee	Indoors	Garden	Outdoors	Two floors	In/outdoors(1)	In/outdoors(2)
DBoW images	277	305	672	1173	1593	1713	2782
Poses	283	307	674	1186	1594	1716	2783
Nodes	52	49	68	167	181	196	339
Vertices	943,721	1,371,560	2,246,028	2,841,135	3,904,113	3,569,842	5,850,152
Process	Timings (ms)						
Frontend	488	589	651	597	467	540	793
iSAM	46	67	110	288	378	271	1140
Deformation	110	105	170	377	381	148	842
Total latency	644	761	931	1262	1226	959	2775

Table 7.9: Computational performance statistics on six datasets using a subsampled pose graph. Quantities shown are at the moment of loop closure. The In/outdoors dataset contains two looping points which are both listed.

	Apartment dataset with full pose graph				
Loop number	1	2	3	4	5
DBoW images	119	526	708	982	1428
Poses	367	1638	2163	2824	3937
Nodes	14	61	80	105	165
Vertices	492,960	2,792,446	3,800,812	4,482,186	6,296,542
Process	Timings (ms)				
Frontend	807	858	1596	703	604
iSAM	29	202	277	230	648
Deformation	51	336	425	425	932
Total latency	887	1396	2298	1358	2184

Table 7.10: Computational performance statistics on the Apartment dataset using an every frame pose graph. Quantities shown are at the moment of loop closure.

	Apartment dataset with subsampled pose graph				
Loop number	1	2	3	4	5
DBoW images	119	529	708	982	1430
Poses	123	531	715	988	1433
Nodes	13	59	77	100	157
Vertices	492,718	2,791,445	3,799,464	4,490,170	6,295,379
Process	Timings (ms)				
Frontend	789	868	1557	789	593
iSAM	19	64	93	88	235
Deformation	31	181	252	285	508
Total latency	839	1113	1902	1162	1336

Table 7.11: Computational performance statistics on the Apartment dataset using a subsampled pose graph. Quantities shown are at the moment of loop closure.

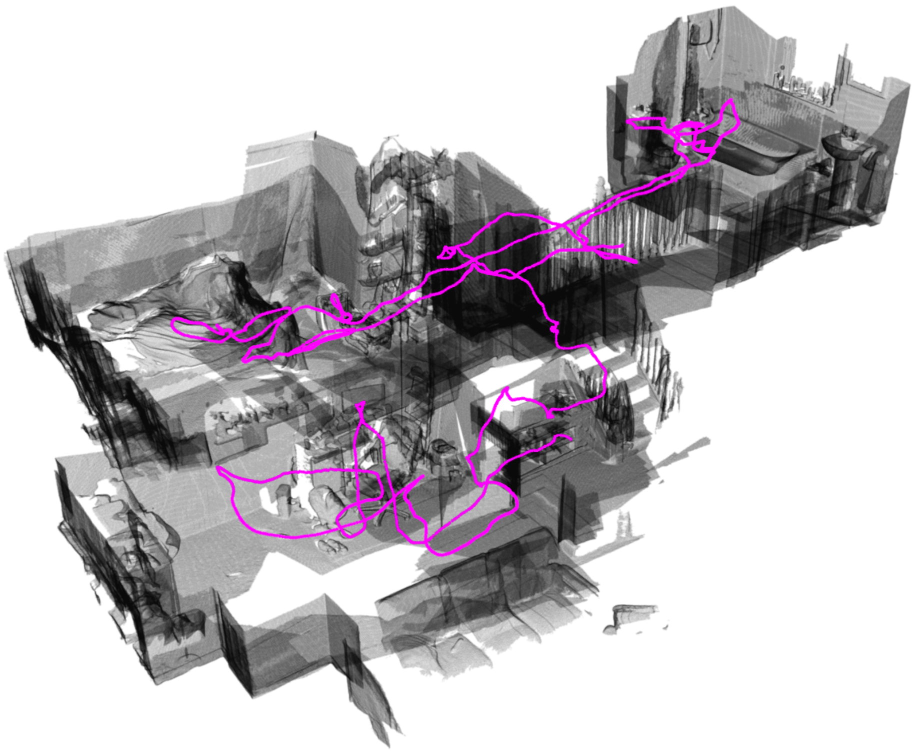


Figure 7.9: Camera trajectory plot within the Apartment dataset, showing the "loopy" path the camera took through the environment.

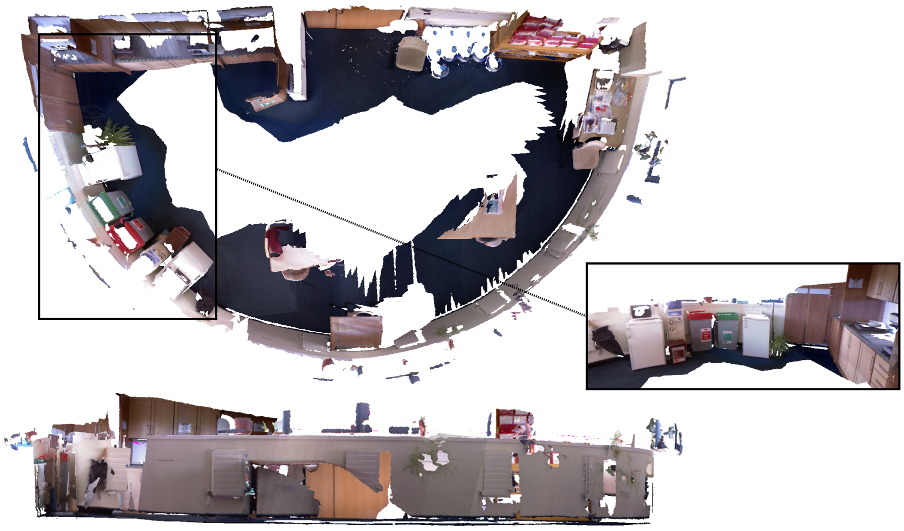


Figure 7.10: Dataset of a small coffee room. Inset shows everyday objects such as bins and fridges are captured in high detail and how the deformation approach works well in smaller environments.

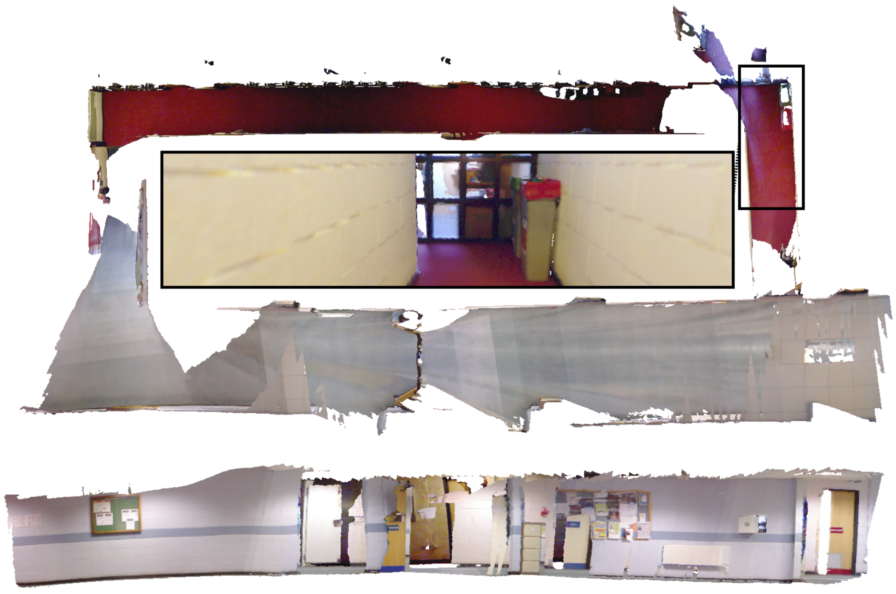


Figure 7.11: Corridor loop closure dataset. The inset shows map consistency at the point of loop closure.

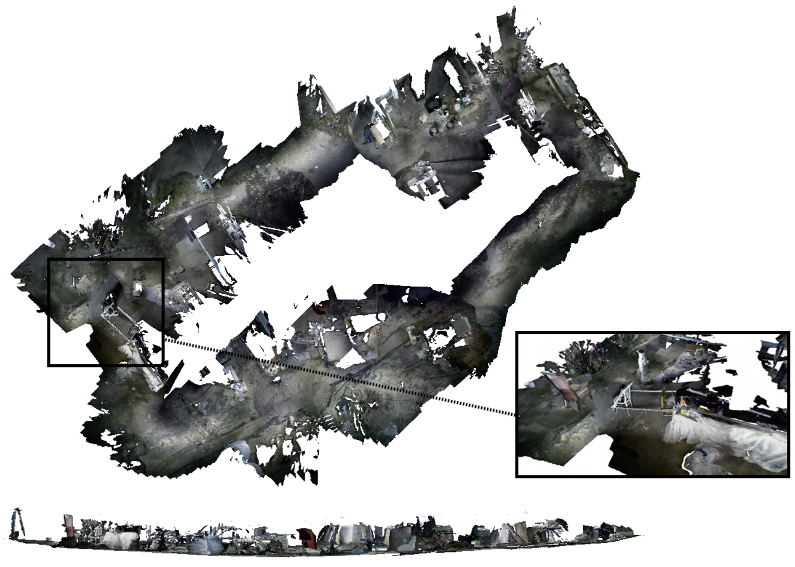


Figure 7.12: Large cluttered outdoor dataset. Inset shows chairs and metal bars are reconstructed well.

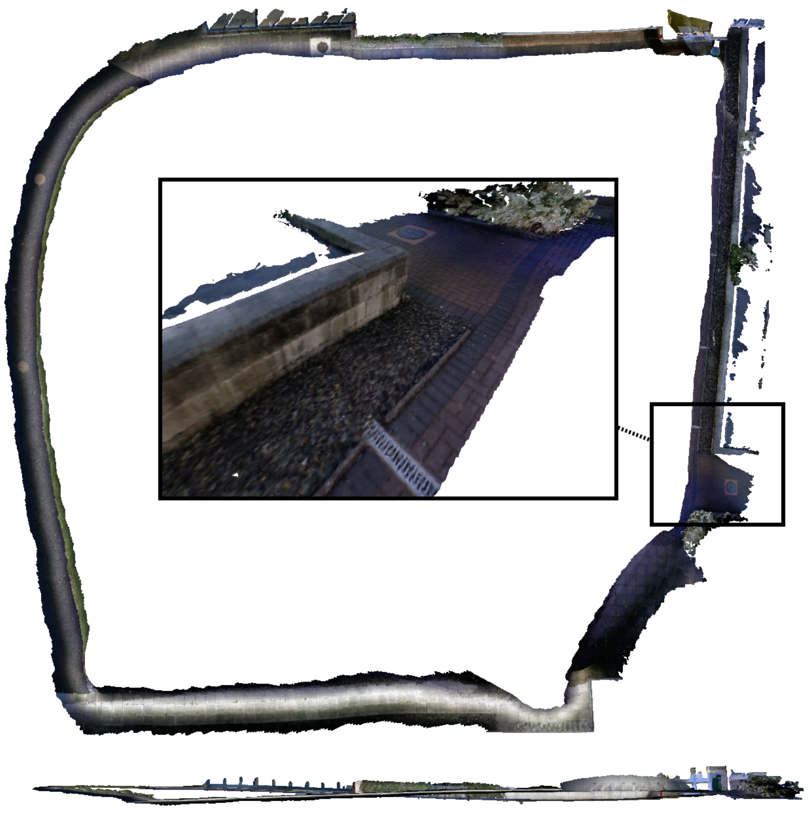


Figure 7.13: Large outdoor dataset. Inset shows brickwork is clearly visible.

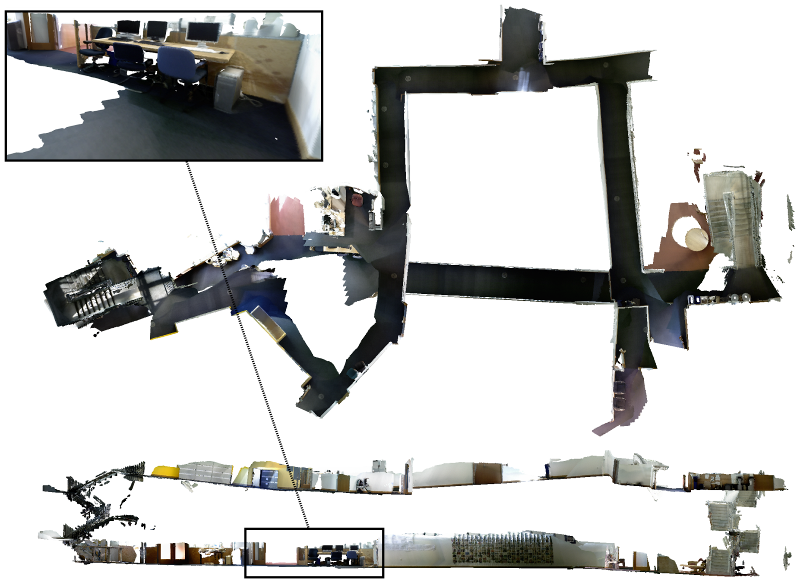


Figure 7.14: Dataset composed of two floors. Inset shows everyday objects such as chairs and computers are captured in high detail.

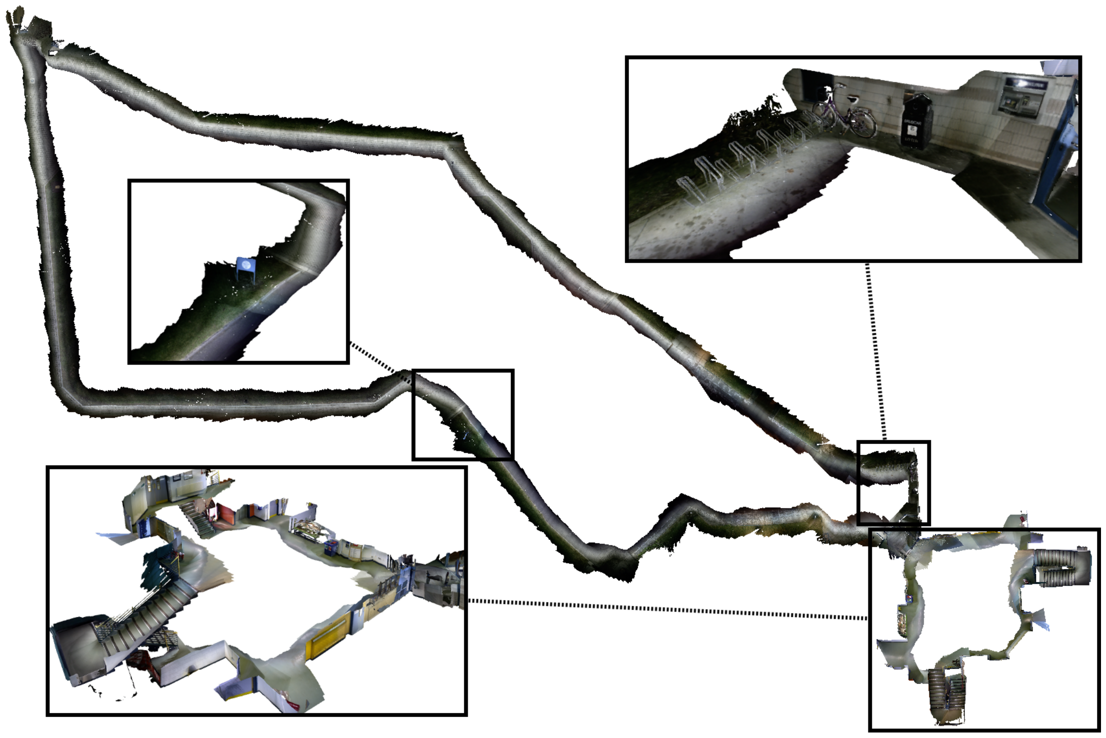


Figure 7.15: Large indoor and outdoor dataset made up of over five million vertices. Insets show the high fidelity of small scale features in the map.

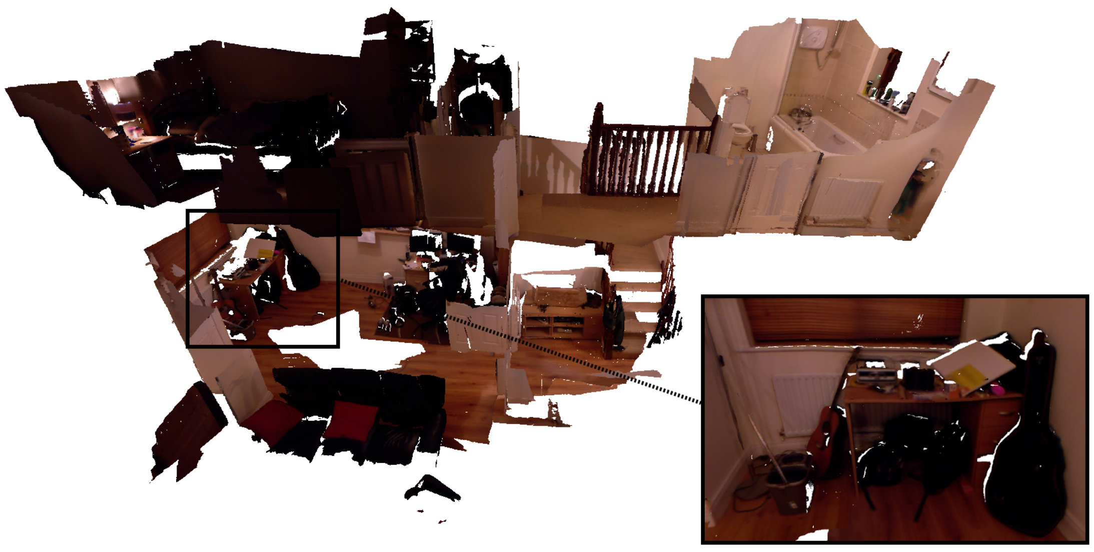


Figure 7.16: Sequence over two floors of an apartment with over six million vertices. Small details such as bathroom fixtures and objects around the environment are clearly reconstructed.

the number of points, planar segments and their geometry. All four datasets have been acquired with the SLAM system described in the previous chapters. A brief description of each of the four evaluated datasets follows (all were captured in human handheld sensor trajectories);

1. A long map spanning a large foyer floor followed by a tall set of stairs.

2. A small cluttered map spanning the area of a studio apartment.

3. A short map spanning a short segment of a corridor environment.

4. A tall map spanning a small seating area followed by a large spiraling staircase and partial upper level.

7.2.1 Incremental Segmentation Performance

To evaluate the performance of the incremental segmentation process listed in Section 6.4 we compare the batch segmentations to the incremental segmentations of the four datasets both qualitatively and quantitatively. We use the open source software CloudCompare (`http://www.danielgm.net/cc/`) to align the batch and incremental models of each dataset together to compute statistics. We quantify the quality of the incremental segmentation versus the batch segmentation by using the “cloud/mesh" distance metric provided by CloudCompare. The process involves densely sampling the batch planar model mesh to create a point cloud model which the incremental model is finely aligned to using ICP. Then, for each vertex in the incremental planar model, the closest triangle in the batch model is located and the perpendicular distance between the vertex and closest triangle is recorded. Five standard statistics are computed over the distances for all vertices in the incremental model: Mean, Median, Standard Deviation, Min and Max. These are listed for all four datasets in Table 7.12.

Figure 7.17 shows heatmap renderings of the “cloud/mesh” error of each incremental segmentation compared to the batch segmentation. Notably in each dataset

Dataset	1	2	3	4
Mean (m)	0.020	0.038	0.111	0.028
Median (m)	0.015	0.004	0.015	0.010
Std. (m)	0.021	0.108	0.251	0.065
Min (m)	0.000	0.000	0.000	0.000
Max (m)	0.157	0.823	1.389	0.719

Table 7.12: Incremental versus batch planar segmentation statistics. All values shown are in metres, on the distances between all vertices in the incrementally segmented model and the nearest triangles in the batch segmented model.

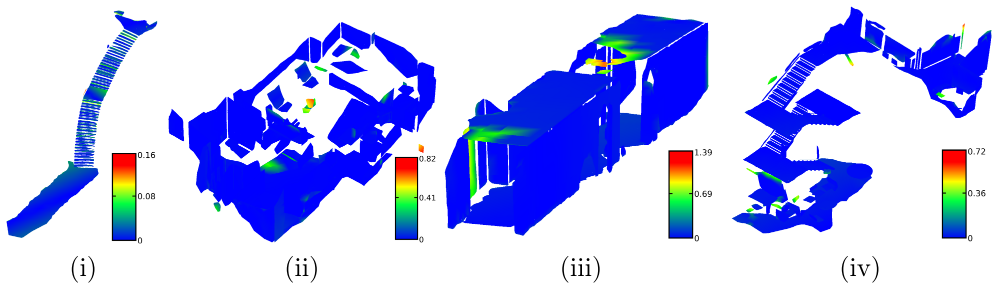


Figure 7.17: Heatmaps based on the distances between all vertices in the incrementally segmented model and the nearest triangles in the batch segmented model for all four datasets. Color coding is relative to the error obtained where blue is zero and tends towards green and red as the error increases.

there are a number of highlighted green planes, these are planes which were not detected in the incremental segmentation model but exist in the batch segmentation. In general the incremental segmentation occasionally fails to segment small planar segments whereas the batch segmentation always finds all planes that match the criterion set out in Algorithm 6.1. Additionally, as the incrementally grown planes use a moving average for the planar segment normal, some planes may have a slightly different orientation when compared to the batch model. This is evident in particular in Figure 7.17 (i) and (iii). Taking this qualitative information into account as well as the statistics in Table 7.12 we find that the incremental segmentation algorithm produces segmentations extremely close to what would be achieved using the batch process and is suitable to use in a real-time system that must generate and use the planar model online.

Dataset	1	2	3	4
Total points	890,207	1,094,910	2,799,744	5,641,599
Planar points	540,230	708,653	1,303,012	2,430,743
QTB decimation	105,663	303,348	189,772	457,678
Our decimation	47,457	84,711	43,257	76,624

Table 7.13: Planar point reduction with our decimation algorithm in comparison to the QTB algorithm.

7.2.2 Triangulation Performance

To assess the triangulation performance, qualitative and quantitative evaluations are presented. A comparison of the triangulation algorithms is shown in Figure 7.22. It can be seen that both algorithms produce a highly simplified triangulation, while preserving the principal geometry of the planar segments. Further assessment of mesh quality is done by measuring the angle distribution across meshes. A naïve simplified planar mesh is set as a baseline, which applies Delaunay triangulation to only the boundary points of a planar segment. The normalised distribution is shown in Figure 7.18, collected from the 400 planar segments of the four datasets. Taking this figure into account along with the qualitative results shown in Figures 6.2 (iv) and 6.4 (v) we can infer that approximately 80% of the triangles from the polygon-based triangulation are isosceles right-angle triangles, resulting from the quadtree-based triangulation. With point-based triangulation, the angles spread over 30°-90°, whereas the naïve boundary-based triangulation shows an even more random distribution. Defining a skinny triangle as one with a minimum angle <15°, the percentages of skinny triangles with boundary-based, point-based and polygon-based triangulation are 28%, 10% and 10%, respectively.

The effectiveness of planar segment decimation is also evaluated. Table 7.13 shows the point count for planar point decimation. Approximately 90% of the redundant points are removed with our algorithm, which is 15% more than the QTB algorithm, despite the fact that both algorithms are based on a quadtree. Part of this reduction

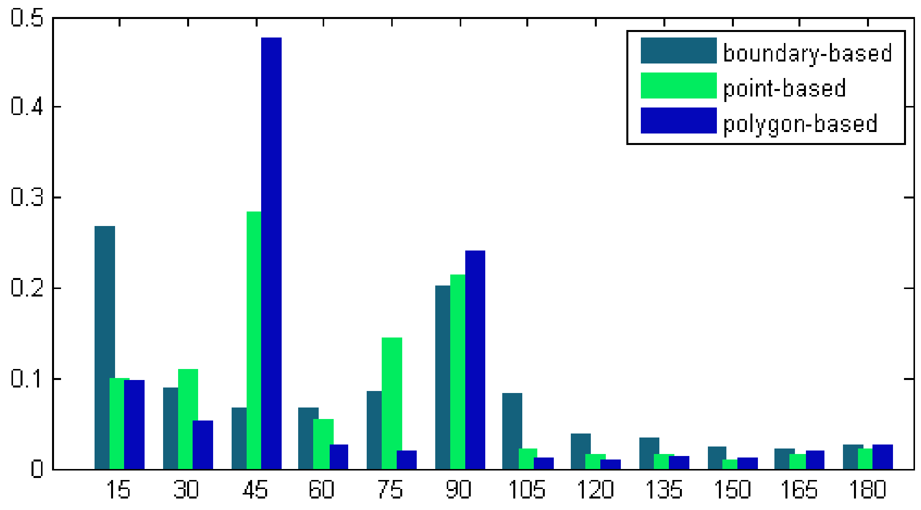


Figure 7.18: Triangulation quality measured with a normalised histogram of the angle distribution of planar meshes.

Dataset	1	2	3	4
GPT	1,020,241	1,350,004	2,293,599	4,665,067
QTB	90,087	288,833	182,648	433,953
Point-based	85,653	161,270	79,988	143,396
Polygon-based	76,367	130,642	66,413	118,275

Table 7.14: Planar mesh simplification with our triangulation algorithms measured with triangle counts, in comparison to GPT and the QTB algorithm.

gain comes from our triangulation methods, which add no extra points once decimation is completed, unlike the QTB algorithm. In Table 7.14, the mesh simplification statistics with triangle counts are also given. We take the triangle count of GPT for non-decimated planar segments as the baseline. In accordance with the point count reduction, both of our algorithms require no more than 10% of the amount of triangles of a non-decimated triangulation, and both perform better than the QTB algorithm. In terms of texture generation performance, Figures 6.6 and 7.22iv show qualitative results. The output textures incorporate almost all visual information contained in the original dense point cloud, enabling a photo realistic and aesthetically-pleasing textured 3D model.

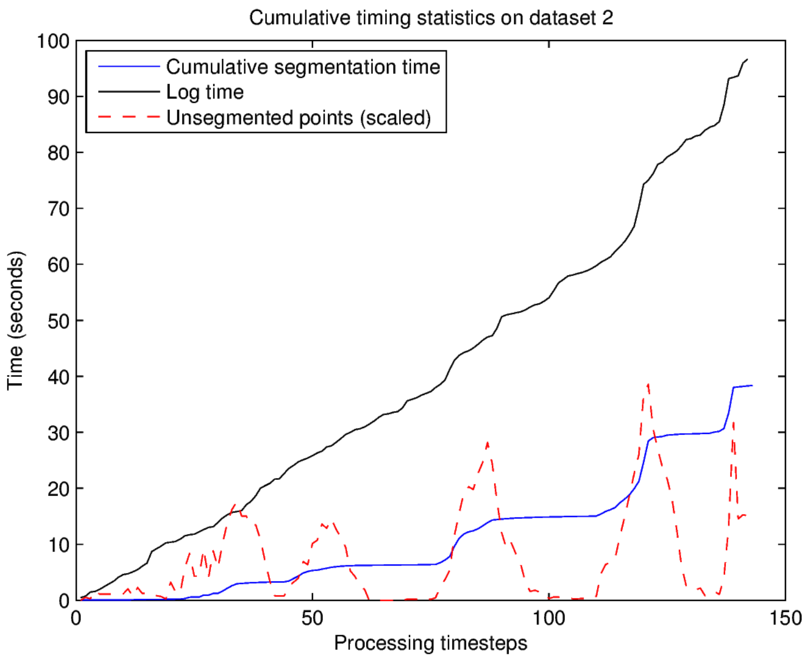


Figure 7.19: Graph comparing cumulative log file time (real world time) versus the time the incremental segmentation method spends processing data on dataset 2. Also shown is the size of the unsegmented point pool over time (linearly scaled).

7.2.3 Computational Performance

We evaluate the computational efficiency of the described algorithms. We firstly evaluate the incremental method for planar segmentation followed by the parallel system used for large-scale batch data processing.

7.2.3.1 Incremental Performance

In contrast to the results presented in Section 7.2.3.2 which include the full pipeline from batch planar segmentation to triangulation, we only evaluate the performance of the incremental planar segmentation algorithm listed in Section 6.4 here. We analyse the online performance of the algorithm as a component of the dense mapping system described in Chapter 5. In this setting, small point cloud slices of the environment being mapping are provided to the incremental segmentation method gradually over time as the camera moves through the area. Figure 7.19 shows a plot of the cumulative

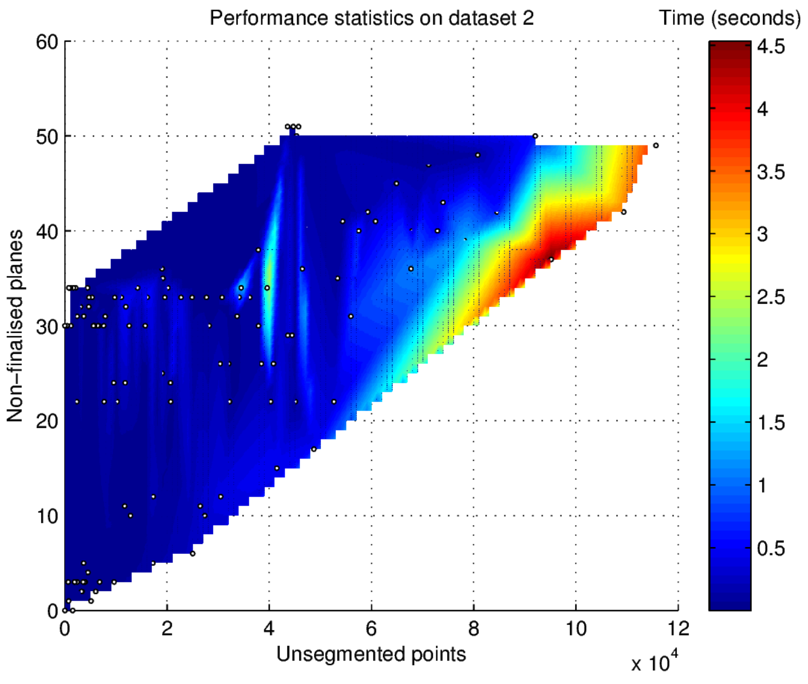


Figure 7.20: Heatmap rendering showing the relationship between the number of unsegmented points, the number of non-finalised planes and the execution time of the incremental segmentation method. The white dots represent the samples used to generate the map (using linear interpolation).

log file time (real world time) versus the time spent incrementally growing planes with the process described in Section 6.4. It can be seen that throughout the mapping process the incremental segmentation method is processing the data it is provided with faster than it is being produced, meaning that real-time online operation is being achieved. Also shown is a linearly scaled line representing the size of the unsegmented point pool over time. The relationship between this line and the segmentation time is immediately evident in how the cumulative processing time of the segmentation method increases when the number of unsegmented points increases. Similarly, the total processing time ceases to increase as the number of unsegmented points tends to zero.

In Figure 7.20 the relationship between the number of non-finalised planes, the number of unsegmented points and the execution time of the segmentation method is visualised. It can be seen that as the number of unsegmented points becomes large the execution time increases quite a lot. However for a fixed number of unsegmented points, in particular above 6×10^4, an increased number of non-finalised planes improves performance. This suggests that growing and merging existing planes is computationally cheaper than adding an entirely new plane. The peak around 4×10^4 unsegmented points came from an early on influx of many new points to the map. Along with the observation of apparent plateauing of cumulative processing timing in Figure 7.19 we can conclude that the number of unsegmented points introduced to the segmentation method at any one time influences computational performance the most. If too many are introduced, for example if the density of the input point cloud is too high or the mapping system is producing data too quickly, real-time performance may be hindered. However this can easily by remedied by downsampling the data provided to the segmentation algorithm.

7.2.3.2 Batch Performance

With batch processing the baseline for performance comparison is standard serial processing with the GPT and QTB algorithms. Table 7.15 shows the execution times. The point-based and polygon-based triangulations are approximately of the same speed, both 2 to 3 times faster than the GPT and QTB algorithms. The results also show that the texture generation algorithm is fast in execution, processing multi-million point datasets in less than 2 seconds. Examining the bottom half of Table 7.15, it is clear that the parallel system architecture has a profound effect on the overall performance. The execution time decreases with an increasing number of triangulation threads. An effect of diminishing returns becomes apparent as the number of triangulation threads increases, due to the overhead associated with the parallel implementation. However, as the per-thread workload increases, such as the inclusion of texture generation, the overhead of parallelisation becomes amortised.

Both point-based and polygon-based triangulation yield accurate and computa-

Dataset	1	2	3	4
Number of planar segments	101	116	66	117
Serial GPT (s)	18.6	24.3	44.2	91.1
Serial QTB (s)	16.7	18.7	38.3	73.1
Serial point-based (s)	6.9	9.8	17.7	40.2
Serial polygon-based (s)	6.9	9.5	17.8	40.0
Serial polygon-based (texture) (s)	8.3	10.0	20.3	41.4
1:1 Polygon-based (s)	6.4	8.1	15.1	33.8
1:1 Polygon-based (texture) (s)	7.6	8.5	17.4	35.2
1:3 Polygon-based (s)	3.6	4.2	8.3	19.2
1:3 Polygon-based (texture) (s)	4.4	4.1	9.2	19.6
1:5 Polygon-based (s)	3.7	3.5	7.9	16.1
1:5 Polygon-based (texture) (s)	4.7	3.5	8.7	16.2

Table 7.15: Efficiency of triangulation and the parallel architecture, measured in seconds. The 1:x ratio denotes 1 segmentation thread with x triangulation threads.

tionally efficient planar segment triangulations with significant point and triangle count reductions, both exceeding the performance of the QTB algorithm. The point-based approach is of low complexity and maintains good triangular mesh properties that are desirable for lighting and computer graphics operations. The polygon-based approach yields higher point and triangle count reductions with a more regularised mesh pattern, capturing information about the scene in the form of principal geometric features, such as the principal orientation of a planar segment. While the polygon-based method produces less triangles, it does generate T-joints in the mesh. Such features are detrimental when employing Gouraud shading and other lighting techniques to render a mesh with colored vertices. The polygon-based and point-based methods offer a trade-off depending on the desired number of triangles or the intended use of the final triangulation. With robot navigation in mind, the low polygon-count models achieved with our system are suitable for use in a primitives-based localisation system, such as the KMCL system of Fallon *et al.* [29].

As can be seen in the left images of Figure 7.21, the gaps between planar and non-planar triangulations are apparent. The gap can be closed by including the boundary vertices of the segmented planes in the non-planar segment GPT triangulation, as

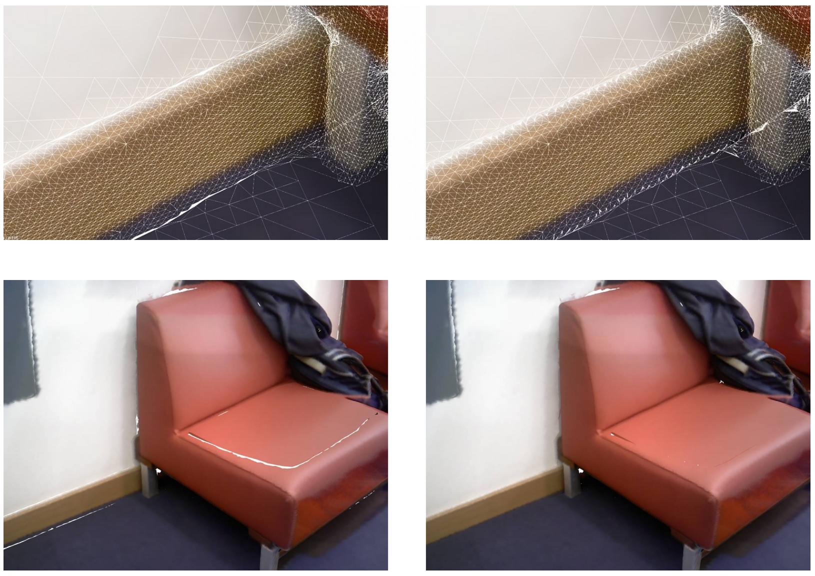


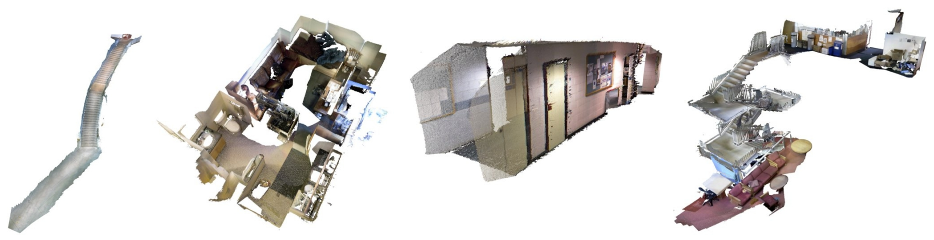
Figure 7.21: Joining of GPT mesh with planar segment triangulations. Left shows unjoined segments and right shows segments joined with interpolated boundary vertices.

shown in Figure 7.21. The number of boundary vertices can be increased with a smaller alpha value when computing the concave hull of each segment or by linearly interpolating between boundary vertices. Extra vertices can also be extracted from the vertex degree grid used in polygon-based triangulation. In our system we chose to leave these gaps open, as this separation gives an easier visual understanding of any map, implicitly providing a separation between structural features (like walls, table tops) and "object" features, useful in automatic scene understanding, manipulation and surface classification.

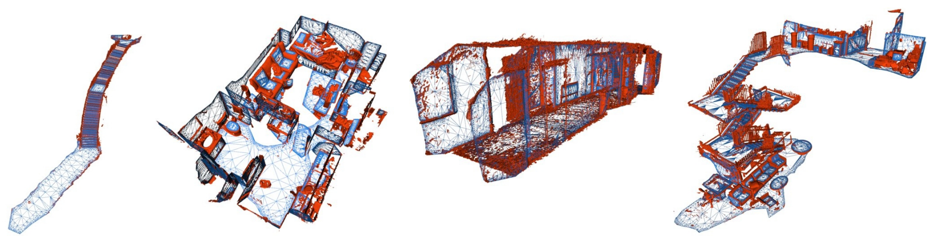
7.3 Multimedia

The interpretation of the experimental results presented in this thesis is greatly enhanced by a set of videos which more clearly highlight the scale and quality of the presented systems. These videos and their respective URLs are described as follows:

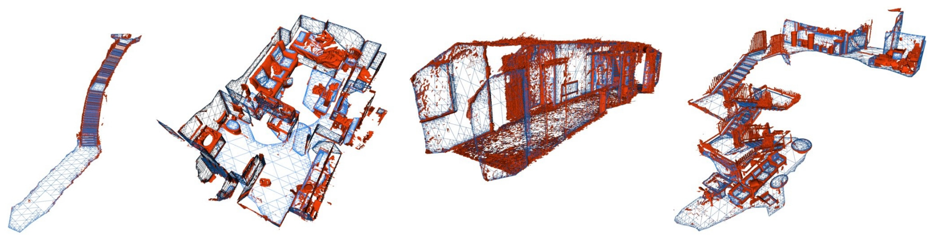
Extended Scale Volumetric Fusion (`http://youtu.be/ggvGX4fwT5g`)



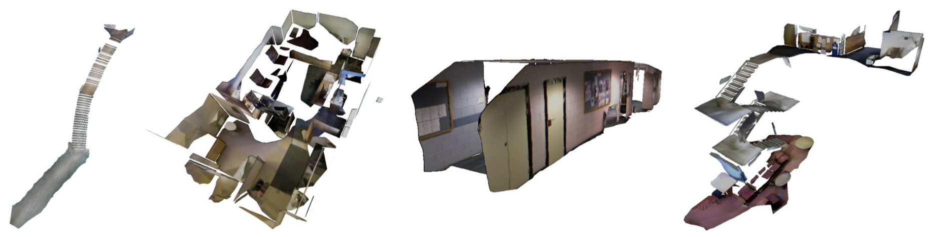
(i) Four dense colored point cloud datasets used for evaluation, numbered 1, 2, 3 and 4 from left to right.



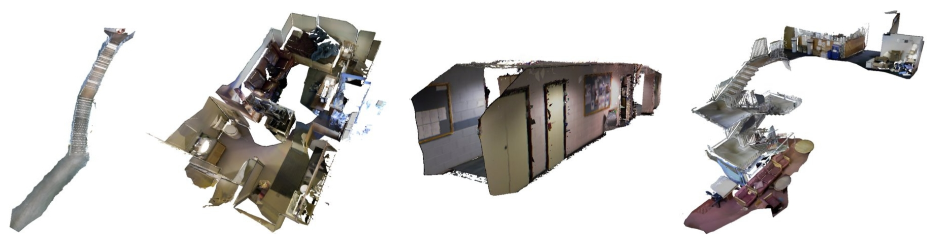
(ii) Point-based triangulation, with planar and non-planar meshes highlighted in blue and orange, respectively.



(iii) Polygon-based triangulation with planar and non-planar meshes highlighted in blue and orange, respectively.



(iv) Textured simplified planar segments from each dataset.



(v) Complete 3D model with our proposed system.

Figure 7.22: Four evaluated datasets (numbered from 1 to 4 from left to right) with various triangulation and texturing results.

This video demonstrates the volume shifting capabilities of the system described in Chapter 3. The basic functionality of the system is shown followed by a number of sample datasets.

Robust RGB-D Visual Odometry (`http://youtu.be/MEugh12dcYA`)

This video demonstrates the color fusion technique described in Chapter 3 and the method for robust real-time RGB-D visual odometry described in Chapter 4. Comparisons are drawn between the different techniques.

Deformation-based Loop Closure (`http://youtu.be/MNw-GeHHSuA`)

This video shows our method for dense large scale loop closure described in Chapter 5. Interpolated map deformation is shown followed by a number of example datasets.

Real-time Multiple Loops (`http://youtu.be/D3yYjaLmiqU`)

This video demonstrates the real-time multiple loop closing capabilities of our SLAM system described in Chapter 5. This video was recorded during the real-time playback of a large in and outdoor dataset.

Planar Simplification (`http://youtu.be/uF-I-xF3Rk0`)

This video shows our techniques for real-time incremental planar simplification and batch planar triangulation described in Chapter 6. Real-time incremental planar segmentation is shown followed by model comparisons of various datasets.

Closed-Loop Robotics (`http://youtu.be/XqDUniEY954`)

This video shows our fully autonomous closed-loop robotics experiment which we describe in Chapter 8. The handheld video stream was recorded in a single take, highlighting the strong real-time aspect of the system.

CHAPTER 8

Applications

In this chapter we present two case studies where the techniques described in this thesis have been applied. The first of these is work on point cloud segmentation by Finman *et al.* [30, 31] and the second a real-time real-world experiment involving a fully closed-loop autonomous robotic platform [128].

8.1 Segmentation

Finman *et al.* have published a number of papers in recent years directly using the output of the SLAM system we have presented in this thesis. The work they presented greatly benefits from the high quality reconstruction obtained with our system, in contrast to the low quality point clouds which are often captured in raw RGB-D frames. The following two subsections each give an overview of a recent publication by Finman *et al.* and how the system we developed has been useful in their own research. This is not a contribution of this thesis but included here as an example of the utility and efficacy of the techniques described in this thesis.

8.1.1 Object Segmentation From Change Detection

In this paper, Finman *et al.* present a system for automatically learning segmentations of objects given changes in dense RGB-D maps over the lifetime of a robot [30]. Using the SLAM system described in this thesis to capture multiple dense maps, they detect changes between mapped regions from multiple traverses by performing a 3D difference of the scenes. Their method takes advantage of the free space seen in each map to account for variability in how the maps were created. The resulting changes from the 3D difference are considered the discovered objects, which are then used to train multiple segmentation algorithms in the original map. The final objects can then be matched in other maps given their corresponding features and learned segmentation method. If the same object is discovered multiple times in different contexts, the features and segmentation method are refined, incorporating all instances to better learn objects over time. This work benefits greatly from the uniformly smooth reconstructions produced by our system. A figure from the original publication [30] is shown in Figure 8.1, where overlapping maps are aligned to detect any changes.

8.1.2 Efficient Incremental Map Segmentation

In this paper, Finman *et al.* present a method for incrementally segmenting large RGB-D maps as they are being created [31]. They propose that segmentation of large scale dense maps is a first step for higher-level tasks such as object detection. Current popular methods of segmentation scale linearly with the size of the map and generally include all points. This method takes a previously segmented map and segments new data added to that map incrementally online. Segments in the existing map are re-segmented with the new data based on an iterative voting method. Their segmentation method works in maps with loops to combine partial segmentations

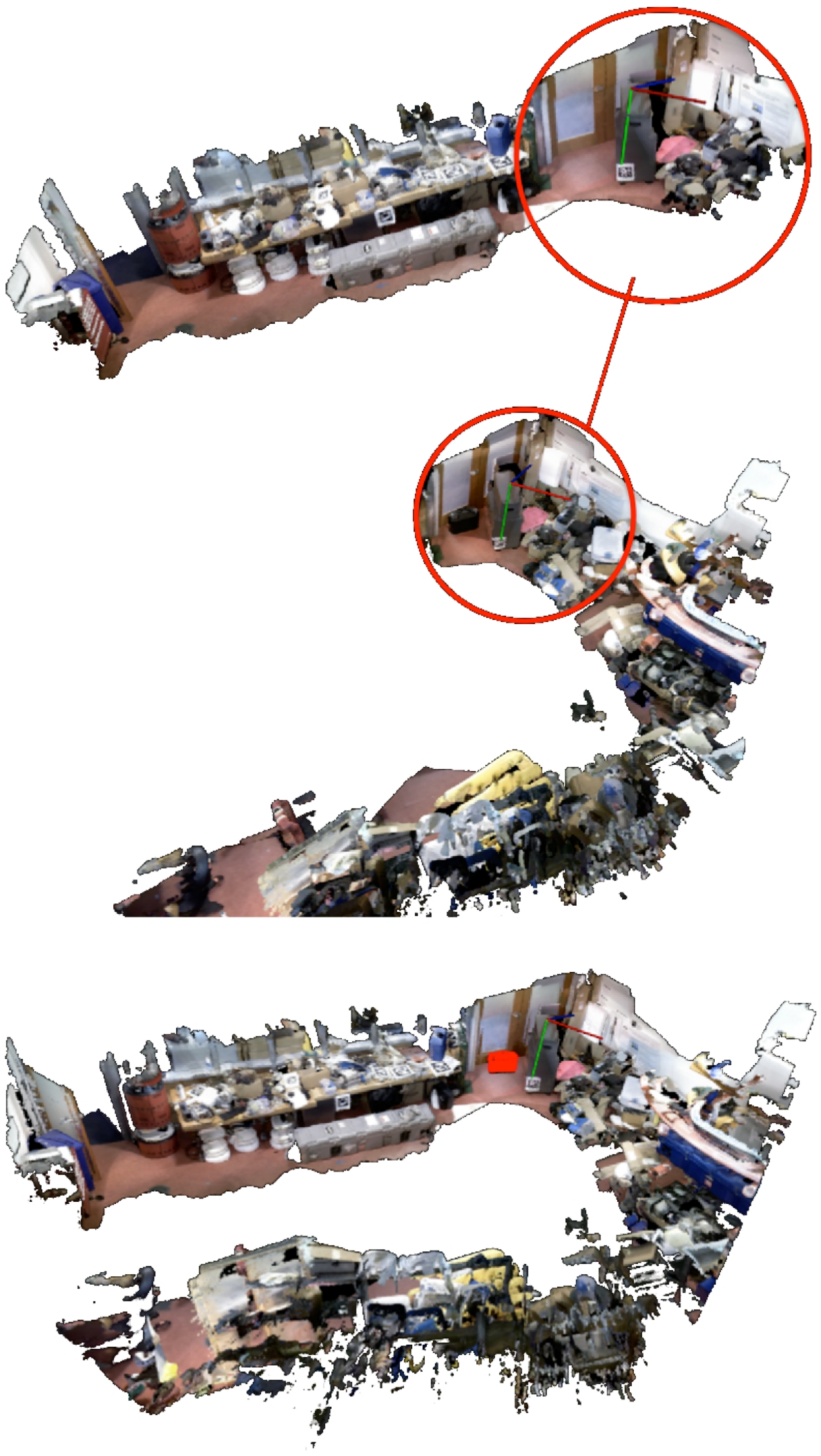


Figure 8.1: Top and center: First two maps for alignment. Note the overlapping regions within the red circles. Bottom: Both aligned maps drawn together with the filtered difference, a suitcase, highlight in red.

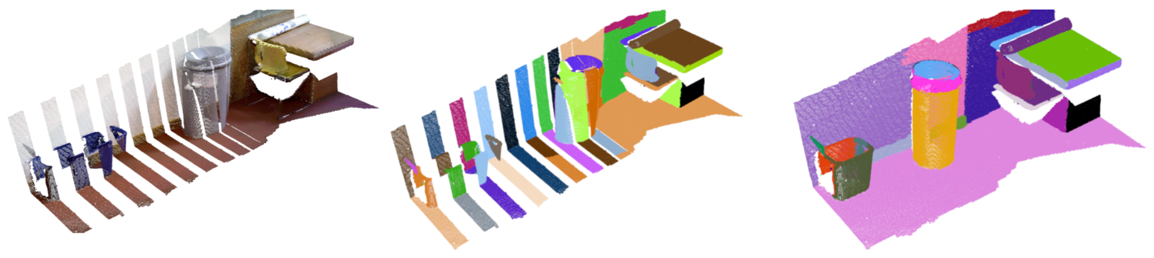


Figure 8.2: Left: RGB-D map being built with new data (left) being added to the full map (right). Center: The new data segmented individually with each segment randomly colored. Right: The newly segmented map with the new data combined incrementally. Note: the left and center pictures have the new data spaced apart for viewing purposes only.

from each traversal into a complete segmentation model. They verify their algorithm on multiple real-world datasets spanning many metres and millions of points in real-time, while also comparing their technique with a popular batch segmentation method for accuracy and timing complexity. This work directly exploits the incremental architecture of our SLAM system in how it incrementally segments individual cloud slices (as described in Chapter 3) which are then combined into the overall segmentation. A figure from the original publication [31] is shown in Figure 8.2, where the cloud slice delineation of an incrementally produced map is shown, along with the individual and joint segmentations.

8.2 Closed-Loop Autonomous Robotic System

In this second case study we present an experiment that marries the work of this thesis with related recent work on robotic perception to form a system capable of completing a simple real-world task in an entirely autonomous fashion. This experiment is perhaps the most practical contribution of this thesis focussing on a complete working autonomous robotic system which has a tangible effect in the real world.

This work is related to a number of other recent efforts that seek to develop and exploit an object-based and/or semantic understanding of a mobile robot's environment. Aydemir *et al.* [4] investigate techniques for active visual search for objects in a robot's environment, exploiting spatial relationships between objects to develop efficient search strategies. Their paper suggests the use of "dense 3D point cloud representation[s] of scenes to guide the search", which is something that is enabled in our work via the framework outlined in Chapters 3 through 5. Other related recent work includes the work of Salas-Moreno *et al.* [99], which develops SLAM++ (an object-oriented approach to SLAM), and Herbst *et al.* [46], which performs automatic discovery of objects via multiple views of a scene. Also related is a large body of recent work by Saxena *et al.* [100] which develops techniques for a PR2 robot to detect, classify and grasp objects in a variety of different contexts.

The task we set out to complete is autonomously determining the location of a preselected object in the physical world. This experiment requires a robotic framework with a number of capabilities including autonomous exploration, dense real-time localisation and mapping, object detection, path planning and motion control. Figure 8.4 shows the two main steps involved in the process, with reference to a detailed explanation of each core component provided in Section 8.2.1.

The physical embodiment of this experiment requires a mobile robotic platform, a laptop (to interface onboard the robot) and a workstation computer. The robot in our experiment is the Clearpath Robotics Turtlebot 2 platform, shown in Figure 8.3. The laptop onboard the turtlebot is equipped with an Intel Core i7-3630QM CPU, 24GB of RAM and an nVidia GeForce 675M GPU with 2GB of memory. The workstation computer is the same as described in Section 7.1.3.

8.2.1 Components

This section describes in detail the steps taken by each of the components labelled alphabetically in Figure 8.4, including the specific instances in the actual experiment carried out.

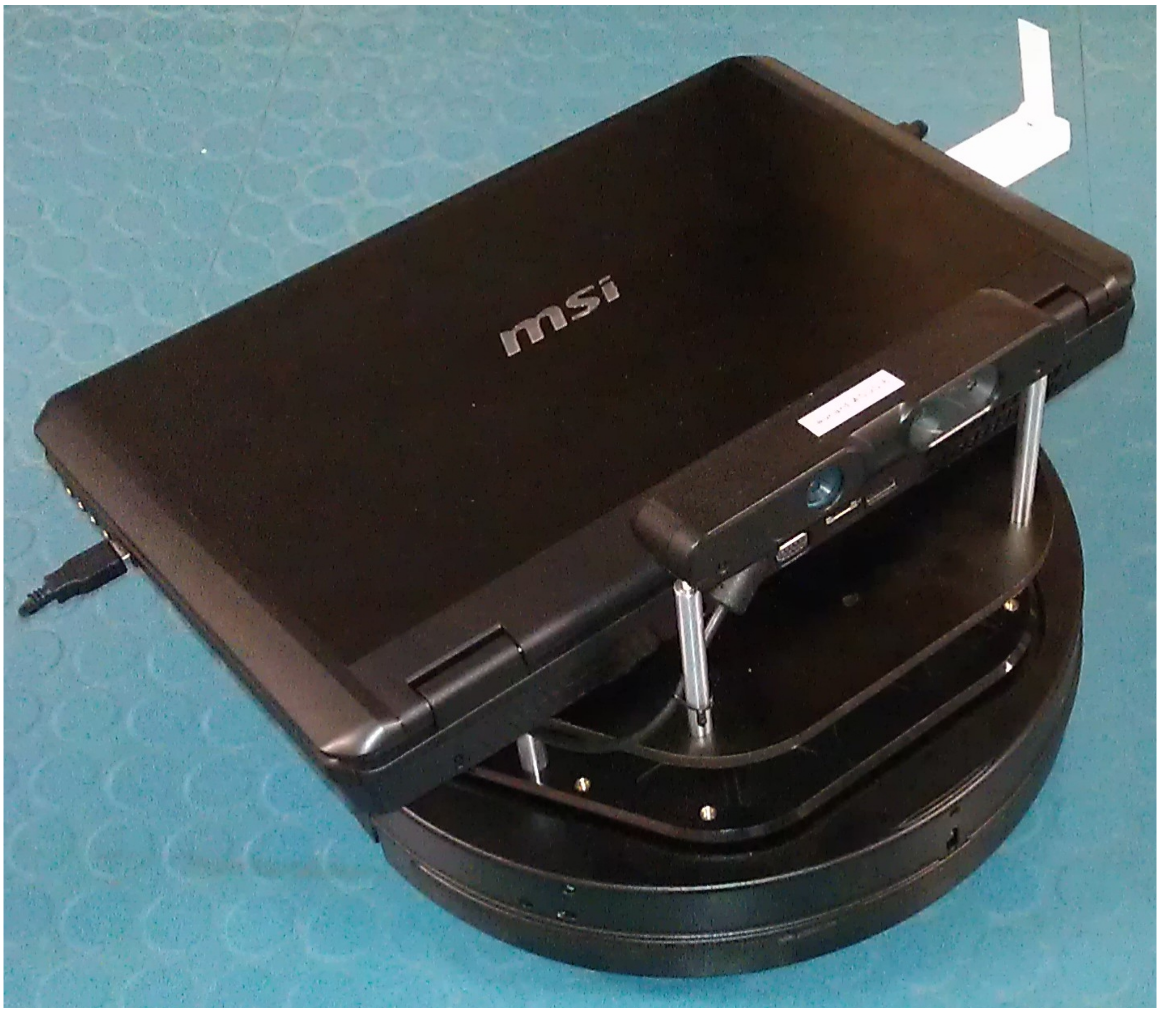
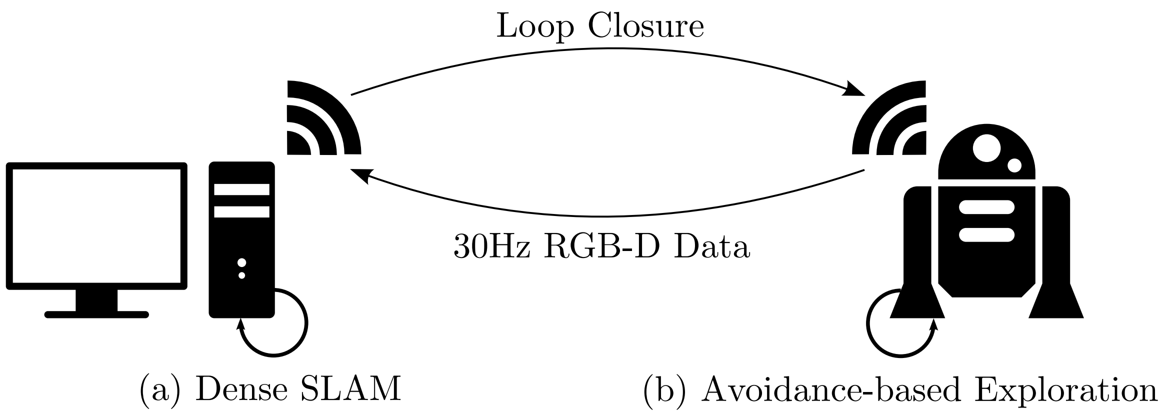
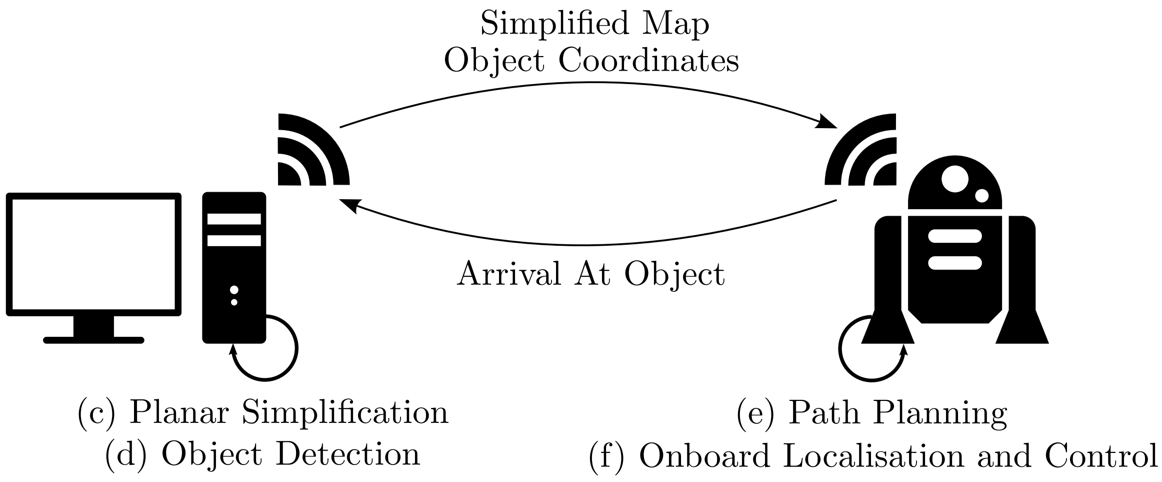


Figure 8.3: Photograph of the Turtlebot 2 platform used in our experiment. It is a 2-wheeled platform with an RGB-D sensor, controlled by an onboard computer (a standard laptop in this scenario).



(i)



(ii)

Figure 8.4: This figure shows the two main steps involved in executing the required task. The top subfigure (i) illustrates the first step in the process. The robot is set to explore the environment while streaming the RGB-D data captured with the onboard sensor back to the workstation over 802.11n WiFi. The workstation uses this data to reconstruction a globally consistent map of the explored environment in real-time, signaling the robot to cease exploration when a loop has occurred. In the second step, shown in subfigure (ii), the workstation simplifies the dense map such that it is suitable for real-time localisation and detects the position of the desired object in the dense map. This information is sent to the robot which then navigates to the detected object using onboard real-time path planning and control against the simplified map. Details for each of the individual components are provided in Section 8.2.1.

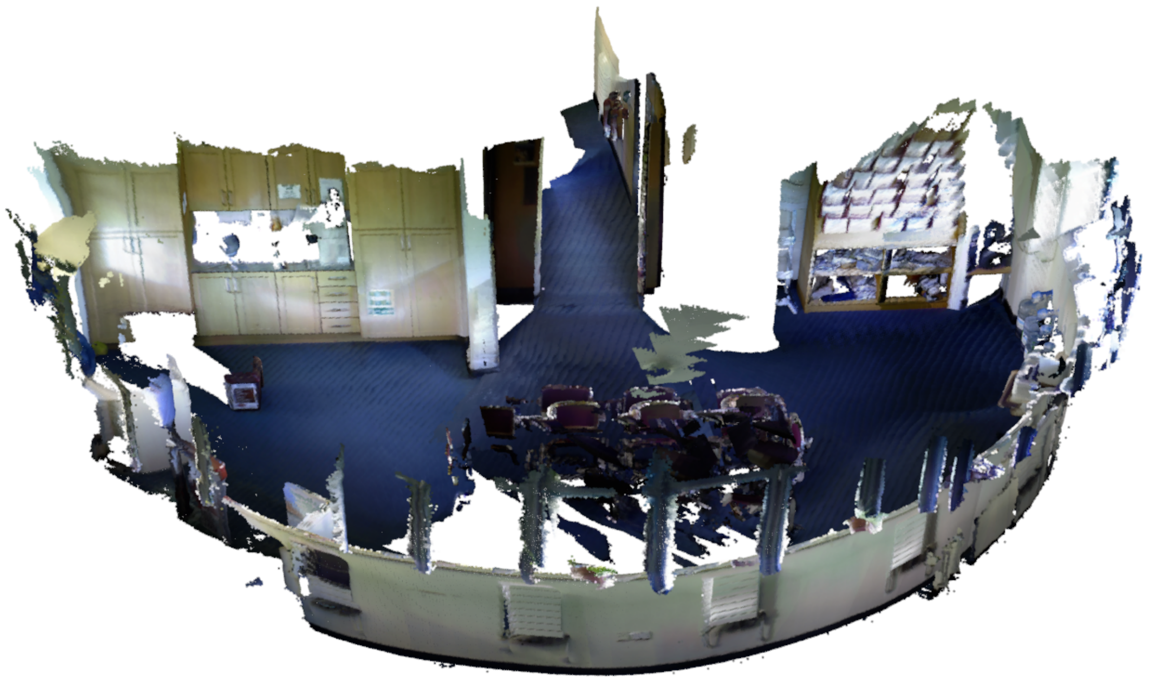


Figure 8.5: Dense reconstruction of the area autonomously explored by the robot.

(a) Dense SLAM - In order to reconstruct the environment explored by the robot in real-time, the SLAM approach outlined in Chapters 3 through 5 is employed on the workstation computer. While the robot explores the real world, the RGB-D data captured by the onboard sensor is streamed in real-time over 802.11n wifi to the workstation machine. Once a loop closure is detected in the explored area, the robot is signalled to stop exploring. Figure 8.5 shows the resulting reconstruction of the explored area.

(b) Avoidance-based Exploration - A simple approach to exploration is adopted in this experiment. Planning is carried out in real-time on the robot using the immediate RGB-D data captured with the onboard sensor. The approach attempts to maintain a constant distance to any surfaces to the left of the robot not lying on the ground plane. This is accomplished by analysing a single line of the depth map at the height of the sensor off of the ground plane. The bearing and forward velocity of the robot is recalculated every frame (i.e. at

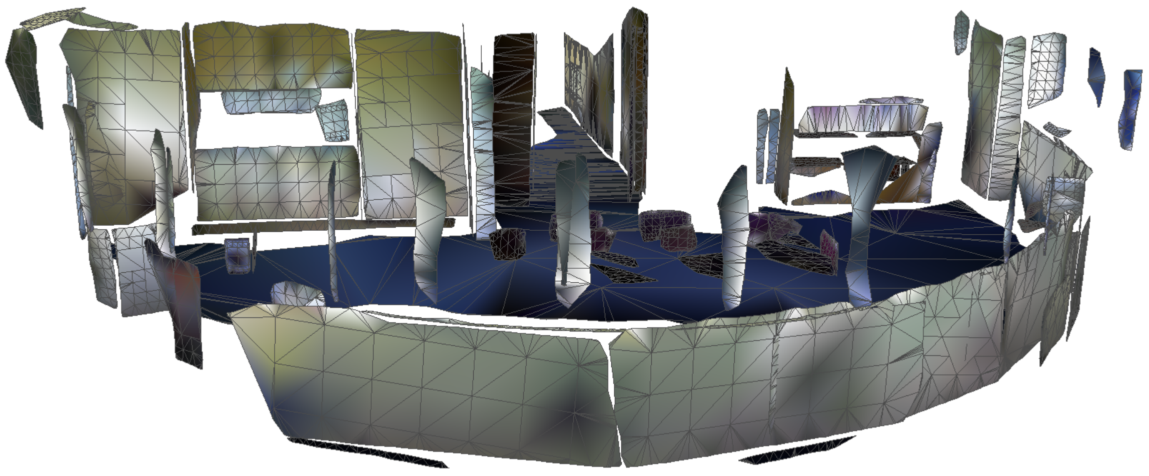


Figure 8.6: Planar simplification of the area autonomously explored by the robot with triangulation overlaid.

30Hz) based on the relative distance to any surfaces detected in the depth map which the robot may collide with. While simple in practice, this method for exploration works well in most environments and is suitable for use as a simple proof-of-concept example. A more sophisticated coverage-based approach could be adopted to improve performance in more complicated environments.

(c) Planar Simplification - As we discuss in Step (f), a simplified planar map of the environment is required for the approach we employ for real-time localisation. For this we apply the planar simplification technique described in Chapter 6 to the dense reconstruction obtained from Step (a). This model is quick to compute and also provides a format in which it is trivial to perform floor plane detection and alignment for 2D path planning (by taking the plane with the largest area with the correct relative transformation to the capturing sensor). Figure 8.6 shows the simplified planar model computed from the dense map. This compact scene model is transferred to the robot wirelessly for autonomous navigation and localisation in the subsequent steps.

(d) Object Detection - In order to query the robot to navigate to a point of in-

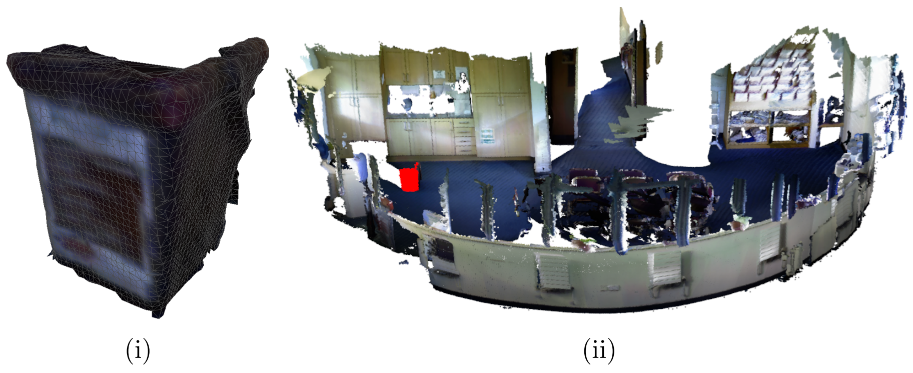


Figure 8.7: From left to right; (i) Learned model of a trash can from a previous scan, with triangulation shown; (ii) Detected position of the object within the mapped environment highlighted in red.

terest, we choose to learn a number of object models as a precursor step to the experiment. From this point, provided steps (a) through (c) have succeeded, we can query the system with a known object model. If the system can locate the object within the dense map provided by Step (a), a path can be planned from the last known location of the robot through the environment using the simplified model provided by Step (c). To learn the segmentation parameters for different object models, we use the approach presented by Finman *et al.* [30]. Figure 8.7 shows a sample object model learned by the system, and the highlighted detection of the object within the dense map provided by Step (a).

(e) Path Planning - The simplified planar model provided by Step (c) includes detection of and alignment with the floor plane. By projecting the remainder of the model onto the floor plane a simple occupancy grid map of the environment can be recovered. From here, the configuration space of the robot can be computed and 2D path planning within the occupancy grid can easily be carried out. In our implementation we seed the path planner with the last known location of the robot and the location of the detected object and run the A* search algorithm to find a path through the occupancy grid. From here we simplify

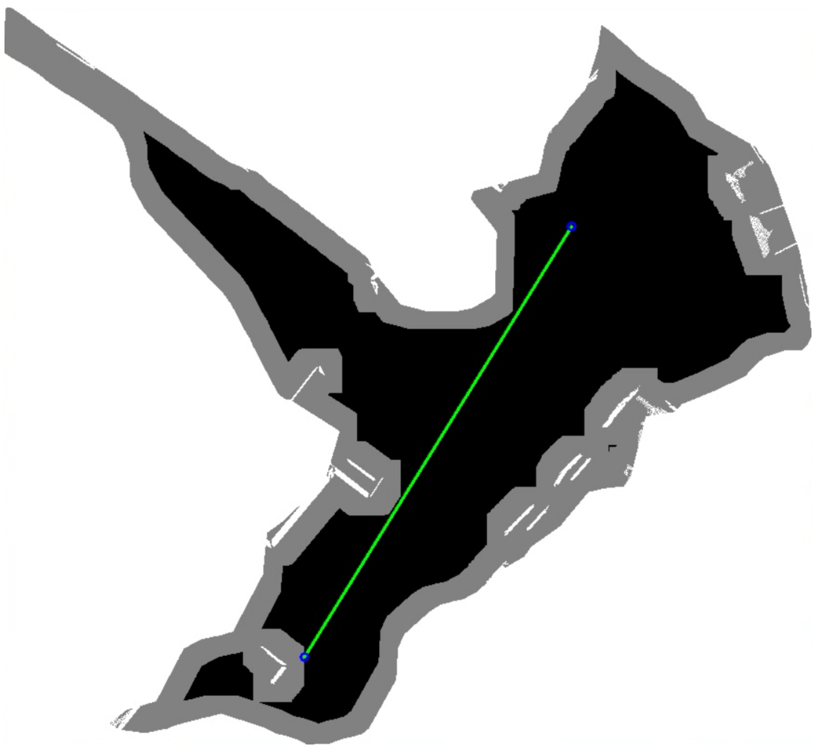


Figure 8.8: This figure shows the path planned from the last known location of the robot (top right) to the location of the detected object in the environment. White space is not considered, while grey space is unoccupied but outside of the configuration space of the robot. The path is shown in green while the control waypoints are shown in blue.

the A* path using a greedy ray-tracing method to compute a set of sparse waypoints within the environment. Figure 8.8 shows the path planned through the environment in our experiment.

(f) Onboard Localisation and Control - Given a simplified model of the environment to localise against and a target location to reach in the map, the robot must autonomously navigate to each point in the planned path in a closed-loop fashion. For this we use the Kinect Monte Carlo Localisation (KMCL) system of Fallon *et al.* [29]. The KMCL system is a particle filter-based localisation system that uses predicted RGB-D frames from within a planar model of an environment as the basis for a likelihood function in comparison to actual RGB-D

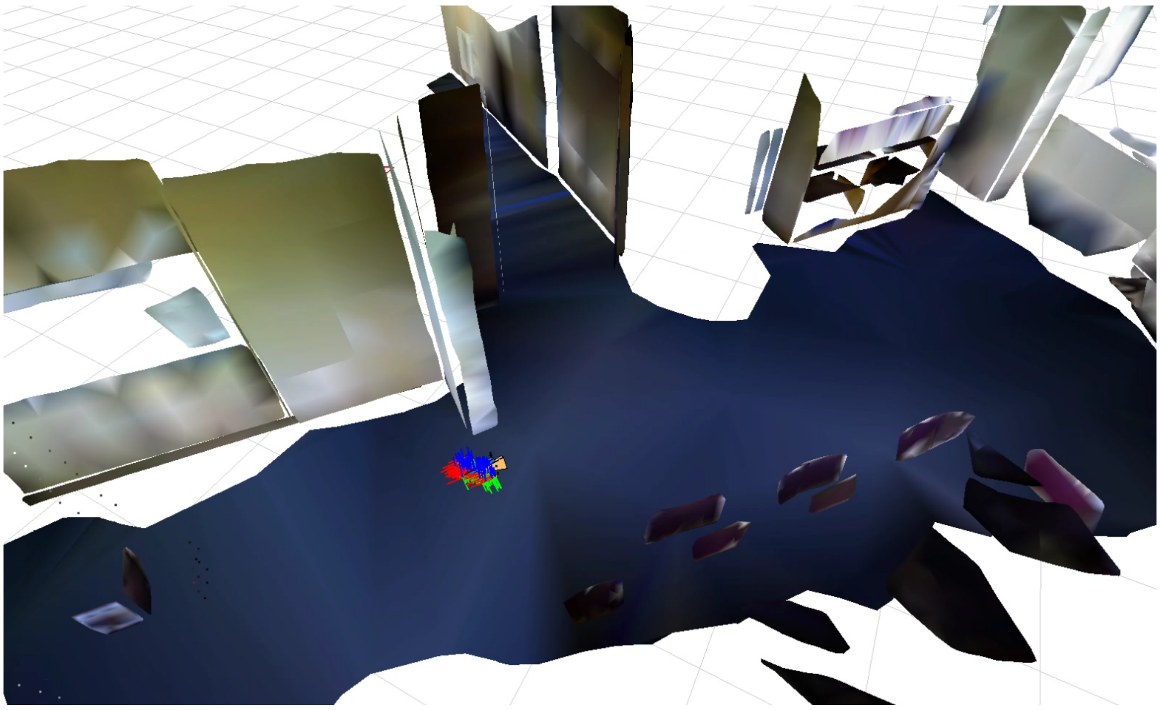


Figure 8.9: This figure shows the KMCL system in action as the robot navigates to a target waypoint in the environment in real-time. Shown is the simplified planar model, as well as a number of particle filter estimates of the robot's current position.

sensor readings. Figure 8.9 shows a screenshot of the KMCL system in action during our experiment. The estimated position of the robot is updated at camera frame rate while a simple proportional controller firstly aligns the orientation of the robot with the location of the next waypoint, before adjusting the forward velocity of the robot to reach the waypoint. Once the robot has reached the position of the desired object in the environment, motion is ceased.

8.2.2 Result

The purpose of this case study was to merge a number of recent advances in RGB-D based perception research to accomplish a simple real-world task using low-cost commodity components. In this sense, the experiment was a success demonstrating clear fitness for purpose. This experiment also highlights the importance of each component of the system and how each is necessary in completing the task. Namely, real-time dense mapping is required to inform the robot that it no longer needs to explore and can immediately access a globally consistent model which can be used for subsequent object detection and future motion planning. The density of the initial map is necessary for performing the detection of a variety of objects commonly found in real-world environments. In contrast to this, quick access to a simplified planar representation of the environment is needed to perform real-time onboard localisation in the mapped area for path planning and motion control.

One observation made during the execution of this experiment was both the compounding of failure rates, and the potential for a cascading effect due to a non-terminal error in the upstream processing resulting in a failure in the downstream processing. As a consequence, in any given mission a number of consecutive runs could be required for the robot to complete the task due to the individual rates of failure of each component of the system compounding together. Examples of the individual failures included frame-drops in the wireless streaming of the raw RGB-D image sequence to the SLAM server, failure of the planar segmentation algorithm to robustly estimate the ground plane (e.g. due to the errors in the reconstruction process), and failure of the object segmentation and hence recognition due to spurious geometry introduced from noise in the reconstruction process. Of the above, we found that the principal source of error in the system was the unreliable nature of the wireless streaming. This would result in dropped frames which in turn would result in a degradation in camera tracking and 3D scene estimation.

This is one of the key observations of the experiment which highlights the fact that the development of techniques which although alone are quite robust and reliable is not necessarily enough when it comes to combining these techniques together into a complete framework when attempting to accomplish a larger, higher level task.

CHAPTER 9

Conclusions

9.1 Thesis Contributions

This thesis has presented a real-time large scale dense visual SLAM system based on RGB-D data capable of producing high quality globally consistent reconstructions which are useful for a number of autonomous operations. The principal contribution of this thesis is the development of a number of methods specifically aimed towards the large scale aspect of dense reconstruction, extending the state of the art beyond small scale environments [86]. In particular this is the first example of an RGB-D based visual SLAM system employing volumetric fusion which (i) can function over extended scale environments, (ii) uses both geometric-based frame-to-model tracking and photometric-based frame-to-frame tracking for camera pose estimation, (iii) is capable of closing large loops in order to update the dense surface reconstruction online in real-time for global consistency, and (iv) provides an alternative lightweight but higher level scene representation more useful for planning and autonomous navigation.

We have provided an extensive evaluation, both quantitatively and qualitatively on common benchmarks and our own datasets demonstrating the system's ability to produce large scale dense globally consistent maps in real-time. These datasets include more traditional forward-facing robot SLAM style datasets and more complicated handheld human-operated camera trajectories. We have also presented an example of a practical robotics application, involving the marriage of the contributions of this thesis with other recent robotics perception work to demonstrate the immediate usefulness of these techniques and indeed of dense methods as a whole.

Taking real-time dense methods into large scale applications is arguably most challenging in terms of computational efficiency, due to the enormous amounts of data which must be processed in a timely manner. Existing techniques for dense methods were typically bounded in scale of execution which required the development of new techniques which "unlocked" the applicability of these methods to larger scales. Within this thesis this includes the development of the efficient cyclical buffer approach to extended scale volumetric fusion, and the implementation of photometric camera pose estimation as a highly parallel GPU-based operation.

Related to this is the novel application of space deformation to extended scale RGB-D based maps. Although used for years in the field of computer graphics, such complicated non-rigid geometric operations were not found to be necessary in the context of RGB-D based visual SLAM until the advent of higher quality fusion-based reconstructions. 3D mapping and reconstruction can be viewed as the inverse of rendering in computer graphics, and hence a problem can be formulated as the application of techniques originally intended for computer graphics to 3D mapping and reconstruction. One of the key challenges in this thesis was formulating these techniques in a framework more suited to real-time incremental operation as is commonly found in the visual SLAM world.

Given this new source of high quality fused map data, in contrast to maps made up of raw RGB-D data, related techniques such as object segmentation and scene understanding can benefit greatly. In this thesis in particular, the uniformity of volumetric fusion-based data is found to be beneficial for planar segmentation and simplification, both in geometric qualities and appearance (texturing) qualities. Provisioning of less noisy data up front relaxes the need for algorithms further down any processing pipeline to be highly robust or overly complicated. This in turns helps in achieving the overarching aim of this thesis to take real-time dense methods to large scale environments and exploit them for real-world robotics applications.

9.2 Future Directions

There are a large number of future directions for the work in this thesis looking at both solving current limitations of the selected approaches and applying the system as it is at present to existing problems. Perhaps the most significant issue with the current approach to extending volumetric fusion to larger scales is the lack of support for reintegrating areas of the map which are revisited into the fusion frontend. This results in aliasing in areas that receive multiple passes. However representing the surface as a set of cloud slices maintains spatiotemporal information about the map which can be used for change detection, scene differencing or even the merging of cloud slices from multiple passes. Reintegration or re-fusing of the mesh-based map in real-time is a challenging problem due to the sheer volume of data. Some existing approaches discussed in Section 2.4 support this but lack a means for correcting for drift or global consistency online in real-time. Commonly adopted space efficient data structures need to be fully restructured upon large updates to the map, which in turn would hinder real-time performance greatly. Other discussed approaches are either offline methods, or sacrifice local surface connectivity to achieve surface refusing. Real-time large scale dense fused 3D reconstruction which supports online drift correction, provides a globally consistent representation of the map at any time and allows map re-use and re-fusing is a challenging problem which we aim to address in our future work.

Related to this is the reliance on appearance-based techniques for loop closure detection, such as DBoW and FABMAP [13, 36]. There are known issues with these techniques in terms of requiring very similar views of a previously visited scene in order to detect a loop closure. Adopting an approach which directly localises (and re-localises) the sensor against the existing large scale map continuously in not only an appearance-based sense but also a geometry-based sense is a viable avenue for future research that would improve place recognition rates and overall reconstruction quality. This tighter approach to place recognition would also make the task of capturing complete watertight models of scenes easier, something which has proven difficult in the RGB-D SLAM community without resorting to approaches which "fill-in" areas of the scene which were not observed [121].

Along the line of more intelligent place recognition is the preliminary work of Finman *et al.* on segment-based place recognition [32]. Higher level semantics and 3D scene understanding is one of the central motivations for dense methods. As the amount of data available to reason about increases, the amount of information which can be extracted from the scene also increases. Semantic object-based SLAM is being explored more and more in the field [99, 105, 33], and relies on rich high quality data to succeed. Scalability of dense methods beyond single buildings and associated techniques for performing semantic analysis of these huge amounts of data is an interesting research question which is becoming more important as these techniques become closer to an everyday usage scenario, in the form of consumer technology.

With the advent of the Google Project Tango[1] platform there is an increasing

[1]`https://www.google.com/atap/projecttango`

interest in taking visual SLAM to commodity mobile platforms [102, 66]. Many current fully dense real-time visual SLAM systems require a high powered GPU device to perform the processing of the enormous amounts of data encountered each frame, or at least a high specification CPU [110]. The application and adaptation of some of the techniques described in this thesis to mobile devices is today a very relevant direction for future work. Many other devices beyond handheld capture and simple wheeled robots could also benefit from dense reconstructions over large scales, in particular quadrotors where it is advantageous to have a rich geometrical environment representation when planning agile flight motions [2].

This ties into the concept of multi-platform collaborative mapping and multi-session visual SLAM. Given a wide range of different platforms with different sensing and processing capabilities, an interesting direction for future work involves bringing together these platforms into a unified framework capable of performing large scale dense and semi-dense reconstruction in real-time in a collaborative fashion. Here the concept of anchor nodes could be employed to connect otherwise disjoint camera pose graphs into a single global pose graph [80]. From here, the deformation-based loop closure technique described in this thesis could be directly applied to not only dense RGB-D fusion-based data but also other sources of 3D data such as stereo vision and lidar. This would enable a widely applicable mapping system which could for instance rely on RGB-D data while indoors and then stereo vision when outdoors in direct sunlight, all under a single framework.

In summary, dense methods present a platform for more advanced autonomous reasoning and robotic perception. There is a strong necessity for these methods to function in real-time, and also scale gracefully in operation beyond the lab to more general environments without sacrificing performance. We are at an age where high performance computing is ubiquitous and the sensing capabilities of low-cost commodity devices are more advanced than ever. The techniques outlined in this thesis have proven themselves to bridge the gap between the sensing platform and the advanced hardware. The result is a system which can provide rich informative dense data that is required for high level autonomous reasoning. The hope is that the work in this thesis will prove a strong basis for future research both in terms of real-time large scale dense visual SLAM and as a platform for providing dense data for secondary research down the robotic perception pipeline.

References

[1] P. Agarwal, G. Grisetti, G. Tipaldi, L. Spinello, W. Burgard, and C. Stachniss. Experimental analysis of dynamic covariance scaling for robust map optimization under bad initial estimates. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Hong Kong, China, 2014.

[2] H. Alvarez, L. Paz, J. Sturm, and D. Cremers. Collision avoidance for quadrotors with a monocular camera. In *Proc. of The 12th International Symposium on Experimental Robotics (ISER)*, 2014.

[3] C. Audras, A. I. Comport, M. Meilland, and P. Rives. Real-time dense RGB-D localisation and mapping. In *Australian Conf. on Robotics and Automation*, Monash University, Australia, December 2011.

[4] A. Aydemir, K. Sjoo, J. Folkesson, A. Pronobis, and P. Jensfelt. Search in the real world: Active visual object search based on spatial relations. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2818–2824, May 2011.

[5] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-

pivoting algorithm for surface reconstruction. *IEEE Trans. on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[6] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. In *Robotics: Science and Systems (RSS)*, June 2013.

[7] D. R. Canelhas, T. Stoyanov, and A. J. Lilienthal. SDF Tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, November 2013.

[8] J. A. Castellanos and J. D. Tardos. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[9] R. Castle, G. Klein, and D. Murray. Wide-area augmented reality using camera tracking and mapping in multiple regions. *Computer Vision and Image Understanding*, 115(6):854 – 867, 2011.

[10] J. Chen, D. Bautembach, and S. Izadi. Scalable real-time volumetric surface reconstruction. In *SIGGRAPH*, Anaheim, CA, USA, 2013. ACM.

[11] J. Chen, S. Izadi, and A. Fitzgibbon. KinÊtre: animating the world with the human body. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, pages 435–444, New York, NY, USA, 2012. ACM.

[12] A. Comport, E. Malis, and P. Rives. Accurate Quadri-focal Tracking for Robust 3D Visual Odometry. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Rome, Italy, April 2007.

[13] M. Cummins and P. Newman. Invited Applications Paper FAB-MAP: Appearance-Based Place Recognition and Mapping using a Learned Visual Vocabulary Model. In *27th Intl Conf. on Machine Learning (ICML)*, 2010.

[14] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, pages 303–312, New York, New York, USA, August 1996.

[15] F. Darema, D. George, V. Norton, and G. Pfister. A single-program-multiple-data computational model for epex/fortran. *Parallel Computing*, 7(1):11 – 24, 1988.

[16] T. A. Davis and W. W. Hager. Modifying a sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 20(3):606–627, May 1999.

[17] A. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1403–1410, 2003.

[18] A. Davison. *Mobile Robot Navigation Using Active Vision.* PhD thesis, University of Oxford, 1998.

[19] F. Dellaert. Square Root SAM: Simultaneous location and mapping via square root information smoothing. In *Robotics: Science and Systems (RSS)*, Cambridge, MA, 2005.

[20] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 2, pages 1322–1328 vol.2, May 1999.

[21] J. Deschaud and F. Goulette. A fast and accurate plane detection algorithm for large noisy point clouds using filtered normals and voxel growing. In *3DPVT*, 2010.

[22] P. Deutsch and J.-L. Gailly. Zlib compressed data format specification version 3.3. In *RFC*, United States, 1996. RFC Editor.

[23] V. Domiter and B. Zalik. Sweep-line algorithm for constrained delaunay triangulation. *Int. J. Geogr. Inf. Sci.*, 22(4):449–462, January 2008.

[24] M. Duckham, L. Kulik, M. Worboys, and A. Galton. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition (Proc. DAGM)*, 41(10):3224–3236, 2008.

[25] E. Eade and T. Drummond. Unified loop closing and recovery for real time monocular SLAM. In *British Machine Vision Conf. (BMVC)*, 2008.

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

[27] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *Intl. Conf. on Computer Vision (ICCV)*, Sydney, Australia, December 2013.

[28] C. Estrada, J. Neira, and J. Tardós. Hierarchical SLAM: Real-time accurate mapping of large environments. *IEEE Trans. Robotics*, 21(4):588–596, 2005.

[29] M. F. Fallon, H. Johannsson, and J. J. Leonard. Efficient scene simulation for robust Monte Carlo localization using an RGB-D camera. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2012.

[30] R. Finman, T. Whelan, M. Kaess, and J. Leonard. Toward lifelong object segmentation from change detection in dense RGB-D maps. In *European Conference on Mobile Robotics (ECMR)*, Barcelona, Spain, Sep 2013.

[31] R. Finman, T. Whelan, M. Kaess, and J. Leonard. Efficient incremental map segmentation in dense RGB-D maps. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Hong Kong, Jun 2014.

[32] R. Finman, T. Whelan, L. Paull, and J. Leonard. Physical words for place recognition in dense RGB-D maps. In *Workshop on Visual Place Recognition in Changing Environments, ICRA*, Hong Kong, Jun 2014.

[33] N. Fioraio, G. Cerri, and L. Di Stefano. Towards semantic kinectfusion. In A. Petrosino, editor, *Image Analysis and Processing - ICIAP 2013*, volume 8157 of *Lecture Notes in Computer Science*, pages 299–308. Springer Berlin Heidelberg, 2013.

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

[35] P. Furgale, T. Barfoot, and G. Sibley. Continuous-time batch estimation using temporal basis functions. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, St. Paul, MN, USA, May 2012.

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

[37] A. Golovinskiy and T. Funkhouser. Min-cut based segmentation of point clouds. In *IEEE Workshop on Search in 3D and Video (S3DV) at ICCV*, September 2009.

[38] A. Golovinskiy, V. G. Kim, and T. Funkhouser. Shape-based recognition of 3D point clouds in urban environments. *Intl. Conf. on Computer Vision (ICCV)*, September 2009.

[39] M. Gopi and S. Krishnan. A fast and efficient projection-based approach for surface reconstruction. In *Proc. Computer Graphics and Image Processing.*, pages 179–186, 2002.

[40] J. Guivant and E. Nebot. Compressed filter for real time implementation of simultaneous localization and map building. In *Field and Service Robots*, pages 309–314, Finland, June 2001.

[41] J. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, volume 1, pages 318–325, 1999.

[42] A. Handa, R. Newcombe, A. Angeli, and A. Davison. Real-time camera tracking: When is high frame-rate best? In *Eur. Conf. on Computer Vision (ECCV)*, volume 7578 of *Lecture Notes in Computer Science*, pages 222–235, 2012.

[43] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *Intl. J. of Robotics Research*, 2012.

[44] P. Henry, D. Fox, A. Bhowmik, and R. Mongia. Patch volumes: Multiple fusion volumes for consistent RGB-D modeling. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Berlin, Germany, June 2013.

[45] P. Henry, D. Fox, A. Bhowmik, and R. Mongia. Patch Volumes: Segmentation-based Consistent Mapping with RGB-D Cameras. In *Third Joint 3DIM/3DPVT Conference (3DV)*, 2013.

[46] E. Herbst, X. Ren, and D. Fox. RGB-D object discovery via multi-scene analysis. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4850–4856. IEEE, 2011.

[47] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[48] A. Howard and N. Roy. The robotics data set repository (radish), 2003. URL `http://radish.sourceforge.net/`.

[49] G. Hu, S. Huang, L. Zhao, A. Alempijevic, and G. Dissanayake. A robust RGB-D SLAM algorithm. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1714 –1719, October 2012.

[50] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an RGB-D camera. In *Proc. of the Intl. Symp. of Robotics Research (ISRR)*, Flagstaff, Arizona, USA, August 2011.

[51] A. Jacobson and O. Sorkine. Stretchable and twistable bones for skeletal shape deformation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH ASIA)*, 30(6):165:1–165:8, 2011.

[52] A. C. Jalba and J. B. T. M. Roerdink. Efficient surface reconstruction from noisy data using regularized membrane potentials. *IEEE Trans. on Image Processing*, 18(5):1119–1134, May 2009.

[53] H. Johannsson, M. Kaess, M. Fallon, and J. Leonard. Temporally scalable visual SLAM using a reduced pose graph. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013.

[54] S. Julier and J. Uhlmann. A counter example to the theory of simultaneous localization and map building. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 4, pages 4238–4243, 2001.

[55] M. Kaess. *Incremental Smoothing and Mapping.* PhD thesis, Georgia Institute of Technology, December 2008.

[56] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. Robotics*, 24(6):1365–1378, December 2008.

[57] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[58] K. S. Karan. Skinning characters using surface-oriented free-form deformations. In *Graphics Interface*, pages 35–42, 2000.

[59] A. Karpathy, S. Miller, and L. Fei-Fei. Object discovery in 3d scenes via shape analysis. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2013.

[60] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proc. of the 4th Eurographics Symposium on Geometry Processing.*, pages 61–70, 2006.

[61] 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 *Third Joint 3DIM/3DPVT Conference (3DV)*, June 2013.

[62] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for RGB-D cameras. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2013.

[63] C. Kerl, J. Sturm, and D. Cremers. Dense visual SLAM for RGB-D cameras. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, November 2013.

[64] K. Khoshelham and S. O. Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[65] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, Nara, Japan, November 2007.

[66] K. Kolev, P. Tanskanen, P. Speciale, and M. Pollefeys. Turning mobile phones into 3D scanners. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, Columbus, Ohio, USA, June 2014.

[67] K. Konolige and M. Agrawal. FrameSLAM: From bundle adjustment to real-time visual mapping. *IEEE Trans. Robotics*, 24(5):1066–1077, October 2008.

[68] K. Konolige and J. Bowman. Towards lifelong visual maps. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1156–1163, October 2009.

[69] K. Konolige, J. Bowman, J. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua. View-based maps. In *Robotics: Science and Systems (RSS)*, Seattle, USA, June 2009.

[70] K. Konolige, J. Bowman, J. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua. View-based maps. *Intl. J. of Robotics Research*, 29(8):941–957, July 2010.

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

[72] Y. Latif, C. Cadena, and J. Neira. Robust loop closing over time. In *Robotics: Science and Systems (RSS)*, Sydney, Australia, 2012.

[73] D. Lee, H. Kim, and H. Myung. GPU-based real-time RGB-D 3D SLAM. In *Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 46 –48, November 2012.

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

[75] L. S. M. Mazuran, G. D. Tipaldi and W. Burgard. Nonlinear graph sparsification for SLAM. In *Robotics: Science and Systems (RSS)*, Berkeley, USA, 2014.

[76] L. Ma, T. Whelan, E. Bondarev, P. H. N. de With, and J. McDonald. Planar simplification and texturing of dense point cloud maps. In *European Conference on Mobile Robotics (ECMR)*, Barcelona, Spain, September 2013.

[77] L. Ma, R. Favier, L. Do, E. Bondarev, and P. H. N. de With. Plane segmentation and decimation of point clouds for 3d environment reconstruction. In *Proc. the 10th Annual IEEE Consumer Communications & Networking Conference*, Jan. 2013.

[78] J. Marquegnies. Document layout analysis in SCRIBO. Technical Report CSI Seminar 1102, Research and Development Laboratory, EPITA, July 2011.

[79] Z. C. Marton, R. B. Rusu, and M. Beetz. On fast surface reconstruction methods for large and noisy datasets. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Kobe, Japan, May 2009.

[80] J. McDonald, M. Kaess, C. Cadena, J. Neira, and J. Leonard. Real-time 6-DOF multi-session visual SLAM over large scale environments. *J. of Robotics and Autonomous Systems*, 61(10):1144–1158, October 2013.

[81] M. Meilland and A. Comport. On unifying key-frame and voxel-based dense

visual SLAM at large scales. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 3-8 November 2013. IEEE/RSJ.

[82] N. J. Mitra, A. Nguyen, and L. Guibas. Estimating surface normals in noisy point cloud data. In *special issue of International Journal of Computational Geometry and Applications*, volume 14, pages 261–276, 2004.

[83] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI Conf. on Artificial Intelligence*, Edmonton, Canada, 2002.

[84] J. Montiel, J. Civera, and A. Davison. Unified inverse depth parametrization for monocular SLAM. In *Robotics: Science and Systems (RSS)*, 2006.

[85] R. A. Newcombe and A. J. Davison. Live dense reconstruction with a single moving camera. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 1498–1505, 2010.

[86] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time Dense Surface Mapping and Tracking. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, Washington, DC, USA, 2011.

[87] R. Newcombe, S. Lovegrove, and A. Davison. DTAM: Dense tracking and mapping in real-time. In *Intl. Conf. on Computer Vision (ICCV)*, pages 2320–2327, Barcelona, Spain, November 2011.

[88] P. Newman. *On the Structure and Solution of the Simultaneous Localisation and Map Building Problem.* PhD thesis, The University of Sydney, 1999.

[89] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3d recon-

struction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*, 2013.

[90] B. Oehler, J. Stueckler, J. Welle, D. Schulz, and S. Behnke. Efficient multi-resolution plane segmentation of 3d point clouds. In *Intelligent Robotics and Applications*, volume 7102 of *Lecture Notes in Computer Science*, pages 145–156. Springer Berlin Heidelberg, 2011.

[91] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2262–2269, May 2006.

[92] E. Olson and P. Agarwal. Inference on networks of mixtures for robust robot mapping. In *Robotics: Science and Systems (RSS)*, Sydney, Australia, July 2012.

[93] B. Pateiro-López and A. Rodríguez-Casal. Generalizing the convex hull of a sample: The r package alphahull. *Journal of Statistical Software*, 34(i05), 2010.

[94] M. Pauly, M. Gross, and L. P. Kobbelt. Efficient simplification of point-sampled surfaces. In *Proc. of the Conference on Visualization*, VIS '02, pages 163–170. IEEE Computer Society, 2002.

[95] K. Pirker, M. Rüther, G. Schweighofer, and H. Bischof. GPSlam: Marrying sparse geometric and dense probabilistic visual mapping. In *British Machine Vision Conf. (BMVC)*, pages 115.1–115.12, 2011.

[96] T. Rabbani, F. van Den Heuvel, and G. Vosselmann. Segmentation of point clouds using smoothness constraint. *Inter. Archives of Photo. Remote Sensing and Spatial Info. Sciences*, 36(5):1–6, 2006.

[97] H. Roth and M. Vona. Moving volume KinectFusion. In *British Machine Vision Conf. (BMVC)*, Surrey, UK, September 2012.

[98] M. Ruhnke, L. Bo, D. Fox, and W. Burgard. Compact RGBD surface models based on sparse coding. In *AAAI Conf. on Artificial Intelligence*, 2013.

[99] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison. SLAM++: Simultaneous localisation and mapping at the level of objects. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, June 2013.

[100] A. Saxena, L. Wong, M. Quigley, and A. Ng. A vision-based system for grasping novel objects in cluttered environments. In M. Kaneko and Y. Nakamura, editors, *Robotics Research*, volume 66 of *Springer Tracts in Advanced Robotics*, pages 337–348, 2011.

[101] C. E. Scheidegger, S. Fleishman, and C. T. Silva. Triangulating point set surfaces with bounded error. In *Proc. of the 3rd Eurographics symposium on Geometry Proc.* Eurographics Association, 2005.

[102] T. Schöps, J. Engel, and D. Cremers. Semi-dense visual odometry for AR on a smartphone. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, September 2014.

[103] G. Sibley, C. Mei, I. Reid, and P. Newman. Planes, trains and automobiles – autonomy for the modern robot. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 285–292. IEEE, 2010.

[104] G. Sibley, C. Mei, I. Reid, and P. Newman. Vast-scale outdoor navigation using adaptive relative bundle adjustment. *Intl. J. of Robotics Research*, 2010.

[105] H. C. Siddharth Choudhary, Alexander Trevor and F. Dellaert. SLAM with object discovery; modeling and mapping. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Chicago, USA, September 2014.

[106] P. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *Intl. J. of Robotics Research*, 5:56–68, 1986.

[107] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer Verlag, 1990.

[108] F. Steinbrücker, J. Sturm, and D. Cremers. Real-Time Visual Odometry from Dense RGB-D Images. In *Workshop on Live Dense Reconstruction with Moving Cameras at the Int. Conf. on Computer Vision (ICCV)*, November 2011.

[109] F. Steinbrücker, C. Kerl, J. Sturm, and D. Cremers. Large-scale multi-resolution surface reconstruction from rgb-d sequences. In *Intl. Conf. on Computer Vision (ICCV)*, Sydney, Australia, 2013.

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

[111] H. Strasdat, J. Montiel, and A. Davison. Real-time monocular SLAM: Why filter? In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2010.

[112] J. Strom, A. Richardson, and E. Olson. Graph-based segmentation for colored 3D laser point clouds. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2010.

[113] J. Stückler and S. Behnke. Multi-resolution surfel maps for efficient dense 3d

modeling and tracking. In *Journal of Visual Communication and Image Representation*, 2013.

[114] J. Stühmer, S. Gumhold, and D. Cremers. Real-time dense geometry from a handheld camera. In *Pattern Recognition (Proc. DAGM)*, pages 11–20, Darmstadt, Germany, September 2010.

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

[116] R. W. Sumner, J. Schmid, and M. Pauly. Embedded deformation for shape manipulation. In *SIGGRAPH*, New York, NY, USA, 2007. ACM.

[117] N. Sünderhauf and P. Protzel. Switchable constraints for robust pose graph slam. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[118] J. Tardós, J. Neira, P. Newman, and J. Leonard. Robust mapping and localization in indoor environments using sonar data. *Intl. J. of Robotics Research*, 21(4):311–330, April 2002.

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

[120] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment – a modern synthesis. In W. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *LNCS*, pages 298–372. Springer Verlag, 2000.

[121] T. Tykkälä, A. I. Comport, and J.-K. Kamarainen. Photorealistic 3D Mapping of Indoors by RGB-D Scanning Process. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 3-7 November 2013.

[122] T. Tykkälä, C. Audras, and A. Comport. Direct Iterative Closest Point for Real-time Visual Odometry. In *The Second international Workshop on Computer Vision in Vehicle Technology: From Earth to Mars in conjunction with the International Conference on Computer Vision*, Barcelona, Spain, November 6-13 2011.

[123] G.-J. van den Braak, C. Nugteren, B. Mesman, and H. Corporaal. Fast hough transform on GPUs: Exploration of algorithm trade-offs. In *Advances Concepts for Intelligent Vision Systems*, volume 6915 of *Lecture Notes in Computer Science*, pages 611–622, 2011.

[124] R. Wagner, U. Frese, and B. Bäuml. 3D Modeling, Distance and Gradient Computation for Motion Planning: A Direct GPGPU Approach. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2013.

[125] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, Jul 2012.

[126] T. Whelan, H. Johannsson, M. Kaess, J. Leonard, and J. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013.

[127] T. Whelan, M. Kaess, J. Leonard, and J. McDonald. Deformation-based loop closure for large scale dense RGB-D SLAM. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, November 2013.

[128] T. Whelan, M. Kaess, R. Finman, M. Fallon, H. Johannsson, 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 *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Berkeley, USA, Jul 2014.

[129] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. Leonard, and J. McDonald. Real-time large scale dense RGB-D SLAM with volumetric fusion. In *International Journal of Robotics Research Special Issue on Robot Vision*, 2014.

[130] T. Whelan, L. Ma, E. Bondarev, P. H. N. de With, and J. McDonald. Incremental and batch planar simplification of dense point cloud maps. In *Robotics and Autonomous Systems ECMR '13 Special Issue*, 2014.

[131] S. B. Williams. *Efficient Solutions to Autonomous Mapping and Navigation Problems*. PhD thesis, ACFR, Univ. of Sydney, Australia, September 2001.

[132] M. Zeng, F. Zhao, J. Zheng, and X. Liu. A Memory-Efficient KinectFusion Using Octree. In *Computational Visual Media*, volume 7633 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2012.

[133] Q. Zhou and V. Koltun. Dense scene reconstruction with points of interest. In *SIGGRAPH*, Anaheim, CA, USA, 2013. ACM.

[134] Q. Zhou, S. Miller, and V. Koltun. Elastic Fragments for Dense Scene Reconstruction. In *Intl. Conf. on Computer Vision (ICCV)*, December 2013.