Imperial College London
Department of Computing

Dense Monocular Perception for Mobile Robotics

Jacek Zienkiewicz

April 2017

Supervised by Prof. Andrew Davison
Co-supervised by Dr Stefan Leutenegger

Submitted in part fulfilment of the requirements for the degree of PhD in Computing and the Diploma of Imperial College London. This thesis is entirely my own work, and, except where otherwise indicated, describes my own research.

Dla Jowity

Abstract

This thesis concerns the problem of providing a mobile robot with detailed perception of its local environment using a passive, monocular camera. We embrace the paradigm of dense visual SLAM and bring it to the domain of small, low-cost robots. This enables us to directly use information collected from all pixels in an image and create dense reconstructions of environments.

We present a complete and self-contained perception system that allows a mobile robot to estimate its ego-motion, perform infrastructure-free auto-calibration and build, in real-time, a detailed map of its environment in the form of a height map from a single, monocular camera. Our system is capable of providing a robot with accurate information in a form directly suitable for local navigation and obstacle avoidance. By adopting more restrictive, task-oriented models and using the domain knowledge about our applications we were able to improve performance and robustness. Furthermore, when designing our algorithms, we put a great emphasis on methods that can be efficiently and in a straightforward manner implemented on parallel architectures, and therefore we can achieve excellent scalability in terms of resolution of input images and environment representation. We believe that this work offers a promising route to a truly usable real-time monocular dense SLAM system for mobile robots.

Acknowledgements

This work would not have been possible without the help and encouragement of many people to whom I am eternally grateful.

First, I would like to thank to my supervisor, Professor Andrew Davison, who provided me with this amazing opportunity, together with his trust and support throughout my studies, as well as the patience and freedom to explore and learn. Pursuing a PhD was one of my biggest dreams and working under Andy's supervision has been a life-changing experience. I am also very grateful to my second supervisor, Dr Stefan Leutenegger, for all his guidance during my final two years of study. Stefan's advice, new insights and perspectives have been highly important for the successful completion of my PhD.

During my studies in London I was very fortunate to meet many brilliant people and it was a great experience to work with them. Special thank you go to Robert Lukierski for being a fantastic friend and lab mate, and for being honest, trustworthy and always willing to help. Thanks are due to Ankur Handa for his willingness to patiently explain and demystify even the most sophisticated mathematical tools and concepts. Ankur's help was fundamental and invaluable to my progress and development as a researcher, especially during the first two years. I would like to thank to other other colleagues and friends from my lab and Imperial College London, especially Akis Tsiotsios, Jan Jachnik, Sanjay Bilakhia, John McCormac and Renato Salas-Moreno, for the many fruitful discussions, advice, collaboration and encouragement they provided. I really appreciate the support I received from Hauke Strasdat and Steven Lovegrove at the very beginning of my PhD. I am also grateful to the people who helped even before I started my PhD, and supported me in my preparation for the research I have undertaken at Imperial, in particular, Marcus Moe, Stuart Duff, Emily Jordening, Ana Plata, and Sira Gonzalez.

Finally, I would like to express my sincere gratitude to all my family. Thank you to my mum, for all the love, faith, and support she gave me to follow my passions and dreams — *dziekuje mamo!* I owe a huge amount to my wonderful wife Jowita, to whom I dedicate this thesis; thank you for your patience, years of sacrifice and unconditional support.

The research presented in this thesis has been funded by Dyson Technology Ltd., for which I am very grateful.

Contents

Chapter 1

Introduction

Contents

1.1 Objectives and Motivation

Cameras are widely considered as a very attractive and suitable sensing modality for mobile robot perception and SLAM (Simultaneous Localisation and Mapping), and after decades of active research we are now seeing increasingly more significant deployments of robotic systems for real-world applications which actively utilise computer vision. Motivating examples for our study are in particular systems in the domain of low-cost robotics aiming at mass-market service tasks, like the recently introduced robotic vacuum cleaners Dyson 360Eye™ (Fig. 1.1) and iRobot Roomba® 980. These robots utilise monocular cameras and build sparse maps of their environments in order to perform localisation and execute efficient cleaning patterns. However, for robust operation, the robots still have to employ a collection of additional sensing modalities including bump sensors and PSD (Position Sensitive Device) for short range obstacle detection and avoidance.

Figure 1.1: The Dyson 360Eye™ robotic vacuum cleaner and similar domestic robots are motivating examples for our study in which we address the question of whether a single camera can provide a mobile robot with enough capabilities to perform autonomous navigation and mapping. (Image reproduced courtesy of Dyson.)

Advancing commodity processing power, particularly from GPUs (Graphics Processing Unit), together with algorithmic improvements, have enabled various recent demonstrations of real-time dense 3D reconstruction from monocular video. A typical approach to live, dense monocular reconstruction builds upon 3D camera motion tracking, followed by high quality multi-view depth map estimation and fusion into a generic 3D representation such as a TSDF (Truncated Signed Distance Function) or surfel cloud. Clearly, robotics applications can draw inspiration from the *Dense Tracking and Mapping (DTAM)* [Newcombe et al., 2011b; Newcombe, 2012] approach and many systems presented in this vein have been impressive, however, we note that there are relatively few examples of moving beyond *showing* real-time dense reconstruction towards *using* it in specific real-world applications, for example for vision-based only navigation. The reasons for that are manifold: monocular reconstruction pipelines are commonly perceived as complex, requiring significant parameter tuning, and they are heavyweight and lack robustness in challenging conditions. Furthermore, the reconstructions are often in a form which needs substantial further processing before they could be applied for any in-the-loop use by a robot, such as path planning or obstacle avoidance. In addition, reconstructing and tracking the world in full 3D might simply not be necessary or computationally justified, for example for a mobile robot moving on the floor in a domestic environment.

In this thesis we adopt the core concepts of dense monocular tracking and mapping, and bring it to the domain of small, low-cost robots. Specifically, we build a real-time system that can provide a robot with a high-quality and detailed geometric perception of its local environment using only a single moving camera. We will rely on carefully chosen models and parameterisation as well as the domain knowledge in order to achieve robustness and address some of the above-mentioned challenges. When designing our system we put great emphasis on algorithms and implementations that make efficient use of highly parallel computing architectures, currently provided via commodity GPUs. We consider this as a strength and a key to the efficiency and scalability of our approach, as it allows us to embrace the paradigm shift in processor architectures and now ubiquitous parallel computing. Most of the recent advances in computing come from parallel and heterogeneous architectures and we expect this trend to continue. If one considers the embedded computing platforms we can anticipate on low-cost mobile devices, it is strongly likely that massively parallel GPU-like, FPGA-like or specialised DSP units will offer the dominant low-power processing resource and parallelisable algorithms will increasingly come to the fore.

We will show that a *dense* and parallelisable vision approach to robot SLAM using a single camera can potentially play the key role in many of the capabilities required for autonomous navigation, ego-motion estimation, depth estimation as well as depth and colour fusion. We intentionally try to use a single camera to solve as many aspects of robot autonomy as possible. From a robot perspective, the system is self-contained, and includes infrastructure-free auto-calibration. In principle it allows us to take any camera, put it on the robot and start autonomous operation.

Although our focus is to use this system for a mobile robot moving in a typical indoor environment, certain elements of our system are quite general and capable of performing SLAM and 3D reconstruction in more general settings. We present particular solutions for each subsystem, *i.e.* camera tracking, depth map estimation and depth map fusion, but the overall architecture is quite flexible, and individual components in the pipeline can be (and sometimes will be) exchanged for other methods. For example, instead of using dense image alignment for camera tracking, one can rely on a feature-based tracking system, or completely skip depth map estimation and use a depth or stereo-camera instead.

1.2 Dense, Monocular Vision: Challenges and Opportunities

There are various different approaches to real-time visual SLAM and in this thesis we build on the core concepts centred around dense and monocular methods. In general, by dense and monocular methods we understand approaches that build dense maps of an environment using a single, passive camera. This is in contrast to sparse methods that usually reconstruct the world in the form of spare point clouds of features, and methods that rely on stereo and depth cameras. In this section we will discuss the key characteristics and challenges associated with a dense, monocular approach and compare it with alternative techniques.

1.2.1 Monocular vs. depth and stereo cameras

The term *monocular* describe a system that uses only a single, passive (RGB or greyscale) camera. Performing visual SLAM using a monocular camera only is a very challenging problem, because in contrast to vision systems that use *e.g.* two synchronised cameras in a stereo setup, or time-of-flight sensors to infer depth, in monocular systems, structure of the scene has to be estimated from camera motion.

With the increasing availability of relatively cheap and advanced depth cameras (*e.g.* Intel RealSense, Microsoft Kinect, PMD pico flexx Time-of-Flight camera), one can challenge the motivation behind monocular systems. Even the cost of equipping a robot with an additional image sensor can seem quite negligible, compared to the potential benefits that a stereo system can offer: from a stereo pair of images, the 3D structure of a scene can be calculated immediately without the need for the robot to move. Whilst active depth and stereo cameras have many advantages, in certain circumstances passive systems can be preferred. We will discuss the strengths and weaknesses of the monocular approach to visual SLAM and will argue that a monocular system still has many interesting properties compared to depth and stereo cameras that can justify its use.

One of the first appealing characteristics of a monocular SLAM system is its simplicity and accessibility. It is very straightforward to create and experiment with such a computer vision system, as all that is required is a single camera connected to a processing unit, such as a laptop computer. By avoiding the hardware complexity associated with depth and stereo cameras (*e.g.* there is no need for synchronisation between the imaging modules as in stereo vision), the full SLAM system can be realised in a very compact form-factor and at a lower cost. In general, this makes the deployment of monocular systems easier, as the algorithms can be implemented using already existing, ad-hoc cameras, and do not require (sophisticated) hardware upgrades.

There are other scenarios where passive systems are preferred over depth cameras, for example due to power consumption or outdoor use. Depth cameras typically require power intensive active illumination, which increases the thermal budget needed for operation, and they tend to not work robustly in strong ambient light, which occurs in typical outdoor settings.

While depth camera technology will surely continue to improve, the resolution and frame-rate of current depth sensors is still quite limited. In that respect monocular systems are more flexible and are mainly limited by computational power. A single camera paired with direct and dense methods (that we will discuss in the next section) can result in highly scalable algorithms that can offer interesting trade-offs when camera frame-rate and resolution are varied [Handa et al., 2012].

Depth cameras are able to perform direct measurements of the 3D world structure but have limited operation range. On the other hand, monocular systems are inherently scale-agnostic, which means that it is impossible to measure the absolute scale of the camera motion or of the reconstructed scene. This can be seen as a limitation, but we consider it as one of the greatest properties of monocular systems, because unlike time-of-flight systems and stereo cameras, there are no limits on minimal and maximal range of estimated depth — it allows the same method to be applied both to planetary surface reconstruction as to close-up reconstructions. A monocular system is effectively a variable baseline stereo, and we can use multiple and various baselines by actively moving the camera. This is a property that we will particularly exploit to obtain high quality detailed reconstructions. By combining multiple frames for multi-view stereo we therefore obtain more robust and higher quality results than by using standard two-view stereo.

Clearly, there are several limitations of monocular passive systems as well. Obviously, passive system do not work in darkness, and a robot using a monocular system will be required to move in order to perceive the world in 3D. Furthermore, in monocular depth estimation we usually assume that the scene is static between individual frames. One of the main drawbacks of passive systems is that they rely on natural texture for both tracking and depth estimation. This problem can be at least partially tackled by improving the quality of input data, for example, by increasing the resolution or using better sensors with higher signal-to-noise ratio. Alternatively, one can increase the amount of data and use multiple frames as described above.

A very interesting research direction and possibility for future work is in creating a system that combines the advantages of depth/stereo cameras together with the best properties of a monocular system. Clearly the sensing modalities can be very complementary, and possibly with other sensors like Inertial Measurement Units (IMU) can allow for creating truly robust and power-efficient SLAM systems. In fact, many of the methods described in this thesis, although designed for a monocular system, can be almost directly used with a stereo/depth camera. This shows that developing a monocular system could help us understand and develop better algorithms for depth cameras. For example we could use multi-view stereo techniques to enhance the output of a depth or stereo camera in order to fill-in missing data or enable its operation at very short distance range, or under strong ambient lighting.

1.2.2 Dense vs. feature-based methods

Apart from building a monocular camera SLAM system, we also aim at using so-called "dense" methods. By dense methods we understand here approaches that allows us to build dense models of environment, but also during the estimation try to directly use image information collected from *all pixels*. For the above reason, although not necessarily synonymous, the term "dense" is sometimes also used to describe "direct" methods. This is in contrast to so-called "feature-based methods" which, during estimation, rely only on a sparse set of distinct features extracted from each image separately. The difference between "direct" and "feature-based" approaches is probably best defined by [Irani and Anandan, 1999] who use the term "direct" to describe methods which recover the unknown parameters, by minimise a cost function based on *measurable image quantities* collected from all pixels in the image (*e.g.* image brightness, or brightness-based cross-correlation *etc.*). On the other hand, the "indirect" or "feature-based" methods, determine the motion and shape by minimising an error measure based on *distances* between corresponding features identified in each image. An example of a direct and dense approach is the Lucas-Kanade method [Lucas and Kanade, 1981] for image alignment, whereas a representative feature-based approach is Bundle Adjustment [Triggs et al., 1999].

The discussion regarding the advantages and disadvantages of direct and feature-based methods has a long tradition in the computer vision community [Sawhney et al., 1999], and both approaches have achieved many significant contributions to the field. In particular, in the area of single camera tracking most of the early progress was possible due to feature-based methods [Davison, 2003; Klein and Murray, 2007]. In recent years, due to novel cameras like Microsoft Kinect and parallel processors, researchers concentrated increasingly on direct and dense methods. There is still ongoing research on monocular camera tracking and mapping, and most of the state-of-the-art SLAM systems utilise ideas from both indirect and direct methods, *e.g.* Semi-direct Visual Odometry (SVO) [Forster et al., 2016] or the method presented by [Engel et al., 2016] which is the first system that is both direct and sparse. On the other hand, one of the best performing, easy to use and robust systems right now, ORB-SLAM [Mur-Artal et al., 2015], is a feature-based method. Which methods and approach one eventually uses depends on many circumstances, such as the camera used, application area and even quality of engineering. Although we do not restrict ourselves to any paradigm and in fact we will also use feature-based tracking in some of our experiments (Chapter 7), for the reasons described below in this research we are mostly interested in direct and dense methods.

One of the advantages of direct methods is that they naturally lead to dense reconstructions that is of a great practical importance for mobile robot perception. Maps created by feature-based methods are typically too sparse for any practical use in robot navigation, such as obstacle detection and avoidance, and any reasoning about semantic information, such as free space. On the other hand, by using all pixels in the image, direct methods can achieve much higher measurement density, and create models of the environment that are directly usable for a mobile robot. We will refer to the term dense, instead of direct, to describe our approach as it is more general and emphasises our interest in not only using all pixels but creating dense models.

Another important characteristic of direct methods is that they tend to be more robust compared to feature-based methods under many conditions that we will encounter in our settings, *e.g.* image degradation due to motion blur, or highly repetitive textures. Furthermore, by explicitly using contributions from all pixels in the image, even in relatively textureless areas where indirect methods struggle to extract features, direct methods still can perform well and robustly.

As mentioned, in our approach we aim at exploiting the power of parallel computing, and we believe that direct and dense methods are best suited for very efficient implementations on parallel processors like GPUs. In general, direct methods tend to be simpler and easier to understand and execute as they consist of many relatively straightforward computations on small independent data bits (pixels) which can be easily run concurrently. Even when parallel processing is not available, direct and semi-direct methods can be implemented efficiently on conventional processors as demonstrated, for example in [Engel et al., 2013; Kerl et al., 2013; Forster et al., 2014].

There are also very interesting challenges and open research questions related to dense and direct methods. For example, despite certain attempts [Whelan et al., 2015], loop-closure is still an unsolved problem for fully dense methods. Furthermore, whereas elegant solutions exist for multi-sensor fusion involving feature-based methods and *e.g.* IMU [Leutenegger et al., 2014], given different sensor characteristics, fusing an IMU with measurements from a visual front-end that is based on a direct method is still an open research problem. Finally, we believe that the direct and dense paradigm to visual SLAM is in the best position to embrace the deep learning revolution in computer vision. Future SLAM systems will very likely enhance their robustness and versatility thanks to elements of deep learning and end-to-end training combined with the fundamental ingredients of direct methods.

1.3 Computer Vision and Robot Perception

In the following we will present a brief review of the history of robot perception and visual SLAM which has inspired the work in this thesis. A more detailed review of related work is included in the individual chapters.

Computer vision has been used for robot perception from the early days of mobile robotics, starting with the seminal work by Moravec [Moravec, 1977, 1980], who first presented an impressive system that allowed a robot to drive autonomously through a cluttered environment guided only by an on-board camera. The robot was equipped with a single camera that was actively moving to obtain stereo information and estimate the 3D locations of obstacles. The system incorporated and implemented all relevant functionalities required for vision based autonomous operation, including automatic camera distortion calibration, vision based ego-motion estimation, stereo-based depth estimation and obstacle detection as well as mapping. Based on the information obtained from the vision system, the robot was able to plan and execute its motion and perform obstacle avoidance. Similarly to our work, Moravec's system was concerned with geometric perception rather than semantic understanding of the environment.

Moravec's work lead to further research in the field of visual perception for autonomous robot navigation, but progress was much slower than originally anticipated. The robotics systems of the 1980s and 1990s, although conceptually impressive, had quite limited capabilities and were able to operate successfully only in highly constrained environments and under a significant amount of human supervision. One of the major limiting factors of those systems was the lack of sufficient processing power (as already observed by Moravec himself), but the systems were also inherently limited due to deficits in the low-level vision components relied upon. This is one of the reasons why much of the research in the following years was focused on reliable and fast low level visual techniques, such as feature detection and matching.

Vision continued to be an integral part of robot perception, but because of the above mentioned limitations, in order to act independently in more complicated surroundings, robots had to rely on a variety of sensors, including sonars [Elfes, 1987; Durrant-Whyte, 1994] and laser range finders, as well as odometry from wheels and IMU. Passive vision was mainly used in stereo setups *e.g.* [Krotkov et al., 1995]. For example, Amber, the robotic platform for Planetary Exploration developed in the 1980s [Bares et al., 1989], relied mostly on a laser scanner for obstacle detection and mapping and used a camera only for the purpose of "semantic" understanding.

The 1990s and 2000s were decades of continuous progress in SLAM, artificial intelligence and state estimation, and in autonomous driving in general. The significant advances made in mobile robotics were best demonstrated in very famous and successful driver-less car competitions: the DARPA Grand Challenge [Thrun et al., 2006b] and DARPA Urban Challenge [Urmson et al., 2008]. The first challenge was to build an autonomous robot capable of traversing over 200 km through a desert terrain in less than 10 hours. The DARPA Urban Challenge required a robot to travel fully autonomously through an urban environment, and included situations such as stop intersections, traffic merges and parking. Although cameras were integral parts of the perception systems, as for example described in [Leonard et al., 2008], most of the teams equipped their robots with at least a few laser range finders. Particularly in terms of accuracy and robustness, laser scanners were superior compared to vision and were critical for safe and reliable operation. In fact, to some degree we still see this trend continue today because most autonomous cars being developed now (with notable examples like Tesla which relies on vision and radar, or MobilEye), in one way or other rely on laser scanners, *e.g.* Velodyne, for safety critical issues and in order to obtain reliable, metric maps of the environment. The era of laser scanners in robotics was nonetheless very important for computer vision, because, as we will see later, many algorithms developed for laser scanners, *e.g.* height map fusion, can be used in conjunction with the data coming from a passive camera. The DARPA Grand and Urban Challenge were also great accelerators for progress and innovation in passive vision. In fact, it is reported that the robot Stanley, winner of the DARPA Grand Challenge, could maintain its high speed and therefore win thanks to long-range vision [Dahlkamp et al., 2006].

Independent of the progress in robotics, the computer vision community continued research in the field of visual SLAM and 3D reconstructions. There are many different techniques for obtaining 3D structure from images, including Shape-from-Shading [Zhang et al., 1999], Shape-from-silhouettes [Fitzgibbon et al., 1998; Hernández and Schmitt, 2004], Depth-from-Defocus [Nayar et al., 1995], and Space Carving [Kutulakos and Seitz, 2000]. There exist also methods that perform monocular scene reconstruction using machine learning techniques, for example based on graphical models that combine local features with global reasoning [Saxena et al., 2005; Hoiem et al., 2005, 2008], or are even able to estimate depth maps from a single image using Deep Networks *e.g.* [Eigen et al., 2014]. However, the most widely and successfully used are approaches based on multi-view geometry and Structure-from-Motion. One reason for this particular success might be that they require the least complex hardware, and put few constraints on the environment, as well as simultaneously tackling the problem of not only 3D structure recovery but also camera motion estimation.

Indeed, existing state-of-the-art image-based geometry reconstruction systems can deliver remarkable results at large scales *e.g.* [Agarwal et al., 2009; Frahm et al., 2010]. These methods perform global optimisation using a multi-stage pipeline that involves many separate and complex steps, including Bundle Adjustment and Structure-from-Motion (*e.g.* Bundler [Snavely et al., 2006], VisualSFM [Wu, 2013], OpenMVG [Moulon et al., 2013]), Multi-View Stereo (*e.g.* [Goesele et al., 2007], CMVS [Furukawa and Ponce, 2007], and SURE [Wenzel et al., 2013]) as well as surface reconstruction (*e.g.* [Kazhdan and Hoppe, 2013]). MVE [Fuhrmann et al., 2014] is an example of a system that incorporates all of these parts in a single framework, but still as distinct stages each of which performs global optimisation on image batches. Incremental reconstruction, *e.g.* adding more images to the already reconstructed scene, is not straightforward and explicitly handled. The overall framework needs hours to operate even on a relatively small set of input images. Because all of these approaches are fundamentally not sequential and highly prohibitive for real-time operation, they are not suitable for a robotics application where (a) processing should be fast and (b) the reconstruction should be updated incrementally as new frames are processed.

For robotics, of much greater interest are the real-time visual SLAM systems that perform incremental reconstruction suitable for in-the-loop control or guidance. Many of the real-time systems described below share some similarities and use the same underlying techniques as off-line methods, but they are designed to operate under significantly different assumptions and circumstances, *i.e.* perform an incremental reconstruction from an incoming stream of video frames, rather than global optimisation on a large scale set of unordered photos.

The first monocular real-time visual SLAM algorithm that relied only on commodity hardware was MonoSLAM [Davison, 2003]. This pioneering work relied on the Extended Kalman Filter, and although the solution was very elegant, was best suited to operate in small environments and could track only a limited number of features. Another prominent example of a real-time monocular SLAM system is the Parallel Tracking and Mapping (PTAM) [Klein and Murray, 2007] who demonstrated extremely robust sparse tracking, by using bundle-adjustment technique and splitting the tracking and mapping tasks into distinct threads. MonoSLAM and PTAM are examples of feature-based methods that represent the world as a sparse feature set, and are generally very well suited for task such as camera tracking; latest state-of-the-art examples include ORB-SLAM [Mur-Artal and Tardós, 2014] and SVO [Forster et al., 2014]. Although certain attempts were made to "densify" the sparse maps of features [Lovegrove, 2011], the maps produced by those methods are certainly not suitable and dense enough to be used as the basis for obstacle detection and robot navigation.

Progress in monocular camera tracking together with increased computational capabilities pushed passive systems towards real-time dense reconstruction. GPU-acceleration enabled implementation of systems that were able to perform depth maps estimation in real-time, *e.g.* [Merrell et al., 2007] developed an algorithm that first computed noisy depth maps from an image sequence and subsequently fused several neighbouring depth maps into a high quality depth map. [Newcombe and Davison, 2010] also utilised the GPU-accelerated techniques and together with PTAM-based camera tracking created a system that was capable of incrementally building a dense 3D mesh model from a single moving camera. The DTAM method [Newcombe et al., 2011b] went even further and was the first to perform not only mapping but also tracking densely. Other prominent examples in the field of real-time, dense monocular reconstruction include work by [Stuehmer et al., 2010] and by [Graber et al., 2011], MonoFusion [Pradeep et al., 2013], and REMODE [Pizzoli et al., 2014]. In the middle ground between feature-based and dense methods are the semi-dense methods with the most notable examples of Semi-dense Visual Odometry [Engel et al., 2013] and LSD-SLAM [Engel et al., 2014].

The arrival of Microsoft's Kinect sensor ushered in an era of affordable, commodity depth cameras in computer vision and robotics. Researchers in robotics immediately realised the potential of depth cameras and [Henry et al., 2010] presented a first impressive system that was able to achieve large scale mapping capabilities thanks to dense surface representation using patches, near real-time 3D camera tracking based on ICP alignment, and loop closure capabilities. The availability of high quality depth maps greatly facilitates camera tracking and 3D reconstruction, and it should not be surprising that most successful systems in real-time dense 3D reconstruction rely on depth cameras, including KinectFusion [Newcombe et al., 2011a], Bylow *et al.* [Bylow et al., 2013], Point-based Fusion [Keller et al., 2013], ElasticFusion [Whelan et al., 2015], Real-time 3D Reconstruction at Scale using Voxel Hashing [Nießner et al., 2013], and Elastic Fragments [Zhou et al., 2013]. With platforms like Google Tango, depth cameras are becoming increasingly accessible, and it is already possible to perform volumetric integration on mobile devices [Kahler et al., 2015].

In this thesis we follow a geometric approach, *i.e.* we aim at performing a geometrical reconstruction of the robot's environment in order to perform certain tasks. There is a significant body of work based on learning and appearance. This is clearly a viable approach, and in fact, the approaches can be complementary and benefit from each other. It is easier to reconstruct if we know what we are reconstructing, but also 3D understanding and modelling greatly improves semantic segmentation and several researchers have tried to exploit this approach *e.g.* , [Bao et al., 2013] and ObjectStereo [Bleyer et al., 2011b]. Prominent examples of learning and appearance based mapping methods were developed, *i.e.* during the DARPA's Learning Applied to Ground Robots (LAGR) project, for example the work of [Hadsell et al., 2009] and [Konolige et al., 2009]. The underlying principle for these methods is as follows: they extract discriminative features from image patches and using their appearance they classify them into different semantic categories suitable for navigation, *e.g.* free space, obstacle, *etc.* These appearance based methods will be discussed in more details in Chapter 5. With the recent resurgence of deep and reinforcement learning, one can anticipate that learning-based approaches will play an increasingly important role in mobile robot perception and autonomous driving. Clearly, this is a very promising route. However, even though certain approaches try to go as far as to learn actions directly from raw input images only, for example learning to navigate in a maze [Mirowski et al., 2016], it is to expected that well-known and established techniques from SLAM will continue to play an important role in systems that require autonomous behaviour, and can by used, for example, to augment deep reinforcement learning as shown in [Bhatti et al., 2016], in order to achieve more consistent and effective behaviours.

1.4 Publications

The work described in this thesis resulted in the following publications:

- Zienkiewicz, J., Lukierski, R., and Davison, A. J. (2013), **Dense, Autocalibrating Visual Odometry from a Downward-looking Camera**. In *Proceedings of the British Machine Vision Conference (BMVC).* [Zienkiewicz et al., 2013];
- Zienkiewicz, J. and Davison, A. J. (2014). **Extrinsics Autocalibration for Dense Planar Visual Odometry**. *Journal of Field Robotics (JFR).* [Zienkiewicz and Davison, 2015];
- Zienkiewicz, J., Leutenegger, S., and Davison, A. J. (2016). **Real-Time Height-Map Fusion using Differentiable Rendering**. In *Proceedings of*

the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS). [Zienkiewicz et al., 2016a];

- Zienkiewicz, J., Tsiotsios, A., Davison, A. J. and Leutenegger, S. (2016). **Real-Time, Monocular Surface Reconstruction Using Dynamic Level-of-Detail**. In *Proceedings of the International Conference on 3D Vision (3DV).* [Zienkiewicz et al., 2016b].

Furthermore, a live demonstration of the multi-scale reconstruction framework from Chapter 7 was presented at:

- Zienkiewicz, J., Leutenegger, S. (2016). **Real-time, Monocular Surface Reconstruction Using Dynamic Level-of-Detail**. Live demonstration at the *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

The following video material demonstrates the algorithms developed in this thesis:

- Real-Time Height Map Fusion using Differentiable Rendering, `https://youtu.be/3NQqeRcSsCw`;
- Monocular, Real-Time Surface Reconstruction using Dynamic Level of Detail, `https://youtu.be/UB_HDJU6LL4`.

1.5 Thesis Outline

The specific chapter-by-chapter breakdown of this thesis is as follows:

- **Chapter 2** introduces the basic notation and concepts required for building a monocular SLAM system, including our method for fast and robust depth map estimation.
- **Chapter 3** describes our approach to visual odometry from a downward looking camera based on whole image alignment.
- **Chapter 4** builds on top of the method presented in the previous chapter and provides a full extrinsics auto-calibration for a mobile robot.

- **Chapter 5** introduces approaches to height map fusion from a stream of depth maps and focuses on efficient GPU implementations.
- **Chapter 6** describes a generative approach to height map fusion and surface reconstruction that relies on the principles of differentiable rendering.
- **Chapter 7** extends the fusion algorithms from the previous chapters to allow multi-scale/multi-resolution surface reconstruction.
- **Chapter 8** concludes the thesis with discussions and suggestions for future work.

Chapter 2

Preliminaries

Contents

2.1 Visual SLAM System for a Mobile Robot

When designing a SLAM system for a mobile robot one has to make several design choices that are determined by several factors, *e.g.* the environment in which the robot is operating, the robot's size and weight, the computational budget, and the desired cost and complexity of the overall platform. In our case, we are considering a relatively small robot moving mostly on a flat surface in indoor environments, and our main goal is to provide the robot with local perception and mapping capabilities.

2.1.1 Hardware and Software Platform

A typical hardware set-up we consider in this thesis is shown in Fig. 2.1. In most of our robotics experiments we used the Pioneer Robot P3-DX platform, a differential drive wheeled mobile robot, carrying an NVIDIA GPU-equipped laptop for real-time

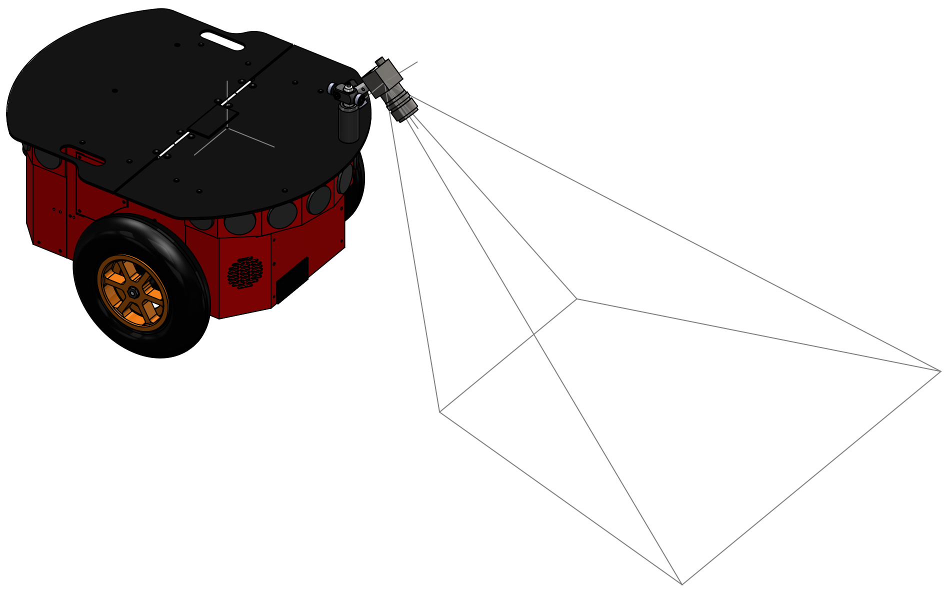


Figure 2.1: A typical hardware set-up considered in this thesis. We use a simple wheeled mobile robot, mount a single camera on it and point it downwards. Here we show how camera view frustum intersects the ground plane. This camera configuration can provide the robot with detailed information about the environment just in front of it, but also some look-ahead required for local motion planning.

vision processing. We mount a single camera on a robot and point it down at the floor at an oblique angle, so that it is looking forward up to approximately 1-3 meters. Fig. 2.2 shows some example images captured by the camera. The camera field of view is dominated by the planar structure of the floor, with small objects like cables lying on it (which are particularly "dangerous" for a small robotic vacuum cleaner and cannot be detected by PSD sensors or a laser scanner), with other typical structures visible such as furniture and walls. We do not put any particular constraints on the camera orientation, and our extrinsics auto-calibration method (which will be presented in Chapter 4) helps with easy experimentation and finding the best position and orientation. However, the chosen configuration allows the robot to obtain detailed information about its very close vicinity, but also sufficient look-ahead to execute motions with velocities that are required for a practical operation (in the range of 0.5—1.5 ms^{-1}).

We have used a standard Point Grey camera capturing images at 640×480 resolution and 30 Hz frame-rate. To compensate for motion blur we reduce the shutter time and use automatic in-built camera settings for gain. The camera has a lens with approximately 80° field of view and is calibrated off-line for intrinsic parameters

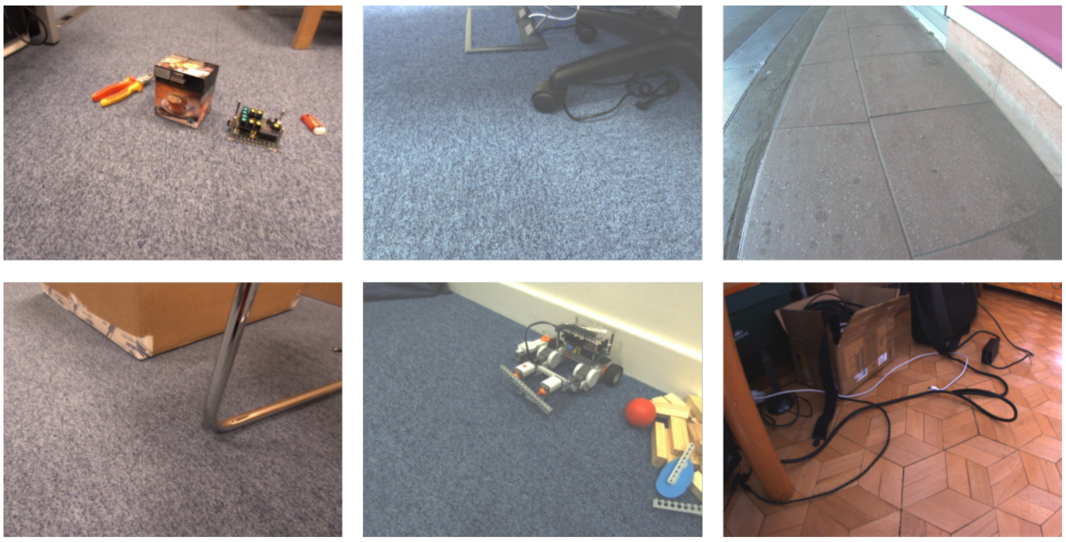


Figure 2.2: Examples of images captured by the camera. Typically, the robot observes a floor surface with small objects lying on it, some vertical structures like walls and furniture.

including significant radial distortion. Note that we undistort our images on the GPU at frame-rate before applying methods which assume perspective projection.

Although we currently use relatively high-end GPUs to enable real-time operation, we intentionally focus on algorithms that are easy to implement on parallel processors, and design them in such a way that it is possible to further optimised and port them to more computational constrained embedded platforms. We achieve this by considering the data flows within modern processors and using techniques that allow for coherent and predictable memory access patterns. Our implementations typically consist of many, relatively simple computations, but ones that are straightforward to execute concurrently. One of the particular strengths of a monocular, parallel approach to visual SLAM is the great scope of scalability of this paradigm. In order to meet particular computational constraints and achieve real-time operation, one can, for example, vary the frame-rate and resolution of the input data.

Most computing units, even embedded platforms for mobile robotics, are already equipped with GPUs, which so far often remain underutilised. For a long time, parallel processors like the GPU were perceived as difficult to program and not power-efficient, but this has changed considerably. In our work we use CUDA which requires NVIDIA's GPUs, in conjunction with modern features of OpenGL, but obviously one can use different programming models and languages, *e.g.* OpenCL,

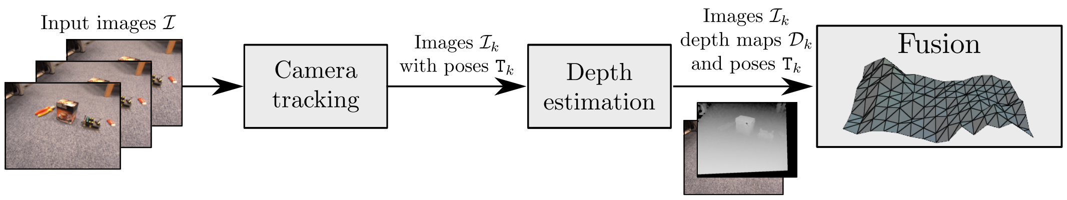

Input images $\mathcal{I}$
Camera tracking
Images $\mathcal{I}_k$ with poses $\mathtt{T}_k$
Depth estimation
Images $\mathcal{I}_k$ depth maps $\mathcal{D}_k$ and poses $\mathtt{T}_k$
Fusion

Figure 2.3: We adapted the fairly standard monocular 3D reconstruction pipeline that roughly consists of three relatively separate stages: camera motion tracking, multi-view depth map estimation as well as the depth and colour fusion.

Vulkan or Apple's Metal. In fact, we believe that many of the parts of our algorithms, *e.g.* the depth estimation method presented later in this chapter, are suitable for implementation on dedicated Visual Processing Units like Movidius Myriad2.

2.1.2 Monocular SLAM Pipeline

In visual SLAM the goal is to incrementally construct and update a map of an unknown environment while simultaneously estimating the position of the camera within this map. To simplify the problem, in our approach we apply the common separation of camera tracking and mapping: in the first step, only the camera motion is estimated, which is subsequently treated as a fixed quantity. Next, we perform depth estimation using a multi-view stereo technique that harnesses the accurate motion estimates. The depth maps and images are subsequently fed into an incremental mapping module in a *loosely-coupled* approach.

This is a relatively standard monocular dense 3D reconstruction pipeline [Newcombe et al., 2011b; Pradeep et al., 2013; Fuhrmann et al., 2014], within which we can identify three relatively separate stages: camera motion tracking, multi-view depth map estimation and finally depth and colour fusion, as shown in Fig. 2.3. This separation not only makes estimation easier, but it also gives additional flexibility. Even though in this thesis we propose methods that take direct advantage of domain knowledge in order to provide solutions that are robust and efficient, and this in particular applies to motion estimation (Chapter 3) auto-calibration (Chapter 4), and depth map fusion (Chapters 5–7), the individual elements of the proposed framework can also be used in different configuration. For example, instead of using only our visual odometry, we will pair the multi-scale surface reconstruction with general 6 DoF motion tracking to perform reconstruction of arbitrary surfaces. The presented monocular pipeline can be also modified and used in conjunction with a depth camera.

This will allow us to simplify the pipeline by entirely omitting depth estimation, but we could still successfully apply the same principles for motion estimation and depth map fusion.

A possible extension and improvement within our framework would consist of a closer integration of its individual elements: for example, instead of performing frame-to-frame tracking only, we could also track the motion with respect to the reconstructed dense model for example in order to reduce tracking drift. Note that in our approach we will not explicitly consider loop-closure, as we are only interested in relatively local perception. However, this is also an interesting direction for future work.

2.2 Camera Model

In this thesis we will work with a standard pinhole camera model, Fig. 2.4, parameterised by horizontal and vertical scaling factors f_u, f_v (which are related to the focal length and pixel size), and the location of the principal point, (u_0, v_0), in image space. To describe the pinhole effect we will generally use the matrix form:

$$\mathtt{K} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} , \tag{2.1}$$

where $\mathtt{K}$ is referred to as the intrinsic matrix.

For the remainder of this thesis we assume that the intrinsics parameters are known, and that the lens distortions not modelled by the pinhole camera have been corrected. Determining camera intrinsics is a straightforward, one-off procedure for a certain sensor and lens, and is out of the scope of this work. In practice, we use the method proposed by [Zhang, 1999].

There are two operations related to the camera model that will recur throughout this thesis: projection and un-projection. Projection describes how a 3D point $\mathbf{P} = (X, Y, Z) \in \mathbb{R}^3$, expressed in the camera local frame of reference, is mapped onto the image plane, and we will denote it by:

$$\mathbf{p} = \pi(\mathtt{K}\mathbf{P}) , \tag{2.2}$$

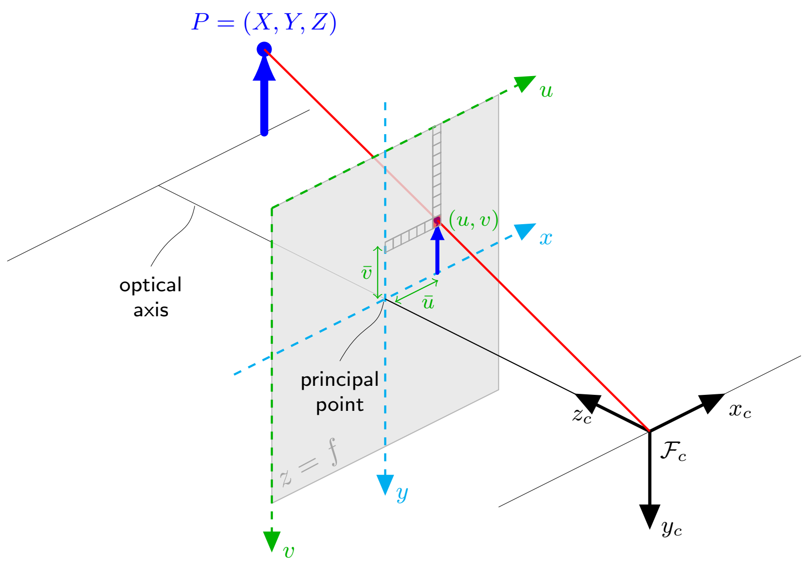


Figure 2.4: The essential geometry describing the effect of a pinhole camera model. The image was adapted from `https://tex.stackexchange.com/questions/96074/more-elegant-way-to-achieve-this-same-camera-perspective-projection-model`, where it was released under a Creative Commons License.

where $\mathbf{p} \in \mathbb{R}^2$ is the point on the image plane and π performs the perspective projection operation defined as:

$$\pi\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \end{bmatrix} . \tag{2.3}$$

Therefore the full projection step is defined as:

$$\mathtt{K}\mathbf{P} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} f_u X + u_0 Z \\ f_v Y + v_0 Z \\ Z \end{bmatrix} , \tag{2.4}$$

$$\mathbf{p} = \begin{bmatrix} f_u \frac{X}{Z} + u_0 \\ f_v \frac{Y}{Z} + v_0 \end{bmatrix} . \tag{2.5}$$

During un-projection we are interested in inverting the projection operation, *i.e.* we want to map a point from an image plane $\mathbf{p} = (x, y) \in \mathbb{R}^2$ into the three-dimensional space. Since in projection we "collapse" a 3D point onto a 2D plane, we are essentially "losing" one dimension, so in general, projection is non-invertible unless we have access to the "lost" dimension. However, if we know or have an estimate of the depth

d of a point $\mathbf{p} = (x, y)$, un-projection is defined by:

$$\mathbf{P} = d\mathtt{K}^{-1}\dot{\mathbf{p}} \ . \tag{2.6}$$

Here, we use the dot notation to describe the homogeneous extension of the point $\dot{\mathbf{p}} = \binom{\mathbf{p}}{1}$, therefore:

$$\dot{\mathbf{p}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} , \tag{2.7}$$

and $\mathtt{K}^{-1}$ is the inverse of the intrinsic matrix $\mathtt{K}$, which is straightforward to compute:

$$\mathtt{K}^{-1} = \begin{bmatrix} \frac{1}{f_u} & 0 & -\frac{u_0}{f_u} \\ 0 & \frac{1}{f_v} & -\frac{v_0}{f_v} \\ 0 & 0 & 1 \end{bmatrix} . \tag{2.8}$$

Therefore, in order to un-project a point, we first homogenise it and multiply by the inverse of the intrinsic matrix to recover the missing dimension:

$$\mathtt{K}^{-1}\dot{\mathbf{p}} = \begin{bmatrix} \frac{1}{f_u} & 0 & -\frac{u_0}{f_u} \\ 0 & \frac{1}{f_v} & -\frac{v_0}{f_v} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x-u_0}{f_u} \\ \frac{y-v_0}{f_v} \\ 1 \end{bmatrix} , \tag{2.9}$$

and then use the known d to calculate its final 3D position in the camera local frame of reference:

$$\mathbf{P} = d \begin{bmatrix} \frac{x-u_0}{f_u} \\ \frac{y-v_0}{f_v} \\ 1 \end{bmatrix} = \begin{bmatrix} d\frac{x-u_0}{f_u} \\ d\frac{y-v_0}{f_v} \\ d \end{bmatrix} . \tag{2.10}$$

The un-projection operation is extensively utilised, *e.g.* during depth map estimation, where we test different depth hypotheses, as well as during height map fusion, where we use the estimated depth values and transform them into height measurements.

2.3 Depth Map Estimation

2.3.1 Introduction

Stereo and depth map estimation is probably one of the most studied subjects in computer vision, and although many interesting and successful methods have been proposed, there seems to be no universal and recommended algorithm that is most widely used. Many methods, despite achieving high scores in stereo benchmarks, do not perform well in practice or are too complex and run extremely slowly (*e.g.* minutes on a single image pair on powerful hardware and GPUs), making them impractical for real-time applications.

A study of the stereo algorithm literature gives good insight into the trends, fashions and progress in computer vision. For a long time, methods utilising probabilistic graphical models were particularly popular, [Kolmogorov and Zabih, 2001; Tran and Davis, 2006; Scharstein and Pal, 2007; Woodford et al., 2008]. The focus was then moved to variational approaches with successful demonstrations like [Newcombe et al., 2011b] and [Ranftl et al., 2012]. Variational methods, although offering very elegant solutions and impressive results, often require manual selection of hyperparameters of the step size involved in the solving of the associated optimisation problems, which makes it difficult to obtain consistent results. Furthermore, although the methods are highly parallelisable, the mathematical apparatus involved can be very complex and difficult to implement correctly. Some of the most compelling results can now be obtained using approaches based on the PatchMatch method [Barnes et al., 2009], *e.g.* PatchMatch Stereo, [Bleyer et al., 2011a], in particular when combined with edge-aware filtering [Lu et al., 2013], or variational methods as in PatchMatch Huber [Heise et al., 2013]. However, the stochastic nature of those methods and the fact that they use very image large patches during computation make them not straightforward to implement on a GPU. Additionally, PatchMatch-based methods do not extend naturally to multi-view stereo. An increasing number of approaches try to leverage recent advances in semantic understanding to improve stereo estimation *e.g.* [Kundu et al., 2014; Yamaguchi et al., 2014], and replace some [Žbontar and LeCun, 2014] or all elements [Eigen et al., 2014] of the standard stereo pipeline with deep learning. However, the solutions that work best and are most widely used in practice often utilise many heuristics and are well-engineered solutions, for example Semi-global Matching (SGM) [Hirschmüller, 2005; Hirschmüller, 2008].

After investigating several stereo algorithms we decided to design our own depth estimation algorithm that is robust and works well in practice and is relatively easy to implement on a GPU and we will describe it in the following section. We use some of the recently proposed, state-of-the-art techniques and also exploit the advantage of a multi-view approach by using multiple frames with different baselines to improve performance. However, we have decided against fully quantifying and evaluating in

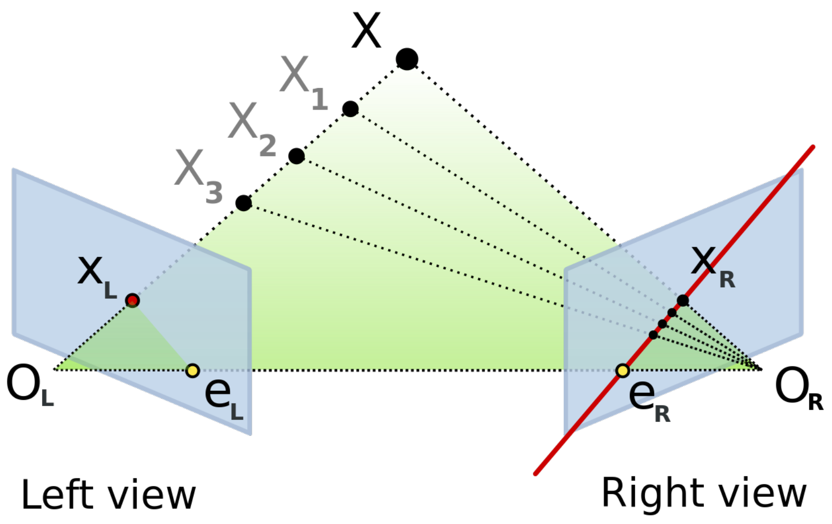


Figure 2.5: The epipolar geometry between two views gives rise to strong constraints that can be exploited during the search for corresponding points in stereo matching. Point $\mathbf{x}_L$ in the left image defines a ray in space, that corresponds to an epipolar line in the right image on which the projection of the 3D point $\mathbf{X}$ lies. (Image obtained from https://upload.wikimedia.org/wikipedia/commons/1/14/Epipolar_geometry.svg, created by Arne Nordmann, GFDL http://www.gnu.org/copyleft/fdl.html, CC-BY-SA-3.0 http://creativecommons.org/licenses/by-sa/3.0/.)

benchmarks or comparing our method with alternative approaches. This is because our later introduced depth map fusion techniques are essentially agnostic to the depth estimation methods and one can replace the depth estimation with one of his/her choice, or even use depth cameras.

2.3.2 Multi-view Stereo Pipeline

The principles of stereo estimation are rather straightforward and well understood: in the most traditional case it is concerned with the estimation of a structure of a scene observed by two cameras from distinct viewpoints. It is assumed that the relative position between the cameras as well as and their internal parameters are known, and the structure is estimated by finding corresponding points in the two views. When performing the search for corresponding points in stereo matching, one can exploit strong constraints that arise from the *epipolar geometry* — the projective geometry between two views [Hartley and Zisserman, 2004]. Here we assume that a pixel in an image can be associated with a ray in space, and therefore, as shown in

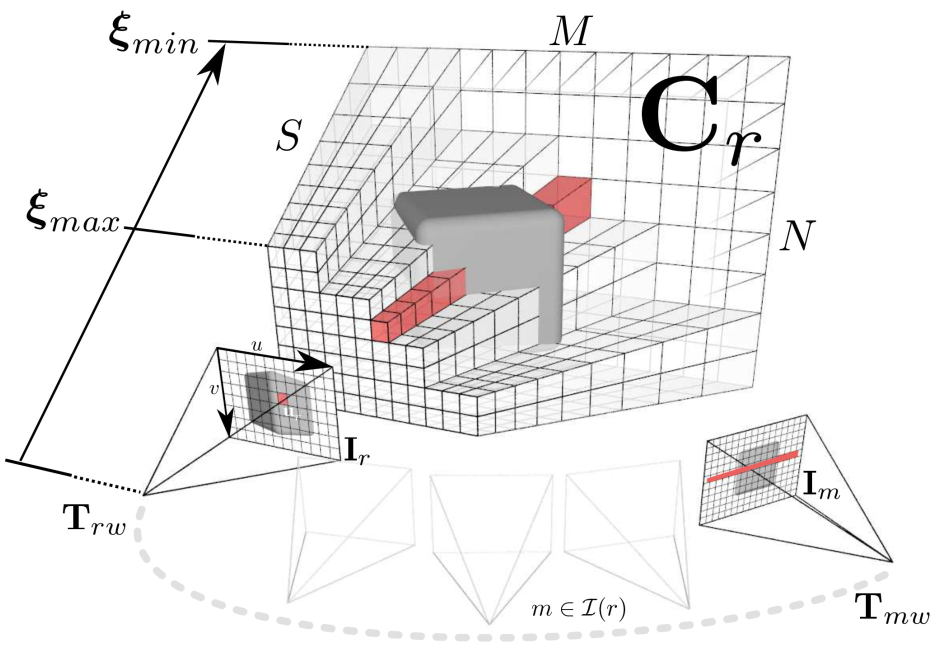


Figure 2.6: Visualisation of the cost volume principle used for dense depth map estimation in a multiple-view stereo. (Image adapted from [Lovegrove, 2011; Newcombe et al., 2011b].)

Fig. 2.5, a point in one view defines to an epipolar line in the other view on which the corresponding point lies.

When designing a stereo algorithm one faces several choices as summarised in [Scharstein and Szeliski, 2001]. With a sufficiently textured scene, even relatively simple methods perform well. However, a lack of texture, non-Lambertian surface properties, changing lighting conditions, bad image quality are some of the factors that can greatly affect the usefulness and quality of results. In monocular depth estimation, we have to deal with additional challenges including tracking failure, incoherent image acquisition settings between consecutive frames (*e.g.* due to auto-gain settings), motion in the scene (the algorithms assume that the scene remains static between frames) and self-shadowing. On the other hand, a monocular set-up also has certain unique characteristics that we can take advantage of. For example, we can easily use a technique referred to as multiple-view stereo (MVS), *i.e.* we can calculate a depth map from more than two images. To implement multiple-view stereo one often uses the concept of cost volume as illustrated in Fig. 2.6. A monocular system is naturally a multi-view, variable baseline stereo system and we can combine multiple frames to improve robustness, or vary the baseline to perform estimation at different scales.

Our depth estimation pipeline exploits the advantages of the monocular multi-view stereo approach, while being relatively simple and easy to implement. We focused on speed and robustness, and our method is fast enough to be run in real-time for every frame at approximately 30 Hz. Rather than spending all effort on depth map estimation, we decided to allocate more resources to depth map fusion (which we will describe in the second half of this thesis). In a nutshell, our approach works as follows. As the camera starts moving we track its motion using, depending on the scenario, either our Dense Visual Odometry (Chapter 3), or ORB-SLAM [Mur-Artal et al., 2015]. While tracking camera motion we maintain a buffer of recent keyframes with various baselines. In order to estimate a depth map, we construct a cost volume, where for each pixel in the current frame we accumulate matching scores between the current frame and a selected set of frames from the buffer of recent keyframes. We utilise the concept of cost volume filtering proposed by [Rhemann et al., 2011] to perform cost aggregation, where in order to achieve smoothness with edge-preserving properties on the estimated depth map we employ the guided image filtering method of [Lu et al., 2012] on the slices of the cost volume. Finally, for each pixel we select the depth with the lowest cost. As a final post-processing step we reject depth measurements with high uncertainty due to lack of texture, and those which fail a standard stereo consistency test due to occlusions. In the following we will describe the individual steps in more detail. Our approach has certain similarities with the work described in [Zhang et al., 2011] (Census transform, use CUDA, use of integral images), but we extended the framework to multi-view stereo and adopted more recent guided-image filtering techniques. We also employ outlier rejection and try to estimate the uncertainty of every depth measurement.

Stereo Matching

Stereo is a correspondence problem, which we typically try to solve by testing how well a point in one image matches with different points along the corresponding epipolar line in a second image, which are found by re-projection using different depth hypotheses. There are numerous methods for assessing how well two image regions match [Hirschmüller, 2007] and they vary in complexity, computational demands, robustness and invariance to certain effects; *e.g.* Sum of Squared Differences (SSD), Sum of Absolute Differences (SAD), Normalized Cross-Correlation (NCC), Truncated absolute difference of colour and the gradient at the matching point [Bleyer et al., 2011a], or Mutual Information [Hirschmüller, 2005]. Recently, methods for computing stereo matching cost with convolutional neural networks were proposed [Žbontar and LeCun, 2014]. The method performs well, but at the moment it is very slow (requiring approximately 90 seconds per image) and the proposed network seems unnecessarily large and complex for a simple matching task (seven fully connected layers and a total of 600000 parameters). However it is to be expected that we will see much faster and more compact architectures not only for matching but for other elements of the stereo pipeline. In our approach we rely on the Census transform [Zabih and Woodfill, 1994] of a 9×7 image patch centred around a pixel together with the Hamming distance for calculating matching cost. One reason for using the Census transform is its robustness to appearance changes due to camera self-shadowing which occur quite often. Furthermore, Census is fast and easy to implement even for relatively large matching window sizes.

Frame Selection and Cost Volume

We utilise the concept of plane sweeping and cost volume aggregation as it is a flexible way of performing multi-view stereo. However, unlike *e.g.* DTAM [Newcombe et al., 2011b], which used multiple small baseline frames, we are more restrictive in the way we select images for stereo matching. We maintain a fixed-size buffer of recent frames that are candidates for matching and a new frame is added to this buffer when the camera has moved sufficiently far from the most recent keyframe. In the multi-scale setting, in order to handle scale variations the threshold for adding new frame is scale dependent. Note that this is entirely independent of the keyframes selected and maintained for example by ORB-SLAM.

In the cost volume we combine the cost from several keyframes. Typically we use a small number of frames (4-7), but with different baselines. This relies on the observation (used in stereo matching, *e.g.* in [Matthies et al., 1989] and [Engel et al., 2013]) that short baselines help to avoid local minima when performing correspondence search, whereas larger baselines improve accuracy, by allowing precise localisation, as illustrated in Fig. 2.7. During depth map estimation, we need to have reasonable estimates of the minimal and maximal depth that we perform the search over. When a camera is fixed on a robot and looking downwards, we can easily calculate the depth range by knowing the distance and orientation of the camera with respect to the floor. In more general 6 DoF tracking, we use the depth of features extracted and tracked by ORB-SLAM to constrain the depth search range.

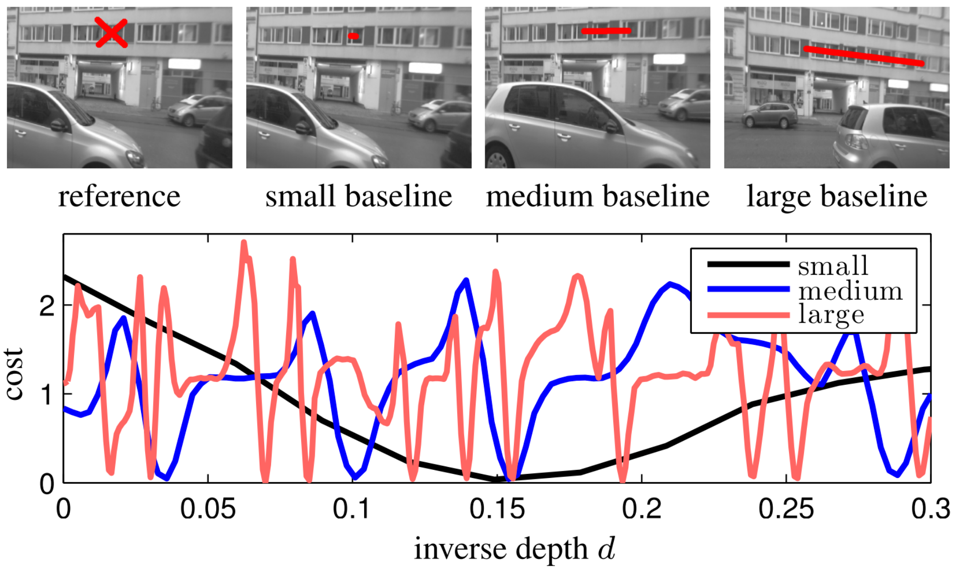


Figure 2.7: Variable baseline stereo (image from [Engel et al., 2013]): in our depth estimation we combine stereo images with different baselines. Small baselines help to avoid local minima when performing correspondence search, whereas larger baselines improve accuracy, by allowing precise localisation.

Cost Aggregation using Guided Filtering

Cost aggregation is performed using a very efficient, parallel implementation of the Cost Volume Filter [Rhemann et al., 2011] which allows the support for aggregation to be selected adaptively. A key observation for the cost volume filtering is that the aggregation step of adaptive support weight algorithms is equivalent to smoothing the stereo cost volume with an edge-preserving filter. The first examples such as [Yoon and Kweon, 2005, 2006] relied on bilateral filters and therefore were limited in their scalability (use of large support) for real-time applications. Instead of using a bilateral filter, [Rhemann et al., 2011] proposed to use the guided image filtering technique introduced by [He et al., 2010]. The guided filter generates output by considering the content of a guidance image, which can be the input image itself or another image. In cost volume filtering we process individual slices of the cost volume by using the current image/frame as a guide. This allows for an edge-preserving smoothing operator similar to a bilateral filter, but gives better results near the edges. One of the main advantages of many guided filters is that their run time is independent of the size of the filter kernel.

In our implementation we use the Cross-based Local Multipoint Filter, CLMF-0 [Lu et al., 2012], a variant of one of the recent guided image filtering techniques. This approach runs in constant time even on a GPU thanks to the use of orthogonal integral images [Zhang et al., 2009]. Our method is not meant to compete in terms of quality of the depth maps with techniques that relies on global optimisation, such as Total Generalized Variation (TGV) [Ranftl et al., 2012] or DTAM [Newcombe et al., 2011b], but it avoids the computational complexity of those methods, and therefore is easier to implement and can run faster, while still offering good regularisation properties in low-texture areas and preserving sharp edges.

Post-processing

The final stage of stereo estimation consists of selecting the depth value with minimal cost, followed by sub-pixel refinement and calculation of depth uncertainty in the way proposed by [Lukierski et al., 2015]. In order to remove unreliable measurements, we reject depth values that do not pass consistency checks or have high uncertainty.

2.3.3 Results

Figs. 2.8 and 2.9 show examples of depth maps calculated with our method, both when the camera is placed on a robot as well as in more general settings. Note that we use greyscale images as the input to our depth estimation method. We can perform depth map estimation in real-time, at about 30 frame per second, using VGA image resolution on a NVIDIA GTX 680. The black areas in the depth maps indicate invalid or unavailable data. Overall, the quality of depth maps is satisfactory, with some missing area due to lack of texture or image overlap. We will use the depth map estimation method presented here for the surface reconstruction in Chapters 5 to 7.

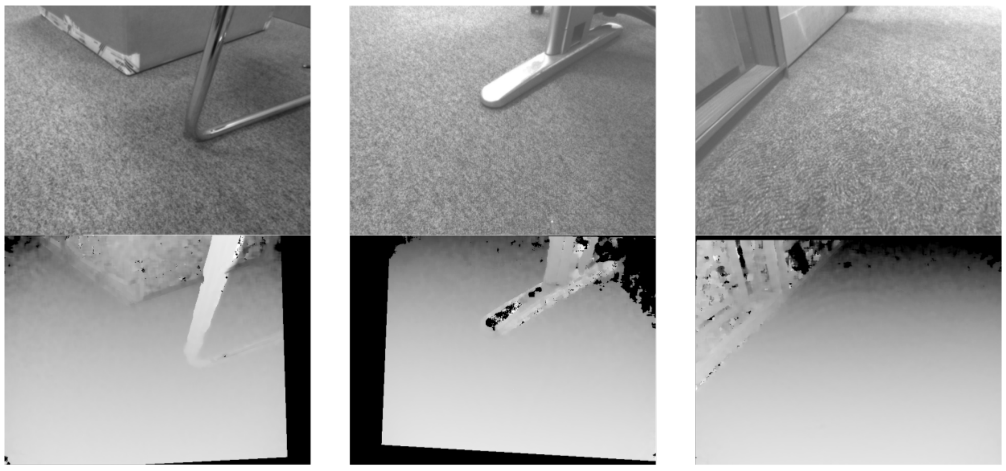


Figure 2.8: Examples of depth maps estimated using our method from the camera placed on a mobile robot.

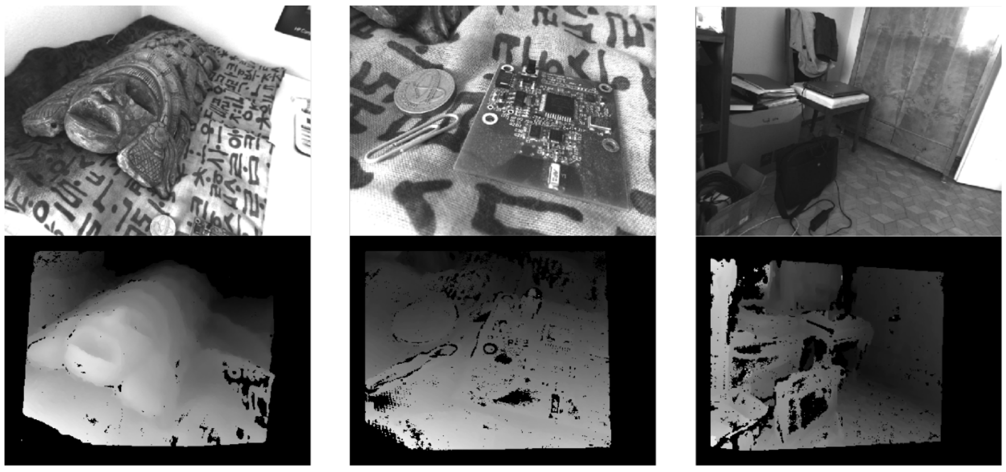


Figure 2.9: Our method can estimate depth maps in real-time also in more general settings, when paired with 6 DoF tracking such as ORB-SLAM.

Chapter 3

Dense Planar Visual Odometry

Contents

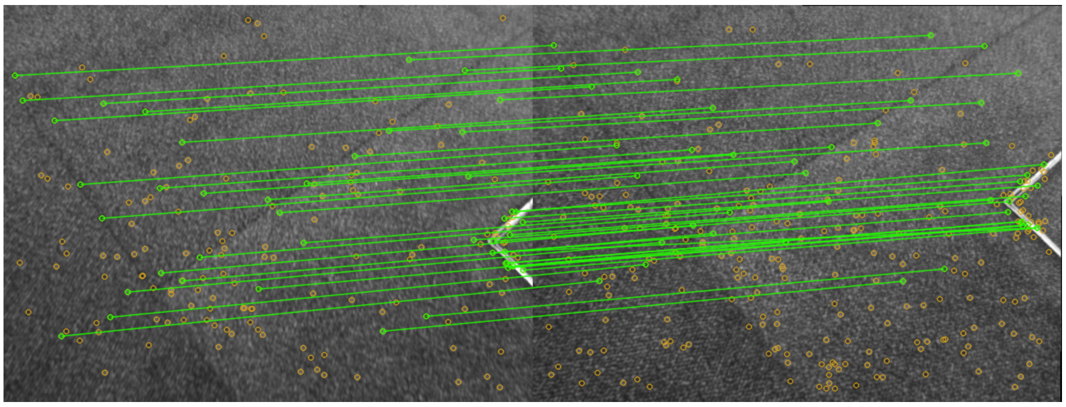
3.1 Introduction

Estimating its incremental ego-motion is one of the fundamental capabilities a mobile robot has to be equipped with, and classically this was performed using dead reckoning on the basis of wheel odometry. However, particularly in the case of light-weight, low-cost robots moving on unpredictable surfaces such as carpet or dirt, wheel odometry can suffer from lack of robustness (*e.g.* due to wheel slippage) or accuracy (tracking provided by wheel odometry is not accurate enough to perform multi-view depth estimation we described in the previous chapter). Visual odometry (VO), *i.e.* ego-motion estimation based on camera tracking, can jump over many of these well-known problems, and it is the first element of the dense perception system we have developed.

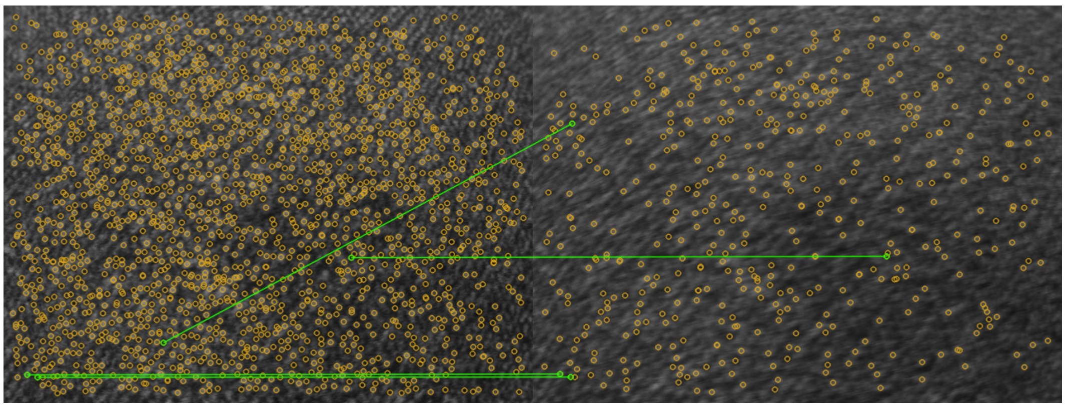
Most VO approaches rely on a feature extraction and mapping pipeline and reconstruct and track the world in full 3D, which makes them heavyweight, complicated and often fragile. In fact, in our experiments we found that in our settings when the camera is looking downwards at floor, standard feature-based methods struggle in a lot of situations (Fig. 3.1) due to the highly repetitive texture observed on many floor and carpet surfaces, or due to motion blur that can easily occur as the camera is very close to the surface. In our approach we take direct advantage of domain knowledge that wheeled mobile robots move on globally or locally planar surfaces, and estimate motion by tracking the natural floor texture which moves past the camera using whole image alignment. Almost every surface, even those which are apparently quite blank and do not lead to the extraction of standard point features, has trackable texture when used in whole image alignment. Furthermore, dense image alignment allows *all* of the surface texture available to contribute to the motion estimate, and results in a simple and efficient tracking system that is well suited for implementation on parallel processors like GPUs. Our method assumes knowledge of the extrinsic pose of the camera relative to the robot frame and performs image alignment directly with respect to the three degrees of freedom of planar robot motion. In Chapter 4 we present an algorithm that solves the necessary extrinsics auto-calibration problem with a one-shot, infrastructure-free method.

The visual odometry method presented here is inspired by [Lovegrove et al., 2011] who showed the power of motion estimation via iterative dense alignment in the specific application of on-road vehicle motion estimation from a rear parking camera by observing the planar road surface. We adopt the core approach of that work, but bring it to the domain of a small indoor robot which drives and rotates rapidly over a variety of real-world surfaces with different texture characteristics. Importantly, unlike the manual extrinsic calibration used by [Lovegrove et al., 2011], in the next chapter we show that our planar VO system can be rapidly auto-calibrated from a short sequence without the need for any special markers or targets or manual measurements.

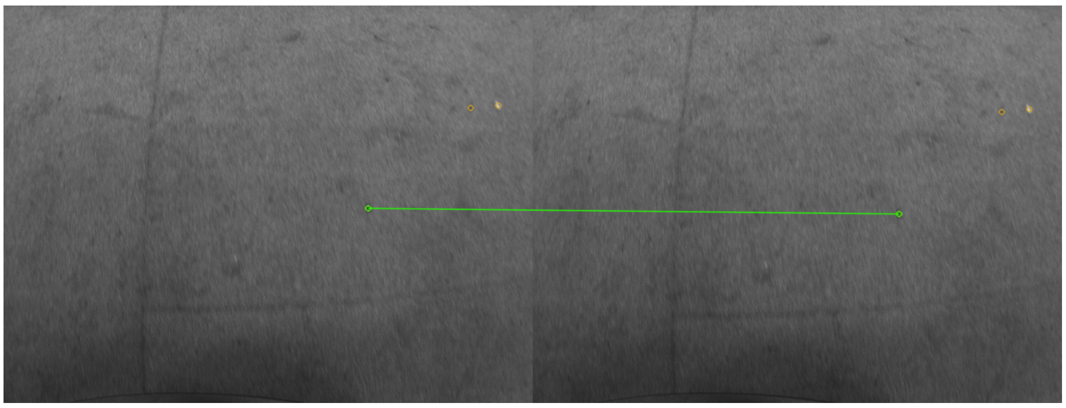
We start by presenting relevant work in the field of visual odometry. Next, we



(a) It is common to perform feature tracking and mapping between consecutive frames in order to estimate incremental camera motion.



(b) However, feature tracking and mapping tend to fail due to image degradation (*e.g.* motion blur).



(c) Methods based on feature tracking and mapping also perform poorly in situations where there is highly repetitive texture or there is a lack of texture.

Figure 3.1: Feature tracking and mapping is not suitable for motion estimation in our settings.

introduce general concepts related to dense image alignment which are then used to derive an efficient method for our planar visual odometry. In Section 3.6 we demonstrate experimentally that unbiased and robust VO is obtained from our system over the full range of dynamics of our experimental robot's motion. Please note that additional experiments on VO are also presented in Chapter 4 in the context of auto-calibration, where we also investigate how violation of our basic assumptions (planar motion and scene) affects the performance of our visual odometry. We conclude this chapter with a discussion and suggestions for improvements.

3.2 Related Work

The term *visual odometry* [Nistér et al., 2004] identifies the important class of problems where accurate but purely incremental motion estimation can usefully be provided by a camera system. This is in contrast to more general visual SLAM systems (e.g. [Davison, 2003]) of the time, aiming at drift-free localisation but with more restrictions on local accuracy and scale of operation.

There have been a number of notable Visual SLAM and VO systems which track general 6 DoF motion, some of them based on stereo vision (*e.g.* the incremental components of [Konolige et al., 2007; Mei et al., 2009]), whereas others only on a single camera *e.g.* PTAM [Klein and Murray, 2007]. The methods for monocular 6 DoF motion tracking are now well understood and established, and state-of-the-art systems include Semi-dense VO [Engel et al., 2013], SVO [Forster et al., 2014] and ORB-SLAM [Mur-Artal et al., 2015]. Instead of using a standard feature detection and matching pipeline, common to most of the above-mentioned SLAM and VO systems, dense and direct approaches track camera motion by registering consecutive images and minimising the photometric cost. Since direct tracking in general requires access to a depth map for every frame, methods based on image alignment were mainly developed for RGB-D cameras [Steinbrücker et al., 2011], stereo [Comport et al., 2007], or where depth is implicit [Lovegrove and Davison, 2010]. A system that first demonstrated dense and direct tracking (and mapping) using a monocular camera was presented in [Newcombe et al., 2011b].

The methods described above are general SLAM and VO systems that are designed to perform tracking an mapping in arbitrary scenes and conditions. VO becomes easier if domain assumptions can be brought strongly into play, and many authors have considered the special case of planar robot motion over a ground surface. As in our work, [Campbell et al., 2005] demonstrated how a single camera can track floor texture in different situations, but using optical flow computation and with a partial forward-looking view to estimate orientation. [Kitt et al., 2011] argued convincingly for taking advantage of prior knowledge that a camera is viewing a planar floor in their system using feature correspondences. [Nourani-Vatani et al., 2009] used a downward, fronto-parallel looking camera and correlation of patches. There are also a wide range of homography-based methods that estimate camera ego-motion using the assumption of a planar scene, *e.g.* [Pirchheim and Reitmayr, 2011; Saurer et al., 2012; Adams et al., 2002]. These methods typically determine the homography by detecting corresponding visual features in two images to the same plane, and compute the camera motion by decomposing the homography matrix [Faugeras and Lustman, 1988]. However, none of these authors went all the way in making the best possible use of the planar scene assumption. Used properly it allows *all the pixels* in a video sequence to contribute to motion estimation via iterative dense alignment. [Lovegrove et al., 2011] showed the power of this approach in the specific application of on-road vehicle motion estimation from a rear parking camera by observing the planar road surface.

3.3 Tracking using Full Image Alignment

We start by explaining the basic principles of tracking using whole image alignment, a method which evolved from the iterative technique introduced by [Lucas and Kanade, 1981]. The key to the Lucas-Kanade approach is to consider tracking as image alignment over some continuous space of possible transformations. In order to achieve this, we define a cost function that measures how well two images are aligned. We then iteratively compute the derivative of the cost function with respect to the parameters at a current estimate and take a linear step towards a minimum via gradient descent.

More specifically, given two images, live $\mathcal{I}^l$ and reference $\mathcal{I}^r$, we are looking for a transformation $\mathtt{T}^{lr} \in \mathbf{SE}(3)$ that describes the camera motion between them and allows us to register (align) them. We parameterise $\mathtt{T}^{lr}(\mathbf{x})$ by a vector $\mathbf{x}$ belonging to the Lie Algebra $\mathfrak{se}(3)$. Since dense image alignment is an optimisation problem, we define an energy function F which measures the discrepancy across the whole image in the form of the sum of squared differences between all pairs of superposed pixels:

$$F(\mathbf{x}) = \frac{1}{2}\|f(\mathbf{x})\|_2^2 \ , \tag{3.1}$$

where $f(\mathbf{x})$ is the per-pixel error term:

$$f(\mathbf{x}) = \mathcal{I}^l(\pi(\mathtt{K}\mathtt{T}^{lr}(\mathbf{x})d_r\mathtt{K}^{-1}\dot{\mathbf{p}}_r)) - \mathcal{I}^r(\mathbf{p}_r) \ . \tag{3.2}$$

Here $\mathbf{p}_r$ denotes a pixel coordinate in the reference image, and (as explained in Chapter 2) the operation $\pi(\mathtt{K}\mathtt{T}^{lr}(\mathbf{x})d_r\mathtt{K}^{-1}\mathbf{p}_r)$ transforms a pixel in the reference image via the current transformation estimate into the live image. In order to perform this back-projection, the depth value d_r of the pixel is required, so in principle, dense image alignment requires access to a depth measurement for at least one of the images. When using a depth camera, the depth map is explicitly given, and for now, will assume this is the case. Later, when deriving our planar visual odometry, we will show that depth can be induced from the plane the camera is observing. As described in Chapter 2, we also assume that the camera intrinsics $\mathtt{K}$ are known, which leaves us with only one unknown element in Eqs. (3.1) and (3.2), the camera motion $\mathtt{T}^{lr}$, parameterised by $\mathbf{x}$.

Note, that instead of using the $l2$-norm as indicated in Eq. (3.1), in practice we employ a robust cost function (*e.g.* Huber norm) in an iteratively reweighted least squares framework to reduce the sensitivity of our algorithms to outliers in the data and other, unmodelled effects. However, for the sake of brevity and clarity, we will omit it in our derivation.

In dense image alignment we do not have to explicitly solve the data association problem as in feature-based methods. Instead, pixel correspondence is given by the current estimate of the camera motion, and we improve this estimate by iteratively minimising the cost function:

$$\arg\min_{\mathbf{x}\in\mathfrak{se}(3)} F(\mathbf{x}) = \arg\min_{\mathbf{x}\in\mathfrak{se}(3)} \frac{1}{2}\|f(\mathbf{x})\|_2^2 \ . \tag{3.3}$$

The objective function can be easily visualised. Fig. 3.2 shows an example of a live and reference image, with a plot of the cost function as we align the images along the x-axis. A plot of the cost function with respect to both translational degrees of freedom of the robot can be seen in Fig. 3.3. Obviously, this generalises to all degrees of freedom of the considered motion. We see a smooth function with clearly

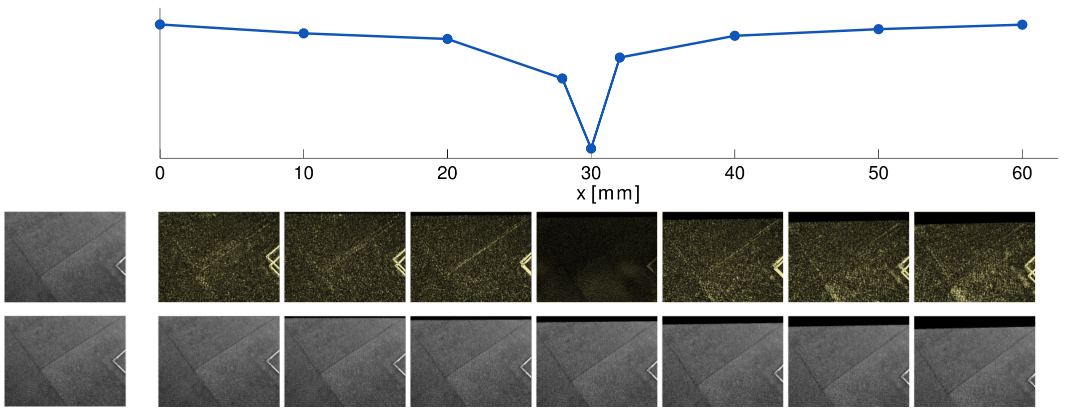


Figure 3.2: When performing dense image alignment, we do not have to explicitly solve the data association problem, but start with some initial estimate of the camera motion and refine it iteratively by minimising a cost function that measures how well two images align. Here we show the effect of aligning two images by sliding one of them along the x-axis. There is a clear and distinctive x value that best registers the images.

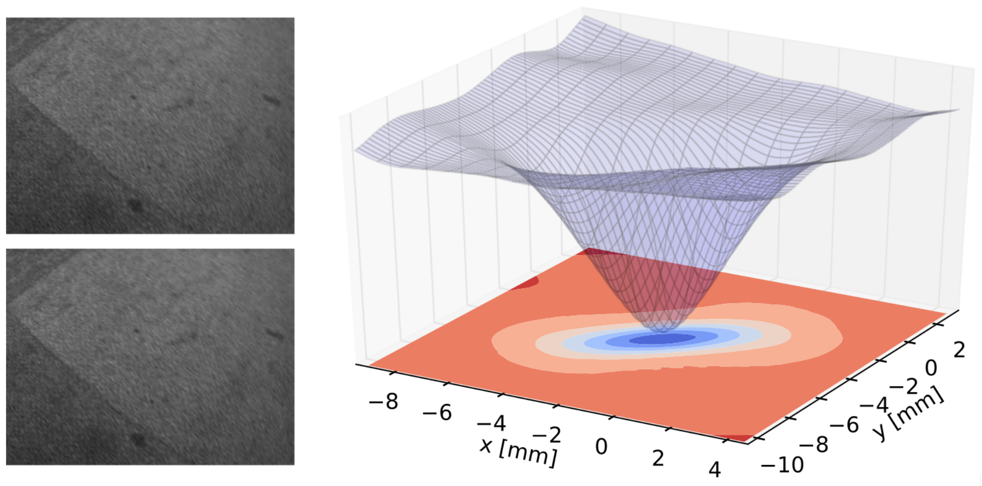


Figure 3.3: Another example of a typical dense tracking cost function. The function depends on all degrees of freedom of camera motion, but here only a 2D translational slice of motion in the x and y directions is shown.

distinctive minima, although the basis of convergence is relatively narrow. However, starting from a good initial estimate (and using other techniques like coarse-to-fine), we can efficiently perform the minimisation of the objective function, as we will describe below.

We now describe our strategy for solving the optimisation problem Eq. (3.3). We start by performing a small re-parameterisation of the original problem. Given a current estimate of the solution $\hat{\mathtt{T}}^{lr}$, we do not parametrise the cost function by $\mathtt{T}^{lr}(\mathbf{x})$, but instead by $\mathtt{T}(\mathbf{x})$, an update matrix that represents small changes to the current estimate of the solution $\hat{\mathtt{T}}^{lr}$ in the following way:

$$f(\mathbf{x}) = \mathcal{I}^l(\pi(\mathtt{K}\hat{\mathtt{T}}^{lr}\mathtt{T}(\mathbf{x})d_r\mathtt{K}^{-1}\dot{\mathbf{p}}_r)) - \mathcal{I}^r(\mathbf{p}_r) \ . \tag{3.4}$$

This allows us to simplify the calculations of the derivative of the cost function, and later on, use an efficient variant of dense image alignment (Efficient Second-Order Minimisation, ESM).

Solving the optimisation problem in Eq. (3.3) iteratively means that at each iteration we find a small update $\hat{\mathbf{x}}$ to our current estimate via the expression (with slight abuse of notation):

$$\hat{\mathbf{x}} = -(\mathtt{J}^\top\mathtt{J})^{-1}\mathtt{J}^\top f(\mathbf{0}) \ , \tag{3.5}$$

where:

$$\mathtt{J} = \boldsymbol{\nabla} f(\mathbf{0}) \ , \tag{3.6}$$

and update $\hat{\mathtt{T}}^{lr}$ using the following rule:

$$\hat{\mathtt{T}}^{lr} \leftarrow \hat{\mathtt{T}}^{lr}\mathtt{T}(\hat{\mathbf{x}}) \ . \tag{3.7}$$

The Jacobian of the cost function, Eq. (3.6), required in the optimisation, can be easily assembled from the derivatives of the individual per-pixel error terms. Using the chain rule we can find the general expression, which is given by:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \left.\frac{\partial \mathcal{I}^l(\mathbf{a})}{\partial \mathbf{a}}\right|_{\mathbf{a}=\pi(\hat{\mathbf{p}}_l)} \left.\frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\right|_{\mathbf{b}=\hat{\mathbf{p}}_l} \mathtt{K}\hat{\mathtt{T}}^{lr}\frac{\partial \mathtt{T}(\mathbf{x})}{\partial x_i} d_r\mathtt{K}^{-1}\dot{\mathbf{p}}_r \ , \tag{3.8}$$

where we wrote $\hat{\mathbf{p}}_l = \mathtt{K}\hat{\mathtt{T}}^{lr}\mathtt{T}(\mathbf{x})d_r\mathtt{K}^{-1}\dot{\mathbf{p}}_r$ for brevity. As evident, the derivative depends on the gradient in the live image, $\frac{\partial \mathcal{I}^l(\mathbf{a})}{\partial \mathbf{a}}$, as well as the derivative of the homogeneous projection operator π, which is straightforward to compute, namely:

$$\frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}} = \frac{\partial \begin{pmatrix} a/c \\ b/c \end{pmatrix}}{\partial \begin{pmatrix} a \\ b \\ c \end{pmatrix}} \tag{3.9}$$

$$= \begin{bmatrix} \frac{\partial a/c}{\partial a} & \frac{\partial a/c}{\partial b} & \frac{\partial a/c}{\partial c} \\ \frac{\partial b/c}{\partial a} & \frac{\partial b/c}{\partial b} & \frac{\partial b/c}{\partial c} \end{bmatrix} = \begin{bmatrix} 1/c & 0 & -a/c^2 \\ 0 & 1/c & -b/c^2 \end{bmatrix} . \tag{3.10}$$

The last element of the formula is the derivative of the transformation $\frac{\partial \mathtt{T}(\mathbf{x})}{\partial x_i}$ with respect to its individual parameters x_i. Thanks to our parameterisation and update rule, Eq. (3.7), we only need to evaluate the gradient at $\mathbf{x} = \mathbf{0}$ (Eq. (3.6)), and this is given by:

$$\left.\frac{\partial \mathtt{T}(\mathbf{x})}{\partial x_i}\right|_{\mathbf{x}=\mathbf{0}} = \mathtt{G}_i^{\mathbf{SE}(3)} , \tag{3.11}$$

where $\mathtt{G}_i^{\mathbf{SE}(3)}$ is the ith generator of the $\mathbf{SE}(3)$ group.

The procedure described above corresponds to the most classical, forward compositional approach to the Lucas-Kanade dense image alignment method. Other, more efficient approaches have also been presented, *e.g.* the inverse-compositional method [Baker and Matthews, 2004]. For a more detailed description we refer to [Lovegrove, 2011]. The main observation here is that the gradient of the per-pixel cost function can easily be computed using the image gradient, and that the computation of the Jacobian is highly parallelisable, as the calculations in Eq. (3.8) are performed for each pixel independently.

3.4 Tracking with a Plane Induced Homography

The fact that we need a depth value d_r for each pixel in order to perform dense image alignment, would seem to prevent us from using this method in conjunction with a monocular camera. However, we will show now that using the assumption that a camera observes a plane, with known orientation of the camera with respect to that plane, means we can still perform dense image alignment. Note that this is exactly the robot configuration we defined in Chapter 2, where we assumed that the camera is fixed on a robot and it is pointed towards the floor, and that the robot moves on a flat surface.

Let us consider a plane $(\mathbf{n}^\top, d_c)$ defined in the camera frame of reference, where $\mathbf{n}$ is the unit vector normal to the plane and d_c is the perpendicular distance from the plane to the origin. Given $\mathbf{n}$ and d_c we can calculate a depth value d for each pixel (u, v) as follows:

$$d(u, v) = \frac{-d_c}{[u \quad v \quad 1]K^{-T}\mathbf{n}} . \tag{3.12}$$

In the simplest case, when the camera is observing the plane from a fronto-parallel view, we have $\mathbf{n} = (0,0,1)^\top$. When the camera orientation with respect to the ground plane is denoted by $\mathtt{R}^{vc}$, the normal vector is $\mathbf{n} = \mathtt{R}^{cv}(0,0,1)^\top$.

This concept can be used to derive a homography, which says that two images (live $\mathcal{I}^l$ and reference $\mathcal{I}^r$) of the same plane are related by a plane-induced homography $\mathtt{H}^{lr}$. This homography transforms pixel coordinates in the reference image $\mathbf{p}_r$ into the live image $\mathbf{p}_l$:

$$\mathbf{p}_l = \pi\left(\mathtt{H}^{lr}\dot{\mathbf{p}}_r\right) \ , \tag{3.13}$$

and depends on the camera motion $\mathtt{T}^{lr}$ and the parameters $(\mathbf{n}^\top, d_c)$ of the plane defined in the camera reference frame:

$$\mathtt{H}^{lr} = \mathtt{K}\mathtt{T}^{lr}(I \mid -\mathbf{n}_{d_c})^\top\mathtt{K}^{-1} \ , \tag{3.14}$$

where $\mathbf{n}_{d_c} = \frac{\mathbf{n}}{d_c}$.

In dense tracking based on a plane induced homography our goal is essentially the same as described in the previous section: we are looking for a motion $\mathbf{x}$ that best registers $\mathcal{I}^l$ and $\mathcal{I}^r$, but no longer need the per-pixel depth values d_r, as they are now induced by the plane. The per-pixel error term therefore becomes:

$$f(\mathbf{x}) = \mathcal{I}^l(\pi(\mathtt{H}^{lr}(\mathbf{x})\dot{\mathbf{p}}_r)) - \mathcal{I}^r(\mathbf{p}_r) \ , \tag{3.15}$$

with:

$$\mathtt{H}^{lr}(\mathbf{x}) = \mathtt{K}\hat{\mathtt{T}}^{lr}\mathtt{T}(\mathbf{x})(I| - \mathbf{n}_{d_c})^\top\mathtt{K}^{-1} \ . \tag{3.16}$$

The computations of the derivatives change only slightly to:

$$\left.\frac{\partial f(\mathbf{x})}{\partial x_i}\right|_{\mathbf{x}=\mathbf{0}} = \left.\frac{\partial \mathcal{I}^l(\mathbf{a})}{\partial \mathbf{a}}\right|_{\mathbf{a}=\pi(\hat{\mathtt{H}}^{lr}\dot{\mathbf{p}}_r)} \left.\frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\right|_{\mathbf{b}=\hat{\mathtt{H}}^{lr}\dot{\mathbf{p}}_r} \left.\frac{\partial \mathtt{H}^{lr}(\mathbf{x})}{\partial x_i}\right|_{\mathbf{x}=\mathbf{0}} \dot{\mathbf{p}}_r \ , \tag{3.17}$$

where:

$$\hat{\mathtt{H}}^{lr} = \mathtt{K}\hat{\mathtt{T}}^{lr}(I| - \mathbf{n}_{d_c})^\top\mathtt{K}^{-1} \ , \tag{3.18}$$

and:

$$\frac{\partial \mathtt{H}^{lr}(\mathbf{x})}{\partial x_i} = \mathtt{K}\hat{\mathtt{T}}^{lr}\frac{\partial \mathtt{T}(\mathbf{x})}{\partial x_i}(I| - \mathbf{n}_{d_c})^\top\mathtt{K}^{-1} \ . \tag{3.19}$$

Again, we can use the gradient of the image and derivative of the projection operator as well as $\mathtt{G}_i^{\mathbf{SE}(3)}$, the generators of the $\mathbf{SE}(3)$ group, to calculate the derivatives of the per-pixel error terms and the whole cost function.

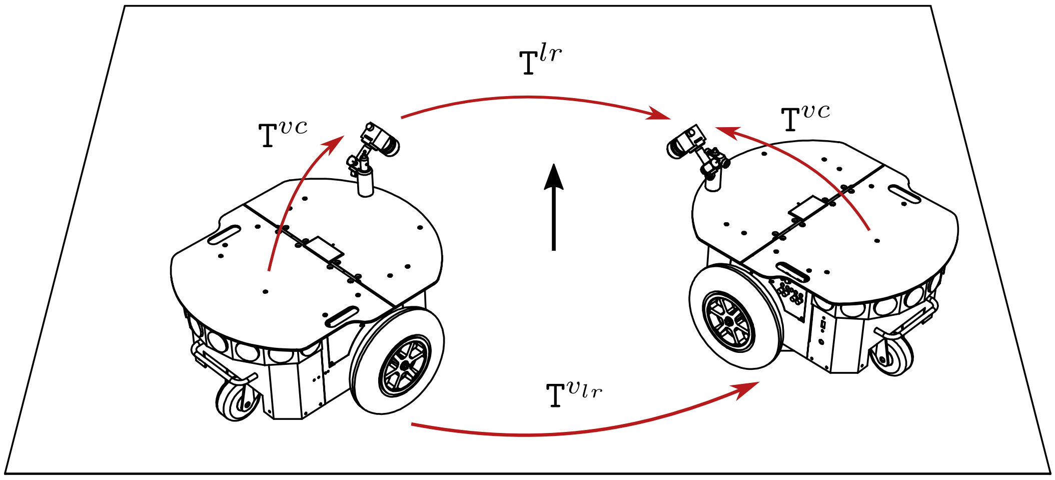


Figure 3.4: We can parameterise the 3D camera motion, $\mathtt{T}^{lr}$ using 2D robot motion, $\mathtt{T}^{v_{lr}}$, and a known transformation from the camera frame of reference to the robot frame of reference $\mathtt{T}^{vc}$.

3.5 Planar Visual Odometry

3.5.1 Image Alignment Parameterised by Planar Motion

We will now use the concepts presented in the previous sections to derive our dense planar visual odometry method for a mobile robot. Recall the typical robot-camera configuration we consider in this thesis: a camera is bolted to the robot in an arbitrary but fixed location, such that its field of view is filled with the planar floor surface. We can distinguish two main frames of references in this set-up: one associated with the camera and another with the robot. We will denote $\mathtt{T}^{vc}$ as the transformation from the camera frame of reference to the robot frame of reference, whereas $\mathtt{T}^{cv} = (\mathtt{T}^{vc})^{-1}$ is its inverse. We define $\mathtt{T}^{vc} = (\mathtt{R}^{vc} \mid \mathbf{t}^{vc})$, where $\mathtt{R}^{vc} \in \mathbf{SO}(3)$. We will refer to the position of the camera in the robot frame of reference by $\mathbf{t}^{vc} = (x_{vc}, y_{vc}, z_{vc})^\top$, and its orientation $\mathtt{R}^{vc}$ using roll, pitch and yaw angles (denoted by α_{vc}, β_{vc}, and γ_{vc} respectively). We will refer to $\mathtt{T}^{vc}$ as camera extrinsics and at this stage we assume that they are known. Our method for auto-calibrating camera extrinsics will be presented in Chapter 4.

Using the notation from the previous sections, $\mathtt{T}^{lr}$ will denote the transformation matrix describing the relative motion of the camera between time-steps when the camera captures two images, a live image $\mathcal{I}^l$ and an overlapping earlier reference image $\mathcal{I}^r$. Even though $\mathtt{T}^{lr}$ is a 3D motion in the camera frame of reference, the observation that the camera is fixed on the robot and therefore is moving parallel to the ground plane allows us to parameterise 3D camera motion using relative 2D robot motion $\mathtt{T}^{v_{lr}}$ in the following way:

$$\mathtt{T}^{lr} = \mathtt{T}^{cv}\mathtt{T}^{v_{lr}}\mathtt{T}^{vc} \ . \tag{3.20}$$

The essential geometry of this problem is shown in Fig. 3.4.

Since the camera pose $\mathtt{T}^{vc}$ as well as $\mathbf{n}_{d_c}$ remain constant (in fact $\mathbf{n}_{d_c}$ is uniquely determined by $\mathtt{T}^{vc}$), and the robot moves on a plane, we only need to determine 3 degrees of freedom of the robot planar motion $\mathbf{x} = (x, y, \theta)^T$. Although $\mathtt{T}^{v_{lr}}$ is in $\mathbf{SE}(2)$ we can refer to it in 3D by raising it into $\mathbf{SE}(3)$ and enforcing that the robot moves on the xy plane:

$$\mathtt{T}^{v_{lr}}(\mathbf{x}) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & x \\ \sin\theta & \cos\theta & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{3.21}$$

Now our goal is to find a homography $\mathtt{H}^{lr}$ parameterised by vehicle planar motion $\mathtt{T}^{v_{lr}}(\mathbf{x})$:

$$\mathtt{H}^{lr}(\mathbf{x}) = \mathtt{K}\mathtt{T}^{cv}\mathtt{T}^{v_{lr}}(\mathbf{x})\mathtt{T}^{vc}(I| - \mathbf{n}_{d_c})^\top\mathtt{K}^{-1} \ . \tag{3.22}$$

that best registers $\mathcal{I}^l$ and $\mathcal{I}^r$. To achieve this, we will perform all of the already familiar steps: we define an energy function F which measures the discrepancy across the whole image in the form of the sum of squared differences between all pairs of superposed pixels:

$$F(\mathbf{x}) = \frac{1}{2}\|f(\mathbf{x})\|_2^2 \ , \tag{3.23}$$

with:

$$f(\mathbf{x}) = \mathcal{I}^l(\pi(\mathtt{H}^{lr}(\mathbf{x})\dot{\mathbf{p}}_r)) - \mathcal{I}^r(\mathbf{p}_r) \ . \tag{3.24}$$

The homography $\mathtt{H}^{lr}$ will be parameterised with $\mathtt{T}(\mathbf{x})$, an update matrix that represents small changes to the current estimate of the solution $\hat{\mathtt{T}}^{v_{lr}}$, in the following form:

$$\mathtt{H}^{lr}(\mathbf{x}) = \mathtt{K}\mathtt{T}^{cv}\hat{\mathtt{T}}^{v_{lr}}\mathtt{T}(\mathbf{x})\mathtt{T}^{vc}(I| - \mathbf{n}_{d_c})^\top\mathtt{K}^{-1} \ . \tag{3.25}$$

The vector $\mathbf{x}$ now belongs to Lie Algebra $\mathfrak{se}(2)$, and we iteratively apply the following rule:

$$\hat{\mathtt{T}}^{v_{lr}} \leftarrow \hat{\mathtt{T}}^{v_{lr}}\mathtt{T}(\hat{\mathbf{x}}) \ , \tag{3.26}$$

in order to update our estimate of the robot motion.

The general expression of the gradient of the per-pixel error terms is given by:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \frac{\partial \mathcal{I}^l(\mathbf{a})}{\partial \mathbf{a}}\bigg|_{\mathbf{a}=\pi(\mathtt{H}^{lr}(\mathbf{x})\dot{\mathbf{p}}_r)} \frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\bigg|_{\mathbf{b}=\mathtt{H}^{lr}(\mathbf{x})\dot{\mathbf{p}}_r} \frac{\partial \mathtt{H}^{lr}(\mathbf{x})}{\partial x_i}\dot{\mathbf{p}}_r \tag{3.27}$$

$$\frac{\partial \mathtt{H}^{lr}(\mathbf{x})}{\partial x_i} = \mathtt{K}\mathtt{T}^{cv}\hat{\mathtt{T}}^{v_{lr}} \frac{\partial \mathtt{T}(\mathbf{x})}{\partial x_i}\mathtt{T}^{vc}(I| - \mathbf{n}_{d_c})^\top \mathtt{K}^{-1} \ . \tag{3.28}$$

3.5.2 Efficient Second-order Minimisation (ESM)

We could use the standard forward compositional minimisation scheme as presented in Section 3.3, but instead, in order to minimise the objective function in Eq. (3.23), we perform an iterative optimisation based on the Efficient Second-order Minimisation (ESM) method [Malis, 2004; Mei et al., 2008]. ESM improves on Lucas and Kanade's original technique [Lucas and Kanade, 1981] by allowing us to minimise a second-order approximation of the objective function using only first-order derivatives, and therefore, as observed by many authors *e.g.* [Lovegrove et al., 2011; Klose et al., 2013; Engel et al., 2014], in general achieves better convergence rates. Most of the steps involved in the optimisation are similar to a standard iterative optimisation, *i.e.* at each iteration we find an updated estimate $\hat{\mathbf{x}}$ of the state via the expression:

$$\hat{\mathbf{x}} = -(\mathtt{J}^\top\mathtt{J})^{-1}\mathtt{J}^\top f(\mathbf{0}) \ . \tag{3.29}$$

However, ESM computes a second-order approximation of the objective function from only first-order derivatives. Therefore now we use:

$$\mathtt{J} = \frac{1}{2}\left(\boldsymbol{\nabla} f(\mathbf{0}) + \boldsymbol{\nabla} f(\hat{\mathbf{x}})\right) \ , \tag{3.30}$$

which means that this approach requires us to evaluate the partial derivatives of the cost function $f(\mathbf{x})$ at $\mathbf{0}$ and at the solution $\hat{\mathbf{x}}$. Using the general expression of the derivative of the per-pixel error terms from Eq. (3.27), we can easily calculate the derivative at $\mathbf{0}$:

$$\frac{\partial f(\mathbf{x})}{\partial x_i}\bigg|_{\mathbf{x}=\mathbf{0}} = \frac{\partial \mathcal{I}^l(\mathbf{a})}{\partial \mathbf{a}}\bigg|_{\mathbf{a}=\pi(\hat{\mathtt{H}}^{lr}\dot{\mathbf{p}}_r)} \frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\bigg|_{\mathbf{b}=\hat{\mathtt{H}}^{lr}\dot{\mathbf{p}}_r} \frac{\partial \mathtt{H}^{lr}(\mathbf{x})}{\partial x_i}\bigg|_{\mathbf{x}=\mathbf{0}}\dot{\mathbf{p}}_r \ , \tag{3.31}$$

where:

$$\hat{\mathtt{H}}^{lr} = \mathtt{K}\mathtt{T}^{cv}\hat{\mathtt{T}}^{v_{lr}}\mathtt{T}^{vc}(I| - \mathbf{n}_{d_c})^\top \mathtt{K}^{-1} \ , \tag{3.32}$$

$$\frac{\partial \mathtt{H}^{lr}(\mathbf{x})}{\partial x_i}\bigg|_{\mathbf{x}=\mathbf{0}} = \mathtt{K}\mathtt{T}^{cv}\hat{\mathtt{T}}^{v_{lr}} \frac{\partial \mathtt{T}(\mathbf{x})}{\partial x_i}\bigg|_{\mathbf{x}=\mathbf{0}} \mathtt{T}^{vc}(I| - \mathbf{n}_{d_c})^\top \mathtt{K}^{-1} \ , \tag{3.33}$$

and $\left.\frac{\partial \mathtt{T}(\mathbf{x})}{\partial x_i}\right|_{\mathbf{x}=\mathbf{0}} = \mathtt{G}_i^{\mathbf{SE}(2)}$, the ith generator of the $\mathbf{SE}(2)$ group.

Calculating the derivatives at $\mathbf{x} = \hat{\mathbf{x}}$ may appear to be more difficult, as it essentially requires knowledge of the solution, we are searching for in the first place. Specifically, for $\mathbf{x} = \hat{\mathbf{x}}$, we need to compute:

$$\left.\frac{\partial f(\mathbf{x})}{\partial x_i}\right|_{\mathbf{x}=\hat{\mathbf{x}}} = \left.\frac{\partial \mathcal{I}^l(\mathbf{a})}{\partial \mathbf{a}}\right|_{\mathbf{a}=\pi(\mathtt{H}^{lr}(\hat{\mathbf{x}})\dot{\mathbf{p}}_r)} \left.\frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\right|_{\mathbf{b}=\mathtt{H}^{lr}(\hat{\mathbf{x}})\dot{\mathbf{p}}_r} \frac{\partial \mathtt{H}^{lr}(\hat{\mathbf{x}})}{\partial x_i}\dot{\mathbf{p}}_r \ . \tag{3.34}$$

In other words, we need to evaluate the gradient of the objective function at the solution $\hat{\mathbf{x}}$. Obviously, it is not possible to do that explicitly without knowledge of the solution, but we can approximate it using the following assumptions. First, we assume that we are close to the solution, hence $\hat{\mathbf{x}}$ is small. Since Lie Algebra $\mathfrak{se}(2)$ defines a locally invariant vector field tangential to the manifold of $\mathbf{SE}(2)$, this assumption allows us to treat $\frac{\partial \mathtt{H}^{lr}(\mathbf{0})}{\partial x_i}$ and $\frac{\partial \mathtt{H}^{lr}(\hat{\mathbf{x}})}{\partial x_i}$ as equivalent, *i.e.* :

$$\frac{\partial \mathtt{H}^{lr}(\mathbf{0})}{\partial x_i} \equiv \frac{\partial \mathtt{H}^{lr}(\hat{\mathbf{x}})}{\partial x_i} \ , \tag{3.35}$$

and the same holds for $\frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}$, namely that:

$$\left.\frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\right|_{\mathbf{b}=\mathtt{H}^{lr}(\hat{\mathbf{x}})\dot{\mathbf{p}}_r} \equiv \left.\frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\right|_{\mathbf{b}=\mathtt{H}^{lr}(\mathbf{0})\dot{\mathbf{p}}_r} \ . \tag{3.36}$$

However, we still need to calculate $\left.\frac{\partial \mathcal{I}^l(\mathbf{a})}{\partial \mathbf{a}}\right|_{\mathbf{a}=\pi(\mathtt{H}^{lr}(\hat{\mathbf{x}})\dot{\mathbf{p}}_r)}$. In ESM we use the observation that at the solution the warped live image should, in principle, equal the reference image. Assuming $\mathtt{H}^{lr}(\hat{\mathbf{x}}) = \hat{\mathtt{H}}^{lr}\mathtt{H}(\hat{\mathbf{x}})$ we can write:

$$\dot{\mathbf{p}}_l = \hat{\mathtt{H}}^{lr}\mathtt{H}(\hat{\mathbf{x}})\dot{\mathbf{p}}_r \ , \tag{3.37}$$

$$\dot{\mathbf{p}}_r = \mathtt{H}(-\hat{\mathbf{x}})\hat{\mathtt{H}}^{rl}\dot{\mathbf{p}}_l \ , \tag{3.38}$$

and therefore at the solution $\hat{\mathbf{x}}$, where the live image should in principle exactly equal the reference image, the following condition should also be satisfied:

$$\mathcal{I}^l(\mathbf{p}_l) = \mathcal{I}^r(\mathbf{p}_r) = \mathcal{I}^r(\pi(\mathtt{H}(-\hat{\mathbf{x}})\hat{\mathtt{H}}^{rl}\dot{\mathbf{p}}_l)) \ . \tag{3.39}$$

Using this assumption, we can express the image derivative part of the Eq. (3.34) as follows:

$$\left.\frac{\partial \mathcal{I}^l(\mathbf{p}_l)}{\partial \mathbf{p}_l}\right|_{\mathbf{p}_l=\pi(\hat{\mathtt{H}}^{lr}\mathtt{H}(\hat{\mathbf{x}})\dot{\mathbf{p}}_r)} = \left.\frac{\partial \mathcal{I}^r(\mathbf{a})}{\partial \mathbf{a}}\right|_{\mathbf{a}=\mathbf{p}_r} \left.\frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\right|_{\mathbf{b}=\dot{\mathbf{p}}_r} \mathtt{H}(-\hat{\mathbf{x}})\hat{\mathtt{H}}^{rl}\frac{\partial \dot{\mathbf{p}}_l}{\partial \mathbf{p}_l} \ , \tag{3.40}$$

with $\dot{\mathbf{p}}_r = \mathtt{H}(-\hat{\mathbf{x}})\hat{\mathtt{H}}^{rl}\dot{\mathbf{p}}_l$ as stated above. Eq. (3.40) still depends on $\hat{\mathbf{x}}$ but now less significantly. Once again we will assume that $\hat{\mathbf{x}}$ is small and will evaluate it to $\mathbf{0}$. Substituting $\mathbf{p}_r = (u_r, v_r)^\top$, $\dot{\mathbf{p}}_r = (u_r, v_r, 1)^\top$, and by using

$$\frac{\partial \dot{\mathbf{p}}_l}{\partial \mathbf{p}_l} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \tag{3.41}$$

Eq. (3.40) simplifies to:

$$\left.\frac{\partial \mathcal{I}^l(\mathbf{p}_l)}{\partial \mathbf{p}_l}\right|_{\mathbf{p}_l = \pi(\hat{\mathtt{H}}^{lr}\mathtt{H}(\hat{\mathbf{x}})\dot{\mathbf{p}}_r)} = \left.\frac{\partial \mathcal{I}^r(\mathbf{a})}{\partial \mathbf{a}}\right|_{\mathbf{a}=\mathbf{p}_r} \begin{bmatrix} 1 & 0 & -u_r \\ 0 & 1 & -v_r \end{bmatrix} \hat{\mathtt{H}}^{rl} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}. \tag{3.42}$$

As evident, in the Efficient Second-order Method we use several assumptions and the gradient of both the live and reference image during the minimisation process to build a second-order approximation of the objective function. Although ESM involves more computation than the first order methods, *e.g.* the inverse compositional approach [Baker and Matthews, 2004], thanks to the second-order approximation, in general, it achieves better convergence properties as documented in [Benhimane and Malis, 2004].

3.5.3 Practical Considerations and Implementation Details

Our dense planar visual odometry assumes that the camera observes a planar floor, however in most cases this condition cannot be satisfied. In order to make our approach more practical, we only perform tracking on the pixels that are identified as ‘floor’. To achieve this, when the camera starts moving, we first assume that only a region directly in front of the robot is planar and use this for tracking. This allows us to perform initial motion estimation accurate enough for calculation of the depth maps (Chapter 2). With the depth maps we can start building a model of the scene and label parts of it as ‘floor’ and ‘non-floor’, as for example shown in Chapter 6, or build an appearance model of the floor and perform tracking only on the pixels that are classified to satisfy the planar assumptions.

There are several well-established techniques (*e.g.* robusitfication of the cost function, coarse-to-fine) that help to make tracking using dense image alignment more robust and practical. For a comprehensive review of these refer to some of these excellent references: [Lovegrove, 2011; Newcombe, 2012; Handa, 2013]. As

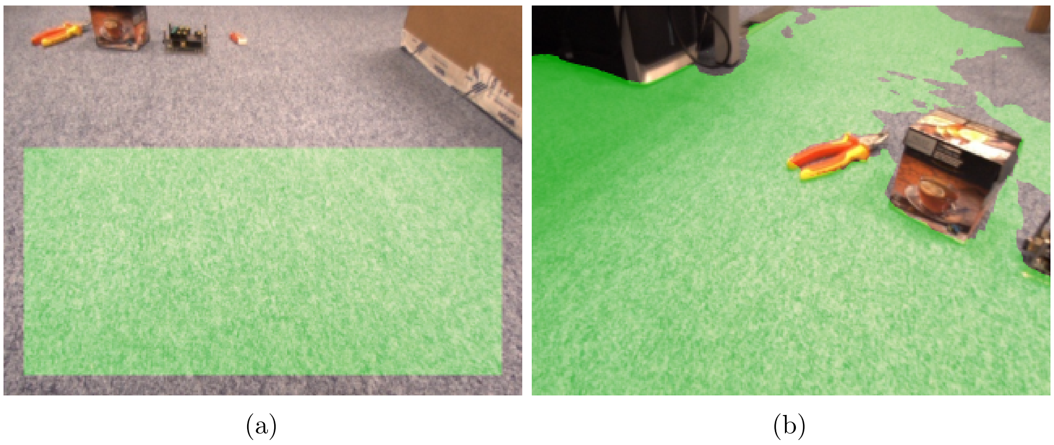


Figure 3.5: Our method does not require that all pixels in the camera field of view belong to the planar floor, and in practice we perform dense image alignment on the image parts that are confidently identified as a planar floor. We start by assuming that only the area directly in front of the robot is planar (a). This is sufficient to perform accurate motion estimation, and allows us to estimate depth maps (Chapter 2) that next can be used to classify pixels into 'floor' and 'non-floor' (b), by for example building appropriate appearance models [Dahlkamp et al., 2006; Hadsell et al., 2009] or performing a surface reconstruction as shown in Chapter 6.

already mentioned, in our implementation, instead of using the $l2$-norm as indicated in *e.g.* Eqs. (3.1) and (3.23), we use a Huber norm to robustify the cost function in an iteratively reweighted least squares framework. This helps us to reject partially nonplanar structures still visible in the field of view as well as other effects that are not directly modelled in our approach, *e.g.* specularities and shadows. Furthermore, we make use of a coarse-to-fine multi-resolution approach to speed up computations and extend the basin of convergence.

Since tracking using image alignment is a nonlinear optimisation problem, it requires good starting conditions in order to avoid local minima and to improve convergence. In our VO system we take advantage of the general smoothness of robot motion relative to our camera capture rate and we initialise the solution $\hat{\mathtt{T}}^{v_l r}$ to the value found on the previous VO timestep.

As mentioned, dense methods are suitable for parallel implementation. Most of the computations involved in our method are spent on evaluation of the per-pixel error terms, Eq. (3.24), and their gradients Eq. (3.27). which are fully parallelisable, and in our CUDA implementation we typically launch one thread per pixel. We use an efficient parallel reduction on GPU (based on techniques described by [Harris, 2008] and [Luitjens, 2014]) to assemble $\mathtt{J}^\top\mathtt{J}$ and $\mathtt{J}^\top f(\mathbf{0})$ from the individual error terms, whereas only the final computations, Eq. (3.29), that are required to solve the normal equation are performed on a CPU. Our method operates very comfortably in real-time on a modern PC with GPU. The mean frame-rate of VGA image resolution processing required for visual odometry is about 270–300 FPS (10 times real-time) on a desktop GTX 480 GPU and about 60–65 FPS on a mid-range laptop GT650M.

3.6 Experiments

We ran a series of experiments to demonstrate that dense planar visual odometry can offer accurate and precise motion estimation and indeed could replace wheel odometry. Since the quality of visual odometry depends also on the quality of the extrinsics calibration we will refrain from the full evaluation until the next chapter, where we perform additional tests and will address various aspects of visual odometry and auto-calibration including an investigation of how the violation of certain assumptions such as planar motion or planarity of the observed scene affects the performance of our method.

Fig. 3.6 shows our test platform, a Pioneer P3-DX robot with an adjustable rigid camera mount, and camera relatively close to the ground. We assume the camera position on the robot is known and that the camera observes entirely the planar floor surface. In the next chapter, which addresses the problem of auto-calibration, we perform an additional evaluation using data obtained from a rear parking camera.

3.6.1 Comparison against Wheel Odometry

As we wanted to evaluate our visual odometry method in different field conditions, with different camera poses, and various surfaces and lighting conditions, obtaining ground truth for all experiments was in general difficult. Even though we tried several possibilities to obtain ground truth for robot motion in order to evaluate the quality of visual odometry once calibrated, we decided to used the wheel odometry of the Pioneer robot as a reference for most experiments. While wheel odometry of course is not usually considered suitable as a ground truth reference, this robot is heavy and precise and turns out to have locally very accurate wheel odometry on surfaces with good grip such as short carpet. To verify this and justify its use as the reference for the rest of our evaluation, we conducted an experiment where wheel odometry was compared against a ground truth reference consisting of an

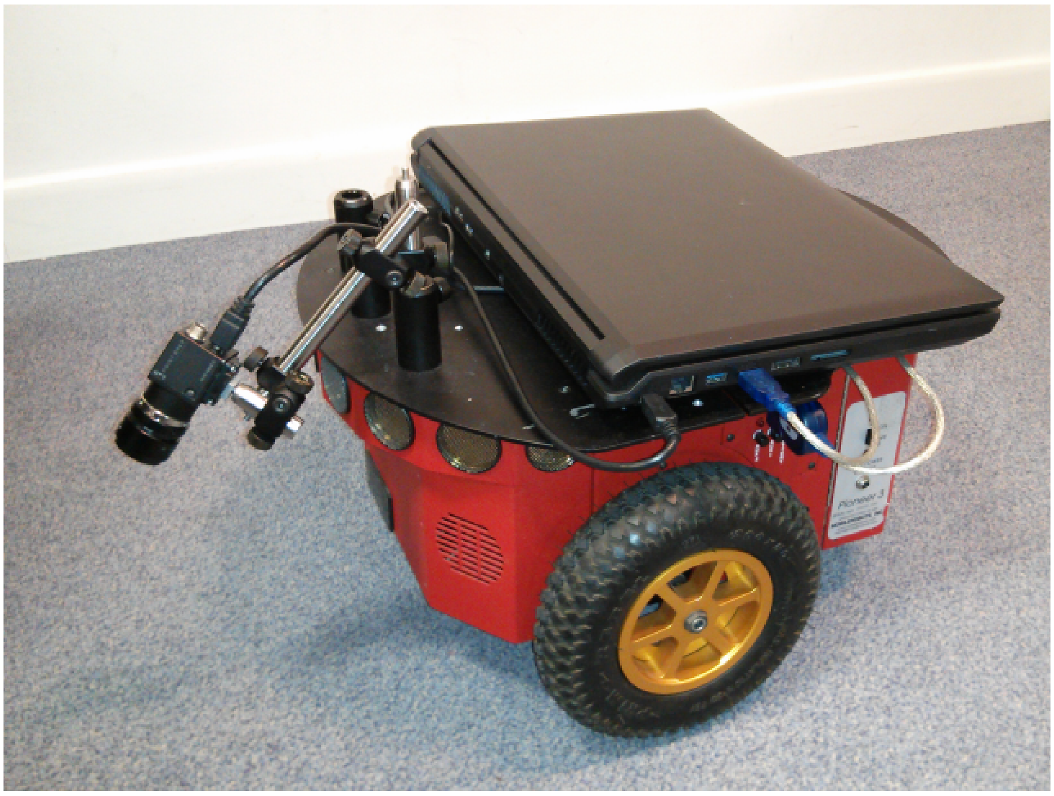


Figure 3.6: Robot Pioneer P3-DX used for in our experiments.

external overhead calibrated camera system observing a target marker placed on top of the robot. The results are shown in Fig. 3.7. What we see is that the Pioneer's wheel odometry is indeed accurate, with no observable bias in incremental motion estimation; in addition the robot has better local motion estimation accuracy as seen in the smoothness of the wheel odometry plot. We decided therefore to base our VO experiments on a comparison with wheel odometry rather than the external camera system, with the large advantage that this easily enables extended experimentation and long trajectories where providing an external ground truth reference might be more challenging and prone to error than the visual odometry itself.

Fig. 3.8 shows examples of different surfaces that we used to test our method. Probably the simplest and most intuitive demonstration of the performance of our visual odometry is a plot that shows superimposed trajectories generated both by wheel and visual odometry, as depicted in Fig. 3.9. We see good overall performance over significant distances apart from bigger errors caused by low-lighting conditions and self-shadowing, issues we will examine later. Here we also show a trajectory

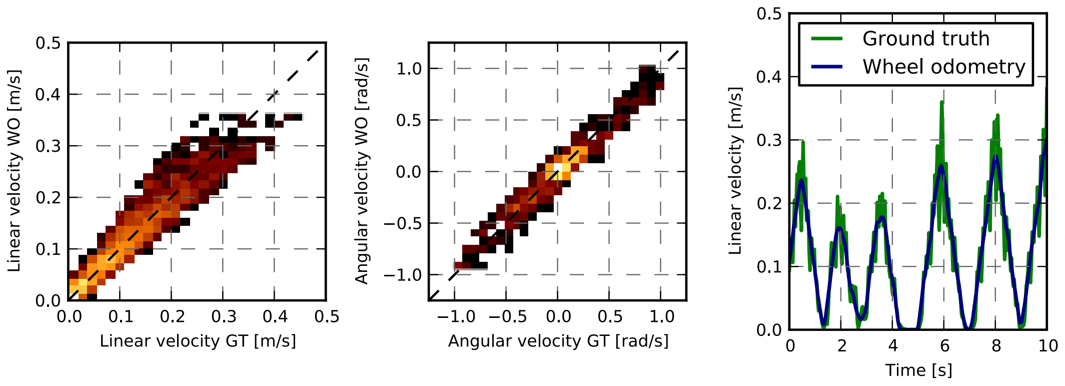


Figure 3.7: Comparison of wheel odometry against external camera-based ground truth for short motions of our indoor robot. We carry out a motion of several minutes, and we obtain the incremental velocity and angular velocity measurements from both wheel odometry and the ground truth reference and plot points on two histograms. Colours in these 2D 'heat map' histograms indicate the frequency of measurements. We see that the data are tightly clustered around the $x = y$ axis in both cases, indicating that the robot's wheel odometry is accurate and unbiased. In fact the robot's wheels give better local motion estimation accuracy, as shown by the increased smoothness of the wheel odometry line in the short time series plot on the right.

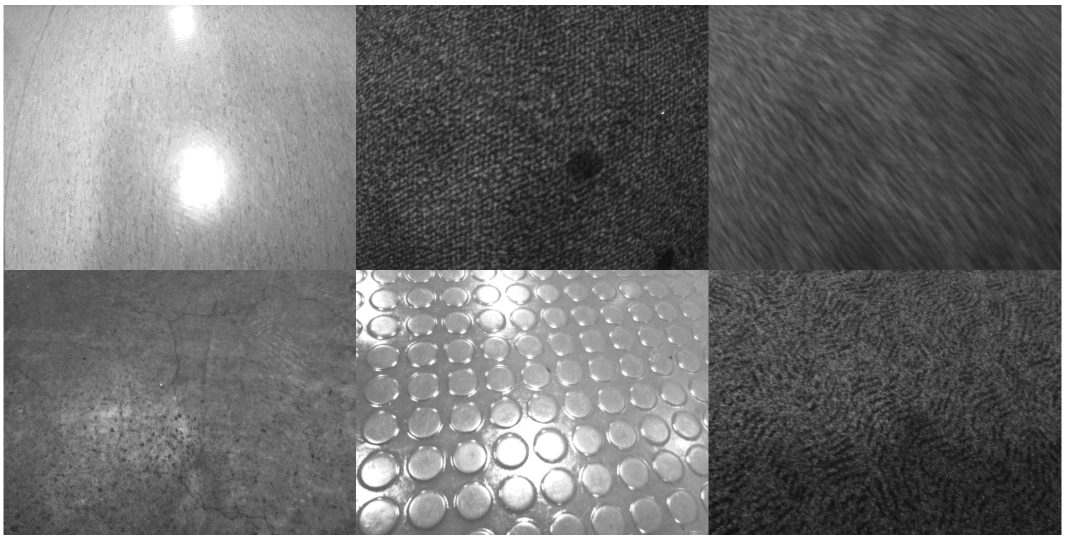


Figure 3.8: We tested our visual odometry using various camera settings and on different types of surface.

generated by uncalibrated visual odometry, where the camera extrinsics were off just by a few millimetres and degrees. This highlights the importance of proper auto-calibration, a subject we will investigate in the next chapter.

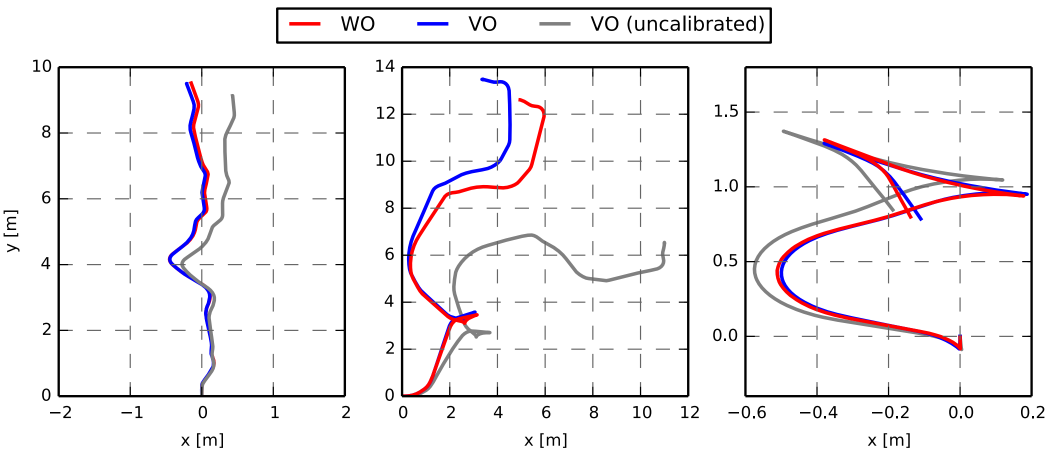


Figure 3.9: Example of different trajectories estimated by WO and calibrated and uncalibrated VO. The uncalibrated trajectories were generated using only a manual calibration, and the camera extrinsics were off by only ±10 mm and ±2.5°. The increased error in the middle figure is due to significant self-shadowing and low lighting conditions.

However, as both VO and WO systems will eventually drift over time, observing trajectory plots is not a strong basis for quantitative evaluation of VO quality. Instead, for quantitative evaluation we analyse the statistics of incremental motion estimation accuracy. Fig. 3.10 summarises the behaviour of VO against WO in the form of 2D histograms. These histograms were created from more than 350 metres of motion (about 45000 frames, 25 minutes) that featured a wide range of motions and turns up to the maximum dynamics of the robot (peak linear velocity 0.8 ms^{-1}, average 0.2 ms^{-1}; peak angular velocity 1.5 rads^{-1}, average 0.24 rads^{-1} respectively) on various surfaces and using 10 different calibrated camera configurations with camera heights from 7 cm to 25 cm. In the histograms, each point represents the estimated velocity (linear and angular on the left and right respectively) plotted against ground truth, with a heat map showing the frequency of each pair of values. The tight packing of these distributions around the $x = y$ axis indicates the unbiased and low uncertainty estimation of robot motion and the good performance of the algorithm most of the time. In the velocity range of up to around 0.6 ms^{-1} we see extremely robust performance. Beyond this speed we see a very small fraction of gross errors that result mostly from a combination of disadvantageous effects like reduced inter-frame overlap, motion blur, shadows cast by the robot or low-light conditions.

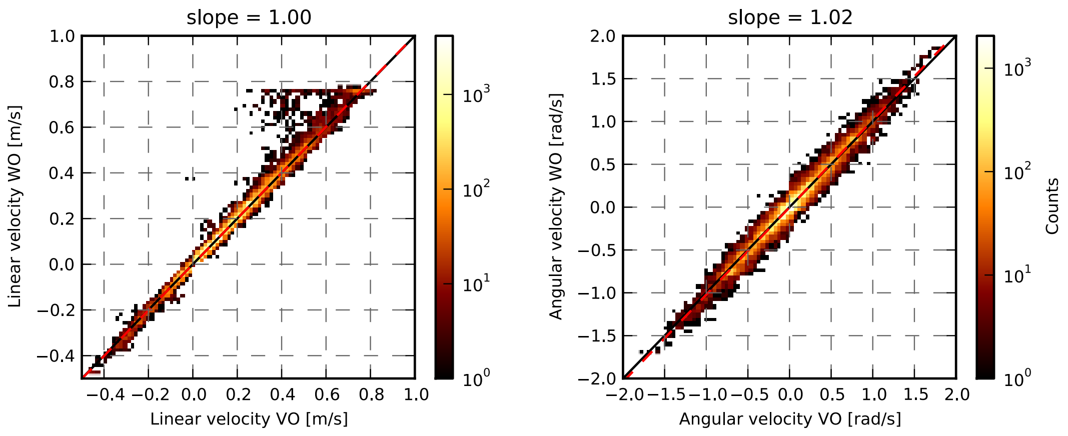


Figure 3.10: 2D heat-map histograms showing the statistical performance of auto-calibrated visual odometry against validated WO ground truth. Histograms were created using about 45000 data points; note the logscale used for the heat map. Gross errors are in fact extremely rare and the distribution is tightly packed about the $x = y$ line indicating unbiased and precise performance: the red dashed line is a fit to our data which is almost perfectly aligned with the desired black line. Without a systematic error we expect the slope of the fitted line to be equal to 1, and indeed we obtain a slope of 1.00 for the linear velocity and 1.02 for the angular velocity.

Fig. 3.11 shows tracking results for quite challenging surface types. The first has little matte texture but is shiny with raised circular studs. The nonplanar bumpy motion of the robot is the cause for more noisy estimation in this case. The second floor is smooth but also shiny with strong specularities. Despite this, with no modification our algorithm gives good results a large fraction of the time.

3.6.2 High Precision Motion Estimation

This experiment investigates and highlights the capability of downward-looking VO to recover extremely high precision local motion. We mounted the camera at around 4 cm from the ground, and drove the robot straight forward at its lowest smooth velocity setting of 2 cms^{-1}. The camera observed and tracked a wooden board floor surface on top of which was placed a ruler with clear millimetre markings (we track only the wooden part observed by the camera). By inspecting the image sequence obtained (Fig. 3.12 top) we were able to confidently confirm the ground truth motion (at 30 Hz) of around 0.7 mm per frame, corresponding to image motion of around 8 pixels per frame.

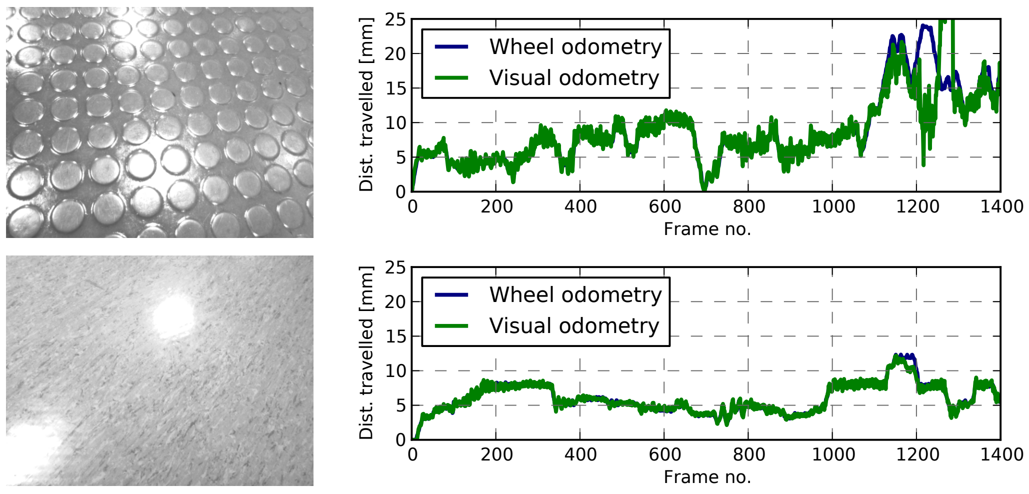


Figure 3.11: Our visual odometry also works on challenging surface types. The time series compare the estimated per-frame distance travelled measured by visual and wheel odometry. The nonplanar motion of the robot over the raised circular studs and strong specularities cause more noisy estimation but our algorithm gives good results most of the time.

The results of planar VO applied to this sequence against idealised ground truth can be seen in Fig. 3.12. We see a remarkable precision in incremental motion estimation, where the per-frame motion is estimated with a standard deviation consistently within 0.1 mm. We suspect that the oscillation observed at the beginning is to real vibration of our robot when it first starts moving, and can be seen to damp when a constant velocity is reached. This oscillation, of only around 0.5 mm peak to peak amplitude, is something not measurable by wheel odometry.

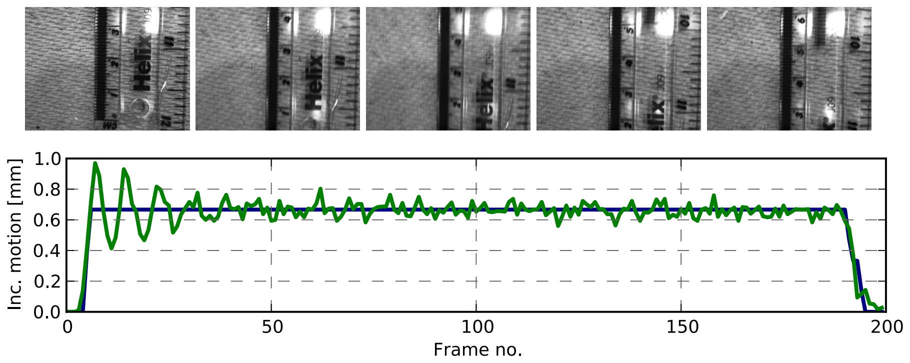


Figure 3.12: Experiment to investigate ultra-high precision local motion estimation. Top: Images from 10 frame intervals. Bottom: VO against idealised ground truth.

3.6.3 Qualitative Results

With high quality camera tracking from dense planar VO we can start creating simple maps of the environment in the form of a planar mosaic as shown in Fig. 3.13. A high quality mosaic indicates both correct estimation of the planar camera motion and of the out-of-plane angles. Note that these mosaics were created using incremental motion only; there is no loop-closure or global optimisation.

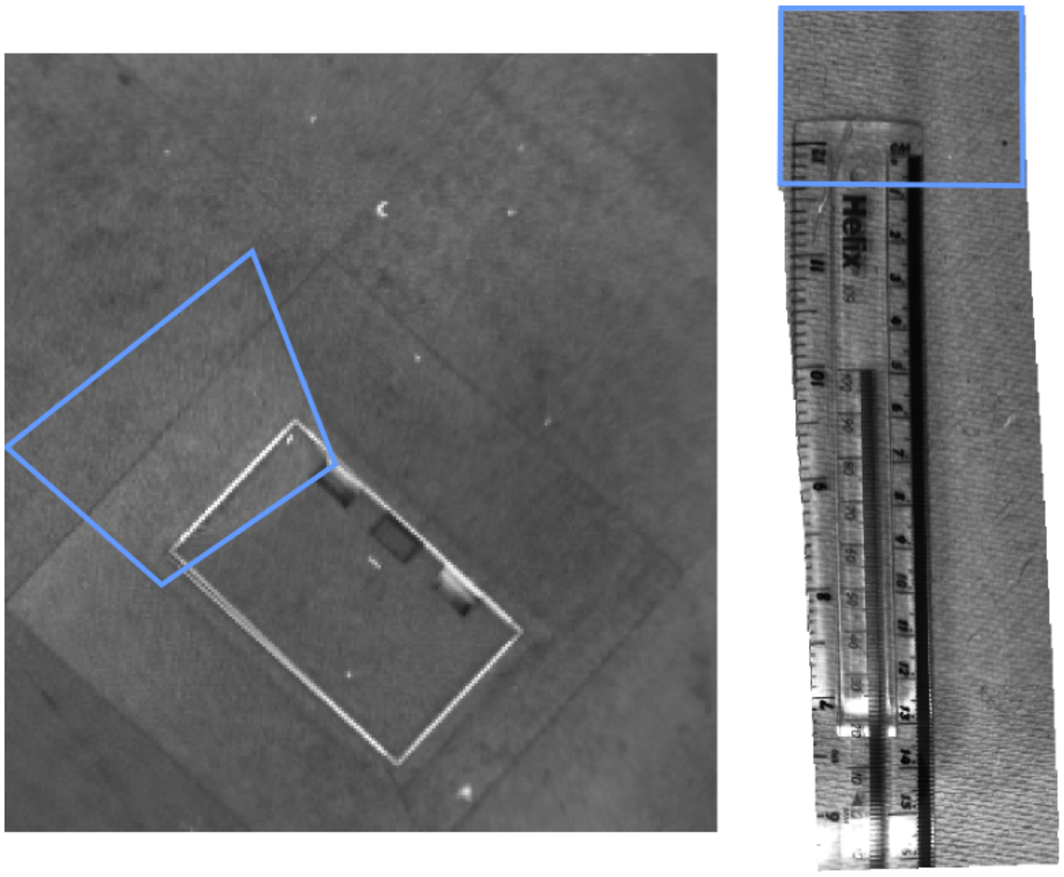


Figure 3.13: Consistent planar mosaics created by simple stitching of frames using the homographies estimated by our visual odometry system. The field of view of a single camera frame is superimposed for comparison.

The accuracy of our visual odometry manifests itself also in the ability to perform multi-view stereo as presented in the previous chapter. The depth maps shown in Fig. 2.8 were calculated using motion estimation based on dense visual odometry (where we identify and track only on the planar part of the scene).

3.7 Conclusion

In this chapter we have demonstrated that a dense planar image alignment approach offers a robust and accurate solution for visual odometry, in particular for a small robot moving in an indoor environment. Although our main assumption that the ground is flat and parallel to the camera motion limits the applicability of our method to certain applications, we have shown how this greatly simplifies VO and enables us to use the image information available in a video sequence in a straightforward manner. With small modifications that allow us to identify and track using only the pixels that directly observe a planar floor, our visual odometry can be applied in more general scenarios, and we will use it throughout the rest of this thesis to estimate robot ego-motion. In the next chapter we also show that the camera extrinsics required for visual odometry can be precisely auto-calibrated in highly practical settings.

The methods based on dense image alignment are well suited to efficient parallel implementation on a GPU, and they are highly scalable, which makes them very attractive for low-cost robots. It will be interesting to see the trade-offs in VO as camera frame-rate and resolution are varied. In fact, an optical mouse can be thought of as an example of our VO approach where we decrease the sensor resolution and increase frame-rate to extremes.

There is great scope for future extension of the method. Instead of tracking frame-to-frame camera motion, in order to reduce drift the method can easily be used to perform motion estimation with respect to a global model, *e.g.* a reconstructed planar mosaic. Although we achieve certain robustness by using a robust cost function, the performance can be further improved by directly modelling and taking into account many effects that can negatively affect the behaviour of dense image alignment. For example the method proposed by [Park et al., 2009] addresses the problem of motion blur, whereas by performing Lucas-Kanade in the Fourier domain [Ashraf et al., 2010; Lucey et al., 2013] one could handle substantial illumination variations.

Chapter **4**

Auto-calibration for Visual Odometry

Contents

4.1 Introduction

In the previous chapter we presented our approach to Visual Odometry for a mobile robot moving on a planar surface. The VO system requires all six degrees of freedom of the $\mathtt{T}_{vc} \in \mathbf{SE}(3)$ transformation between the robot's drive frame and the camera frame to be estimated, and in this chapter we present a simple procedure that allows us to calibrate this transformation fully automatically.

Because we assume that the downward-looking camera directly observes the plane on which the robot is locally driving, we have strong information to help the calibration procedure. This means that of the six unknown extrinsic parameters, two can be estimated reliably by capturing a very short sequence of video as the robot moves, and without needing any external reference. These are the roll and pitch angles of the camera, since only a correct estimate of these will lead to inter-frame homography warps of the planar texture which are consistent with movement over the same plane. We experimentally demonstrate the ease and accuracy of this first part of the auto-calibration procedure.

Calibrating the remaining four parameters — the camera's yaw angle, and its translational position relative to the robot frame — cannot be achieved without some kind of external reference, but we clearly show that there are several highly simple and practical forms this can take which leads to rapid and precise full calibration, by formulating the problem as a pose-graph optimisation with the help of wheel odometry or a nonholonomic constraint on the vehicle motion. First, if we have a sequence of camera trajectory where the robot makes a small number of movements and turns with accompanying synchronised wheel odometry on a good surface, we present a graph optimisation algorithm which produces an unbiased estimate of the full calibration. Note that the purely relative information from wheel odometry is sufficient here, and we do not need to perform full SLAM or have an absolute external motion reference. We perform auto-calibration using only wheel odometry as an additional reference, and take the uncertainty in that wheel odometry into account, only requiring that the wheel odometry can be assumed to have zero-mean errors incrementally on a region with good surface grip to achieve unbiased camera extrinsics estimation.

We then examine even weaker cases for a robot which does not have wheel odometry, and show that very generally applicable assumptions about the nonholonomic motion of most wheeled robots are sufficient for accurate auto-calibration of all but two of the remaining degrees of freedom of the robot-camera transformation. If we then further augment this with simple labels about which parts of the calibration trajectory involve forward or backward robot motion, which should be available from the robot's control signal even if it has no odometry, we can calibrate everything apart from the height of the camera above the plane, which requires just one manual measurement or known object to resolve.

A wide range of real-world surfaces permit successful and precise auto-calibration and planar visual odometry, and we demonstrate results on surfaces such as carpet, vinyl floor tiles, concrete, grass and wooden boards. We analyse the performance of our auto-calibration techniques against ground truth where possible, and check them for consistency by repeatedly achieving the same auto-calibration results over different motions and surface types. Furthermore we analyse how violation of planar assumptions affects the performance of our methods.

The chapter is organised as follows. We start by presenting related work on extrinsics auto-calibration in Section 4.2, and then proceed to describing the core of our approach that includes vision-based (Section 4.4) and graph-based auto-calibration (Section 4.5). An extensive experimental evaluation is conducted in Section 4.6, followed by a discussion and conclusions in Section 4.7.

4.2 Related Work

Auto-calibration of the robot-sensor configuration and, more generally, methods for auto-calibrating the extrinsic transformations between multiple sensors *e.g.* [Underwood et al., 2010] are integral elements of the deployment process of any robotic platform or multi-sensor system.

Calibration of the camera extrinsics in most cases requires special reference targets [Martinelli et al., 2006; Antonelli et al., 2010], or is achieved by calibrating with reference to some carefully engineered ground truth positioning system. [Knorr et al., 2013] presented a system that exploits a planar assumption for multi-camera extrinsics calibration. Their system is based on a Kalman Filter and does not require any external reference as the ground plane serves as a natural reference object. Similarly to our method, the calibration procedure relies on plane-induced homographies between successive frames. Additionally [Miksch et al., 2010] used the assumption that the road surface visible in the images is approximately flat to calculate the orientation of the camera with respect to the vehicle frame of reference. However, their method relies on feature extraction and mapping, puts constraints on the vehicle motion by forcing it to move in straight line, and does not extend naturally to multiple frames (a final parameter is determined by a recursive filter which averages various estimates over time).

Our graph-based auto-calibration is quite general and merits comparison with other work on auto-calibrating the extrinsics of outward-looking sensors on mobile robots, *e.g.* [Censi et al., 2008] and especially with [Kümmerle et al., 2011a] who performed a calibration against a full SLAM system. In their work, [Brookshire and Teller, 2011] demonstrated that external incremental pose measurements are sufficient to recover sensor calibration provided that degenerate trajectories are avoided. However, to the best of our knowledge, our work is the first that demonstrates how to use nonholonomic constraints for sensor extrinsics calibration. In the robot vision and tracking literature there are only a few examples of applying nonholonomic motion constraints to sensing directly. In their feature-based VO system, [Scaramuzza et al., 2009b] used a one-point RANSAC outlier rejection scheme based on nonholonomic constraints to reduce the computational burden of data association and therefore speed up egomotion estimation. [Scaramuzza et al., 2009a] used nonholonomic constraints together with the known offset between camera and vehicle frame of reference to estimate the absolute scale in SfM. [Fossati and Fua, 2008] did not directly apply nonholonomic constraints, but instead showed that incorporating the local direction of travel while estimating motion can greatly improve visual tracking, since the pose of an object has a direct influence on its direction of travel.

4.3 Preliminaries

The goal of auto-calibration is to estimate the pose of the camera relative to the robot which carries it. Its translational position is determined by the position vector (x_{vc}, y_{vc}, z_{vc}) and its orientation can be represented in the form of roll, pitch and yaw angles. Our auto-calibration method builds on top of dense image alignment technique from the previous chapter and it has two distinct steps. First, the roll and pitch angles of the camera are estimated purely from a short image sequence without the need for targets or markers, or constraints on robot trajectory. Correct estimates of roll and pitch angles make it possible to track planar camera motion and we then acquire a camera trajectory featuring a short sequence of manoeuvres which is required for the second step of calibration. There, the remaining 4 DoF, the yaw angle and translations of the camera, are estimated by considering the estimated planar camera motion in the context of the robot's whole trajectory. There are two methods available here. In the most general case these 4 DoF can be calibrated by bringing in relative motion estimates from wheel odometry as an external motion.

Since synchronised and unbiased odometry may not be available on many platforms, we also present an alternative for the weaker but common case where a wheeled robot's motion can be assumed to be nonholonomic, showing that a further 2 DoF (the x_{vc} coordinate of the camera position and the yaw angle) can be determined without the need for any source of reference. A final extension of our calibration framework allows us to obtain a reasonable estimate of the y_{vc} camera coordinate without any precise measurements from wheel odometry, using only motion direction priors, *i.e.* simple information on whether the robot was moving forward or backward at each point on its trajectory. The last degree of freedom, the height of the camera above the plane which determines the overall scale of motion estimation, in this case needs to be estimated with one manual measurement (directly of the camera height, or perhaps by recognising a single known object or marking in the visual scene).

4.4 Vision-based Calibration

In a previous chapter, in Section 3.5, we demonstrated how to track camera motion expressed in terms of a planar robot motion, provided that the full 6 DoF transformation between the robot and camera frames $\mathtt{T}^{cv}$ is given. Here we will present a method for calibrating the first 2 DoF of the full transformation, sufficient to allow planar camera motion tracking.

We first assume that the camera is aligned with the robot frame of reference, *i.e.* that $x_{vc} = 0$, $y_{vc} = 0$, $z_{vc} = 1$, $\gamma_{vc} = 0$, and therefore we have:

$$\mathtt{T}^{vc} = \left(\mathtt{R}^{vc} \mid (0,0,1)^\top\right) \quad . \tag{4.1}$$

Let us recall Eq. (3.20):

$$\mathtt{T}^{lr} = \mathtt{T}^{cv}\mathtt{T}^{v_{lr}}\mathtt{T}^{vc} \ .$$

Now, since the frames of references are aligned, we can think of $\mathtt{T}^{v_{lr}}$ as a camera planar motion, and there are only two unknowns in $\mathtt{T}^{vc}$, roll and pitch angles, $\mathbf{y} = (\alpha_{vc}, \beta_{vc})^\top$, that define the camera orientation matrix $\mathtt{R}^{vc}$.

By estimating camera orientation $\mathtt{R}^{vc}$ with respect to the ground plane we can simultaneously estimate the normal of the plane. Unlike [Silveira et al., 2008] we are not using inverse depth to estimate the normal, but instead we exploit the fact that the unit plane normal vector $\mathbf{n}$ is uniquely defined by the camera orientation $\mathtt{R}^{cv}$ and depends only on the roll and pitch angles:

$$\mathbf{n} = \mathtt{R}^{cv}\,(0,0,1)^\top \ . \tag{4.2}$$

We can now readily parameterise the homography in Eq. (3.22) by two sets of parameters: the planar motion of the camera from frame to frame $\mathbf{x} = (x, y, \theta)^\top$, and the orientation of the camera with respect to the plane using roll and pitch angles $\mathbf{y} = (\alpha_{vc}, \beta_{vc})^\top$:

$$\mathtt{H}^{lr}(\mathbf{x},\mathbf{y}) = \mathtt{K}\mathtt{T}^{cv}(\mathbf{y})\mathtt{T}^{v_{lr}}(\mathbf{x})\mathtt{T}^{vc}(\mathbf{y})(I| - \mathbf{n}(\mathbf{y}))^\top\mathtt{K}^{-1} \ . \tag{4.3}$$

Using Eq. (4.1) and Eq. (4.2) we can simplify and rewrite the homography in Eq. (4.3) into the following form:

$$\mathtt{H}^{lr}(\mathbf{x},\mathbf{y}) = \mathtt{K}\mathtt{R}^{cv}(\mathbf{y})\mathtt{M}\mathtt{T}^{v_{lr}}(\mathbf{x})\mathtt{N}\mathtt{R}^{vc}(\mathbf{y})\mathtt{K}^{-1} \ , \tag{4.4}$$

with two constant matrices:

$$\mathtt{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \quad \text{and} \quad \mathtt{N} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \ . \tag{4.5}$$

Similarly to Eq. (3.25) we choose to parameterise the homography by a transformation matrix $\mathtt{T}(\mathbf{x})$ that represents a small change to the estimate of planar motion $\hat{\mathtt{T}}^{v_{lr}}$, as well as by an update matrix $\mathtt{R}(\mathbf{y})$ representing a small update to the current estimate of camera orientation $\hat{\mathtt{R}}^{cv}$:

$$\mathtt{H}^{lr}(\mathbf{x},\mathbf{y}) = \mathtt{K}\hat{\mathtt{R}}^{cv}\mathtt{R}(\mathbf{y})\mathtt{M}\hat{\mathtt{T}}^{v_{lr}}\mathtt{T}(\mathbf{x})\mathtt{N}\mathtt{R}(-\mathbf{y})\hat{\mathtt{R}}^{vc}\mathtt{K}^{-1} \ . \tag{4.6}$$

For $\hat{\mathtt{T}}^{v_{lr}}$ we use the same update rule as in Eq. (3.26), whereas for the rotation matrix $\hat{\mathtt{R}}^{cv}$ the following update rule is applied:

$$\hat{\mathtt{R}}^{cv} \leftarrow \hat{\mathtt{R}}^{cv}\mathtt{R}(\hat{\mathbf{y}}) \ , \tag{4.7}$$

where $\hat{\mathbf{y}}$ belongs to the Lie Algebra $\mathfrak{so}(3)$ and represents a small update to the estimate.

By calculating the partial derivatives of homography $\mathtt{H}^{lr}$ with respect to its parameters we obtain the essential building blocks necessary to apply the dense image alignment machinery introduced in Chapter 3. The partial derivative of $\mathtt{H}^{lr}$ around $\mathbf{x} = \mathbf{0}, \mathbf{y} = \mathbf{0}$ with respect to $\mathbf{x}$ is equal to:

$$\left.\frac{\partial \mathtt{H}^{lr}(\mathbf{x}, \mathbf{y})}{\partial x_i}\right|_{\mathbf{x},\mathbf{y}=\mathbf{0}} = \mathtt{K}\hat{\mathtt{R}}^{cv}\mathtt{M}\hat{\mathtt{T}}^{v_{lr}}\frac{\partial \mathtt{T}(\mathbf{x})}{\partial x_i}\mathtt{N}\hat{\mathtt{R}}^{vc}\mathtt{K}^{-1} \ , \tag{4.8}$$

while the derivative with respect to $\mathbf{y}$ is expressed in the following form:

$$\left.\frac{\partial \mathtt{H}^{lr}(\mathbf{x}, \mathbf{y})}{\partial y_i}\right|_{\mathbf{x},\mathbf{y}=\mathbf{0}} = \mathtt{K}\hat{\mathtt{R}}^{cv}\frac{\partial (\mathtt{R}(\mathbf{y})\mathtt{M}\hat{\mathtt{T}}^{v_{lr}}\mathtt{N}\mathtt{R}(-\mathbf{y}))}{\partial y_i}\hat{\mathtt{R}}^{vc}\mathtt{K}^{-1} \ . \tag{4.9}$$

Finally, we can use the fact that:

$$\left.\frac{\partial \mathtt{R}(\mathbf{y})}{\partial y_i}\right|_{\mathbf{y}=\mathbf{0}} = \mathtt{G}_i^{\mathbf{SO}(3)} \quad \text{and} \quad \left.\frac{\partial \mathtt{R}(-\mathbf{y})}{\partial y_i}\right|_{\mathbf{y}=\mathbf{0}} = -\mathtt{G}_i^{\mathbf{SO}(3)} \ , \tag{4.10}$$

where $\mathtt{G}_i^{\mathbf{SO}(3)}$ is the ith group generator for $\mathbf{SO}(3)$, and apply the product rule to show that:

$$\left.\frac{\partial (\mathtt{R}(\mathbf{y})\mathtt{M}\hat{\mathtt{T}}^{v_{lr}}\mathtt{N}\mathtt{R}(-\mathbf{y}))}{\partial y_i}\right|_{\mathbf{y}=\mathbf{0}} = \mathtt{G}_i^{\mathbf{SO}(3)}\mathtt{M}\hat{\mathtt{T}}^{v_{lr}}\mathtt{N} - \mathtt{M}\hat{\mathtt{T}}^{v_{lr}}\mathtt{N}\mathtt{G}_i^{\mathbf{SO}(3)} \ . \tag{4.11}$$

Having arrived at these expressions we can now efficiently use dense image alignment to find not only the planar frame to frame camera motion but also the normal of the plane parallel to which the camera is moving.

Although it is possible to estimate the plane normal from just two images, to better constrain and improve the robustness of the estimation process we combine multiple frames into a local dense map and jointly estimate the motion between consecutive keyframes as well as the parameters of the plane normal. When the overlap between the last keyframe and the current frame falls below a defined threshold we add this frame to the map. In practice for estimating roll and pitch angles we usually used maps containing between 5 and 50 frames.

4.5 Graph-based Calibration

At this step of calibration, we assume that the roll and pitch angles were calibrated using the method described in the previous section, and therefore that we can accurately track camera motion with respect to the ground plane. The remaining 4 DoF, the camera position, (x_{vc}, y_{vc}, z_{vc}) as well as the yaw angle, need now to be estimated in order to relate the camera motion to the robot motion. In the following we will refer to VO as the (up-to-scale) planar motion obtained from camera tracking using already calibrated roll and pitch angles only, and use wheel odometry (WO) to mean measurements coming from wheel odometry. Therefore all the motions considered in this section are assumed to be planar.

There are two distinct graph-based calibration methods presented: one relies on external motion estimation source, whereas the second method exploits general motion assumptions.

4.5.1 Calibration against Wheel Odometry

In this type of auto-calibration we are interested in finding a set of parameters $(x_{vc}, y_{vc}, z_{vc}, \gamma_{vc})$ that best explain the incremental measurements from VO and the corresponding WO measurements. Our formulation is similar to that proposed in [Brookshire and Teller, 2011] and is based only on incremental measurements, but we model the problem directly as a factor-graph (as proposed in [Kümmerle et al., 2011a]). This formulation allows us to benefit from all the techniques available for graph optimisation and solve the problem very efficiently.

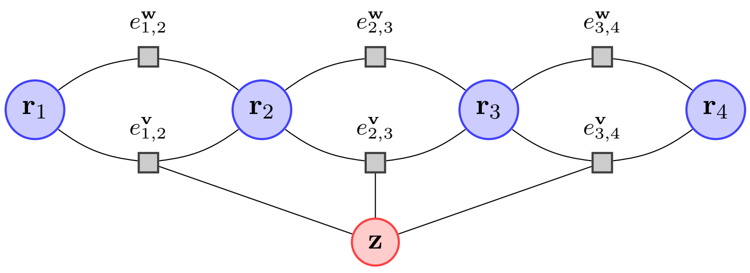


Figure 4.1: Factor graph formulation of our calibration against wheel odometry. Variable nodes $\mathbf{r}_i$ represent robot poses along the trajectory, and one additional node $\mathbf{z} = (x_{vc}, y_{vc}, z_{vc}, \gamma_{vc})$ represents the camera calibration. Factor nodes $\mathbf{w}_{i,i+1}$ (rectangles) with error functions $e^{\mathbf{w}}_i$ represent the measurement obtained from wheel odometry, whereas $\mathbf{v}_{i,i+1}$ with error function $e^{\mathbf{v}}_i$ are obtained from camera tracking.

An example factor graph is shown in Fig. 4.1. In the graph the nodes $\mathbf{r}_i = (x_{r_i}, y_{r_i}, \theta_{r_i})$ represent robot poses along the trajectory, and one additional node $\mathbf{z} = (x_{vc}, y_{vc}, z_{vc}, \gamma_{vc})$ represents the camera configuration. Every time an odometry measurement is available we add a new node to the graph. We assume that camera and wheel odometry are well synchronised, and that jitter and synchronisation errors are negligible. Two consecutive nodes $\mathbf{r}_i$ and $\mathbf{r}_{i+1}$ are connected by two factors, one representing the measurement obtained by wheel odometry $\mathbf{w}_{i,i+1} = (x_{w_{i,i+1}}, y_{w_{i,i+1}}, \theta_{w_{i,i+1}})$ and the second the visual odometry measurement $\mathbf{v}_{i,i+1} =$

$(x_{v_{i,i+1}}, y_{v_{i,i+1}}, \theta_{v_{i,i+1}})$. Visual odometry factors $\mathbf{v}_{i,i+1}$ are also connected to the camera configuration $\mathbf{z}$. There are no loops in this graph and the graph is fixed by the first node. In the graph we are primarily interested in determining the camera calibration $\mathbf{z}$, though as the robot poses $\mathbf{r}_i$ are also variable, as a byproduct of the camera calibration we obtain an estimate of the robot trajectory.

The error function in the wheel odometry factors $e_i^{\mathbf{w}}$ measures how well the parameters $\mathbf{r}_i, \mathbf{r}_{i+1}$ satisfy the constraint arising from the wheel odometry measurement $\mathbf{w}_{i,i+1}$:

$$e_i^{\mathbf{w}}(\mathbf{r}_i, \mathbf{r}_{i+1}, \mathbf{w}_{i,i+1}) = (\mathbf{r}_{i+1} \ominus \mathbf{r}_i) \ominus \mathbf{w}_{i,i+1} \ , \tag{4.12}$$

where $\ominus$ is the inverse of the usual motion composition operator $\oplus$.

The error function in the visual odometry factors $e_i^{\mathbf{v}}$ measures how well the parameters $\mathbf{r}_i$, $\mathbf{r}_{i+1}$ and $\mathbf{z}$ satisfy the constraints arising from the visual odometry measurement $\mathbf{v}_{i,i+1}$:

$$e_i^{\mathbf{v}}(\mathbf{r}_i, \mathbf{r}_{i+1}, \mathbf{z}, \mathbf{v}_{i,i+1}) = (\mathbf{r}_{i+1} \ominus \mathbf{r}_i) \oplus f(\mathbf{z}, \mathbf{v}_{i,i+1}) \ , \tag{4.13}$$

where f is the function that transforms the motion from the camera frame of reference to the robot frame of reference using $(x_{vc}, y_{vc}, z_{vc}, \gamma_{vc})$.

The goal of graph optimisation is to find a set of parameters $(\mathbf{r}^\star, \mathbf{z}^\star)$ that maximise the likelihood of the observed data, *i.e.* that minimise the total error in the graph:

$$\sum_i e_i^{\mathbf{v}\top} \hat{\Omega}_i^{\mathbf{v}} e_i^{\mathbf{v}} + \sum_i e_i^{\mathbf{w}\top} \Omega_i^{\mathbf{w}} e_i^{\mathbf{w}} \ , \tag{4.14}$$

where $\Omega_i^{\mathbf{w}}$ is the information matrix of the measurement $\mathbf{w}_{i,i+1}$ and $\hat{\Omega}_i^{\mathbf{v}}$ is the projection of the information matrix $\Omega_i^{\mathbf{v}}$ using the current estimate of $\mathbf{z}$. Least squares estimation on this hyper graph is performed using g^2o [Kümmerle et al., 2011b].

4.5.2 Calibration using Practical Assumptions

Nonholonomic Calibration

Calibration against wheel odometry works well in practice and is the most universal method. However, many mobile robots belong to a class of nonholonomic systems and therefore are subject to motion constraints, which can also be used for auto-calibration even when odometry is not available. We show that for a sensor that is placed on a nonholonomic mobile platform and that can measure its incremental motion in its own frame of reference, we are able to estimate the x_{vc} component of the sensor's position on the robot, as well as the yaw angle γ_{vc}, just using nonholonomic constraints without the need of wheel odometry or any other reference system.

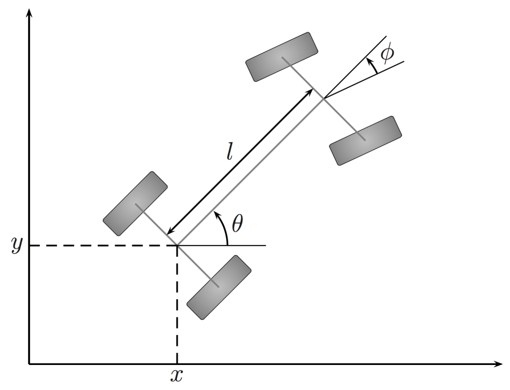
In robotics, a *nonholonomic system* refers to a platform for which the dimension of the admissible velocity space (number of control variables) is smaller than the dimension of the configuration space (number of degrees of freedom). Such systems are therefore also called *underactuated*. The most common family of nonholonomic systems in robotics are wheeled robots (*e.g.* unicycle, differential drive or car-like robots), whose kinematic models are derived from the *rolling without slipping* assumption. Rolling without slipping assumption means that the translational and rotational velocities of a rolling wheel are not independent and is related to the fact that most wheels are not designed to slide sideways, but are required to roll in the direction they are pointing. Consequently, this imposes velocity constraints on rolling vehicles [LaValle, 2006]. One of the most intuitive examples of a nonholonomic system is a simple car with rear-wheel drive, as shown in Fig. 4.2. The configuration space of a car has three degrees of freedom (its (x, y)-position and orientation θ), but only two control variables (steering angle ϕ and its driving velocity). A car-like robot can only move forward or backward in a direction perpendicular to the orientation of its rear wheels axis, and it cannot drive sideways because its wheels would have to slide instead of roll. Exactly these constraints make certain aspects of car driving, like parallel parking, challenging [Laumond, 1998].

More specifically, the rolling without slipping constraint can be expressed by the following formula:

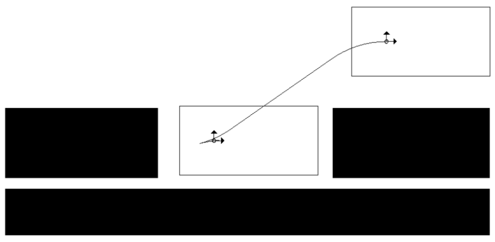
$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \ , \tag{4.15}$$

where (x, y, θ) is used to denote the configuration of a robot at a time step t and its velocities are represented by $\dot{x}$ and $\dot{y}$. Therefore, the nonholonomic constraints limit robot's velocity in its current configuration and forces the vehicle to move tangentially to its main axis. Consequently, nonholonomic constraints do not restrict the (x, y, θ)-configurations a car or robot can take, but how it can reach them.

Our calibration method is not limited to a car-like robot but it is applicable to various types of nonholonomic robots, whose kinematic models can be derived from the *rolling without slipping* assumption formulated above, including a differential



(a) Kinematic model of a car.



(b) Due to nonholonomic constraints, in order to park a car in parallel, a special trajectory has to be executed [Laumond, 1998].

Figure 4.2: Car is a simple and intuitive example of a nonholonomic system. It has three degrees of freedom, *i.e.* (x, y)-position of the rear wheels axis and orientation θ, but only two control variables, *i.e.* a steering angle ϕ and its driving velocity. The assumption that car's wheels cannot roll without slipping limits the manoeuvres (velocities) the car can instantaneously execute in its current configuration and forces the vehicle to move tangentially to its main axis. The nonholoconomic constraints make certain car manoeuvres, *e.g.* parallel parking, challenging.

drive robot like the Pioneer P3-DX used in most of our experiments. The key observation when using nonholonomic constraints for extrinsic calibration, is that even though a planar trajectory $\mathtt{T}^c(t)$ measured by a camera (or another sensor) in its frame of reference does not need to satisfy nonholonomic constraints, the resulting estimated robot trajectory:

$$\mathtt{T}^v(t) = \mathtt{T}^{vc}\mathtt{T}^c(t)\mathtt{T}^{cv} \ , \tag{4.16}$$

should satisfy these motion constraints. Therefore, we are looking for calibration parameters that produce a nonholonomic vehicle trajectory from a given camera trajectory. Note that here we consider the whole trajectories. A camera trajectory is created from a sequence of consecutive incremental planar motions, and therefore we use t to denote time in Eq. (4.16). The variables in the transformation $\mathtt{T}^{vc}$ from camera frame of reference to the robot frame of reference are now $(x_{vc}, y_{vc}, \gamma_{vc})$. In fact, it turns out that for correct values of x_{vc} and γ_{vc} we can satisfy nonholonomic constraints independent of y_{vc} (see Section 4.5.2 below for more details), and therefore we cannot determine y_{vc} by means of this method. Additionally, using this algorithm we cannot estimate the absolute scale, unless the sensor can measure its motion metrically. Therefore, we can only use this algorithm for estimating two degrees of

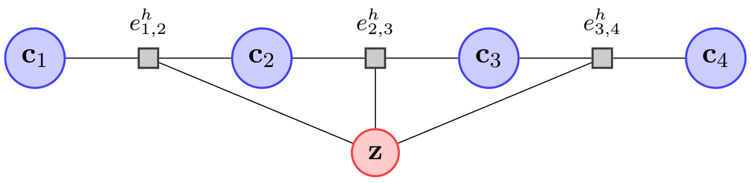


Figure 4.3: Factor graph formulation for nonholonomic calibration. Nodes $\mathbf{c}_i$ correspond to the camera poses and are fixed, whereas a single variable node $\mathbf{z} = (x_{vc}, \gamma_{vc})$ represents camera extrinsic parameters. Factor nodes $h_{i,i+1}$ (rectangles) with corresponding error functions, $e^h_{i,i+1}$, represent violations of nonholonomic constraints.

freedom.

A graph-based formulation of this calibration strategy is depicted in Fig. 4.3. Fixed nodes $\mathbf{c}_i$ represent camera poses along the trajectory, and there is only a single variable node in the graph, $\mathbf{z}(x_{vc}, \gamma_{vc})$, for camera extrinsics calibration parameters. The nodes are connected by factors $h_{i,i+1}$ that do not store any measurements but penalise deviation from nonholonomic motion $e^h_{i,i+1}$ using Eq. (4.15):

$$e^h_{i,i+1} = \dot{x}_{r_{i,i+1}} \sin\theta_{r_i} - \dot{y}_{r_{i,i+1}} \cos\theta_{r_i} \ . \tag{4.17}$$

As mentioned, the nonholonomic error does not depend on y_{vc}. The robot poses $(x_{r_i}, y_{r_i}, \theta_{r_i})$ required for calculating the nonholonomic error are obtained from the camera poses $\mathbf{c}_i$ and current calibration parameters $\mathbf{z}$ using Eq. (4.16). To evaluate the error, it is also necessary to calculate the velocities from the poses. We estimate a velocity at every node by calculating the constant velocity that bring the robot from the current to the next pose. Although this is only an approximation, it works well in practice. Again, least squares estimation on this graph is performed using g^2o.

Limitations of the Nonholonomic Calibration

A limitation of nonholonomic calibration is that it does not provide enough constraints to recover y_{vc}. When calculating the nonholonomic error as defined in Eq. (4.17) one has to use the values measured in the camera frame of reference $(\dot{x}_c, \dot{y}_c, \dot{\theta}_c)$ and θ_c, together with the current estimates of the calibration parameters $(x_{vc}, y_{vc}, \gamma_{vc})$ to obtain the velocities in the robot frame of reference $\dot{x}_r$ and $\dot{y}_r$ as well as the robot orientation θ_r. One can show that these values depends on each other in the following way:

$$\begin{aligned} \dot{x}_r &= \dot{x}_c \cos\gamma_{vc} - \dot{y}_c \sin\gamma_{vc} + \dot{\theta}_c \left(y_{vc}\cos\theta_c + x_{vc}\sin\theta_c\right) \\ \dot{y}_r &= \dot{x}_c \sin\gamma_{vc} + \dot{y}_c \cos\gamma_{vc} - \dot{\theta}_c \left(x_{vc}\cos\theta_c - y_{vc}\sin\theta_c\right) \\ \theta_r &= \theta_c \end{aligned} \tag{4.18}$$

Recall the nonholonomic error formulation (Eq. (4.17)) that assumes rolling without slipping (we drop certain subscripts for brevity):

$$e^h = \dot{x}_r \sin\theta_r - \dot{y}_r \cos\theta_r \tag{4.19}$$

Substituting Eq. (4.18) into Eq. (4.19) leads to:

$$\begin{aligned} e^h = {} & \dot{x}_c \sin\theta_c \cos\gamma_{vc} - \dot{y}_c \sin\theta_c \sin\gamma_{vc} + \dot{\theta}_c \sin\theta_c \left(y_{vc}\cos\theta_c + x_{vc}\sin\theta_c\right) \\ & - \dot{x}_c \cos\theta_c \sin\gamma_{vc} - \dot{y}_c \cos\theta_c \cos\gamma_{vc} + \dot{\theta}_c \cos\theta_c \left(x_{vc}\cos\theta_c - y_{vc}\sin\theta_c\right) \end{aligned} \tag{4.20}$$

Simplifying the equation and using the trigonometric identity ($\sin^2\theta_c + \cos^2\theta_c = 1$), we can reduce the error to the following formula:

$$\begin{aligned} e^h &= \dot{x}_c \sin\theta_c \cos\gamma_{vc} - \dot{y}_c \sin\theta_c \sin\gamma_{vc} - \dot{x}_c \cos\theta_c \sin\gamma_{vc} - \dot{y}_c \cos\theta_c \cos\gamma_{vc} + \dot{\theta}_c x_{vc} \\ &= \dot{y}_c \cos(\gamma_{vc} - \theta_c) + \dot{x}_c \sin(\gamma_{vc} - \theta_c) + \dot{\theta}_c x_{vc} \end{aligned} \tag{4.21}$$

The error e^h depends on the velocities measured in the camera frame of reference $(\dot{x}_c, \dot{y}_c, \dot{\theta}_c)$, the corresponding camera orientation θ_c, as well as x_{vc} and γ_{vc}. However it does not depend on the y_{vc}-coordinate of the camera pose on the robot, and therefore we cannot estimate this quantity using nonholonomic the calibration method.

Calibration using Known Direction of Motion

The fact that we cannot recover y_{vc} using nonholonomic calibration does not necessarily mean that we need to use well-synchronised wheel odometry. Instead we show that using weaker constraints, namely information about whether the robot was moving forward or backward (*i.e.* whether its linear velocity v was positive or negative), is sufficient to constrain y_{vc}. This data can come, for example, from the commands the robot was supposed to execute, or from a very rudimentary odometry system. Obviously, a trajectory used for calibration with the help of this method should include a balanced number of sections of backward and forward motion. Also, correct values of x_{vc} and γ_{vc} are necessary at this stage.

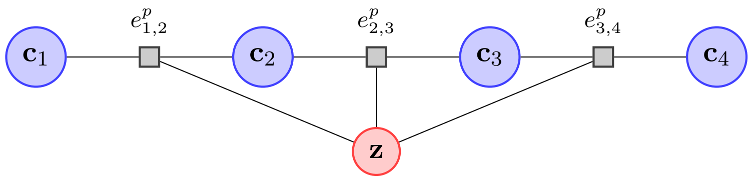


Figure 4.4: Factor graph calibration using motion direction priors. Factor nodes $p_{i,i+1}$ (rectangles) store information (sign) about whether the robot was moving forward (+) or backward (−).

The graph shown in Fig. 4.4 is not significantly different from the nonholonomic graph and it mainly consists of fixed camera poses, $\mathbf{c}_i$, calculated from the incremental measurements. There is one additional node for camera extrinsic configuration $\mathbf{z}$. The factors in the graph, $p_{i,i+1}$ (denoted as rectangles), now store *prior* information (sign) about whether the robot was moving forward (+) or backward (−). To calculate the error in a factor we first need to calculate the robot linear velocity $v_{i,i+1}$ between two consecutive robot poses using the current estimate of the calibration parameters. The error depends on y_{vc} only (the parameter we want to estimate), though proper values of x_{vc} and γ_{vc} are required for the calculation. We use the following (heuristic) formula for error calculation in the graph:

$$e^p_{i,i+1} = \left\{ \begin{array}{ll} 0 & \text{if } \operatorname{sgn}(v_{i,i+1}) = prior_{i,i+1} \\ |v_{i,i+1}| & \text{otherwise} \end{array} \right\} . \tag{4.22}$$

The graph optimisation framework is quite flexible and allows the possible inclusion of additional factors, *e.g.* representing smoothness of motion or constraints on the robot's dynamics that represents the limits of its acceleration and velocity.

Even though we formulated the calibration as a graph optimisation problem with a heuristic error function, it can be useful to think about it as classification task, where we are looking for a value y_{vc} that minimises the number of mismatches between the priors (forward/backward motion) and the motion direction calculated from visual odometry.

4.6 Experiments

4.6.1 Experimental Configuration and Ground Truth Reference

We designed a series of experiments to validate our auto-calibration method using the robotic platform described in Chapters 2 and 3. We also performed an additional evaluation of both visual-odometry and auto-calibration using data obtained from a rear parking camera, located at a height of about one metre, on a passenger vehicle travelling through an urban setting at speeds of up to 45 km/h; this data set was collected by Renault and previously used in published work [Lovegrove et al., 2011]. The road vehicle data were also captured at 640×480 resolution and 30 Hz frame-rate, but the camera field of view was approximately 45°. By testing our method on the car dataset from Renault we could make use of the high quality ground truth trajectory data available from a PHINS system capable of estimating with high-accuracy the vehicle's position and orientation. This system consists of a precise Inertial Measurement Unit made of 3 fibre optic gyroscopes and 3 pendulum type accelerometers, a bi-frequency GPS receiver and the vehicle wheel odometry.

Obtaining ground truth for extrinsics calibration is generally quite difficult; it is cumbersome to measure precisely a full 6 DoF camera pose as the camera frame of reference is hidden inside a camera housing. To verify that our auto-calibration procedure can produce correct results, we tested it against a standard camera calibration procedure that makes use of a chessboard pattern (similar to [Zhang, 1999]). We placed the chessboard pattern on the floor and first acquired a few images of it from different perspectives. Next, we mounted and fixed the camera on the robot and captured an image of the chessboard pattern again. The standard procedure estimates camera intrinsics as well as the 6 DoF of the camera with respect to the chessboard pattern. However, since the position of the chessboard pattern with respect to the robot frame of reference cannot be reliably measured, only 3 DoF can be used as our ground truth, namely the roll and pitch angles as well as the height above the chessboard pattern. These are equivalent to the roll and pitch angles and height of the camera in the robot frame of reference.

The run time for calibration depends on the number of frames used for calculating the camera orientation, and the length of trajectory used for estimating the remaining 4 degrees of freedom. In our test cases we were able to run continuous auto-calibration using 10 frames and hundreds to thousands poses at a rate of a few frames per second.

4.6.2 Calibration on Different Surfaces

In order to evaluate the performance and stability of our auto-calibration algorithm we tested the procedure using three different camera configurations (Fig. 4.5) on a

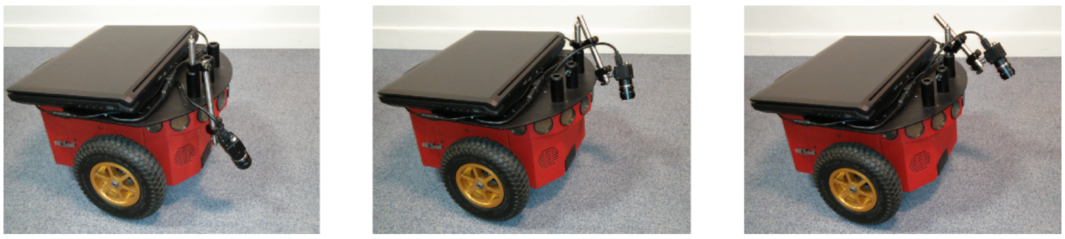


Figure 4.5: We performed verification of the auto-calibration for 3 different camera poses.

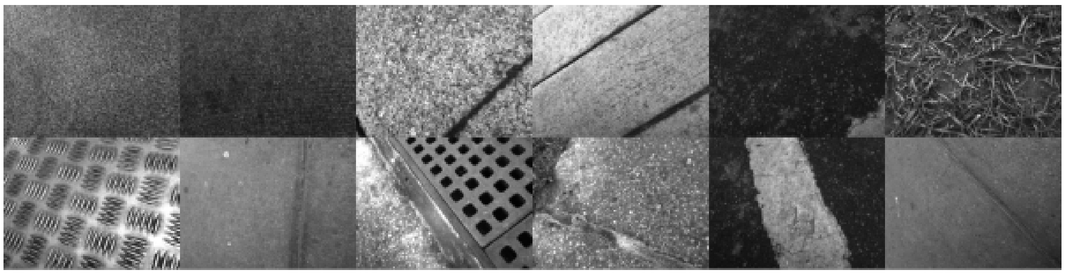


Figure 4.6: We tested our system on various different surfaces, both indoors and outdoors.

wide range of surface types (Fig. 4.6). In some cases, the assumptions about a planar scene and the planarity of the robot motion were violated. Each individual run was on average 15–20 metres long, but we subdivided every trajectory into multiple, overlapping, approximately 4 metre long trajectories starting with every new frame. This allowed us to perform a statistical analysis and provide distributions of the parameters, as from every individual run we had thousands of data points. For every sub-trajectory we estimated the roll and pitch angle of camera using the first 10 frames, whereas the remaining degrees of freedom were calculated using the rest of the available robot trajectory (every time we started with the same initial conditions, $x_0 = 200$ mm, $y_0 = 0$ mm, $z_0 = 200$ mm, $\alpha_0 = 0°$, $\beta_0 = 0°$, $\gamma_0 = 0°$).

Table 4.1 shows the results from our auto-calibration method together with the ground truth measurements for three different camera configurations. For each individual configuration the estimates converge to the same results independent of the surface and in multiple trials and we can conclude that the algorithm is stable. Only on a very uneven surface (sequence 'outdoor 4') do the estimates diverge from the rest of the measurements. The estimates of the camera height, as well as the orientation, seem to be most stable, as indicated by a relative low variance, whereas the estimates of the x_{vc} and y_{vc} coordinates are more noisy.

a) Camera configuration 1

	x [mm]	y [mm]	z [mm]	roll [°]	pitch [°]	yaw [°]
ground truth			**178.7**	**12.4**	**17.6**	
carpet 1	244.1 ±4.3	-18.5 ±5.2	179.1 ±0.8	12.3 ±0.3	17.7 ±0.4	-9.2 ±0.5
carpet 2	241.8 ±2.1	-16.9 ±2.5	178.9 ±0.6	12.3 ±0.3	17.5 ±0.3	-9.2 ±0.3
outdoor 1	245.7 ±5.9	-18.9 ±4.5	178.5 ±0.9	12.1 ±0.9	17.5 ±1.3	-9.5 ±0.2
outdoor 2	250.2 ±3.4	-12.9 ±3.4	180.1 ±0.4	12.2 ±1.2	17.1 ±1.7	-9.7 ±0.2
outdoor 3	244.3 ±2.1	-19.2 ±2.2	178.1 ±0.5	12.0 ±1.4	17.4 ±1.9	-9.3 ±0.2
outdoor 4	254.2 ±9.5	-3.6 ±17.8	174.1 ±6.9	11.9 ±2.0	17.3 ±2.8	-10.0 ±0.7

b) Camera configuration 2

	x [mm]	y [mm]	z [mm]	roll [°]	pitch [°]	yaw [°]
ground truth			**216.4**	**29.8**	**-4.6**	
carpet 1	191.7 ±2.1	-107.8 ±3.6	218.1 ±0.7	30.1 ±1.0	-4.4 ±0.6	75.8 ±0.3
outdoor 1	195.5 ±1.2	-104.6 ±1.8	218.3 ±0.2	29.7 ±0.8	-4.7 ±0.7	75.9 ±0.1
outdoor 2	198.7 ±2.7	-106.0 ±1.6	218.5 ±0.9	29.5 ±1.3	-4.7 ±1.5	75.6 ±0.4

b) Camera configuration 3

	x [mm]	y [mm]	z [mm]	roll [°]	pitch [°]	yaw [°]
ground truth			**146.9**	**-26.6**	**4.7**	
carpet 1	222.6 ±4.6	80.6 ±4.5	148.0 ±0.7	-26.8 ±0.4	4.5 ±0.3	-18.6 ±0.3
outdoor 1	231.0 ±2.6	78.5 ±4.2	148.6 ±1.4	-26.4 ±1.1	4.3 ±1.1	-18.5 ±0.2
outdoor 2	226.6 ±1.4	79.7 ±1.6	149.3 ±0.4	-26.4 ±1.8	4.5 ±1.3	-18.2 ±0.3

Table 4.1: Calibration on different surfaces. About 1000-2000 subtrajectories were used on each surface separately for calculation of mean and standard deviation. Ground truth values were obtained using chessboard pattern method described in Section 4.6.1.

The most stable results in terms of camera orientation are obtained on a carpet surface, when the surface on which the robot is driving is mostly flat. On the other hand, outdoors, where the robot was driving over rough and slightly uneven surfaces, the variance of the estimates increases; this is caused by a locally nonplanar motion and nonplanar structures in the field of view. As demonstrated in the previous chapter (Fig. 3.9), even relatively small miscalibration can eventually lead to significant errors in visual odometry. Therefore, it is recommended to perform the auto-calibration on mostly flat surfaces and with moderate robot velocities, and to increase the number of frames used for calculating camera orientation to improve the stability.

Whereas the process of estimating camera orientation with respect to the plane of motion is independent of the robot's trajectory, estimation of the remaining degrees

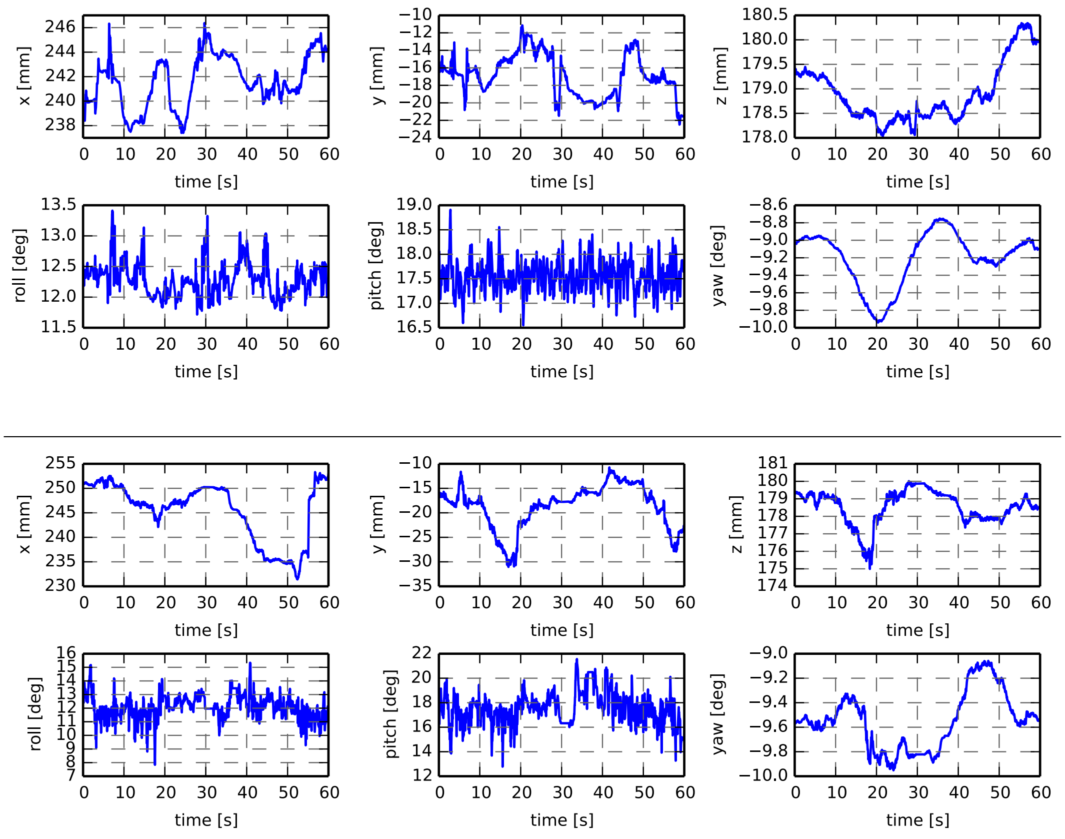


Figure 4.7: The development over time of our estimates of extrinsics parameters of camera configuration 1 on the ‘carpet 1’ (top) and ‘outdoor 1’ (bottom) surface when auto-calibration is run online. Every time point represents another trajectory segment. The values of the roll and pitch angles are stable and independent of the robot trajectory. On the other hand, the remaining degrees of freedom change as different trajectory segments are used.

of freedom (camera metric position and the yaw angle) is governed by other rules. Firstly, it depends on proper frame-to-frame tracking and therefore proper values of camera orientation. If the first step of our auto-calibration is unsuccessful, it cannot be compensated for by graph optimisation and it is impossible to recover unbiased values of the 4 DoF. Secondly, certain trajectories are degenerate and prevent unambiguous calibration. As in [Brookshire and Teller, 2011], calibration cannot be recovered when the robot is driven in a straight line only, or when the camera experiences only concentric, circular motion. In practice these cases are easily avoided by varying the robot’s velocities and the trajectory geometry.

Fig. 4.7 illustrates the behaviour of estimates as time evolves for the same configur-

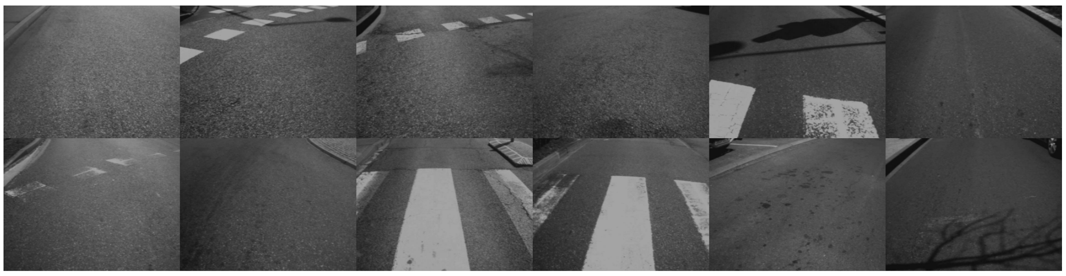


Figure 4.8: Examples of frames obtained from a parking camera. Most of the time the camera observes a flat road surface, with small fraction of images containing nonplanar structure, *e.g.* pavement or passing cars.

ation on two different surfaces. Here every time point represents another trajectory segment. We observe in general stable behaviour, and see that the roll and pitch angles are independent of the trajectory. Sudden jumps in the values of x_{vc}, y_{vc}, z_{vc} and yaw angle correspond to different robot motions occurring during trajectory segments, which impose slightly different constraints on the calibration graph. Even though the 4 metre long trajectory-segments used in this evaluation were enough to avoid degenerate cases, we generally observed that longer trajectories are desirable for more stable estimates of x_{vc}, y_{vc}, z_{vc} and the yaw angle γ_{vc}.

4.6.3 Odometry from a Road Vehicle's Rear Parking Camera

Next we evaluated our auto-calibration system in a completely different settings, using a video stream that was captured from a standard rear parking camera as a vehicle was moving in an urban environment. The data comes from Renault [Lovegrove et al., 2011]. The camera was placed at about one metre above the ground, a position significantly higher than in the experiment with the Pioneer robot, and was viewing the road surface directly behind the vehicle. Some of the frames captured are depicted in Fig. 4.8. Most of the time the camera observed a flat road surface, but in some parts of the images nonplanar structures are visible. These were mostly pavement (sidewalk) or other passing vehicles. Our test vehicle also experienced some nonplanar motion, when it was turning at high speed and driving over speed bumps. Consequently, this data set allows us to examine the effects of violating planar assumptions both with respect to the planarity of the motion as well as the planar-environment assumption. For the experiment we have full, high-accuracy, ground truth for visual odometry obtained from a PHINS system.

At the beginning of the evaluation we ran our standard calibration procedure:

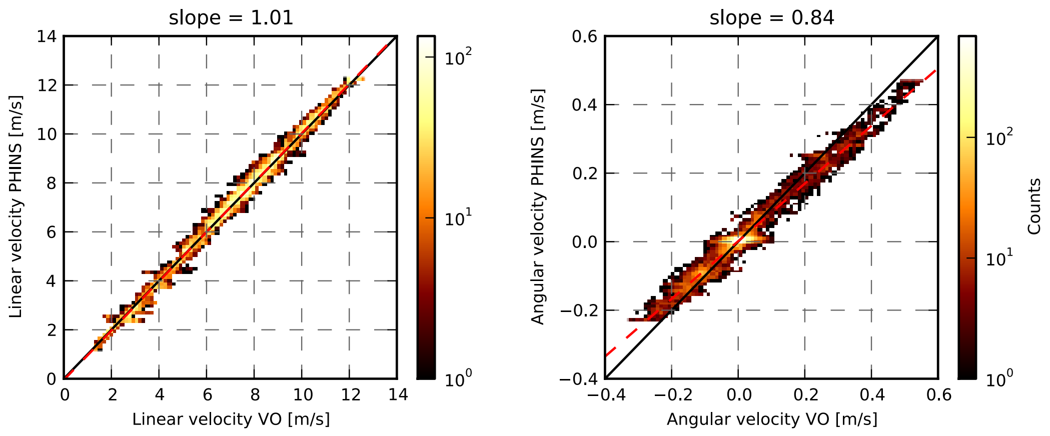


Figure 4.9: 2D histograms of the velocities measured by auto-calibrated visual odometry and the ground truth system. The frequency of a measurement is represented by colour in the heat map. We observe unbiased estimation of the linear velocity (left). The dashed red line represents a straight line fit to the data, and in this case it is perfectly aligned with the $x = y$ line. There is a systematic error measured in the angular velocity (right) as the fitted line diverges from the $x = y$ line. This is also confirmed by the slope of 0.84 for the line fitted to the measurements. Histograms were created using about 10000 data points.

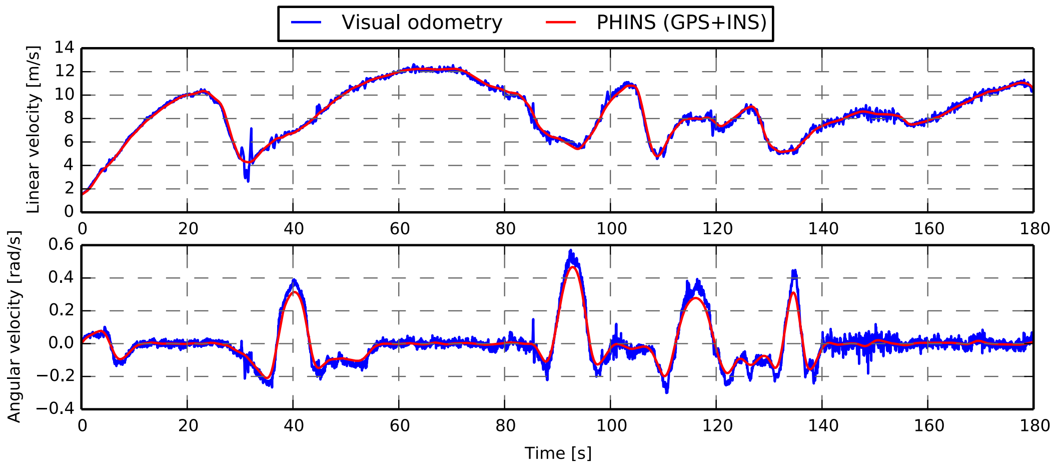


Figure 4.10: Comparison of the velocities measured by visual odometry and the PHINS ground truth system for the first 3 minutes of the sequence. On one hand we see accurate estimation of the linear velocity with only a small error around 30–35 seconds of the sequence, when the vehicle is driving over a speed bump. On the other hand, when the vehicle is performing turning manoeuvres, visual odometry systematically overestimates the angular velocity of the vehicle.

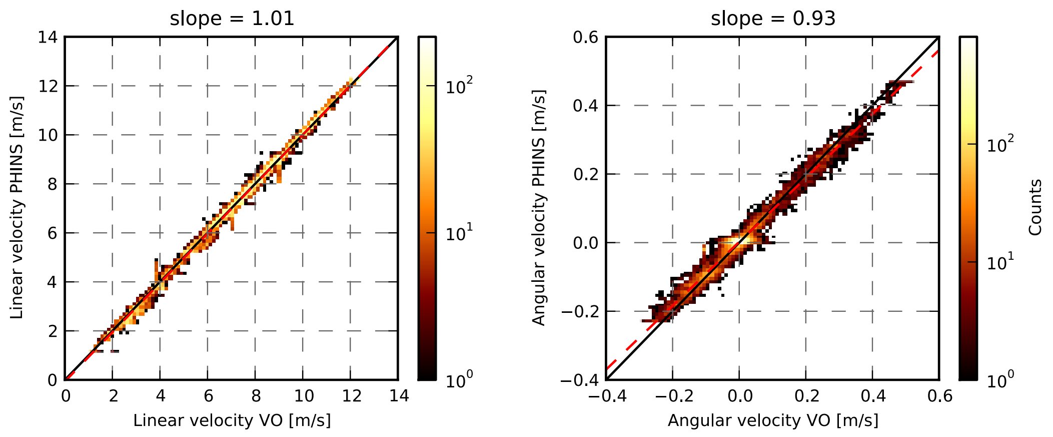


Figure 4.11: Estimating a local plane and tracking motion with respect to it seems to improve the performance of the visual odometry, accounting for the roll of the road vehicle during high speed turns. We still see unbiased and even more accurate estimation of the linear velocity. The systematic error in the angular velocity is reduced but not eliminated, as the slope of the line fitted is now 0.93 compared to 0.84 when global settings were used (note that a slope of 1 would indicate unbiased behaviour). Histograms were created using approximately 10000 data points.

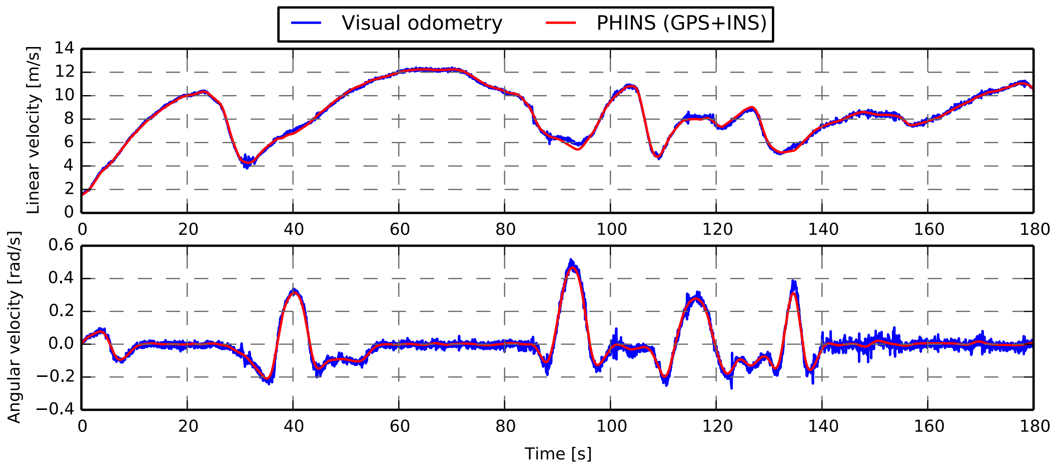


Figure 4.12: Improvement in the tracking obtained by continuous calculation of a local plane of motion is also visible in the time series. The estimation of the linear velocity is smoother and less noisy, whereas the angular velocity is no longer permanently overestimated.

first camera orientation was estimated using a short sequence of frames and next the remaining 4 DoF were calculated using a short sequence of the trajectory (about 200 metres long). After the camera extrinsics were available, we executed our planar visual odometry on the whole 2.5 km long trajectory. The results are summarised in the form of a 2D histogram in Fig. 4.9. Additionally, in Fig. 4.10, we present a short sequence of velocities measured by visual odometry and the PHINS ground truth system in the form of a time series.

The estimation of the linear velocity is accurate and unbiased, as indicated by the tight packing of the points along the $x = y$ axis in the histogram. However, there is a clearly observably systematic error in the measured angular velocity. This is confirmed in the time series included in Fig. 4.10 where we can recognise that the visual odometry over-estimates the angular velocity. This can indicate that when the vehicle is turning, the camera is locally experiencing a nonplanar motion and the estimated orientation is not valid. To confirm this observation we ran an additional evaluation, where the orientation of the camera with respect to the plane of motion was continuously estimated using the five most recent frames, and we calculate the planar motion with respect to the locally estimated plane. As depicted in Figs. 4.11 and 4.12 this procedure helps to reduce the systematic error in angular velocity, but it does not eliminate it completely. As a bonus, we obtain less noisy estimation of the linear velocity, although this comes at the extra cost required for performing continuous calibration.

Fig. 4.13 provides insight into what is happening when the vehicle is performing a turn, and helps to explain why we observe a systematic error in angular velocity. There we have plotted angular velocity measured by the ground truth system against the camera roll angle obtained during the continuous auto-calibration. The roll angle varies in a range of about $\pm 2.5°$ and is proportional to the current vehicle angular velocity. We suspect that as the vehicle is turning at a relatively high speed, vehicle suspension causes the car to lean to the side and consequently changes the camera roll angle to violate the planar motion assumption. We also analysed the behaviour of the pitch angle during the test drive and were expecting to see a similar pattern during acceleration and braking of the vehicle. However, this was not confirmed by the data. Apparently, the vehicle was not accelerating at high enough rates to cause a significant front/back tilt of the camera.

A change in the camera pitch angle is observable in another situation, namely when

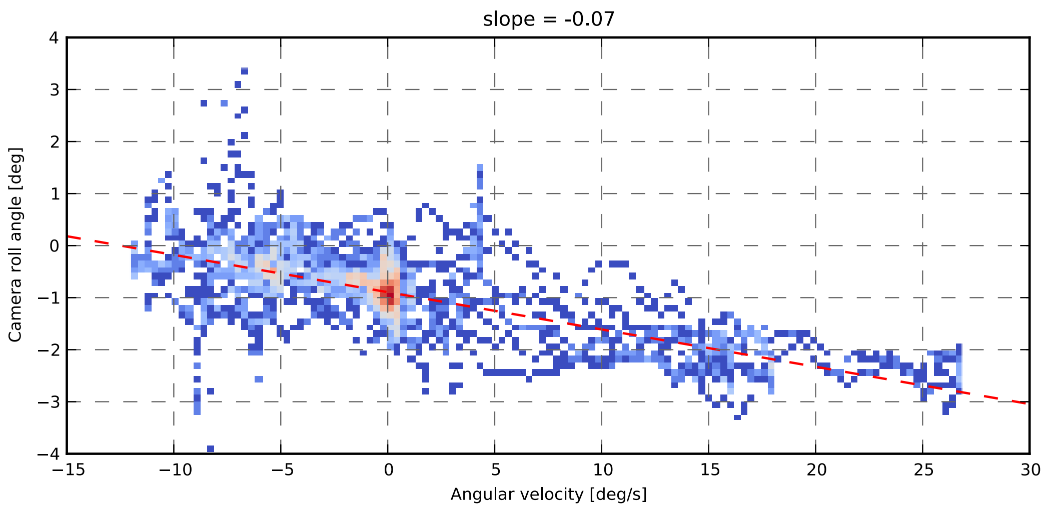


Figure 4.13: Angular velocity against measured camera orientation with respect to the local plane of motion. The roll angle varies in the range of about $\pm 2.5°$ and is correlated with the angular velocity. The red dashed line represents the fit and the slope of the line -0.07 can be understood as a measure of the dependence between these two parameters.

the vehicle was driving over speed bumps. This occurs, for example, at around 30–35 seconds of the time sequence (see Fig. 4.10) and is manifested by an increased error in both linear and angular velocity. Continuously estimating the camera orientation improves the performance of the tracking, which is best visible in Fig. 4.14. There we present a close-up of the two situations when the vehicle is driving over speed bumps. The pitch angle varies significantly in the range of about $\pm 10°$, and especially in the first sequence a clear pattern is visible: the first oscillation is due to the front of the vehicle passing the bump, and the second oscillation is due to the rear wheels driving over the bump. Obviously, the orientation of the camera is changing from frame to frame, so it is important to remember that estimated roll and pitch angles are only an approximation for the five most recent frames.

When the car is driving over speed bumps we can observe yet another effect, a violation of the planar environment assumption, which we will analyse now. As mentioned and shown in Fig. 4.8 in many frames nonplanar structures were visible from the camera. In most situations these elements were a pavement or other car passing by, and occupied only a small fraction of the image, usually close to the image borders. Since we are using a robust norm, our system can handle these situations to some degree and we did not observe any particular systematic error

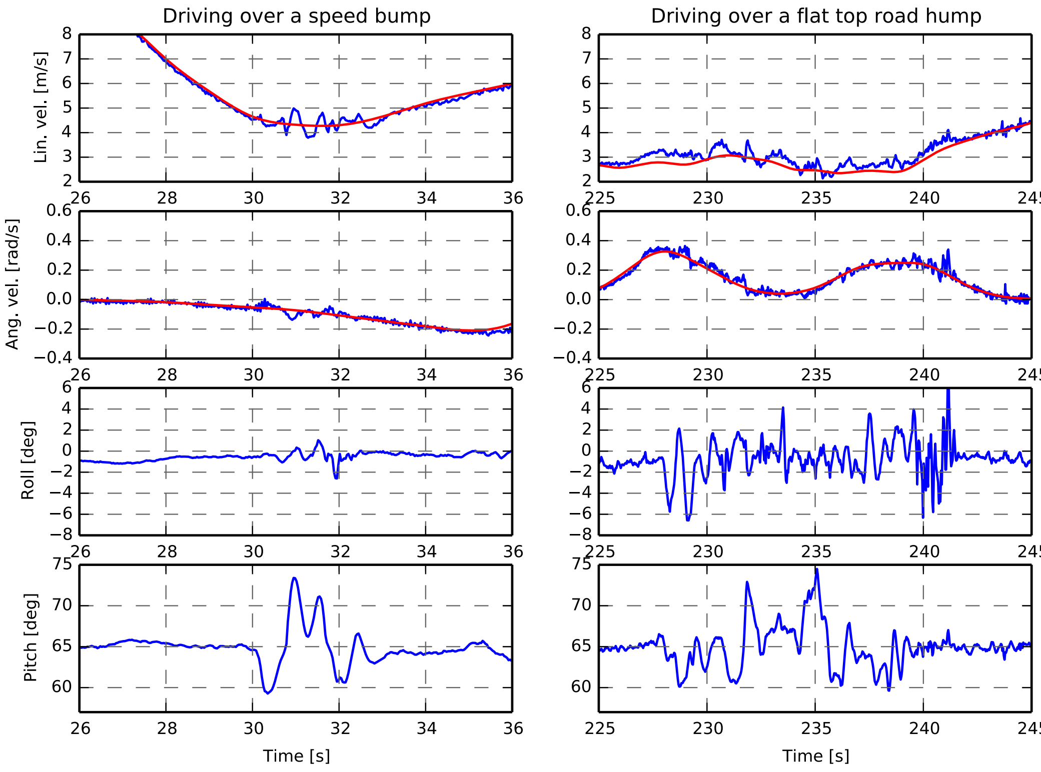


Figure 4.14: Driving over speed bumps causes a violation of the planar motion assumption. However we can detect a change in the pitch angle and by estimating a local plane of motion, at least partially compensate for errors.

due to small fractions of nonplanar objects in the scene. Fig. 4.15 shows a frame where a large fraction of the image is occupied by a passing car and also illustrates which pixels were automatically marked as outliers during the iterative reweighted least-squares estimation of planar frame-to-frame motion (Section 3.5). Obviously, in this situation we also benefit from a large image overlap between consecutive frames and the mostly uniform texture of the nonplanar object.

Structures visible in the camera field of view which do not belong to the main plane of motion, do not significantly affect the auto-calibration. To illustrate that, we ran auto-calibration twice on a short sequence where some part of a pavement was continuously visible. An example frame of this sequence is depicted in the top left corner in Fig. 4.8, and the pavement occupies about 10% of the image. In the first run we were using the whole image, and the second time we manually masked and ignored the nonplanar part of the images. Fig. 4.16 shows that there are no obvious differences in the roll and pitch angle estimated during these two runs. A pavement

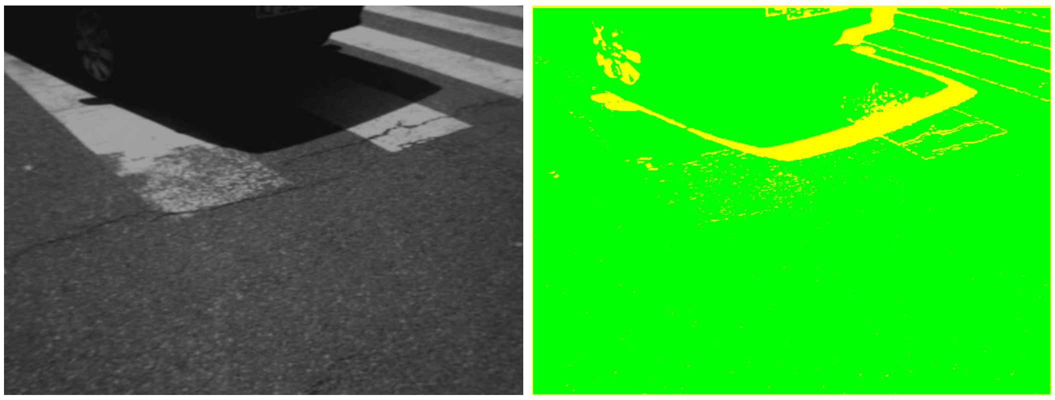


Figure 4.15: Left: An example of a frame where a significant fraction of the image is occupied by a nonplanar structure. Right: During the iterative reweighted least-squares estimation of frame to frame motion (Section 3.5), some pixels are automatically marked as outliers (yellow) by the robust norm used in the cost function.

(sidewalk) has quite a similar appearance to the road surface and the camera moves parallel to it as well. We could probably expect more significant degradation of the performance auto-calibration with more non-road structures, *e.g.* walls, trees, in the camera field of view. As already pointed out, it is recommended to perform calibration under conditions that satisfy the assumptions. The only situation when our auto-calibrating visual odometry failed during this test case happened when the effects of significant nonplanar motion and nonplanar structures were combined with self-shadowing.

For the completeness of our evaluation in Table 4.2 we include results of the same test as in Section 4.6.2. Here we divided the whole 2.5 km long trajectory into a sequence of shorter, approximately 200 metre long sections and ran the auto-calibration on each individual sequence. As already pointed out, the estimated values of the roll and pitch angles vary along the trajectory, but generally the values match the ground truth parameters.

	GT	estimate
roll [°]	0.0	-0.87 ±0.68
pitch [°]	66.0	65.02 ±1.35
height [mm]	1015	1031.99 ±8.51

Table 4.2: Mean and standard deviation of estimated roll and pitch angles, as well as the height of the camera.

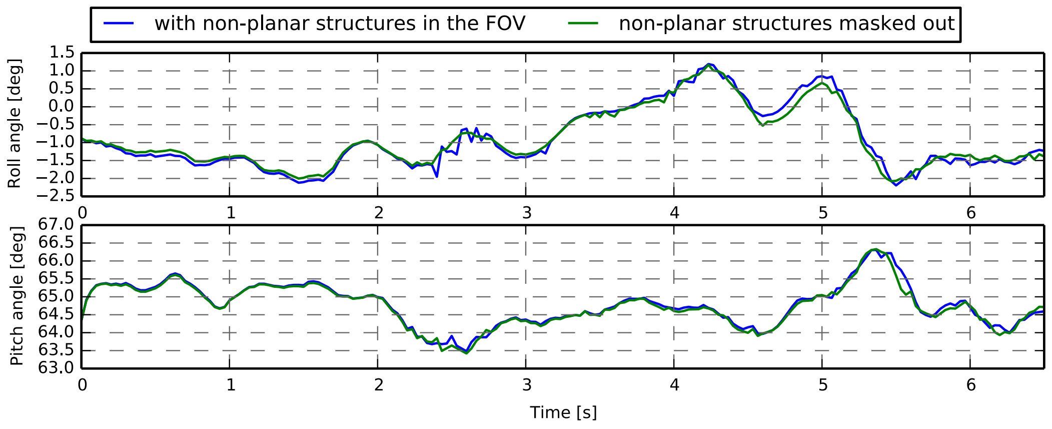


Figure 4.16: Calibration with nonplanar structures in the field of view. We do not observe a significant difference between the values obtained on a sequence where a nonplanar structure was visible all the time, and the same sequence but with the nonplanar structures manually masked out.

4.6.4 Calibration using Nonholonomic Constraints and Direction of Motion

In the final experiment we tested the ability of the method that exploits the nonholonomic constraints of a robot and motion priors to accurately estimate the calibration of the camera. This calibration procedure has two stages: first, a nonholonomic graph is built and estimates of x_{vc} and yaw angle, γ_{vc} are produced. Here, no source of reference is used. Next, a second graph is constructed, where we use the already estimated x_{vc} and γ_{vc} together with information about the direction of motion to determine the remaining degree of freedom, the y_{vc} coordinate of the camera pose in the robot frame of reference. For both graphs the same sequence of camera motion can be used. We assumed that the height of the camera is known (we used the values obtained using the standard procedure as it makes comparisons easier), and that the roll and pitch angles were known and the frame-to-frame tracking was correct.

For three different camera configurations (the same as in Section 4.6.2, on the surface 'carpet 1') we compare the estimates obtained from the standard calibration procedure that uses wheel odometry with the nonholonomic calibration. As with the experiments on different surfaces (Section 4.6.2), we divided the longer trajectory into a set of shorter, overlapping trajectories. This way we obtained hundreds of test trajectories that allow us to perform a statistical analysis of the performance of the proposed method. For all trajectories the initial conditions were the same, $x_0 = 200$,

	x [mm]	yaw [°]
config. 1: standard	241.7 ±5.5	-9.2 ±0.6
config. 1: nonholonomic	241.6 ±5.4	-9.2 ±0.5
config. 2: standard	197.9 ±1.2	75.4 ±0.1
config. 2: nonholonomic	198.6 ±2.7	75.4 ±0.1
config. 3: standard	223.0 ±5.8	-18.4 ±0.4
config. 3: nonholonomic	224.9 ±5.4	-18.4 ±0.4

Table 4.3: Results of the calibration using the standard procedure that relies on wheel odometry and nonholonomic calibration. Note a very good agreement between the parameters obtained using the two independent strategies for three different camera configurations.

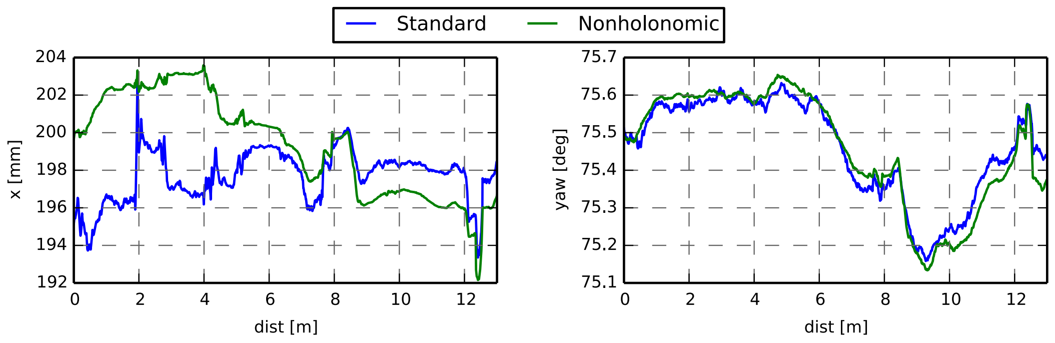


Figure 4.17: Comparison of the calibration using nonholonomic constraints and the standard procedure for the second tested configuration in Table 4.3.

and $\gamma_0 = 0$. The results are presented in Table 4.3[1].

We obtained a remarkable agreement between the values calculated using the two different calibration strategies. Fig. 4.17 shows how the estimates evolve over time for one of the tested camera configurations. We see that the time series behave quite similarly and the difference in the estimated x_{vc} values can be ascribed to the fact that in the standard calibration procedure the height is estimated continuously, whereas for the nonholonomic calibration a constant value of camera height was used.

Having shown that x_{vc} and γ_{vc} can be reliably estimated using nonholonomic calibration, we are ready to demonstrate that the remaining degree of freedom, the y_{vc}

[1]The values obtained for standard calibration differ slightly from the values presented in Section 4.6.2 (Table 4.1) as here it was performing only a continuous graph calibration, whereas previously we ran a completed 6 DoF calibration for each trajectory segment.

coordinate of the camera pose in the robot frame of reference, can be obtained using our motion priors formulation, *i.e.* when for every VO measurement we only know whether the robot was moving forward or backward. Obviously, this formulation puts more requirements on a trajectory that are suitable for calibration. In fact we found that most of the trajectories the robot executed during our test were not suitable for this kind of calibration, as the robot was mainly moving forward. A suitable trajectory features a rich combination of forward and backward motions with a range of angular and linear velocities.

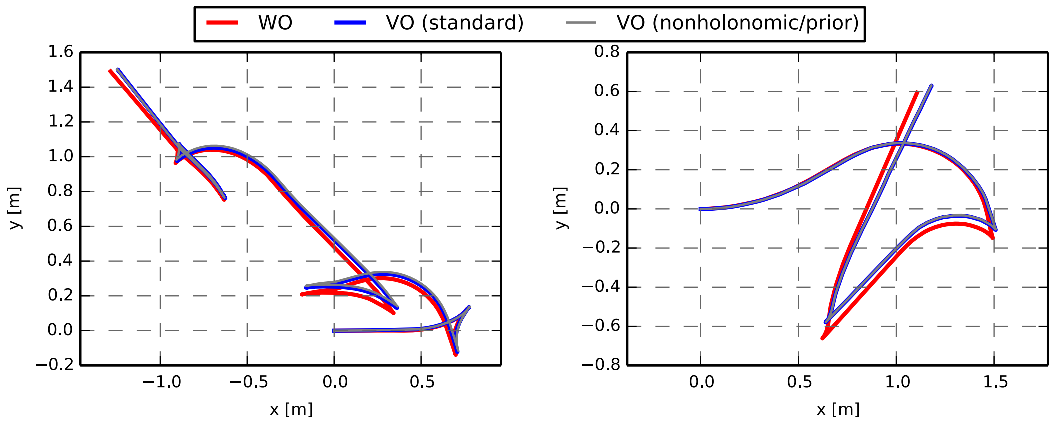


Figure 4.18: Robot trajectories obtained from WO ground truth compared with VO calibrated using our standard method and nonholonomic/direction of motion calibration. We see very good results independent of the calibration method.

To perform this evaluation we executed the following steps. Assuming that the roll and pitch angles are already estimated, and the scale is also known, we first performed nonholonomic calibration to determine x_{vc} and the yaw angle. Next, we used the wheel odometry to calculate the directions of robot motion between consecutive camera frames and constructed a graph as explained in Section 4.5.2. Fig. 4.18 compares trajectories obtained using standard graph based auto-calibration as well as the proposed nonholonomic/motion direction calibration; a comparison of values obtained is also shown in Table 4.4. We can see that for the two independent calibration strategies, both the calibration values and the trajectories match very closely.

	x [mm]	y [mm]	yaw [°]
config. 1: standard	241.6	-9.1	27.1
config. 1: nonholonomic + prior	241.8	-3.7	27.0
config. 2: standard	243.3	-24.6	-9.2
config. 2: nonholonomic + prior	243.2	-25.1	-9.1

Table 4.4: Calibration results obtained using nonholonomic constraints and direction of motion compared with the values calculated for the same trajectories using the standard calibration procedure that relies on synchronised wheel odometry. Trajectories are depicted in Fig. 4.18. We see a very good agreement between these two methods.

4.7 Conclusions

In this chapter we demonstrated that our dense planar visual odometry can be precisely auto-calibrated in highly practical settings. We separated the calibration into two steps: vision and graph-based calibration. First we showed that the camera roll and pitch angles can be simultaneously estimated while performing visual tracking. This was possible by exploiting one of our main assumptions that the ground is flat and that the camera moves parallel to it, and by parameterising the dense image alignment with respect to both planar motion and camera orientation. The graph-based calibration allows us to estimate the remaining degrees of freedom of the camera pose on the robot. The graph-based approach is a more general method and is not restricted to cameras, but can also be used in conjunction with other types of sensors. The only requirement is that a sensor is able to measure incremental planar motion. In case of laser scanners this can be achieved, *e.g.* using ICP. We demonstrated the practicality of our approach through an extensive evaluation. In particular, the method based on nonholonomic constraints permits usable and infrastructure-free auto-calibration, which opens the door to fault detection and lightweight re-calibration in the field.

A possible extension of our work can use continuous-time trajectory representations, *e.g.* [Lovegrove et al., 2013; Furgale et al., 2012] to take advantage of information and enable calibration from unsynchronised sensors. In addition, the ease that continuous-time representations offer in formulating and evaluation constraints on a robot's motion and velocity can be beneficial in particular for nonholonomic calibration.

Chapter 5

Height Map Fusion from Depth Maps

Contents

5.1 Introduction

With the visual odometry system from Chapter 3 and the multi-view stereo approach from Chapter 2 we can perform efficient depth map estimation and therefore equip a robot with the capability to perceive its environment in 3D. However, since the depth maps tend to be noisy and ambiguous, their direct usefulness, in particular for robot navigation, is quite limited and therefore it is required to fuse them into a consistent model of the robot's environment. Usually the monocular reconstruction approach has concentrated on high quality multi-view depth map estimation, and then fusion into a generic 3D representation such as a truncated signed distance function (TSDF), a surfel cloud or a mesh. Some of the systems presented in this vein have been impressive, but we note that there are few examples of moving beyond *showing* real-time dense reconstruction towards *using* it in applications. The live reconstructions are often in a form which needs substantial further processing before they could be useful for any in-the-loop use such as path planning or scene labelling.

We show that a way of achieving efficient, practical and robust use of dense monocular reconstruction is to take a more application-directed approach to depth map fusion and environment representation. What is important in applications including mobile robotics is to calculate measurements from a 3D model such as the drivable paths of maximum pre-defined height or distances to contact along particular directions; or segmentation into semantically meaningful regions or objects. Starting with this chapter we will present different methods and approaches to height map estimation, and show that height maps can offer a viable alternative to more generic environment representations, as they achieve an interesting balance between expressiveness and complexity. The key underlying idea behind height mapping is to represent a robot's environment as a surface and only store the 2.5D elevation information on a two dimensional grid. In contrast to generic 3D representations, *e.g.* volumetric approaches, height maps are very compact (they consume significantly less memory), and their limited expressiveness can sometimes result in improved robustness. Furthermore, a height map is directly applicable for a wheeled mobile robot, and can easily be used for free-space detection. We will also see that height maps fusion can be expressed and extended in various interesting ways. Although we focus here on a height maps as a data structure for representing a robot's environment, the methods presented here can be understood in a broader context as dense surface reconstruction from a set of depth maps.

We define height map fusion as fast and incremental reconstruction from a stream of depth maps where each depth measurement has an estimate of its uncertainty. This is in contrast to the existing large body of work that defines reconstruction as a single, batch process. In incremental fusion, the map has to represent all measurements and observations up to the current time point, and with each new measurement we do not want to recompute everything from scratch.

In this chapter we will lay down the foundations of height map fusion, and present two simple and efficient ways of creating surface models from noisy depth estimates. The two following chapters, Chapters 6 and 7, will build on top of the results presented here.

5.2 Background

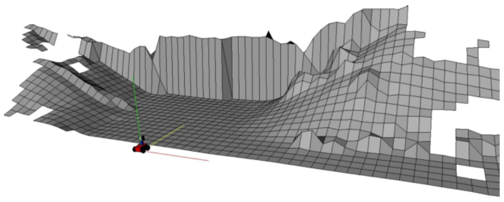
We will start by reviewing different representations for a robot's environment as well as techniques for depth map fusion. An overview of the most relevant work with respect to height mapping will be presented in Section 5.2.2.

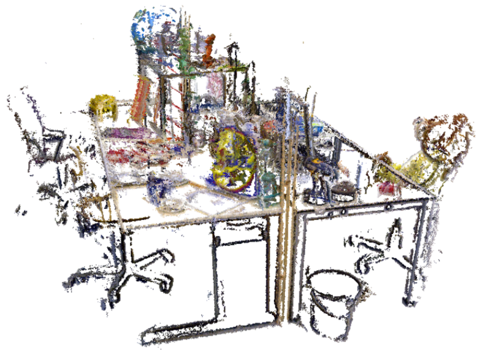
5.2.1 Environment Representations and Depth Map Fusion

There are many approaches for depth map fusion and possible representations of a robot's environment, and they differ considerably in their complexity, expressiveness and how easily can they be applied to particular tasks. Factors like the memory and computational requirements, as well as scalability, can also play an important role when designing and selecting a method for representation and reconstruction. We will review here the most commonly used approaches such as occupancy mapping, point clouds and volumetric methods.

Occupancy mapping [Elfes, 1987] is one of the oldest and simplest techniques for mobile robot map representation. It was first developed for use with sonars that horizontally scan a robot's surrounding and later also successfully adapted for laser scanners used in a similar fashion. An occupancy map models the environment as 2D grid where each cell stores a probability of being free or occupied. The occupancy measurement function is modelled directly and the map can be updated efficiently thanks to the log-odds approach. Using depth maps for 2D occupancy mapping is however not straightforward as it is not obvious how to model probability of occupancy from depth measurements when the surface is observed from an oblique

(a) Occupancy grid map (Image from [Thrun, 2002]).


(b) Height map (Image from [Pfaff et al., 2007]).


(c) Semin-dense point cloud (Image from [Mur-Artal and Tardós, 2015]).

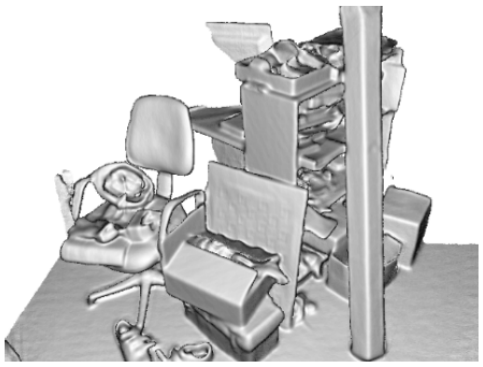

(d) Volumetric reconstruction using TSDF (Image from [Newcombe et al., 2011a]).

Figure 5.1: Examples of different possible representations of a robot's environment.

angle. [Wurm et al., 2010] presented an extension of occupancy mapping to volumetric, 3D environments, but at the cost of increased complexity and memory requirements. Occupancy maps are still used in different robotics applications, *e.g.* [Lukierski et al., 2015] created occupancy maps from omnidirectional depth maps to enable rapid understanding of the free space around a robot and an efficient strategy for further exploration. The main advantage of occupancy mapping is that it can be directly applied to robot navigation, but it is very limited in any further tasks.

Occupancy mapping can be seen as an example of simple semantic mapping, and therefore somewhat related are approaches that perform a semantic segmentation of the scene directly from images based on appearance. Quite often these methods were used as a sensing modality complementary to conventional laser scanners or stereo systems. [Dahlkamp et al., 2006] presented a semi-supervised road detection method based on a monocular camera. They trained on-line a Gaussian Mixture Model (GMM) of the drivable surface based on the measurements from a laser scanner and the output from a probabilistic terrain analysis method [Thrun et al., 2006a], and used it to detect drivable area far ahead the vehicle, beyond the range of laser scanners. It was reported that this approach allowed the vehicle to maintain higher speed compared to the system based on the laser scanners only, and therefore win the DARPA Grand Challenge.

Another example of learning long-range vision for autonomous off-road driving that uses quite similar principles is the method proposed by [Hadsell et al., 2009]. This approach relies on the input from a stereo camera and a real-time classifier is trained on discriminative features extracted from (large) image patches using a pre-trained neural network. The network itself is not trained in real-time. The algorithm is capable of distinguishing five classes: super-ground, ground, footline, obstacle and super-obstacle. Appearance-based methods were also developed within DARPA's Learning Applied to Ground Robots (LARG) project [Konolige et al., 2009; Procopio et al., 2009], which focused on autonomous, often off-road, outdoor driving. The settings the systems were designed for tend to be relatively texture-rich, and unlike indoor environments, quite consistent in appearance. Furthermore, one of the main limitations of purely appearance-based approaches is that colour might not be discriminative enough and does not provide the contextual information required for recognition. As pointed out by [Dahlkamp et al., 2006]: "A brown object on a brown dirt road without significant shadows is undetectable".

Much richer geometric representation of an environment can be obtained using approaches that explicitly map the world in 3D. Here we can distinguish point and surfel clouds as well as volumetric methods. Point clouds of features are commonly used in various successful vision-based SLAM systems, *e.g.* PTAM [Klein and Murray, 2007], OKVIS [Leutenegger et al., 2014], ORB-SLAM [Mur-Artal et al., 2015], and LSD-SLAM [Engel et al., 2014]. Points on a map are usually stored in a set of distinct keyframes, but the points themselves are un-organised and unconnected so they cannot represent a surface. Although very efficient for monocular camera tracking and mapping, this representation is rather abstract and too sparse to be useful for robot navigation since it is not equipped with a notion of free-space or any other semantic meaning.

The concept of keyframes has been extended to dense surfaces, *e.g.* in [Newcombe and Davison, 2010; Newcombe et al., 2011b], were a map is represented as a collection of detailed textured depth maps that can produce a surface patchwork with millions of vertices. This approach produces visually pleasing reconstructions, but, when applied to a practical system one has to solve may non-trivial issues including a strategy for keyframe selection and data association, or a lack of sensible uncertainty representation.

Recently, surfel-based methods have received significant attention. These methods use a technique from computer graphics that represents objects using a set of small surface elements (surfels), [Pfister et al., 2000], and can create dense models. Surfel-based methods work well with dense depth maps obtained using monocular systems [Weise et al., 2009], as well as with quality inputs from active hand-held depth cameras [Keller et al., 2013]. Surfels can support loop closure [Weise et al., 2009; Whelan et al., 2015] and are capable of generating visually impressive large scale reconstructions. The main advantage of surfel-based methods is their scalability. However, similarly to feature-based methods, it is still rather an abstract representation that cannot model closed-surfaces and requires significant post-processing in order to be applied to particular tasks.

Volumetric approaches, in particular based on the signed distance function (SDF) [Curless and Levoy, 1996], are probably the most expressive representations for 3D reconstruction (as shown *e.g.* in [Newcombe et al., 2011a]) and in particular for a robot and motion planning they are quite suitable as at any point in the space they have readily available information about the closest distance to an obstacle. Furthermore, the fusion of dense depth maps into 3D using a signed distance function is very straightforward. However, the cubic memory requirement limits the scalability of volumetric approaches and requires use of octree-data structures or other memory management [Nießner et al., 2013; Prisacariu et al., 2014; Kahler et al., 2015].

5.2.2 Height Mapping

Height maps offer an interesting compromise between purely planar segmentation into free space and obstacles and more generic full 3D representations, and they have been used in mobile robotics for decades. Different techniques have been proposed for height mapping and we will review here the most relevant. Depending on the field and the application many alternative names are used to describe height maps:

e.g. elevation maps, height fields or digital elevation maps (DEM) and digital terrain maps (DTM).

One of the first examples of height mapping was employed in the Ambler project for a legged robot designed for planetary exploration [Bares et al., 1989; Herbert et al., 1989; Kweon and Kanade, 1992]. There, the map was represented using a grid as a set of independent height cells and the measurements, obtained from a stereo system, were fused using the Kalman filter update rule in a similar way as we will describe in Section 5.4. Simplification of the environment as a surface was particularly beneficial for a legged robot operating outdoors and on uneven surfaces. The same simple concept of independent height cells and a per-cell Kalman filter to estimate the elevation was an important building block of navigation and perception systems for many other autonomous mobile platforms, *e.g.* in [Lacroix et al., 2002]. [Triebel et al., 2006] and [Pfaff et al., 2007] applied height maps for outdoor localisation and mapping using laser scanners. Their approach extended traditional height maps in order to perform loop closure as well as offering the ability to model overhangs through the concept of multi-level height maps. The paper by [Fankhauser et al., 2014] presents a method for robot-centric height mapping, where a local height map moves along with the robot's motion. As the robot is moving, the uncertainty of its position and orientation are explicitly handled in the fusion algorithm.

[Gallup et al., 2010a] used a height map as an efficient and powerful model for street-level reconstruction of vertical house facades in a Google Street View fashion. In this context the main advantage of a height map over more general 3D reconstruction techniques was the ability to produce purely vertical structures and continuous surfaces without holes. Also the method was efficient, and able to produce more compact models at the expense of losing small details. The method was used to perform a batch rather than an incremental reconstruction in a following way. Dense depth maps obtained from a stereo camera are first fused into a volumetric occupancy map, from which a height representation is extracted. Subsequently, a triangular mesh is created from the height map, and it is textured using the captured images. [Gallup et al., 2010b] further extended the algorithm to support n-layer height maps and therefore model overhangs, whereas [Häne et al., 2011] used two level height maps for representing indoor environments in a form suitable for robot navigation. They also used a variational formulation of the problem that enabled regularised results. It is noteworthy that the described algorithms rely on GPU parallelisation to achieve high performance.

5.3 Height Map Estimation Preliminaries

5.3.1 Surface Model

A height map is a two-dimensional grid representation of a surface, where each cell on the grid stores a height value of the surface. The connectivity and the interpolation scheme between the height values define different types of surfaces. We obtain a piecewise constant model when we assume that each height is independent, or a piecewise linear model is essentially a triangular mesh, whereas a bilinear surface is obtained when we perform a bilinear interpolation between the heights.

More precisely, for the purpose of this work we will consider as a height map a parameterised surface $\mathcal{S}$ that, given a position on the surface (x, y), can predict a height value:

$$z = f(x, y) \ . \tag{5.1}$$

In particular, for the purpose of this thesis we will only deal with linear models, *i.e.* models for which any point on the surface can be expressed as a linear combination of some basis functions:

$$f(x, y) = \sum_{i=1}^{n} h_i B_i(x, y) \ , \tag{5.2}$$

where $B_i(x, y)$ denotes a basis function, and h_i is an element of a vector parameterising the surface. The type of basis function and its domain, as well as the nature of the coefficients parameterising the surface, determines a family of surfaces, and generally with this formulation we can obtain surfaces of different parameterisations and complexities. In the following we will only concentrate on the piecewise constant and piecewise linear (triangular mesh) surface models parameterised by a vector of height $\mathbf{h}$, mainly because of the computational advantages of working with these representations. However, our formulation can easily be extended to other types of surface models, *e.g.* piecewise bilinear surfaces, cubic B-splines, or those parameterised by wavelet coefficients. Furthermore, we will assume that the resolution and the topology of the surface representation are fixed.

The somehow abstract notion of the basis functions is, in practice, simple to implement and the evaluation of the surface at any point is straightforward. In a

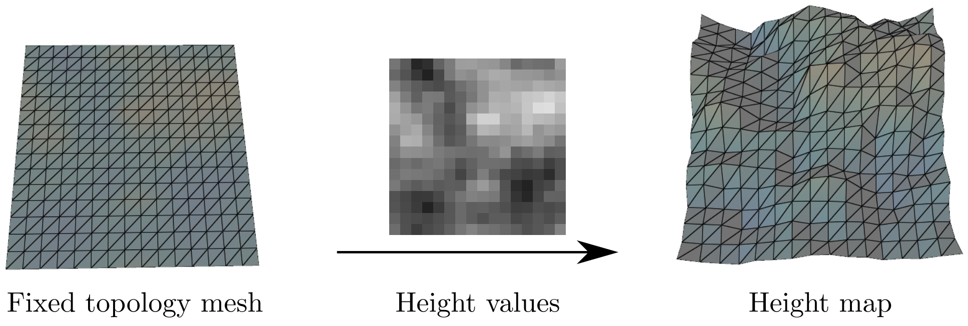


Figure 5.2: In most cases we will represent the surface using a fixed topology, triangular mesh where each vertex has only one degree of freedom, its height.

piecewise constant surface representation model, each basis function B_i is given by an indicator function:

$$B_i(x, y) = \begin{cases} 1, & \text{if } x_i < x \leq x_{i+1} \text{ and } y_i < y \leq y_{i+1} \ , \\ 0, & \text{otherwise} \ , \end{cases} \tag{5.3}$$

where the x_i's and y_i's define an non-overlapping domain for the basis function. This means that in order to predict a height value for a particular point in a piecewise constant model, we simply need to round the coordinates and perform a look-up into memory.

When points associated with heights are treated as vertices of a triangular mesh, the surface can be thought of as of piecewise linear 2D spline. Using a triangular mesh allows us to obtain a continuous but not smooth surface. To evaluate a point (x, y) on a surface modelled by a triangular mesh, we calculate the barycentric coordinates (u, v, w) of that point within the triangle the point is associated with, and then calculate the height by taking the weighted average of the heights spanning the triangle:

$$z = uh_u + vh_v + wh_w \ . \tag{5.4}$$

Therefore, the basis function in this case has the form of barycentric interpolation.

5.3.2 Measurements

The input to our fusion technique is a stream of depth maps. We denote a depth map by $\mathcal{D}$ and summarise it by a vector $\mathbf{d} \in \mathbb{R}^m$, where $\mathtt{T}_{wc}$ is an associated camera pose. We do not put any assumptions or constraints on the nature of the depth maps: we will work with the depth maps obtained using our approach described in Chapter 2; however, our fusion method can be also applied to the data received from a RGB-D camera.

In Chapter 2 we showed how we can transform each depth measurement d into a 3D point and therefore a height measurement z. To recall: consider a pixel location (u, v) with the associated depth measurement d. Given the camera intrinsic matrix $\mathtt{K}$, the current camera pose $\mathtt{T}_{wc}$, and the depth measurement we can project the image point $\mathbf{p}_c = (u, v)$ into the world frame of reference as follows:

$$\mathbf{p}_w = \mathtt{T}_{wc} d \mathtt{K}^{-1} \dot{\mathbf{p}}_c \ . \tag{5.5}$$

Here, $\mathbf{p}_w = (x_w, y_w, z_w)$, represents a point in the world frame of reference. The point $\mathbf{p}_w$ can be seen as a height measurement, z_w, at a grid location (x_w, y_w).

Using our notation for $\mathtt{K}$ and $\mathtt{T}_{wc}$ we can explicitly express the height as:

$$h_w = \left(r_{33} + r_{31} \frac{u - u_0}{f_u} + r_{32} \frac{v - v_0}{f_v} \right) d + t_3 \ , \tag{5.6}$$

or generally:

$$\mathbf{p}_w = \mathbf{a} d + \mathbf{t} \ , \tag{5.7}$$

where the vector $\mathbf{a}$ is given by:

$$\mathbf{a} = \begin{bmatrix} r_{13} + r_{11} \frac{u-u_0}{f_u} + r_{12} \frac{v-v_0}{f_v} \\ r_{23} + r_{21} \frac{u-u_0}{f_u} + r_{22} \frac{v-v_0}{f_v} \\ r_{33} + r_{31} \frac{u-u_0}{f_u} + r_{32} \frac{v-v_0}{f_v} \end{bmatrix} \ . \tag{5.8}$$

We also assume that each depth measurement has an associated uncertainty estimate, and we transform this into height uncertainty using the rules of propagation of uncertainty.

5.3.3 Surface Estimation as a Least Squares Problem

We formulate surface estimation as a least squares problem, where we fit our height observation into a surface model. Specifically, given a set of m measurements $P_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ and a surface parameterised by a vector $\mathbf{h} \in \mathbb{R}^n$, our goal is to find a surface that minimises the sum of squared errors between observations z_i and predictions $f(x_i, y_i)$ from the model:

$$\sum_{i=1}^{m} \frac{1}{\sigma_{z_i}} (z_i - f(x_i, y_i))^2 \ . \tag{5.9}$$

As previously stated, we only consider models where each point on a surface can be expressed by a linear combination of the surface parameters $\mathbf{h}$. Thus, by denoting $\mathbf{z} \in \mathbb{R}^m$ as vector of height measurements, we can express the minimisation problem as follows:

$$\arg\min_{\mathbf{h}} F(\mathbf{h}) = \arg\min_{\mathbf{h}} ||\mathbf{z} - \mathtt{J}\mathbf{h}||^2_{\Sigma_h} \ , \tag{5.10}$$

where each row of the matrix $\mathtt{J} \in \mathbb{R}^{m \times n}$ corresponds to a single height measurement, and Σ_h is the diagonal covariance matrix representing the uncertainties of the height measurements.

The error function $F(\mathbf{h})$ in Eq. (5.10) is a quadratic function of the vector $\mathbf{h}$:

$$\begin{aligned} F(\mathbf{h}) &= ||\mathbf{z} - \mathtt{J}\mathbf{h}||^2_{\Sigma_h} & (5.11)\\ &= \mathbf{h}\mathtt{J}^\top \Sigma_h \mathtt{J}\mathbf{h} - 2\mathbf{h}^\top \Sigma_h \mathtt{J}^\top \mathbf{z} + \mathbf{z}^\top \Sigma_h \mathbf{z} \ , & (5.12)\end{aligned}$$

thus finding the least squares estimate of the surface corresponds to solving the following normal equation:

$$\mathtt{J}^\top \Sigma_h \mathtt{J}\mathbf{h} = \mathtt{J}^\top \Sigma_h \mathbf{z}. \tag{5.13}$$

The structure of the matrix $\mathtt{J}^\top\mathtt{J}$ depends on the surface model we use in the estimation. As we already mentioned, the surface may consist of bivariate polynomials, tensor product B-splines or NURBS, piecewise Bezier patches or triangular splines, but in the following we will consider only two types of surface: piecewise constant and piecewise linear (triangular mesh). We will describe efficient ways for implementing surface estimation using those representations.

5.4 Simple Height Map Fusion

The most commonly used and straightforward method for height map estimation treats each cell on a map as independent and performs per cell a simple averaging. Fusion in this form is still least squares estimation, but one of a special structure: the fact that each cell is independent makes the $\mathtt{J}^\top\mathtt{J}$ matrix in Eq. (5.13) diagonal, which renders solving the normal equation trivial. This is the reason why one rarely forms and solves the normal equation explicitly, but instead represents each height on a map as Gaussian distribution $\mathcal{N}(h_p, \sigma_p^2)$ parameterised by a mean height, h_p, and a variance σ_p^2.

In order to fuse a height measurement z_i with associated position on the map (x_i, y_i) and height uncertainty σ_i^2, we first identify the cell closest to the height measurement $(x_p, y_p) = (round(x_i), round(y_i))$, and next we update the distribution of the height $\mathcal{N}(h_p, \sigma_p^2)$ based on a noisy height observation $\mathcal{N}(z_i, \sigma_i^2)$ using the standard formula for adding two Gaussian distributions [Herbert et al., 1989]:

$$\begin{aligned} h_p &\leftarrow \frac{h_p\sigma_i^2 + z_i\sigma_p^2}{\sigma_p^2 + \sigma_i^2} \ , \\ \sigma_p^2 &\leftarrow \frac{\sigma_p^2\sigma_i^2}{\sigma_p^2 + \sigma_i^2} \ . \end{aligned} \tag{5.14}$$

This formula performs incremental weighted averaging, with weights proportional to the inverse variance of the measurements.

5.4.1 Parallel Implementation

The fact that we treat each height independently results in a very simple and fast algorithm. The method also has minimal memory requirements. To represent the map, for each cell we only need to store its mean height and uncertainty; on a GPU this can be quite conveniently implemented using `float2`.

However, because multiple measurements can be mapped to the same cell, the update computations described in Eq. (5.14) make the algorithm serial, *i.e.* before we can process with a new measurement we have to make sure that both steps in Eq. (5.14) from a previous measurement have been completed. This is not a problem when the algorithm is implemented using a single thread on a CPU that proceeds through all depth measurements one after another. However, implementing the algorithm on a parallel processor is more problematic. On a GPU, in the extreme case we can execute as many threads as the number of measurements in a depth map and thus it is very likely that multiple threads will try to access (read) and update (write) the same cell (memory) location. This can lead to inconsistency and so called race conditions, where the result of an operation completed by one thread can be overwritten by another thread.

In order to prevent race conditions, access to the critical memory locations has to be serialised. In concurrent programming, there are various techniques that allow coordinated access to resources, and they can be divided into three categories:

- locking, based on locks and mutexes,
- wait-free, using atomic operations natively supported by the hardware, where each thread updates memory atomically,

- lock-free.

In the locking strategy all threads try to obtain a lock. The one that succeeds, first completes its work and then releases the lock. Locking guarantees exclusive access to data, but there are several issues with it, particularly when implemented on a GPU. Due to warp divergence, using locks on a GPU is far from straightforward and can easily lead to deadlocks. Furthermore, locks/mutexes can be effective when the number of resources that need to be protected is moderate. In our case we would need a lock for each height cell, *i.e.* the total number of the locks would be equal to the size of the map, therefore greatly increasing the memory requirements, and negatively impacting the efficiency of the method.

Wait-free methods rely on atomic operations that are intrinsically supported by most processors. An atomic operation is an uninterruptible read-modify-write memory operation at a specific address that serializes contentious updates from multiple threads. The advantage of atomic operations is that they are relatively quick compared to locks, and do not suffer from deadlock and convoying. However, their main disadvantage is that standard atomic operations are limited to very specific functions and data types, and often these are not enough to synthesise more complicated operations efficiently. With CUDA 7.5, NVIDIA GPUs support only the following operations: addition (`atomicAdd`), subtraction (`atomicSub`), exchange (`atomicExch`), min (`atomicMin`), max (`atomicMax`), increment (`atomicInc`), decrement (`atomicDec`), and compare-and-swap (`atomicCAS`); and only using certain data types, *i.e.* `int`, `unsigned int`, and `unsigned long long int`. In other programming models (*e.g.* OpenCL) and processors the availability of atomic operations might vary.

In our problem it seems difficult to synthesise the height update, Eq. (5.14) efficiently using atomics and without race conditions, as there are two separate memory reads, two updates and two writes:

- read current height, h_p,
- read current sigma, σ_p^2,
- calculate new height,
- calculate new sigma,

- write new height,
- write new sigma.

Fortunately, the atomic operation *compare-and-swap* (`atomicCAS`) allows for implementation of arbitrary atomic operations and therefore the creation of more sophisticated algorithms. Compare-and-swap writes a new value into a location only if the latter's contents match a supplied old value. The fact that `atomicCAS` can be used to mimic any coordination primitive enables implementation of so called lock-free methods. Specifically, in the lock-free style, all threads try to do the work, write their result, and at least one always succeeds. The threads that fail, repeat.

However, a read-modify-CAS still only applies if the read/write transaction is a single operation. In our case we have to simultaneously read and write both the h_p and σ_p^2. To circumvent this problem we use the fact that NVIDIA GPUs support `atomicCAS` operations also on a 64-bit memory types (`unsigned long long int`) and we use the `float2` data type to encapsulate the mean and sigma. The full read-modify-CAS sequence for the height update is as follows: a thread first reads the value, next computes a new value to write, and subsequently tries to write it using `atomicCAS` (compare-and-swap). If the value changes concurrently, the `atomicCAS` will fail and the thread tries again. Under contention, exactly one thread is guaranteed to succeed. The Listing 1 demonstrates lock-free implementation of Eq. (5.14) in CUDA:

Here we proceed as follows: first we read from a memory location the current estimate of a height and its uncertainty. Next we update their values using observed data, and attempt to write it back to memory using `atomicCAS`. If the value changes concurrently, the `atomicCAS` will fail and we will have to perform the sequence again, *i.e.* read, compute, and try to write.

The described method works well in practice as long as the number of threads trying to update the same memory location is moderate, otherwise a lot of resources are wasted on computations that are discarded. To prevent this we can restrict the number of threads we launch in parallel, *e.g.* one thread per depth map row, that process the data within a row serially. A more efficient parallel implementation can be achieved by explicitly forming the normal equation, Eq. (5.13), *i.e.* assembling the matrix $\mathtt{J}^\top\mathtt{J}$ as well as the vector $\mathtt{J}^\top\mathbf{z}$, and *solving* it when needed. As we already

Listing 1 Lock-free implementation of Eq. (5.14) using CUDA.

```
__device__
float2 atomicKFUpdate(float2* addr, float mean_o, float var_o)
{
  unsigned long long int* ptr  = (unsigned long long int*)addr;
  unsigned long long int  current = *ptr;
  unsigned long long int  new_val;
  unsigned long long int  old_val;
  do
  {
    // Atomically, read-modify-write
    old_val = current;

    // Kalman Filter update
    float2 normal  = __longlong_as_float2(old_val);
    float  mean_p  = normal.x;
    float  var_p   = normal.y;
    float  inv_var = 1.0f/(var_o + var_p);

    mean_p = (var_p*mean_o + var_o*mean_p)*inv_var;
    var_p  = (var_p*var_o)*inv_var;
    // Kalman filter update ends here;

    new_val = __float2_as_longlong(make_float2(mean_p,var_p));
    current = atomicCAS(ptr, old_val, new_val);
  } while (old_val != current);
  return __longlong_as_float2(current);
}
```

stated, the $\mathtt{J}^\top\mathtt{J}$ matrix is diagonal, with one entry per height cell, and for a height map of size $k \times k$, we can store it conveniently on a grid with the identical size. A similar strategy can be applied to the vector $\mathtt{J}^\top\mathbf{z}$. Solving the normal equation associated with this problem is trivial. The entries on the diagonal of $\mathtt{J}^\top\mathtt{J}$ are simply the sum of the σ_i of all measurement uncertainties associated with each cell so far, whereas $\mathtt{J}^\top\mathbf{z}$ is the weighted sum of measured heights. In order to update the normal equation, we still have to perform atomic operations, (two `atomicAdd`, one for the LHS and one on the RHS of Eq. (5.13)), but it is generally faster than the lock-free implementation based on `atomicCAS`. We will use a similar concept of representing the $\mathtt{J}^\top\mathtt{J}$ matrix using a fixed size grid in the following sections.

5.5 Fusion into a Triangular Mesh

Now we will extend our fusion to surface types that explicitly model the connectivity between individual height cells on the grid as is the case in a triangular mesh. The advantages of this approach are that we do not have to rely on heuristics to perform data association (*e.g.* to the nearest cell), and that by modelling the dependencies between height cells we can incorporate smoothness priors into the surface model. Furthermore, when using a mesh, we can quite naturally incorporate colour estimation into our framework. Later on, in Chapters 6 and 7, we will see many interesting extensions of this fusion method that explicitly rely on triangular meshes as a surface representation.

5.5.1 Mesh Representation

In the following we will use a mesh of fixed topology as illustrated in Fig. 5.3. Unlike the simple height fusion, where we typically think of a height map as a collection of independent height points, in this representation each vertex on the mesh is connected with up to 6 neighbouring vertices.

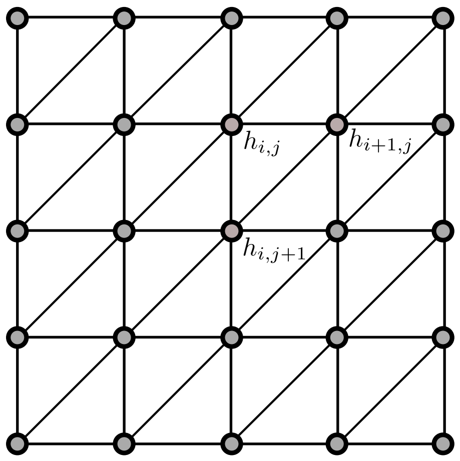


Figure 5.3: Basic structure of the mesh used for fusion.

A simple triangular mesh is essentially a piecewise linear model, where the values between vertices are calculated using barycentric interpolation. A 3D point $\mathbf{p} = (x_i, y_i, z_i)$, can be associated with a triangle of the surface, and we can predict the height at (x_i, y_i) by using barycentric coordinates within this triangle $\mathbf{v}_i = (\alpha_i, \beta_i, \gamma_i)^\top$, in the following way:

$$\hat{z}_i = \alpha_i h_1^{\triangle_i} + \beta_i h_2^{\triangle_i} + \gamma_i h_3^{\triangle_i} \ , \tag{5.15}$$

where $h_1^{\triangle_i}, h_2^{\triangle_i}, h_3^{\triangle_i}$ represent the heights of the triangle associated with the point (x_i, y_i, z_i).

The conversion from the grid coordinates of a point into barycentric coordinates within a triangle on that grid, $(x_i, y_i) \rightarrow (\alpha_i, \beta_i, \gamma_i)$, is straightforward. Assume that

$(x_1^\triangle, y_1^\triangle)$, $(x_2^\triangle, y_2^\triangle)$, and $(x_3^\triangle, y_3^\triangle)$ are the grid coordinates of the 3 vertices defining a triangle. The barycentric coordinates $(\alpha_i, \beta_i, \gamma_i)$ of a point (x_i, y_i) within that triangle are given by:

$$\begin{aligned}
\alpha_i &= \frac{(y_2^\triangle - y_3^\triangle)(x - x_3^\triangle) + (x_3^\triangle - x_2^\triangle)(y - y_3^\triangle)}{(y_2^\triangle - y_3^\triangle)(x_1^\triangle - x_3^\triangle) + (x_3^\triangle - x_2^\triangle)(y_1^\triangle - y_3^\triangle)} \\
\beta_i &= \frac{(y_3^\triangle - y_1^\triangle)(x - x_3^\triangle) + (x_1^\triangle - x_3^\triangle)(y - y_3^\triangle)}{(y_2^\triangle - y_3^\triangle)(x_1^\triangle - x_3^\triangle) + (x_3^\triangle - x_2^\triangle)(y_1^\triangle - y_3^\triangle)} \\
\gamma_i &= 1 - \alpha_i - \beta_i
\end{aligned} \tag{5.16}$$

In general, the computations can be further simplified, because the distances between the vertices on the grid are fixed and known.

To recall, we assume that we have m height measurements, and therefore give rise to the following set of equations:

$$\begin{aligned}
\alpha_1 h_1^{\triangle_1} + \beta_1 h_2^{\triangle_1} + \gamma_1 h_3^{\triangle_1} &= z_1 \\
\alpha_2 h_1^{\triangle_2} + \beta_2 h_2^{\triangle_2} + \gamma_2 h_3^{\triangle_2} &= z_2 \\
&\dots \\
\alpha_k h_1^{\triangle_m} + \beta_k h_2^{\triangle_m} + \gamma_k h_3^{\triangle_m} &= z_m \ ,
\end{aligned} \tag{5.17}$$

where $\triangle_i$ indicates the triangle a particular height measurement is projected onto. Multiple measurements can be associated with the same triangle, and the set of *linear* equations in Eq. (5.17) can be written as:

$$\mathsf{J}\mathbf{h} = \mathbf{z} \ , \tag{5.18}$$

a form we already know from Eq. (5.10). Here, $\mathsf{J} \in \mathbb{R}^{m \times n}$, $\mathbf{h} \in \mathbb{R}^n$ and $\mathbf{z} \in \mathbb{R}^m$, but note that matrix J has only 3 non-zero entries per row. In Section 5.3.3 we showed that estimating the surface from the set of measurements thus amounts to solving the following normal equation:

$$\mathsf{J}^\top \mathsf{J}\mathbf{h} = \mathsf{J}^\top \mathbf{z} \ . \tag{5.19}$$

Here we omitted the covariance matrix of the measurements Σ_h for brevity. However, because of the connectivity between variables in the mesh, $\mathsf{J}^\top \mathsf{J}$ is not diagonal any more, and solving Eq. (5.19) becomes more difficult compared to the simple height fusion method discussed previously. In the following we will describe how to efficiently construct, store and solve the normal equation associated with the mesh.

5.5.2 Storing and Updating $\mathtt{J}^\top\mathtt{J}$ Matrix

A first important observation about the matrix $\mathtt{J}^\top\mathtt{J}$ on the left-hand side of Eq. (5.19) is that it is symmetric and sparse and has a regular structure that reflects the topology of the mesh used. In our case, a single vertex can be connected to only up to 6 neighbouring vertices, so $\mathtt{J}^\top\mathtt{J}$ contains per row a diagonal entry and only up to 6 non-zero off-diagonal entries, as shown in Fig. 5.4. There are several standard ways for storing a sparse matrix, *e.g.* Coordinate List (COO), Compressed Sparse Row (CSR), Compressed Sparse Column (CSC), but they were designed for general sparse matrices, where the sparsity pattern might not be known in advance, and with emphasis on efficient implementation of different operations (*e.g.* matrix construction, matrix-vector product). Since in our case the sparsity pattern is well-known, and we are solving a problem on a regular grid, we can design a custom storage method that will greatly improve and facilitate a parallel implementation.

In our implementation we store the vector of heights $\mathbf{h}$ on a 2D grid, similarly to the method proposed for the simple height fusion. However, we can also exploit the symmetry of the $\mathtt{J}^\top\mathtt{J}$ matrix and its sparsity pattern and rather than storing it using a generic sparse matrix data structure we can represent $\mathtt{J}^\top\mathtt{J}$ conveniently on a regular grid: for a mesh of size $k \times k$ we need a grid of the size $(2k-1)\times(2k-1)$. This is best visualised by the example in Fig. 5.4. Consequently, we also store the right-hand side of the equation, the vector $\mathtt{J}^\top\mathbf{z}$ using a $k \times k$ grid.

Another very important observation about the $\mathtt{J}^\top\mathtt{J}$ matrix is that we do not have to form and calculate $\mathtt{J}$ explicitly, and perform matrix-matrix and matrix-vector multiplication to obtain $\mathtt{J}^\top\mathtt{J}$ and $\mathtt{J}^\top\mathbf{z}$, but instead each individual height measurement can directly update the entries of $\mathtt{J}^\top\mathtt{J}$ and $\mathtt{J}^\top\mathbf{z}$. We have already seen that the coefficients within the matrix $\mathtt{J}$ are simply barycentric coordinates. Consequently, each height measurement z_i updates $\mathtt{J}^\top\mathtt{J}$ at 6 locations (3 diagonal and 3 off-diagonal entries) associated with its triangle, using the coefficient obtained by taking the outer product (weighted when we take uncertainty into account) of the barycentric coordinates $\mathbf{v}_i$ (Eq. (5.15)):

$$\mathbf{v}_i\mathbf{v}_i^\top = \begin{bmatrix} \alpha_i^2 & \alpha_i\beta_i & \alpha_i\gamma_i \\ \alpha_i\beta_i & \beta_i^2 & \beta_i\gamma_i \\ \alpha_i\gamma_i & \beta_i\gamma_i & \gamma_i^2 \end{bmatrix} . \tag{5.20}$$

The above formula also gives an insight into the nature of the entries in the $\mathtt{J}^\top\mathtt{J}$

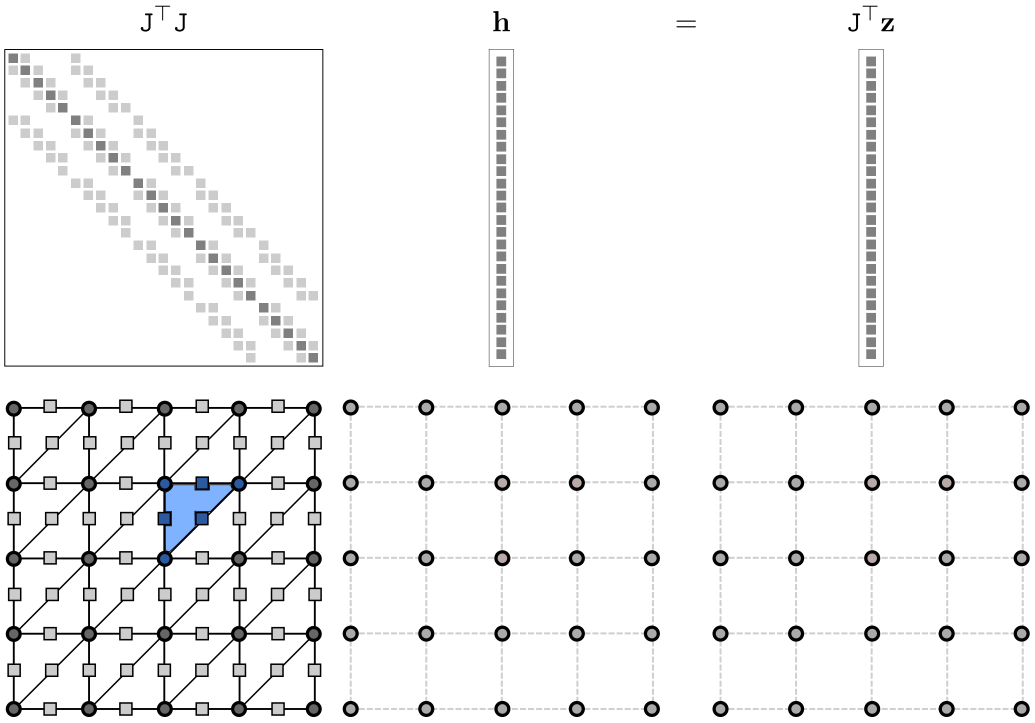


Figure 5.4: We can represent the normal equation $\mathtt{J}^\top\mathtt{J}\mathbf{h} = \mathtt{J}^\top\mathbf{z}$ conveniently on a regular grid. To store the $\mathtt{J}^\top\mathtt{J}$ matrix we exploit its symmetry, and we only require $(2k-1)\times(2k-1)$ grid to represent a $\mathtt{J}^\top\mathtt{J}$ for a $k \times k$ mesh. The blue triangle on the left-hand side of the equation indicates how a single height measurements updates coefficients of the $\mathtt{J}^\top\mathtt{J}$ matrix. There, dots indicate the diagonal entries, whereas squares represent the off-diagonal entries of the $\mathtt{J}^\top\mathtt{J}$ matrix.

matrix. We can see that the entries along the diagonal of $\mathtt{J}^\top\mathtt{J}$ equal the sum of squares of the barycentric coordinates associated with each vertex. When performing the matrix updates on a GPU we have to use atomic operations (`atomicAdd`) when updating individual entries of $\mathtt{J}^\top\mathtt{J}$ and $\mathtt{J}^\top\mathbf{z}$ in order to avoid race conditions, as multiple depth measurements can contribute to the same triangle.

5.5.3 Smoothness Prior and Regularisation

In our fusion approach we can easily incorporate certain priors on the surface properties, *e.g.* that the gradient of the height fields is smooth. This and many other reasonable measures of smoothness can be expressed by a quadratic and positive semidefinite functional. That means that instead of solving the original problem in Eq. (5.13), we will be solving the augmented normal equation:

$$(\lambda\Lambda + \mathtt{J}^\top\mathtt{J})\mathbf{h} = \mathtt{J}^\top\mathbf{z} \ , \qquad (5.21)$$

where λ is a hyper-parameter controlling the regularisation, and $\mathtt{\Lambda}$ is a sparse, positive semidefinite matrix. In practice we only regularise the solution very weakly by assuming that neighbouring vertices have similar values.

5.6 Iterative Solvers

Once the normal equation, Eq. (5.13), is created, we need to solve it in order to estimate the parameters of the height field. Since the matrix $\mathtt{J}^\top\mathtt{J}$ is symmetric, positive definite we could find the solution using Cholesky factorisation and back substitution. However, as the size of the problem increases, this approach becomes impractical, and for large scale least squares problems iterative solvers are more widely used. Iterative solvers preserve the sparsity pattern of the matrix $\mathtt{J}^\top\mathtt{J}$, and are easier to implement on parallel processors. An iterative solver will also have advantages in recursive estimation as we will see in the following section. For an iterative solver we need an initial guess which is subsequently refined. In our case, a reasonable initial estimate can be obtained by running our simple height averaging method. In the following we will present two solvers, the conjugate gradient method and the Gauss-Seidel algorithm.

For the clarity of the derivation, from now on, we will consider solving an equation of the form:

$$\mathtt{A}\mathbf{x} = \mathbf{b} \ , \tag{5.22}$$

effectively substituting $\mathtt{A} = \mathtt{J}^\top\mathtt{J}$, $\mathbf{x} = \mathbf{h}$, and $\mathbf{b} = \mathtt{J}^\top\mathbf{z}$.

5.6.1 Conjugate Gradient

The conjugate gradient method is a very popular choice for solving linear systems of equations involving symmetric positive definite matrices as in our case. It operates based on the observation that solving Eq. (5.22) is equivalent to the problem of minimising the convex quadratic function defined by:

$$\phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top\mathtt{A}\mathbf{x} - \mathbf{b}^\top\mathbf{x} \ , \tag{5.23}$$

because $\mathtt{A}\mathbf{x} - \mathbf{b}$ is the gradient of $\phi(\mathbf{x})$, and at the minimiser, the gradient of $\phi(\mathbf{x})$ should be zero. Indeed, the gradient $\frac{\partial\phi}{\partial\mathbf{x}}$ of the $\phi(\mathbf{x})$ is:

$$\frac{\partial\phi}{\partial\mathbf{x}} = \frac{1}{2}\mathbf{A}^\top\mathbf{x} + \frac{1}{2}\mathbf{A}\mathbf{x} - \mathbf{b} \ , \tag{5.24}$$

and since the matrix $\mathtt{A}$ is symmetric, the expression reduces to:

$$\frac{\partial \phi}{\partial \mathbf{x}} = \mathbf{Ax} - \mathbf{b} \; . \tag{5.25}$$

Conjugate gradient, similarly to the steepest descent method [Nocedal and Wright, 2006], solves Eq. (5.22) iteratively by taking successive steps $\mathbf{p}_k$ in the directions that minimise the objective function (Eq. (5.23)). However, unlike steepest descent, which often takes steps in the same direction multiple times [Shewchuk, 1994, p. 21], descent direction vectors $\mathbf{p}_k$ are $\mathtt{A}$-orthogonal, which guarantees that in each search direction, one takes exactly one step. Vectors $\mathbf{p}_i$ and $\mathbf{p}_j$ are $\mathtt{A}$-orthogonal if they satisfy $\mathbf{p}_i^\top \mathtt{A}\mathbf{p}_j = 0$. In general, this results in much faster convergence and prevents the zigzag pattern characteristic of simple steepest descent. The overall structure of the conjugate gradient method is outlined in Algorithm 1.

Algorithm 1 Conjugate gradient method.

```
1: Given $\mathbf{x}_0$
2: Evaluate $\phi_0 = \phi(\mathbf{x}_0)$ and $\mathbf{g}_0 := \nabla\phi(\mathbf{x}_0) = \mathtt{A}\mathbf{x}_0 - \mathbf{b}$
3: Set $\mathbf{p}_0 := -\mathbf{g}_0$, and $k := 0$
4: while $\mathbf{g}_k \neq 0$ do
5:     $\alpha_k := \frac{\mathbf{g}_k^\top \mathbf{g}_k}{\mathbf{p}_k^\top \mathtt{A}\mathbf{p}_k}$
6:     $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
7:     $\mathbf{g}_{k+1} := \mathbf{g}_k + \alpha_k \mathtt{A}\mathbf{p}_k$
8:     $\beta_{k+1} := \frac{\mathbf{g}_{k+1}^\top \mathbf{g}_{k+1}}{\mathbf{g}_k^\top \mathbf{g}_k}$
9:     $\mathbf{p}_{k+1} := -\mathbf{g}_{k+1} + \beta_{k+1}\mathbf{p}_k$
10:    $k := k + 1$
11: end while
```

The implementation of conjugate gradient is rather straightforward and also its memory requirements are modest. We only need to store the gradient vector of the objective function $\mathbf{g}_k$ and the vector $\mathbf{p}_k$, each of them being the size of the original problem, *i.e.* vector $\mathbf{x}$. For practical reasons, we also store the $\mathtt{A}\mathbf{p}_k$ vector.

When using conjugate gradient for surface reconstruction, we start with an initial estimate of the height, and first calculate the gradient of our objective function, Eq. (5.23). In practice this requires a matrix-vector multiplication, where we multiply the current estimate of the heights $\mathbf{h}$ by the $\mathtt{J}^\top\mathtt{J}$ matrix. On GPU we launch multiple threads in parallel, one thread per entry in the output vector, and each thread calculates its value by accessing up to 7 values in the matrix and 7 neighbouring height values (in the height vector). We use the same procedure in the main loop of the algorithm, whenever matrix-vector multiplication is required, *i.e.* lines 5 and 7 in Algorithm 1. The main computational resources on a GPU are spent on performing dot products, lines 5 and 8, in order to calculate the step sizes α_k and β_k. On GPUs dot product requires performing a reduction, which is not always trivial to implement efficiently. Other steps within the main loop consist of 3 simple vector additions.

Conjugate gradient iterates until a convergence criteria is met, *i.e.* $\mathbf{g}_k = 0$. In practice we stop when the norm of the gradient falls below a certain threshold. One of the limitations of the conjugate gradient algorithm is that its convergence rate depends on the condition number of the $\mathtt{A}$ matrix. In order to speed up convergence one often uses a modified version of the algorithm based on preconditioning that we will discuss in the following section.

Preconditioned Conjugate Gradient

Preconditioning involves transforming the linear system to improve the distribution of the eigenvalue (and condition number) of the $\mathtt{A}$ matrix. To achieve that, instead of solving the original equation:

$$\mathtt{A}\mathbf{x} = \mathbf{b} \ , \tag{5.26}$$

we solve a transformed problem:

$$\mathtt{M}^{-1}\mathtt{A}\mathbf{x} = \mathtt{M}^{-1}\mathbf{b} \ , \tag{5.27}$$

where $\mathtt{M}$ is a symmetric, positive definite matrix that approximates $\mathtt{A}$, but it is easier to invert. The convergence rate now depends on the eigenvalues (and condition number) of $\mathtt{M}^{-1}\mathtt{A}$, which when $\mathtt{M}$ is selected properly can result in solving the transformed system much quicker than the original one.

To circumvent the problem that the matrix $\mathtt{M}^{-1}\mathtt{A}$ might not be symmetric, positive definite, we can use the fact that there is always a factorisation such that $\mathtt{M} = \mathtt{C}^\top\mathtt{C}$. Now, we use matrix $\mathtt{C}$ to transform $\mathbf{x}$ into $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} = \mathtt{C}\mathbf{x} \ , \tag{5.28}$$

and express the quadratic function $\phi(\mathbf{x})$ in term of the vector $\hat{\mathbf{x}}$ as follows:

$$\begin{aligned} \phi(\hat{\mathbf{x}}) &= \hat{\mathbf{x}}^\top\mathtt{C}^{-\top}\mathtt{A}\mathtt{C}^{-1}\hat{\mathbf{x}} - \mathbf{b}^\top\mathtt{C}^{-1}\hat{\mathbf{x}} \\ &= \hat{\mathbf{x}}^\top\left(\mathtt{C}^{-\top}\mathtt{A}\mathtt{C}^{-1}\right)\hat{\mathbf{x}} - \left(\mathtt{C}^{-\top}\mathbf{b}\right)^\top\hat{\mathbf{x}} \ . \end{aligned} \tag{5.29}$$

The original system of equations:

$$\mathtt{A}\mathbf{x} = \mathbf{b} \ , \tag{5.30}$$

has been thus transformed to the following problem:

$$\left(\mathtt{C}^{-\top}\mathtt{A}\mathtt{C}^{-1}\right)\hat{\mathbf{x}} = \mathtt{C}^{-\top}\mathbf{b} \ , \quad \hat{\mathbf{x}} = \mathtt{C}\mathbf{x} \ , \tag{5.31}$$

which is first solved for $\hat{\mathbf{x}}$ and subsequently for $\mathbf{x}$ [Shewchuk, 1994, p. 40]. The matrices $\mathtt{C}^{-\top}\mathtt{A}\mathtt{C}^{-1}$ and $\mathtt{M}^{-1}\mathtt{A}$ have the same eigenvalues and condition number. In practice, we do not have to compute the matrix $\mathtt{C}$, but we can work directly with the matrix $\mathtt{M}$ as outlined in Algorithm 2.

Algorithm 2 Preconditioned conjugate gradient method.

```
 1: Given $\mathbf{x}_0$ and preconditioner $\mathtt{M}$
 2: Evaluate $\phi_0 = \phi(\mathbf{x}_0)$ and $\mathbf{g}_0 := \nabla\phi(\mathbf{x}_0) = \mathtt{A}\mathbf{x}_0 - \mathbf{b}$
 3: Solve $\mathtt{M}\mathbf{y}_0 = \mathbf{g}_0$ for $\mathbf{y}_0$                  ▷ i.e. $\mathbf{y}_0 = \mathtt{M}^{-1}\mathbf{g}_0$
 4: Set $\mathbf{p}_0 := -\mathbf{y}_0$, and $k := 0$
 5: while $\mathbf{g}_k \neq 0$ do
 6:     $\alpha_k := \frac{\mathbf{g}_k^\top \mathbf{y}_k}{\mathbf{p}_k^\top \mathtt{A}\mathbf{p}_k}$                      ▷ determine step-size
 7:     $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
 8:     $\mathbf{g}_{k+1} := \mathbf{g}_k + \alpha_k \mathtt{A}\mathbf{p}_k$          ▷ calculate new gradient at $\mathbf{x}_{k+1}$
 9:     Solve $\mathtt{M}\mathbf{y}_{k+1} = \mathbf{g}_{k+1}$ for $\mathbf{y}_{k+1}$   ▷ i.e. calculate $\mathbf{y}_{k+1} = \mathtt{M}^{-1}\mathbf{g}_{k+1}$
10:     $\beta_{k+1} := \frac{\mathbf{g}_{k+1}^\top \mathbf{y}_{k+1}}{\mathbf{g}_k^\top \mathbf{y}_k}$
11:     $\mathbf{p}_{k+1} := -\mathbf{y}_{k+1} + \beta_{k+1}\mathbf{p}_k$
12:     $k := k + 1$
13: end while
```

The remaining question is how to choose $\mathtt{M}$. Often it is recommended to select $\mathtt{M}$ in such a way that system $\mathtt{M}\mathbf{y} = \mathbf{g}$ amounts to a simplified version of the original system $\mathtt{A}\mathbf{x} = \mathbf{b}$. One popular option is to construct $\mathtt{M}$ based on the incomplete Cholesky factorisation of the matrix $\mathtt{A}$, *i.e.* we set $\mathtt{M} = \tilde{\mathtt{L}}^\top\tilde{\mathtt{L}}$, where $\tilde{\mathtt{L}}$ represents an incomplete factorisation of $\mathtt{A}$, and makes $\mathtt{C}^{-\top}\mathtt{A}\mathtt{C}^{-1} = \tilde{\mathtt{L}}^{-1}\mathtt{A}\tilde{\mathtt{L}}^{-\top} \approx \mathtt{I}$. Incomplete Cholesky factorisation can be a very powerful preconditioner, however, we found it highly impractical for an efficient and simple parallel implementation. A very simplest (although not always the most effective) alternative is to use a Jacobi preconditioner, which creates $\mathtt{M}$ based only the diagonal entries of the matrix $\mathtt{A}$. This appears to be a sensible strategy in our case, as the entries on the diagonal are formed from the sum of squares of the barycentric coordinates, and indeed one can think of it as a simplified version of the original problem.

5.6.2 Gauss-Seidel Algorithm

Gauss-Seidel and similar relaxation-based approaches (*e.g.* SOR) are quite popular ingredients of multigrid methods for solving partial differential equations, and they have already been used successfully in many computer vision applications, *e.g.* in [Szeliski, 1990]. With the Gauss-Seidel algorithm we can directly exploit the regular grid structure of our surface reconstruction problem and we will show that this results in a simple yet very powerful iterative solver that presents an interesting alternative to the conjugate gradient method.

Recall our system of equations:

$$\mathtt{A}\mathbf{x} = \mathbf{b} \ ,$$

and let us write the individual components of $\mathtt{A}$, $\mathbf{x}$ and $\mathbf{b}$:

$$\mathtt{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} , \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} , \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} . \tag{5.32}$$

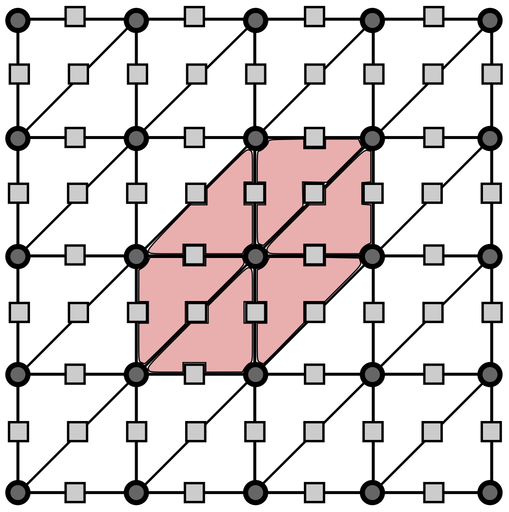
Let us also denote by x_i^k the value of x_i at iteration k and by $x_i^{(k+1)}$ the value at iteration $k+1$.

Rather than using a global loop that updates all components of the solution vector $\mathbf{x}$ at once, a single iteration of the Gauss-Seidel method progresses through the individual elements of the vector $\mathbf{x}$ one after another, and updates them using the following element-wise formula (referred to also as the relaxation formula):

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right) . \tag{5.33}$$

Thus, computation of update value $x_i^{(k+1)}$ within iteration $k+1$ uses only a small subset of the entries in matrix $\mathtt{A}$ and vector $\mathbf{b}$, as well as values from the solution vector that have already been updated $x_i^{(k+1)}$, and values from the previous iteration x_j^k. Intuitively, we solve for $x_i^{(k+1)}$ by fixing and assuming that the remaining elements x_j of the solution vector are correct, and progress through the system of linear equations in $\mathtt{A}\mathbf{x} = \mathbf{b}$, equation after equation.

Since the matrix $\mathtt{A}$ associated with our reconstruction problem has only one non-zero value on the diagonal and up to six non-zero off-diagonal elements (Fig. 5.4), in


(a) Visualisation of the $\mathtt{J}^\top\mathtt{J}$ matrix stored on a grid and the memory access pattern (red area) for an element-wise update, Eq. (5.33). Dots on the grid indicate the diagonal entries of $\mathtt{J}^\top\mathtt{J}$, whereas squares represent the off-diagonal element of the matrix for a single, element-wise Gauss-Seidel iteration.

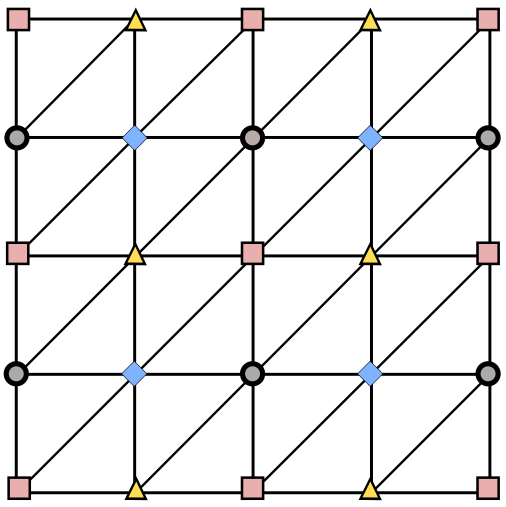

(b) Four-colour based reordering of the variables (vector $\mathbf{h}$) on a grid. Variables within a single grid (*e.g.* squares) are fully independent of each other, and therefore can be updated in parallel.

Figure 5.5: Our parallel implementation of the Gauss-Seidel algorithm exploits and benefits from the regular grid structure of the surface reconstruction problem.

order to update variable x_i using the element-wise update from Eq. (5.33), we only need to access its six surrounding neighbours within the vector $\mathbf{x}$ and the associated off-diagonal entries of the matrix $\mathtt{A}$. In our grid-like storage of $\mathtt{J}^\top\mathtt{J}$, $\mathbf{h}$, and $\mathtt{J}^\top\mathbf{z}$, this corresponds to very local computations and a predictable memory access pattern. In Fig. 5.5a we depicted the variables involved in a single element-wise computation. The computations for a single iteration of the Gauss-Seidel algorithm are therefore much simpler compared to conjugate gradient, as we do not have to perform any matrix-matrix and matrix-vector multiplication, or dot products. Furthermore, as all computations can be performed in-place, implementation of the Gauss-Seidel solver consumes less memory.

The element-wise update rule is straightforward to execute, but the dependencies between unknown, *i.e.* the fact that computations for of particular unknown depend on the updated values of unknowns that have been computed before makes the Gauss-Seidel algorithm essentially sequential. This means that in a standard form the algorithm cannot be implemented efficiently in parallel, and that theoretically we have to sweep through the vertices in our mesh one after another. Fortunately, by using a special ordering of variables in certain cases we can design a parallelisable version of the Gauss-Seidel method.

Since each vertex (variable) on the grid is connected only to a small number of neighbouring vertices, we can identify several subsets of variables, where, within each subset, variables are decoupled from each other. This is referred to as *variable ordering*, and one typical strategy is to perform so-called variable colouring, where a single *colour* defines a maximal subset of variables that are decoupled from each other [Shapira, 2008, p. 93]. Thus, the computations within a single colour can be carried out fully in parallel since there are no data dependencies [Trottenberg et al., 2001, p.176]. In our case we can divide the vertices on the mesh into 4 subgrids (four colours), as shown in Fig. 5.5b. Within a single Gauss-Seidel iteration, the variables are relaxed in the following order: the squares are relaxed in the first stage, the triangles are relaxed in the second stage, the circles are relaxed in the third stage, and the diamonds are relaxed in the final stage.

5.6.3 Comparison of the Solvers

Using a small synthetic dataset (a height map of a size 51×51 with approximately 30000 measurements) and our own reference implementations we evaluated the four iterative solvers: Gauss-Seidel method, standard conjugate gradient, and two versions of preconditioned conjugate gradient, using Jacobi and incomplete Cholesky preconditioners. Fig. 5.6 illustrates the convergence rates of the algorithms on the considered dataset, whereas Table 5.1 includes the total processing times and number of iterations required for each of the methods to solve the problem to a required accuracy ($\epsilon \leq 0.01$).

Looking at Fig. 5.6, as one could expect, the method based on incomplete Cholesky factorisation achieves superior performance in terms of convergence rate. However, upon closer inspection of the total processing times in Table 5.1 we can see that the Jacobi-preconditioned conjugate gradient as well as Gauss-Seidel method perform slightly faster despite requiring greater number of iterations to solve to problem to the same accuracy. This can be attributed to the fact that the method based on Cholesky preconditioning involves matrix factorisation at the beginning of the optimisation process as well as calling a triangular solver at each iteration and this creates additional overhead. Therefore, the computational complexity of a

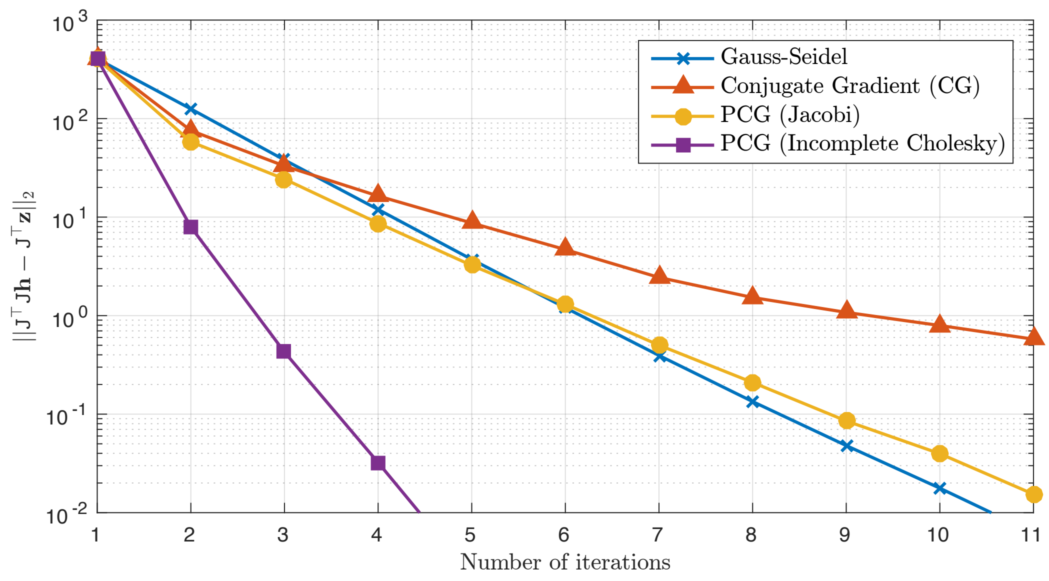


Figure 5.6: Comparison of convergence rates for different iterative solvers.

	CG	PCG (Jacobi)	PCG (Cholesky)	GG
processing time [sec]	0.717	0.334	0.457	0.426
number of iterations	28	11	4	10

Table 5.1: Average processing times and number of iterations required to solve a synthetic least squares problem using reference implementations of four algorithms: CG — conjugate gradient, PCG (Jacobi) — Jacobi-preconditioned conjugate gradient, PCG (Cholesky) — incomplete Cholesky-preconditioned conjugate gradient and GS — Gauss-Seidel method.

single iteration of preconditioned conjugate gradient based on incomplete Cholesky factorisation is higher compared to methods that use simpler preconditioner such as Jacobi. This indicates that in many situations rather than performing small number of complex iterations, it might be advantageous to run more iterations of a simpler method.

When comparing Gauss-Seidel method with the algorithms based on conjugate gradient, at first, the convergence rate seems to be slower. However, that changes as the computations progress and within just 10 iterations we are able to solve the reconstruction problem two orders of magnitude better than the standard conjugate gradient algorithm, and also more accurately than the method based on the Jacobi preconditioning. The total processing times for Gauss-Seidel and Jacobi-preconditioned conjugate gradient are comparable, but by exploiting the structure of the problem the computations within a single Gauss-Seidel iteration are simpler and incur significantly less memory transfer, which makes the Gauss-Seidel method a very attractive solver for the surface reconstruction problems we defined in this chapter.

5.7 Recursive Estimation

In this work we are interested in an incremental reconstruction method where we can process each depth map as it arrives, without the need for recomputing the surface from scratch. Fortunately, this is easily achievable within our framework and we process data as follows: with every new frame, we first update $\mathtt{J}^\top\mathtt{J}$ and $\mathtt{J}^\top\mathbf{z}$ and subsequently run a few iterations of the iterative solver. However, it is not necessary that the solver converges; since we are solving a linear least squares problem iteratively, we can always stop the solver and add new data (*i.e.* update the $\mathtt{J}^\top\mathtt{J}$ and $\mathtt{J}^\top\mathbf{z}$ according to Eq. (5.20)), and then resume the optimisation. Note that all previous measurements are summarised in $\mathtt{J}^\top\mathtt{J}$ and $\mathtt{J}^\top\mathbf{z}$, and that the computations and memory requirements are bounded. From the estimation perspective, this approach corresponds to an Information Filter, with matrix $\mathtt{J}^\top\mathtt{J}$ being the inverse covariance matrix and vector $\mathtt{J}^\top\mathbf{z}$ the information vector.

5.8 Discussion

In this chapter we presented the foundations for an incremental surface reconstruction from a stream of depth maps. Many concepts introduced here will be used as the building blocks for the height map fusion using differentiable rendering method in Chapter 6, and for the multi-scale surface reconstruction method in Chapter 7.

A height map is a very interesting dense representation of an environment, in particular for the small indoor robots we consider in this thesis. Depending on the surface model used and data association strategy we can realise many different variants of height map fusion that can be suitable even for very resource limited platforms and implemented both on CPU or GPU. Since we explicitly formulated surface reconstruction as an optimisation problem, there are many well-understood avenues for improvement such as the use of a robust cost function and adding regularising smoothness priors, as well as the use of a coarse-to-fine approach to speed up convergence. We have shown that by using a fixed topology, regular structure mesh to represent the surface we can solve the optimisation efficiently; in particular the Gauss-Seidel method is very simple and easy to implement.

The ease of using height maps comes however with certain compromises. Obviously, one of the limitations of height maps is that they are unable to represent vertical or overhanging structures appropriately. This limitation could be at least partially mitigated by extending the model to so-called multi-level surface maps [Triebel et al., 2006]. Another great opportunity for an extension is adding support for loop closure. At the moment, when performing reconstruction we assume a single, global height map. One possibility would be to allow the robot to create multiple local height maps as it moves and explores the scene, and align the maps once a loop closure is detected.

Chapter 6

Surface Reconstruction using Differentiable Rendering

Contents

6.1 Introduction

In this chapter we present a novel method for incremental height map estimation. Our core representation for reconstruction remains the same, we use a height (and colour) map defined as a triangular mesh on a regular 2D grid and take as input a stream of depth maps (and images). The main difference compared to the methods presented before is that now we use a generative model and Bayes theorem to derive a more formal solution to surface reconstruction. In a generative approach, we start by formulating a forward observation model and the measurement likelihood function that defines the probability of the observed data given a model. The forward model is parameterised with respect to variables that directly affect the observations, in our case per-vertex height and colour, and we seek for a reconstruction that best explains the observations. In order to evaluate the forward model (and thus the likelihood function) we use a standard computer graphics pipeline and predictive rendering, whereas in order to calculate the derivatives required to obtain the maximum likelihood estimate, we explicitly differentiate the rendering pipeline, and hence, we refer to our approach as *differentiable rendering*. The term *differentiable rendering* was coined by [Loper and Black, 2014] who defined it as a process that: (1) supplies pixels as a function of model parameters and (2) supplies derivatives of the pixel values with respect to those parameters. One of the advantages of this approach is that it can be implemented very efficiently: thanks to the hardware acceleration of modern GPUs rendering can be performed very fast, and when done properly, calculating the derivatives comes at almost negligible additional cost.

Traditionally, in computer vision, generative approaches have been mainly used with standard RGB or grayscale images only. However, in our approach we use the generative model with the depth maps as well, as it improves performance and robustness. We perform estimation in a *loosely-coupled* fashion, that means that we assume that camera poses are given, and we rely on the image measurements to infer surface colour, and separately use depth measurements to infer surface geometry. Specifically, at each new input frame, given a current estimate of the surface and camera pose, we perform a predictive depth and colour rendering and compare it with the observed depth map and colour frame as shown in Fig. 6.1. The error between the rendered and measured frames together with the gradients calculated using differentiable rendering are used for iterative optimisation of the height and colour of each observed surface cell. Similarly to the approach presented in Section 5.7, in

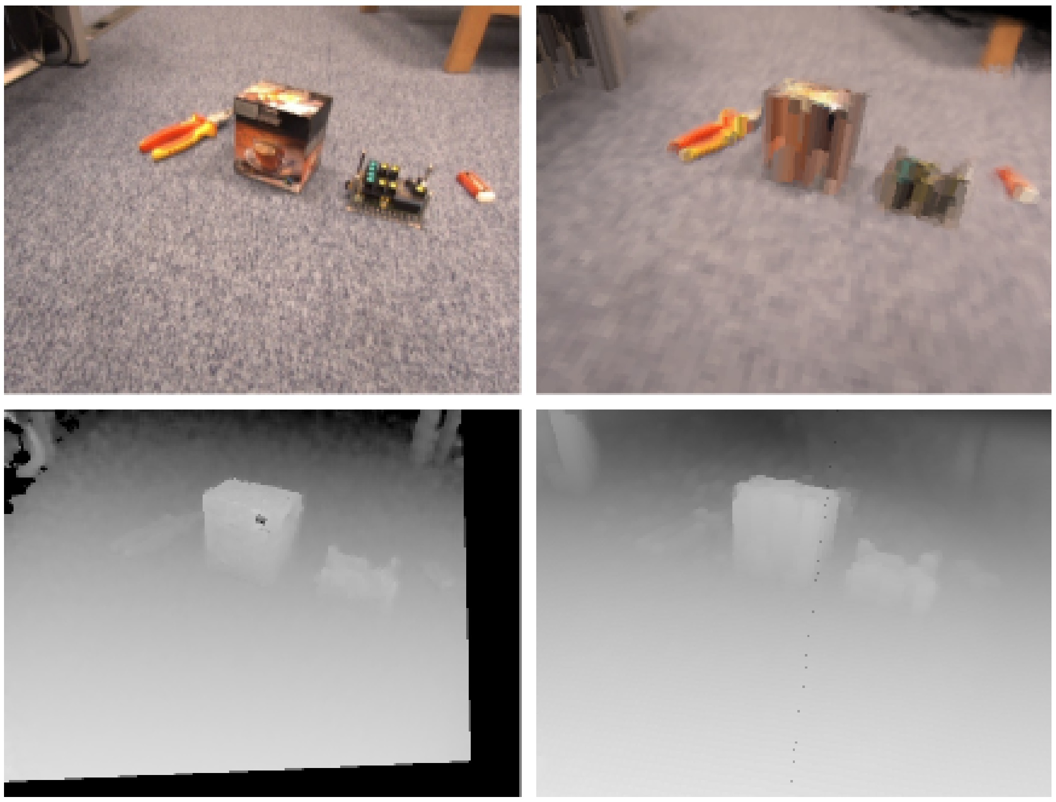


Figure 6.1: Left: an example of a monocular RGB image and motion stereo depth map, which are inputs to our fusion algorithm. By using a generative model that explicitly relates changes in surface parameters with measurements, we are looking for a height map reconstruction that best explains our observations. Right: output RGB and depth image predicted from our differentiable renderer.

order to allow for an incremental reconstruction we also maintain and update the full posterior distribution over the surface parameters in the information form.

Overall, the proposed approach allows us to formulate and derive a solution to the height mapping problem in a more general, elegant and probabilistically sound fashion. For example, now we work directly with our observations, *i.e.* depth measurements and their uncertainty, and do not have to rely on techniques for propagation of uncertainty, or use heuristics to perform data association. In fact, one can see the approaches presented in Chapter 5 as simplified versions of the method described here.

However, using the generative model also brings certain challenges. Whereas previously we were solving a linear least squares problem, now, since our forward model (the rendering) is nonlinear, the resulting optimisation problem is also nonlinear.

As a consequence, we have to use a nonlinear iterative solver that requires a good initialisation. Furthermore, the overall optimisation process is more complex and it needs to be carefully monitored in order to ensure stable behaviour and avoid getting trapped in a local minimum (in fact we cannot guarantee reaching the global optimum as the problem is not only nonlinear, but also nonconvex). During the iterative optimisation we have to evaluate the forward model multiple times, which results in increased computational load. A further consequence of using a generative model and the nonlinearity of the problem is that the incremental reconstruction is now implemented using an extended information filter, *i.e.* we still store the full posterior distribution in information form, but we update it using *linearised* error terms. In contrast to the method from the previous chapter, where we could always stop the iterative solver to add new data to the problem, before we can update the posterior distribution we now have to ensure that optimisation has converged.

In this chapter we also introduce an alternative method for representing the information matrix, compared to the grid-based approach proposed before. Instead of storing one global information matrix for the whole triangular mesh, we maintain and update multiple information filters, each on a *per-triangle* basis. This can be thought of as representing the matrix in an unassembled form, where one can still construct the global matrix from the individual filters. One of the advantages of such an approach is increased flexibility as now we can work with meshes of arbitrary topology, and do not have to rely on regular grid structure.

The remainder of this chapter is organised as follows. We will first review some prior work that showed how generative approaches could be used to solve problems in computer vision, with a particular emphasis on 3D and surface reconstruction. Next we will move to the core algorithm presented in this chapter, showing that we can explicitly use a dense height map model to permit efficient incremental inference using a generative model and differentiable rendering. This will be followed by experiments that demonstrate the viability and practicality of our approach. We will conclude the chapter with a discussion and briefly describe possible extensions of our approach.

6.2 Literature Review

Generative methods have a long history in computer vision and they are at the heart of many successful methods, for example, appearance-based object tracking [Bibby and Reid, 2008; de La Gorce et al., 2008], as well as human face and pose estimation [Parke and Waters, 2008; Poppe, 2007]. Approaches based on generative models are often referred to as *analysis-by-synthesis*, or *vision as inverse graphics* because they typically involve some sort of inverting the imaging process in order to infer the characteristics of the observed scene. Our method is partially inspired by OpenDR [Loper and Black, 2014], a differentiable renderer framework that demonstrates how the standard computer graphics rendering pipeline can be used to obtain generative models suitable for solving various computer vision problems. Thanks to automatic differentiation, certain approximations, and the usage of the off-the-shelf solvers and the Python scripting language, one can easily experiment with different models and algorithms. This is a very powerful framework and highlights the potential of generative approaches but the implementation is generally too slow and unoptimised for any real-time application.

The idea of using a generative model to perform Bayesian 3D surface reconstruction was probably first presented in [Cernuschi-Frias et al., 1986] and further developed in [Hung et al., 1988; Cernuschi-Frias et al., 1989; Hung and Cooper, 1990]. In this work only three primitive surface types (planes, cylinders and spheres) were considered, but the method is applicable to other parameterised surfaces as well. Unlike for example standard stereo algorithms, which perform estimation in the image domain by performing search along epipolar lines, estimation takes place directly in the parameter space of the 3D models. [Hung et al., 1991] extended the framework and derived a sequential Bayesian estimator for 3D surface parameters, where similarly to our algorithm, information extracted from previous images was summarised in a quadratic form.

Closely related to our approach is also the work on Bayesian surface reconstruction from NASA, initiated by [Cheeseman et al., 1996], and further developed in a series of papers that highlight the flexibility and advantages of a Bayesian approach, such as the ability to perform super-resolution, [Morris et al., 1999; Smelyanskiy et al., 2000], ability to incorporate camera calibration in the whole generative model [Morris et al., 2001] and creating a whole unified framework [Smelyanskiy et al., 2001]. Various different aspects like parameterisations and approaches to solve the problem were presented in [Smelyanskiy et al., 2002; Jalobeanu, 2004; Jalobeanu et al., 2004; Stutz, 2005]. Surface reconstruction was explicitly defined as an inverse problem: given a set of images infer the most probable surface that could have generated them, and Bayes theorem was used to derive a formal solution to this problem. One started by constructing a likelihood function that gives the probability of each pixel value given the imaged surface and observation conditions. The likelihood of the whole image was assumed to be just the product of likelihoods of each pixel, *i.e.* the measurement error of a pixel was conditionally independent of the value of its neighbours. The surface was modelled using a discrete uniform grid, with surface properties given at each grid cell, *i.e.* surface emitance (combination of surface albedo and illumination conditions) and elevation. The approach allowed for the inclusion of priors in their reconstruction in the form of neighbour correlations.

The two described approaches present very elegant solutions to surface reconstruction, however they are distinct to our method in several ways. First, although an extension for recursive reconstruction was discussed, these were primarily off-line, batch methods that considered all images at once. Furthermore, the methods estimated the surface directly from the observed images, which although the most general approach, typically results in a lack of robustness and requires starting with an already very good initial estimate of the surface because of the rather small basin of convergence of the associated optimisation problem. In our approach we take an incremental real-time approach and integrate camera tracking. Furthermore, we estimate the surface by directly rendering and fusing depth maps, rather than estimating the surface properties only from RGB images.

[Liu and Cooper, 2010, 2011, 2014] presented an interesting approach to more general 3D reconstruction based on Bayesian modelling and inverse ray tracing. Their method is based on a fixed resolution volumetric representation, modelled using Random Markov Fields (MFR), and one seeks the parameters of the model (in this case for each voxel, two properties are assigned: binary occupancy (0, or 1) and RGB colour) that best explain the observed RGB images. Although the method presents and focuses on efficient inference, it can only handle relatively small resolution volumes ($100^3 - 1000^3$), and a very small number of images (reported datasets are of size 16-49). Processing times are not reported and the method is formulated as a batch process where we need to maintain all observations and therefore is not suitable for incremental reconstruction. Furthermore, the method is not scalable, as

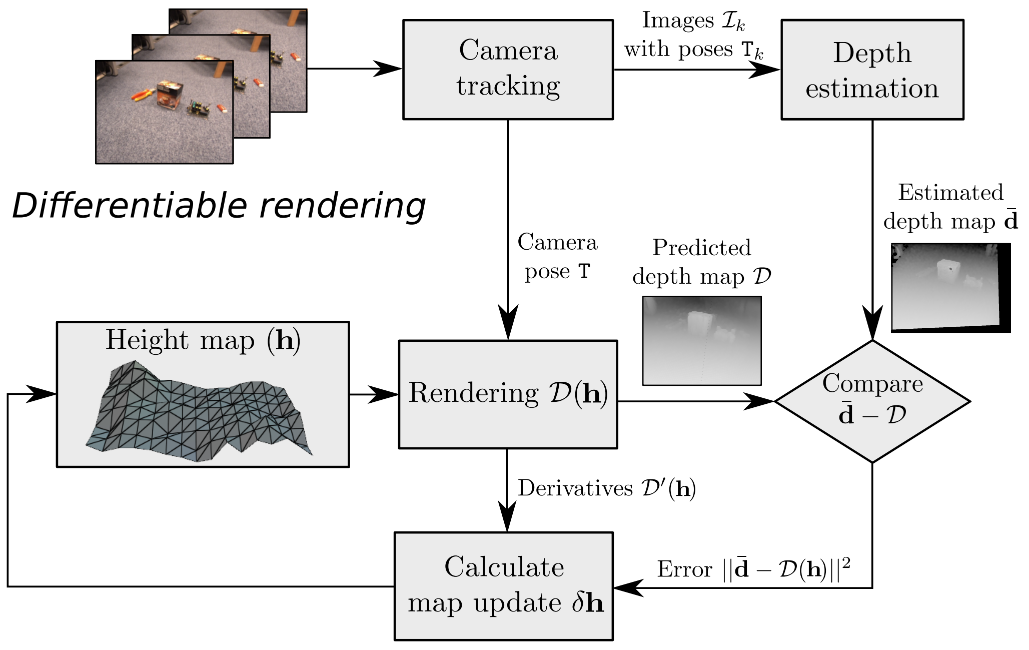

Camera tracking
Images $\mathcal{I}_k$ with poses $\mathtt{T}_k$
Depth estimation
Differentiable rendering
Camera pose $\mathtt{T}$
Predicted depth map $\mathcal{D}$
Estimated depth map $\bar{\mathbf{d}}$
Height map ($\mathbf{h}$)
Rendering $\mathcal{D}(\mathbf{h})$
Compare $\bar{\mathbf{d}} - \mathcal{D}$
Derivatives $\mathcal{D}'(\mathbf{h})$
Calculate map update $\delta\mathbf{h}$
Error $||\bar{\mathbf{d}} - \mathcal{D}(\mathbf{h})||^2$

Figure 6.2: An overview of our system. Given a sequence of images from a single moving camera we continuously estimate the camera motion and use it for depth estimation. Using differentiable rendering, the calculated depth maps are fused into a height map represented by a triangular mesh. Note that rendering in our approach is used quite differently compared to more traditional depth map fusion methods, *e.g.* KinectFusion [Newcombe et al., 2011a]. Firstly, whereas KinectFusion performs raycasting of a signed distance function (SDF) and uses predicted surface for the frame-to-model tracking, in our method we perform frame-to-frame tracking, and use the predictive rendering to *fuse* a depth map into a triangular mesh. Secondly, a fusion of depth measurements into a SDF can be performed in one step, whereas our approach is implemented as an iterative optimisation, which means that in order to fuse a single, observed depth map into a model, we need to perform several predictive renderings and comparisons of the observed and predicted depth maps.

the complexity grows with the number of images.

6.3 Incremental Surface Reconstruction based on a Generative Model

6.3.1 Overview

Our goal is to recursively estimate a surface model from a sequence of images from a camera moving over a scene. Fig. 6.2 shows an overview of our method: certain elements of the pipeline, *e.g.* camera tracking and depth estimation, are already known from previous chapters. The main novelty in this approach is in formulating incremental reconstruction as a recursive nonlinear optimisation problem, where as each new frame arrives we compare it with a generative rendering of our current surface estimate and make an appropriate Bayesian update. In this respect, we formulate nonlinear residuals as the difference between the (inverse) depths as measured in the current frame and the (inverse) depth predictions generated by the rendered model; at the same time, we compare predicted and observed colours (not shown in Fig. 6.2). In order to obtain a recursive formulation that allows us to keep *all* past measurements, we linearise these error terms and keep them as priors that are jointly minimised with the residuals of the current frame.

A crucial element of our method, which enables highly efficient operation, is a differentiable renderer implemented using the standard OpenGL computer graphics pipeline. Given a current surface model and a camera pose it can render a predicted image and depth for each pixel together with the derivatives of these quantities with respect to the model parameters at almost no extra computational cost.

6.3.2 Generative Model

Our approach is inspired by the probabilistic model of the image formation and rendering process illustrated in Fig. 6.3. We parameterise a surface by its geometry $\mathtt{G}$ and its appearance $\mathtt{A}$. Given a camera with an associated pose $\mathtt{T}$ in the scene, we can render a predicted image $\mathcal{I}$ and an inverse depth map $\mathcal{D}$. In our method we do not model lighting and surface properties (such as normals) explicitly, but assume ambient light and Lambertian surfaces.

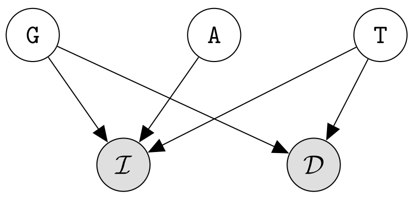


Figure 6.3: A graphical model of the image formation and rendering process used to derive our fusion approach.

The joint distribution that models the image formation process in Fig. 6.3 is given by:

$$P(\mathcal{I}, \mathcal{D}, \mathtt{G}, \mathtt{A}, \mathtt{T}) = P(\mathcal{I}|\mathtt{G}, \mathtt{A}, \mathtt{T})P(\mathcal{D}|\mathtt{G}, \mathtt{T})P(\mathtt{G})P(\mathtt{A})P(\mathtt{T}) \ . \tag{6.1}$$

The relation between image observations and surface estimation can be expressed using Bayes rule:

$$P(\mathtt{G}, \mathtt{A}, \mathtt{T} | \mathcal{I}, \mathcal{D}) \propto P(\mathcal{I}, \mathcal{D} | \mathtt{G}, \mathtt{A}, \mathtt{T}) P(\mathtt{G}) P(\mathtt{A}) P(\mathtt{T}) \ , \tag{6.2}$$

which allows us to derive a maximum *a posteriori* (MAP) estimate of the camera pose and surface:

$$\arg\max_{\mathtt{G},\mathtt{A},\mathtt{T}} P(\mathcal{I}, \mathcal{D} | \mathtt{G}, \mathtt{A}, \mathtt{T}) P(\mathtt{G}) P(\mathtt{A}) P(\mathtt{T}) \ . \tag{6.3}$$

The term $P(\mathcal{I}, \mathcal{D} | \mathtt{G}, \mathtt{A}, \mathtt{T})$ is a likelihood function which we will be able to evaluate and differentiate using our renderer. The terms $P(\mathtt{G})$, $P(\mathtt{A})$, $P(\mathtt{T})$ represent prior knowledge that we might have about the geometry, appearance and trajectory, *e.g.* obtained from previous measurements or general properties of the surface, such as its smoothness.

To simplify the problem, we treat the camera poses as given by a tracking module. Furthermore, since in our loosely-coupled fusion we use the colour images to generate a depth map that determines the height field, we ignore the dependency of the colour image on the height values. Note that this is a conservative assumption letting us treat the colours and height fields independently:

$$\arg\max_{\mathtt{G}} P(\mathcal{D} | \mathtt{G}) P(\mathtt{G}) \ , \tag{6.4a}$$

$$\arg\max_{\mathtt{A}} P(\mathcal{I} | \mathtt{A}) P(\mathtt{A}) \ . \tag{6.4b}$$

In essence, we alternate between height (Eq. (6.4a)) and colour fusion (Eq. (6.4b)), first estimating the geometry using an observed inverse depth map and subsequently we fuse the colour image into the height field while keeping the geometry fixed. In the following we focus on the depth map fusion, but the derivation can be extended to colour estimation in a straightforward manner. Currently, colour information is mainly used for meaningful display, however one can use it to perform camera tracking with respect to the estimated model in order to reduce tracking drift.

6.3.3 Reconstruction as a Nonlinear Optimisation Problem

Eq. (6.4a) can be used to formulate the optimisation problem that needs to be solved in our height map estimation. The geometry of a height field is fully parameterised by the vector of heights $\mathbf{h} \in \mathbb{R}^n$, which is the quantity we want to estimate. We represent the priors $P(\mathtt{G})$ using a multivariate Gaussian probability distribution $\mathcal{N}^{-1}(\boldsymbol{\eta}, \Lambda)$ in a canonical form parameterised by the information vector $\boldsymbol{\eta}$ and matrix Λ. By taking the negative logarithm of Eq. (6.4a) we obtain the following minimisation problem:

$$\arg\min_{\mathbf{h}} F(\mathbf{h}) \ , \tag{6.5}$$

where:

$$F(\mathbf{h}) = F_d(\mathbf{h}) + F_p(\mathbf{h}) \ . \tag{6.6}$$

The cost function thus consists of two terms, the data term $F_d(\mathbf{h})$ and the prior term $F_p(\mathbf{h})$:

$$F_d(\mathbf{h}) = \|\bar{\mathbf{d}} - \mathcal{D}(\mathbf{h})\|^2_{\Sigma_{\bar{\mathbf{d}}}} \ , \tag{6.7a}$$

$$F_p(\mathbf{h}) = \frac{1}{2}\mathbf{h}^\top \Lambda \mathbf{h} - \boldsymbol{\eta}^\top \mathbf{h} + c \ , \tag{6.7b}$$

where $\bar{\mathbf{d}} \in \mathbb{R}^m$ is a column vector that represents the observed inverse depth map with associated measurement uncertainties modelled by (diagonal) covariance matrix $\Sigma_{\bar{\mathbf{d}}}$, and $\mathcal{D}(\mathbf{h})$ is a nonlinear (rendering) operator, $\mathcal{D} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that predicts a vector of depths using the current estimate of $\mathbf{h}$. The prior term (Eq. (6.7b)) is a standard quadratic form associated with a Gaussian probability distribution $\mathcal{N}^{-1}(\boldsymbol{\eta}, \Lambda)$.

Instead of back-projecting the depth measurements and expressing them in terms of height measurements, as described in Chapter 5, we now directly compare the measured depths with the depths rendered from the model, and seek parameters of the model that minimise the discrepancy between observed and predicted values. The cost function term $F_d(\mathbf{h})$ related to the measurements can thus be expressed as a sum of individual per-pixel inverse depth error terms:

$$e_i(\mathbf{h}) = \bar{d}_i - d_i(\mathbf{h}), \tag{6.8}$$

between a measured $\bar{d}_i$ and a predicted inverse depth d_i:

$$F_d(\mathbf{h}) = \sum_{i=1}^{m} \frac{1}{\sigma^2_{d_i}} (\bar{d}_i - d_i(\mathbf{h}))^2 = \sum_{i=1}^{m} \frac{1}{\sigma^2_{d_i}} e_i(\mathbf{h})^2 \ = \frac{1}{2}\mathbf{e}(\mathbf{h})^\top \mathbf{e}(\mathbf{h}). \tag{6.9}$$

The minimisation problem in Eq. (6.5) is a nonlinear least squares problem, which is typically solved using iterative approaches, and will later describe two algorithms: Gauss-Newton and the nonlinear conjugate gradient method. Iterative algorithms start with some initial estimate and use various strategies to gradually improve the solution. A shared characteristic of these methods is that they require access to the gradient of the objective function in order to calculate a direction that minimises the objective function.

The gradient of the objective function $F(\mathbf{h})$ as defined in Eq. (6.5) is given by:

$$\boldsymbol{\nabla} F(\mathbf{h}) = \boldsymbol{\nabla} F_d(\mathbf{h}) + \boldsymbol{\nabla} F_p(\mathbf{h}) \ . \tag{6.10}$$

The gradient of the term associated with the surface priors is straightforward to compute:

$$\boldsymbol{\nabla} F_p(\mathbf{h}) = \Lambda\mathbf{h} - \boldsymbol{\eta} \ , \tag{6.11}$$

whereas the gradient of the data term is given by [Sun and Yuan, 2006, p. 25]:

$$\boldsymbol{\nabla} F_d(\mathbf{h}) = 2\mathcal{D}'(\mathbf{h})^\top \Sigma_{\bar{\mathbf{d}}}^{-1}(\bar{\mathbf{d}} - \mathcal{D}(\mathbf{h})) \ , \tag{6.12}$$

where $\mathcal{D}'(\mathbf{h})$ is the derivative of the rendering operator $\mathcal{D}$ and is called the Jacobian matrix $\mathtt{J}_d(\mathbf{h})$:

$$\mathtt{J}_d = \begin{bmatrix} \frac{\partial d_1}{\partial h_0} & \frac{\partial d_1}{\partial h_1} & \cdots & \frac{\partial d_1}{\partial h_n} \\ \frac{\partial d_2}{\partial h_0} & \frac{\partial d_2}{\partial h_1} & \cdots & \frac{\partial d_2}{\partial h_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial d_m}{\partial h_0} & \frac{\partial d_m}{\partial h_1} & \cdots & \frac{\partial d_m}{\partial h_n} \end{bmatrix} \ . \tag{6.13}$$

Although the number of measurements (m) and size of the state space (n) are high, the Jacobian $\mathtt{J}_d(\mathbf{h}) \in \mathbb{R}^{m\times n}$ linking them is sparse. Since we assume that each depth value depends only on 3 vertices, $\mathtt{J}_d$ has only 3 non-zero entries per row. Note how this is related to the *linear* system of equations, Eq. (5.17), we discussed in a previous chapter. The structure of the Jacobian $\mathtt{J}_d$ here and the matrix $\mathtt{J}$ defined there is essentially the same. However, a consequence of using a generative model is that our data association strategy has changed (we use predictive rendering instead of back-projection). As a result, we are dealing with a *nonlinear* problem, where the Jacobian $\mathtt{J}_d$ can only be used to *linearise* the objective function around a current estimate.

As in Chapter 5, we do not explicitly form the Jacobian matrix and perform matrix-vector multiplication, but instead, we calculate the gradient vector by summing the contributions from the individual, per-pixel error terms. The gradient of $F_d(\mathbf{h})$ is therefore given by:

$$\boldsymbol{\nabla} F_d(\mathbf{h}) = 2\sum_{i=1}^{m} \frac{1}{\sigma_{d_i}^2} \mathtt{E}_i(\mathbf{h}) e_i(\mathbf{h}) \ , \tag{6.14}$$

where $\mathtt{E}_i(\mathbf{h}) = \boldsymbol{\nabla} e_i(\mathbf{h})$ is the derivative (Jacobian) of a per-pixel error term. Since each error term depends on three values in $\mathbf{h}$ (we denote them by $\mathbf{h}_j$), we only need to calculate the derivative of the rendered depth with respect to vertices of the associated triangle, and can assemble the gradient of the function F_d from this individual per-depth Jacobians $\mathtt{E}_i = \boldsymbol{\nabla} e_i(\mathbf{h}) \in \mathbb{R}^3$ and associated error terms $e_i(\mathbf{h})$. In the following subsection we will describe our strategy to obtain $\mathtt{E}_i$.

6.3.4 Differentiable Rendering

In our rendering we explicitly model the ray-triangle intersection and perform analytical differentiation of this operation. Furthermore we assume that each pixel "observes" only a single triangle. There are alternative and more sophisticated approaches. For example, [Smelyanskiy et al., 2000] carefully model the rendering process and take into consideration surface normals and contributions from multiple triangles into pixel colour, whereas [Loper and Black, 2014] proposed an approximate way to calculate the derivative.

Let $\mathbf{r}(t)$ be a ray, parameterised by its starting point $\mathbf{p} \in \mathbb{R}^3$ and direction vector $\mathbf{o} \in \mathbb{R}^3$, $\mathbf{r}(t) = \mathbf{p} + t\mathbf{o}$, with $t \geq 0$. For each pixel in the image we can calculate a ray using camera intrinsics and the centre of the camera frame of reference as the origin. Let $\triangle$ be a triangle in $\mathbb{R}^3$ parameterised by 3 vertices, $\mathbf{v}_0$, $\mathbf{v}_1$, $\mathbf{v}_2$.

Ray-triangle intersection can be easily found using, for example, the classical algorithm of [Möller and Trumbore, 1997], which, when the ray intersects the triangle, yields a vector $(t, \alpha, \beta)^\top$, where t is the distance to the plane in which the triangle lies and α, β are the barycentric coordinates of the ray intersection point with respect to the triangle $\triangle$. Values of t, α, and β are the essential elements required to render the depth and colour for a particular pixel: t is directly related to the depth, whereas the barycentric coordinates are used to interpolate the colour $\mathbf{c}$ based on the RGB colour triangle vertices $(\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2)$ in the following way:

$$\mathbf{c} = (1 - \alpha - \beta)\mathbf{c}_0 + \alpha\mathbf{c}_1 + \beta\mathbf{c}_2 \ . \tag{6.15}$$

Fig. 6.4 depicts this ray-triangle intersection problem.

The rendered inverse depth d_i of pixel i depends only on the geometry of the triangle that a ray is intersecting (and the camera pose that is assumed fixed). As we model the surface using a height map, each vertex has only one degree of freedom,

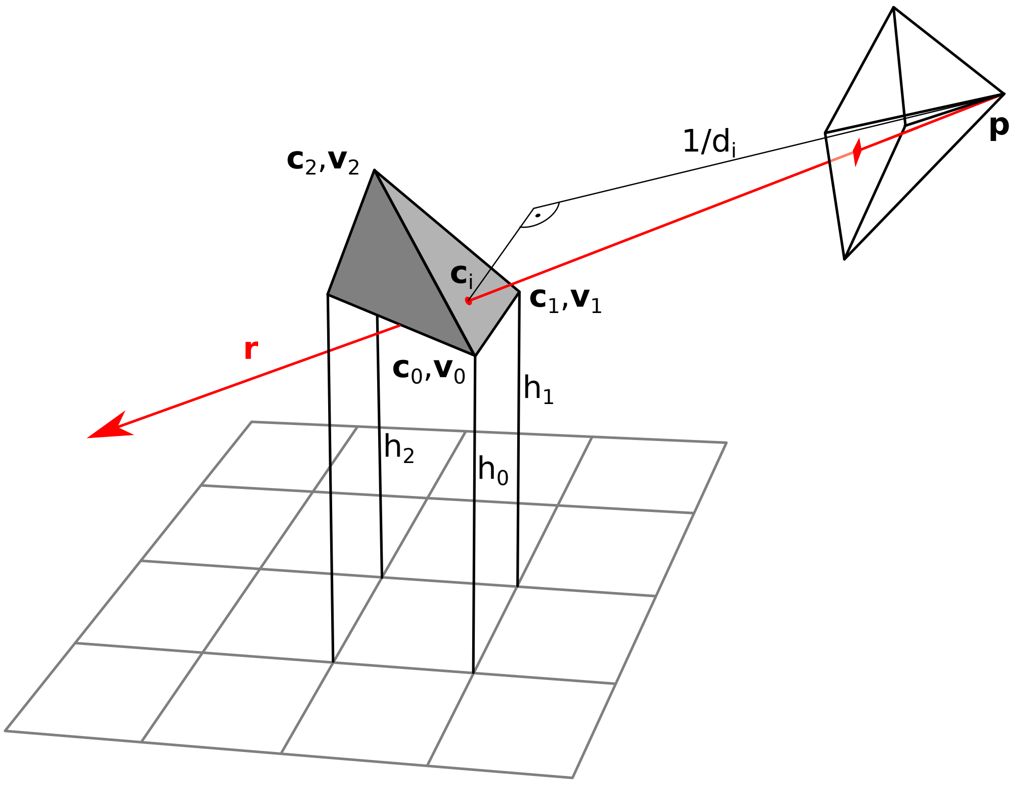


Figure 6.4: The essential geometry of ray-triangle intersection. By directly modelling the rendering process we can easily find the derivatives of a rendered depth (d_i) and colour (c_i) with respect to the parameters of the surface model.

its height h. Assuming that the ray intersects the triangle j specified by heights h_0, h_1, h_2, at distance $1/d_i$, we can express the derivative as follows:

$$\mathtt{E}_i = \frac{\partial d_i}{\partial \mathbf{h}_j} = \begin{bmatrix} \frac{\partial d_i}{\partial h_0} & \frac{\partial d_i}{\partial h_1} & \frac{\partial d_i}{\partial h_2} \end{bmatrix} . \tag{6.16}$$

The individual partial derivatives can be derived easily using the chain and product rules, and eventually used to calculate the gradient of the objective function as described in Eq. (6.14).

6.3.5 Height Map Fusion through Linearisation

Thus far we have defined the general objective function required to perform surface reconstruction and described how to calculate its gradient. Details on how exactly to minimise the objective function will be provided in Section 6.4. Since we want to incrementally reconstruct the surface and process the data as it arrives, we will now present our approach to recursive estimation. Our fusion method is both simple and principled, and can be seen as a generalisation of the approach presented in Section 5.7: all previous observations are maintained and accumulated in the form of the prior term $F_p(\mathbf{h})$ (the quadratic cost function) that serves as constraints on the vertices during the optimisation.

Specifically, with every new depth map $\bar{\mathbf{d}}$, given current prior term $F_p(\mathbf{h})$, *i.e.* Λ, $\boldsymbol{\eta}$ and c, we minimise the function in Eq. (6.5) to find the current estimate of height values, $\hat{\mathbf{h}}$. After optimisation converges we linearise the objective function $F_d(\mathbf{h})$ associated with the data term at the estimated solution $\hat{\mathbf{h}}$:

$$F_d(\mathbf{h}) \approx F_d^l(\mathbf{h}) = \|\bar{\mathbf{d}} - \Big(\hat{\mathbf{d}} + \hat{\mathtt{J}}_d \cdot (\hat{\mathbf{h}} - \mathbf{h})\Big)\|^2 \ , \tag{6.17}$$

where $\hat{\mathbf{d}} = \mathbf{d}(\hat{\mathbf{h}})\Big|_{\mathbf{h}=\hat{\mathbf{h}}}$ and $\hat{\mathtt{J}}_d = \mathtt{J}_d(\mathbf{h})|_{\mathbf{h}=\hat{\mathbf{h}}}$. Note that we omit here the measurement covariance matrix $\Sigma_{\bar{\mathbf{d}}}$ for the brevity of the derivation. We can now use the quadratic approximation of the objective function given by:

$$F_d^l(\mathbf{h}) = \|\overbrace{\Big(\bar{\mathbf{d}} - \hat{\mathbf{d}} - \hat{\mathtt{J}}_d\hat{\mathbf{h}}\Big)}^{\mathbf{r}} + \hat{\mathtt{J}}_d\mathbf{h})\|^2 = \|\mathbf{r} + \hat{\mathtt{J}}_d\mathbf{h}\|^2 \tag{6.18a}$$

$$= \mathbf{r}^\top\mathbf{r} + 2\mathbf{r}^\top\hat{\mathtt{J}}_d\mathbf{h} + \mathbf{h}^\top\hat{\mathtt{J}}_d^\top\hat{\mathtt{J}}_d\mathbf{h} \ , \tag{6.18b}$$

to update our prior term. Thus, to fuse the depth measurement vector $\bar{\mathbf{d}}$ into the height field, we simply augment the prior term in Eq. (6.7b) using the quadratic model $F_d^l(\mathbf{h})$ derived above:

$$\Lambda^+ = \Lambda + \hat{\mathtt{J}}_d^\top\hat{\mathtt{J}}_d \ , \tag{6.19a}$$

$$\boldsymbol{\eta}^+ = \boldsymbol{\eta} + 2\mathbf{r}^\top\hat{\mathtt{J}}_d \ , \tag{6.19b}$$

$$c^+ = c + \mathbf{r}^\top\mathbf{r} \ . \tag{6.19c}$$

Note that we do not have to keep the value of the linearisation point, nor the previous depth measurements, as all information is captured in the linearised error term. This operation can be seen as equivalent to the measurement update step in the Extended Information Filter [Thrun et al., 2004].

When using a regular grid structure for height mapping, we can store the prior terms from Eq. (6.19) (*i.e.* Λ and $\boldsymbol{\eta}$) on a grid as presented in the previous chapter. Here, however, we propose an alternative approach that does not require us to form the information matrix and vector explicitly but instead we accumulate the *per-pixel* inverse depth quadratic costs on a *per-triangle* basis. This means that for each triangle j we keep a quadratic function of the form:

$$f_j(\mathbf{h}) = \mathbf{h}_j^\top \Lambda_j \mathbf{h}_j + \boldsymbol{\eta}_j^\top \mathbf{h}_j + c_j \ . \tag{6.20}$$

where $\Lambda_j \in \mathbb{R}^{3\times3}$, $\boldsymbol{\eta}_j \in \mathbb{R}^3$, and $\mathbf{h}_j \in \mathbb{R}^3$ is a vector of heights that defines the triangle j.

Updating the per-triangle information filter is straightforward. Recall the individual per-pixel error term as in Eq. (6.8). As explained, after the optimisation has converged, we approximate this error term linearly around the current estimate $\hat{\mathbf{h}}_j$ as:

$$e_i \approx e_i^l = \bar{e}_i + \mathbf{E}_i \delta \mathbf{h}_j = \bar{e}_i - \mathbf{E}_i \hat{\mathbf{h}}_j + \mathbf{E}_i \mathbf{h}_j \ . \tag{6.21}$$

Fusion of a single depth measurement $\bar{d}_i$ into the height map thus consists of a simple addition of the linearised error (Eq. (6.21)) to the corresponding triangle's cost function (Eq. (6.20)):

$$f_j^+ = f_j + \frac{(e_i^l)^2}{\sigma_{d_i}^2}. \tag{6.22}$$

Multiplying this out and rearranging provides us with the updated coefficients (Λ_j^+, $\boldsymbol{\eta}_j^+$ and c_j^+) of the per-triangle quadratic cost:

$$\begin{aligned} \Lambda_j^+ &= \Lambda_j + \frac{\mathbf{E}_i^\top \mathbf{E}_i}{\sigma_{d_i}^2} \ , \\ \boldsymbol{\eta}_j^+ &= \boldsymbol{\eta}_j + \frac{2}{\sigma_{d_i}^2} (\bar{e}_i - \mathbf{E}_i \hat{\mathbf{h}}_j) \mathbf{E}_i \ , \\ c_j^+ &= c_j + \frac{(\bar{e}_i - \mathbf{E}_i \hat{\mathbf{h}}_j)^2}{\sigma_{d_i}^2}. \end{aligned} \tag{6.23}$$

The overall prior term $F_p(\mathbf{h})$ and matrix Λ and vector $\boldsymbol{\xi}$ can be assembled from the individual per-triangle error functions, so the overall cost function (Eq. (6.5)) we have to minimise amounts to:

$$F(\mathbf{h}) = \sum_j f_j(\mathbf{h}) + \sum_i \frac{1}{\sigma_{d_i}^2} (e_i(\mathbf{h}))^2 \ . \tag{6.24}$$

Note that this per-triangle formulation gives us more flexibility compared to the approach discussed in Section 5.5.2, as now we are not restricted to any particular grid topology and can work with meshes of irregular vertex connectivity. Still, the memory and computational requirements are bounded: the number of linear cost terms is bounded by the number of triangles in the mesh, whereas the number of nonlinear (inverse) depth error terms is bounded by the number of pixels in the camera. This is of course an important property for real-time operation.

6.4 Nonlinear Solvers

We will now describe Gauss-Newton and nonlinear conjugate gradient method, two algorithms that can be used to solve the nonlinear least squares problem involved in our fusion approach. Both methods utilise the gradient of the objective function, but differ in the way the direction for the minimisation is calculated at each iteration.

6.4.1 Gauss-Newton Method

Gauss-Newton algorithm is a commonly used method for solving nonlinear least squares problem: starting with some initial estimate it solves a series of approximations of the original nonlinear problem, gradually improving the estimate of the solution. Specifically, at each iteration of Gauss-Newton method we try to find a small incremental update $\delta\mathbf{h}$ to the current estimate $\mathbf{h}_0$ by linearising the nonlinear objective function and form its quadratic approximation around $\mathbf{h}_0$. We have already encountered this approached when we presented fusion through linearisation (Eq. (6.17)), but now we parameterise the quadratic function with respect to $\delta\mathbf{h}$:

$$F(\delta\mathbf{h}) = \|\bar{\mathbf{d}} - (\mathbf{d}_0 + \mathtt{J}_d^\top \delta\mathbf{h})\|^2 \ , \tag{6.25}$$

where $\mathbf{d}_0 = \mathcal{D}(\mathbf{h}_0)$, and $\mathtt{J}_d$ is the gradient of the objective function at $\mathbf{h}_0$. The (linearised) objective function can be rewritten in the following way:

$$F(\delta\mathbf{h}) = \|\underbrace{(\bar{\mathbf{d}} - \mathbf{d}_0)}_{\mathbf{r}} - \mathtt{J}_d\delta\mathbf{h}\|^2 = \|\mathbf{r} - \mathtt{J}_d\delta\mathbf{h}\|^2 \ , \tag{6.26}$$

where $\mathbf{r}$ is the per-pixel depth error (residual). Expanding Eq. (6.26) leads to:

$$\begin{aligned} F(\delta\mathbf{h}) = \|\mathbf{r} - \mathtt{J}_d\delta\mathbf{h}\|^2 &= (\mathbf{r} - \mathtt{J}_d\delta\mathbf{h})^\top(\mathbf{r} - \mathtt{J}_d\delta\mathbf{h}) \\ &= \mathbf{r}^\top\mathbf{r} - 2\delta\mathbf{h}^\top\mathtt{J}_d^\top\mathbf{r} + \delta\mathbf{h}^\top\mathtt{J}_d^\top\mathtt{J}_d\delta\mathbf{h} \ . \end{aligned} \tag{6.27}$$

The vector $\delta\mathbf{h}$ can be found by setting the first derivative of the objective function (Eq. (6.27)) to zero:

$$\frac{\partial F}{\partial \delta\mathbf{h}} = -2\mathtt{J}_d^\top\mathbf{r} + 2\mathtt{J}_d^\top\mathtt{J}_d\delta\mathbf{h} = \mathbf{0} \tag{6.28}$$

and solving the associated normal equation:

$$\mathtt{J}_d^\top\mathtt{J}_d\delta\mathbf{h} = \mathtt{J}_d^\top\mathbf{r} \ . \tag{6.29}$$

Although forming the normal equation is not problematic in our case (due to sparsity pattern induced by mesh structure), and in fact we perform necessary steps when updating the quadratic form that represents our prior, the large scale nature of the problem makes solving the Eq. (6.29) more challenging. For the reasons discussed in Section 5.6 factorisation approaches are impractical in these circumstances, but instead we could apply iterative methods (*e.g.* conjugate gradient or Gauss-Seidel). The overall structure of the Gauss-Newton algorithm for height map fusion using differentiable rendering is outlined in Algorithm 3.

By closely investigating the listings in Algorithm 3 we can identify the challenge of solving our optimisation problem using the Gauss-Newton method. In principle, the algorithm consists of two loops: the main, outer loop is the standard nonlinear optimisation and involves the calculation of the incremental Gauss-Newton step. However, there is another iterative method executed in line 6, that solves the normal equation required to calculate the direction vector $\mathbf{p}_k$. Although in the previous chapter we presented very efficient methods for solving normal equations related to our reconstruction problems, there the normal equation had to be solved only once per frame, whereas now we have to solve it within each Gauss-Newton iteration. Furthermore, note that the vector $\mathbf{p}_k$ gives us only a direction in which we should try to minimise the cost function and we still have to calculate the proper step size α_k, in order to avoid divergence.

Algorithm 3 Gauss-Newton method

```
1: Given $\mathbf{h}_0$
2: Set $k := 0$
3: do
4:     Linearise the problem at $\mathbf{h}_k$ (Eq. (6.25))                 ▷ Render
5:     Form normal equation (Eq. (6.29))
6:     Solve $\mathtt{J}_d^\top \mathtt{J}_d \delta\mathbf{h} = \mathtt{J}_d^\top \mathbf{r}$, set $\mathbf{p}_k = \delta\mathbf{h}$        ▷ e.g. using Gauss-Seidel
7:     Do line search to determine step size $\alpha_k$                    ▷ Render
8:     $\mathbf{h}_{k+1} = \mathbf{h}_k + \alpha_k \mathbf{p}_k$
9:     $k := k + 1$
10: while $\mathbf{p}_k \neq 0$
```

6.4.2 Nonlinear Conjugate Gradient

The Gauss-Newton method can be very efficient for solving various nonlinear least squares problems, and by taking second order steps it can achieve a high convergence rate, in particular when the cost function is well approximated by the quadratic function. However, this comes at the cost of the high complexity of the algorithm, as presented in the previous section.

The conjugate gradient algorithm for solving linear least squares problems that we outlined in Chapter 5 can be modified and also used for solving nonlinear problems, and generally is far simpler to implement and execute. Unlike the Gauss-Newton method, nonlinear conjugate gradient is an indirect, matrix-free approach, that does not require forming and explicitly solving the full normal equation, but instead calculates the descent direction only by evaluating the gradient of the objective function. This makes the method especially suitable for solving large scale nonlinear optimisation problems, and in fact it has been successfully applied before to various computer vision tasks, *e.g.* in large scale bundle adjustment [Agarwal et al., 2010]. There are many variants of nonlinear conjugate gradient that differ in the way they update the vector describing the direction of descent; in Algorithm 4 we outline the overall structure of the Fletcher and Reeves variant of the conjugate gradient method [Nocedal and Wright, 2006].

Algorithm 4 Nonlinear conjugate gradient, Fletcher and Reeves version.

```
1:  Given $\mathbf{h}_0$
2:  Evaluate $F_0 = F(\mathbf{h}_0)$ and $\mathbf{g}_0 = \boldsymbol{\nabla} F(\mathbf{h}_0)^\top$        ▷ Render
3:  Set $\mathbf{p}_0 := -\mathbf{g}_0$ and $k := 0$
4:  while $\mathbf{g}_k \neq 0$ do
5:      Do line search to determine step size $\alpha_k$                        ▷ Render
6:      $\mathbf{h}_{k+1} = \mathbf{h}_k + \alpha_k \mathbf{p}_k$
7:      Calculate gradient $\mathbf{g}_{k+1} = \boldsymbol{\nabla} F(\mathbf{h}_{k+1})^\top$          ▷ Render
8:      $\beta_{k+1} := \frac{\mathbf{g}_{k+1}^\top \mathbf{g}_{k+1}}{\mathbf{g}_k^\top \mathbf{g}_k}$
9:      $\mathbf{p}_{k+1} := -\mathbf{g}_{k+1} + \beta_{k+1} \mathbf{p}_k$
10:     $k := k + 1$
11: end while
```

As evident, the optimisation algorithm is rather straightforward: it only requires the ability to evaluate the objective function and its gradient, as well as simple vector operations like addition and dot product. The ▷ *Render* in lines 2, 5 and 7 in Algorithm 4 highlights the execution of the differentiable rendering. Similarly to the Gauss-Newton method, at each iteration of the nonlinear conjugate gradient method it is required to perform a line search that determines the step size α_k in the descent direction, which can lead to several evaluations of the cost function. Therefore, the key to the efficiency of this method is very fast differentiable rendering.

6.5 Implementation

Efficient implementation of the differentiable rendering approach requires some careful consideration. Our method can be divided into roughly two parts: one performs (differentiable) rendering, whereas the second part is dedicated to solving the optimisation problem as well as updating and maintaining the prior term. The solver can be implemented at ease using standard high performance computation frameworks, for example CUDA or OpenCL (as we did in Chapter 5). However, when implementing differentiable rendering the choice is not obvious.

The main computationally expensive step in differentiable rendering consists of establishing the ray and triangle correspondence. Once the ray-triangle association problem is solved, calculating the intersection and its derivatives comes at an almost negligible cost. Graphics pipelines like OpenGL are best suitable and highly optimised for *standard* rendering, but not necessarily for *differentiable* rendering. Thus, when implementing our approach we have identified the following options:

- implement rendering and differentiation fully within the OpenGL rendering pipeline,
- implement our own, custom differentiable rendering pipeline, *e.g.* using OpenCL or CUDA,
- use OpenGL/CUDA interoperability to implement different parts of the algorithm in different frameworks.

The first option assumes that the rendering and calculation of the derivatives (of the ray-triangle intersection problem) can be performed within the fragment shaders of the OpenGL rendering pipeline. Although feasible, there are several issues associated with this approach. OpenGL uses a technique called rasterisation to implement rendering, whereas our approach to differentiation operates more on the principles of ray tracing (we explicitly model the ray triangle intersection). Thus, an efficient and correct implementation of differentiable rendering requires expertise and a good understanding of the OpenGL rendering pipeline (*e.g.* how a rasteriser works and the nature of computations within fragment shaders), and its many subtleties and limitations. Furthermore, support for fairly modern OpenGL features like `Image Load Store` available from the OpenGL 4.2 standard, or even some non-standard extensions, *e.g.* support for atomic float operations (`NV_shader_atomic_float`), is beneficial during the implementation and might not be available on all GPUs.

The second option, *i.e.* implementation of a custom, differentiable renderer entirely using CUDA (or OpenCL), although theoretically offering the greatest flexibility and control over the rendering process, can rarely be the best solution. Even though, thanks to the regular, grid structure of the height map one can use, for example, ray marching (and the Digital Differential Analyser (DDA) algorithm) to efficiently find the ray-triangle intersection for every pixel, we found it difficult to obtain rendering rates on par with the optimised OpenGL implementation.

We have tested all discussed approaches, and found that the third one, which combines OpenGL and CUDA offers best flexibility and performance. In this approach we first use a standard OpenGL pipeline to solve the ray-triangle association problem for each pixel, and then in CUDA we calculate ray-triangle intersections and their derivatives. We can achieve that by a very simple combination of vertex and fragment shaders that for each pixel *render* the (x, y)-coordinates of a point on a grid the pixel is observing (alternatively, one can also render an associate triangle ID). The output from OpenGL is then passed to CUDA, where within a compute kernel determining a ray-triangle association simplifies to a single memory look-up. All subsequent computations, including solving the optimisation problem, are also performed in CUDA. With this technique, thanks to the regular, grid structure of the height map, rendering can be performed very efficiently and robustly: depending on height field and image resolution, we can achieve rendering rates of thousands frames per seconds on a high-end GPU (NVIDIA GTX 980). The main computational load comes from the implementation of the nonlinear solver.

6.6 Experiments and Evaluation

To demonstrate the practicality of our approach for height mapping and robot perception we performed a series of experiments using both synthetic and real data experiments. In the robot experiment we use our standard platform based on the Pioneer P3-DX robot and a single VGA resolution camera mounted at a height of about 30 cm above the ground, pointing downwards and looking ahead approximately 1-2 metres. In most of the experiments we used cells of size 10 mm. We applied camera tracking based on visual odometry (Chapter 3), and used the estimated height map to detect and track the motion only of the planar parts in images. For depth map estimation we use the multi-view stereo algorithm presented in Chapter 2. The visualisation in Fig. 6.5 demonstrates that we were able to reconstruct the essential geometry of the environment as well as the very small objects on the floor, like cables or pliers.

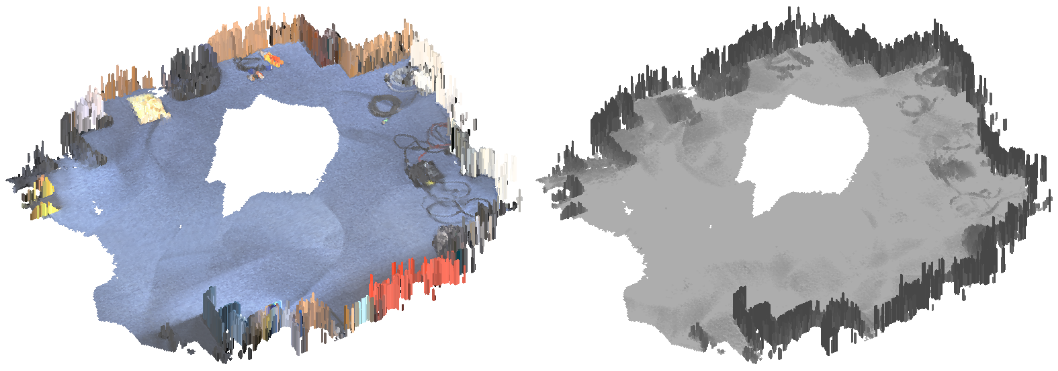


Figure 6.5: A visualisation of a height map reconstructed using the differentiable rendering approach. With the proposed method we are able to map whole rooms in real-time and are still able to recover small details lying on the floor. Our method not only produces geometry but also estimates the texture of the environment.

6.6.1 Performance

Despite certain complexities of the generative approach (especially compared to the simple height map fusion from Section 5.4) we are able to achieve real-time performance even on relatively moderate computational platforms that could be deployed on a small mobile robots. Table 6.1 shows run time measurements for the whole algorithm and its individual components on two different GPUs, the NVIDIA GTX 980 and NVIDIA GT 650M. The latter is a mid-level mobile GPU, comparable in performance with NVIDIA's embedded board Jetson TX1. Jetson TX1 has a credit-card footprint and power consumption of about 15W. Note that the effective frame rate can be even higher, as in our system the *Depth Estimation* and *Fusion* do not have to be executed for each video frame — these are performed only when there is a sufficient baseline due to camera motion.

Run time	GTX 980	GT 650M
Tracking [ms]	2.18	14.5
Depth Estimation [ms]	5.40	24.7
Fusion [ms]	3.41	26.9
Total run time [ms]	10.99	66.9
Total frame rate [fps]	**91.0**	**15.1**

Table 6.1: Timings for the reconstruction algorithm on two different GPUs.

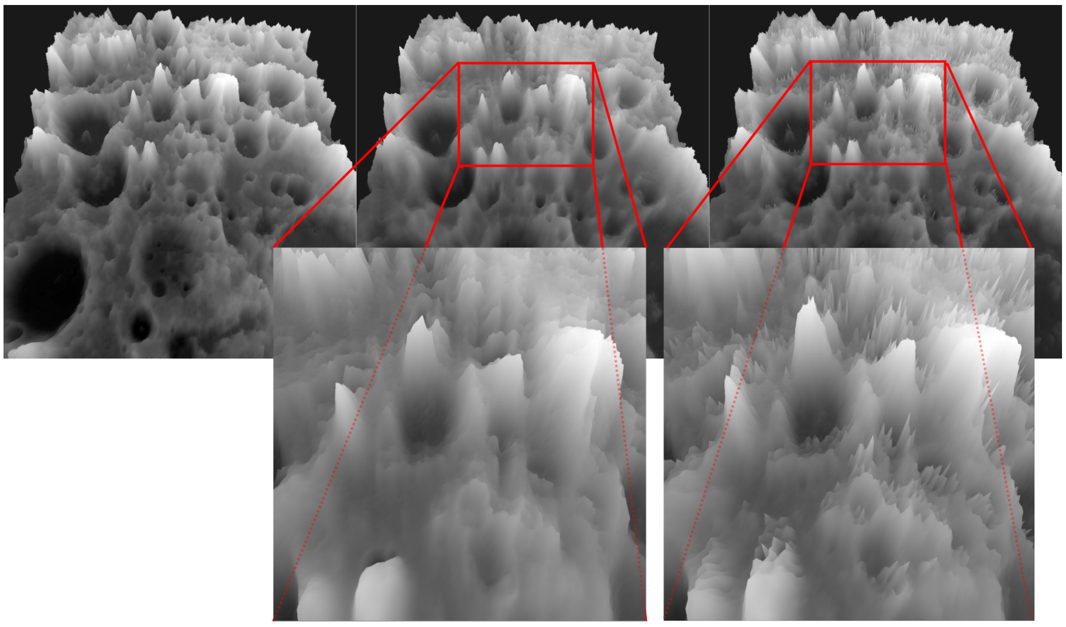


Figure 6.6: Evaluation of the fusion algorithms on synthetic data: (left) ground truth, (middle) reconstruction using the differentiable rendering method, (right) reconstruction using simple height map fusion from Section 5.4. In general, both methods are able to accurately reconstruct the surface from just a few depth maps. However, by incorporating smoothness priors, a method based on optimisation better deals with situations where the resolution of the height map is high compared to the density of measurements (*e.g.* due to oblique viewing angle).

6.6.2 Synthetic Data

To demonstrate the correctness of the differentiable rendering approach, we evaluated it using synthetic "moon" data (Fig. 6.6) and compared with the simple height map fusion algorithm from the previous chapter that treats all height cells as independent. Both methods are capable of accurately reconstructing the surface. However, in the methods based on optimisation (the differentiable rendering presented here and the method from Section 5.5) by modelling the connectivity between vertices of the height map and incorporating smoothness priors, we can better handle situations that arise when the camera is looking at the scene from a very oblique angle. There, for the points that are far from the camera, the resolution of the height map is often high compared to the density of the depth measurements; this leads to reconstruction artefacts when the height vertices are assumed to be independent.

6.6.3 Comparison against a Generic 3D Reconstruction with a Depth Camera

We qualitatively compared our monocular height map estimation to a more generic 3D reconstruction method, which uses a depth camera. In the comparison we used ElasticFusion [Whelan et al., 2015], a system that takes high quality depth maps obtained by a Kinect camera and is capable of creating globally consistent 3D models. Our method, like any incremental, open-loop system, is subject to drift and is not designed to directly compete with ElasticFusion in terms of global accuracy. However, as shown in Fig. 6.7 our approach is capable of creating maps of the ground with similar local accuracy.

6.6.4 Free Space Detection

A desirable property of a height map is that it can be directly used for robot navigation and obstacle avoidance. For example, one can determine the drivable area by simply thresholding the height values. Fig. 6.8 illustrates the results of applying this approach to our reconstructions. There, for each pixel in an image, we check the height of the associated grid cell and label it as free space or not based on a fixed, 1 cm threshold. The created free space mask is subsequently overlaid onto the observed image. Despite the fact that a height map cannot correctly model overhangs, our approach exhibits desirable behaviour even in these scenarios. The method in its current implementation is robust, especially for the task of free space detection. We attribute this to the smoothing behaviour of the height map representation that we use in our method. The performance of our method is best illustrated in the following video: `https://youtu.be/3NQqeRcSsCw`. The simple thresholding method is also used to detect the planar areas of the environment, which we can use for tracking using planar visual odometry (Chapter 3).

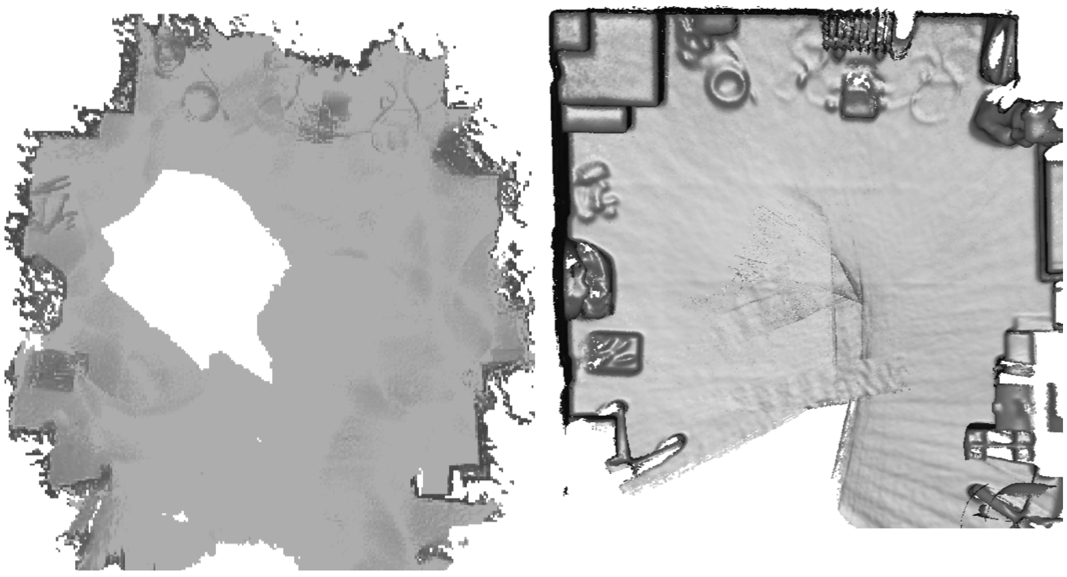


Figure 6.7: A comparison of a reconstruction from our height fusion (left) with high quality reconstruction obtained using ElasticFusion (right). ElasticFusion [Whelan et al., 2015] uses depth maps from a Kinect-like depth sensor and is capable of creating globally consistent models, whereas our method is open-loop and based on monocular depth estimation. This comparison demonstrates that our system is capable of estimating the essential geometry of the room, and still reconstructs details on the floor.

6.7 Conclusions and Discussion

We have demonstrated that a generative approach based on a carefully chosen height map model and efficient differentiable rendering can allow usable real-time monocular dense reconstruction and rigorous incremental probabilistic fusion. The presented approach is optimal up to linearisation errors and discards no information, while the computational complexity is bounded. Using current GPUs, rendering can be done extremely efficiently and the proposed generative approach directly utilises the standard rendering pipeline (*e.g.* OpenGL). This is in line with the main motivation of our work, where we focus on algorithms that can be well expressed and implemented using commodity parallel processors and frameworks. We can also see this as an example of using computer graphics methods and techniques to benefit computer vision. In the following chapter, when presenting a multi-resolution method for surface reconstruction, we will take those ideas a step further.

However, using a generative approach comes at a cost: the complexity and computa-

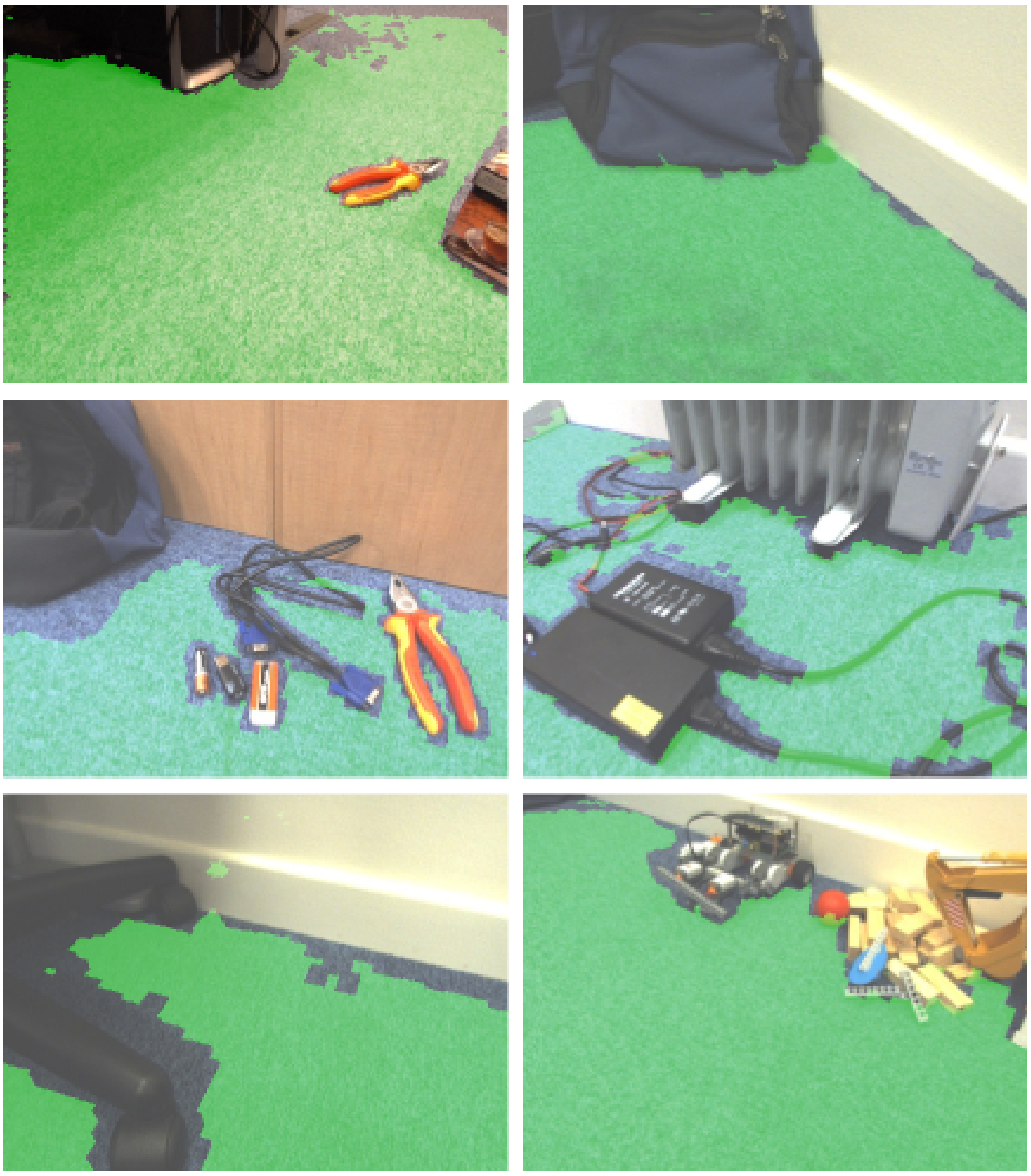


Figure 6.8: Free space detected (green overlay) by simple thresholding of the reconstructed elevation map.

tional demands of height fusion have greatly increased compared with the approaches from the previous chapter. A subtle change in the strategy for data association that is a consequence of using a generative approach (through rendering instead of back-projection), makes the associated optimisation problem nonlinear and thus more difficult to solve. Whereas previously, when iteratively solving a linear least squares problem, we could always pause the solver and add new data to the problem, now we have to optimise until convergence before we can process additional data, as otherwise we could introduce *linearisation* errors. The comparison between these two approaches highlights the importance of choosing the right parameterisation and approximation when designing algorithms.

In general, one can consider the method derived from generative model as a more formal (and elegant) approach to surface reconstruction, and see the approaches presented in the previous chapter as application-driven approximations and simplifications. This demonstrates the versatility and flexibility of height map approaches for modelling and mapping of the environment of small mobile robots moving in an indoor environment. A particularly interesting aspect of height map fusion is it promising scalability, both in terms of input image resolution and scene representation. We can envision a very efficient implementation, which should be possible at lower resolution, perhaps for extreme resource-limited platforms (a very simple height map fusion approach from Section 5.4 can be easily implemented on both CPU and GPU). On the other hand, depending on the computational resources available and requirements we can push the limits of the system towards very high resolution fusion and use more sophisticated approaches that model full posterior probability, capture both geometry and colour and, for example, can allow tracking using full image alignment with respect to the estimated model.

A particular advantage of the generative approach is the great scope for the expansion of this methodology. Within a probabilistic framework we can easily model and express many aspects of the image formation process, such as camera intrinsics, radial distortions, rolling shutter, blur, camera gain — and improve accuracy by jointly estimating these quantities. The method presented in this chapter is a loosely-coupled approach, but theoretically it should be possible to create a unified framework for calibration, tracking, reconstruction, that goes straight from the observed intensities into surface geometry and appearance without the intermediate step of depth map estimation and depth map fusion. However, it still remains an open question and a subject to future research if using a fully generative approach based on rendering is a sensible and practical solution for robot perception. As many authors have found [Smelyanskiy et al., 2000; de La Gorce et al., 2008; Gargallo et al., 2007; Delaunoy et al., 2008], for this approach to work, it requires a good initialisation of the geometry (which means we might still need to calculate depth maps) and proper handling of occlusion boundaries which makes it less robust and practical; for example, in the gradient flow method [Delaunoy and Prados, 2011], the initialisation is obtained from a visual hull. A possible extension of our work could use fully a generative model and image-based reconstruction as a post-processing and refinement step, where we use a batch of images to improve the model and recover details that were not possible to obtain using depth maps only.

CHAPTER 7

Multi-scale Surface Reconstruction using Dynamic Level of Detail

Contents

7.1 Introduction

One of the limitations of the previously discussed approaches for the surface reconstruction is that they rely on a fixed sized grid. Fusing the depth measurements with different scales into a single resolution representation of the environment is problematic, and often leads to system design decisions which incur performance or quality penalties. For example, using a fine resolution throughout will result in high memory consumption and can lead to aliasing artefacts when the density of the measurements is low relative to the resolution of the model. On the other hand, when the resolution is too coarse, the ability to capture small details in the scene is severely limited.

The problem of using a fixed resolution representation is particularly apparent in a monocular system: a 3D reconstruction system based on a moving monocular camera is effectively a variable-baseline multi-view-stereo system, and unlike a depth camera or stereo rig, does not have a fixed minimum or maximum range. As a camera browses a scene we can estimate depth maps at different scales; we can use small baselines when the camera is close to objects to capture fine details, and when the camera is far away, we can observe the global, coarser structure. Furthermore, even within a single image we can observe the environment at different scales: for a mobile robot with the camera looking forwards that we consider in this thesis, due to the oblique angle the camera is looking at the scene, the required resolution directly in front of the robot is much higher compared to the objects and parts of the environment further away from the robot.

In this chapter we present an efficient and scalable algorithm for multi-resolution fusion that naturally supports and harnesses the superior characteristics of a monocular system and can deal with a wide range of scales in real-time. The key requirement for our method is to represent the surface using a triangular mesh. We will use height map fusion method based on the Gauss-Seidel solver from Chapter 5 as an underlying surface reconstruction method, but the multi-scale approach presented here can theoretically be used in conjunction with all three surface reconstruction approaches presented so far, including the differentiable rendering approach.

There are two main concepts that we will utilise in the multi-scale framework: Dynamic Level of Detail (LOD) and an implicit multi-scale surface representation based on Laplacian pyramid decomposition. Dynamic (or sometimes also called continuous) level of detail is an important computer graphics technique for maintaining interactive rendering rates and avoiding aliasing artefacts. We use it to dynamically and adaptively determine the best required resolution for each part of the model when fusing a measurement. To represent the environment at multiple scales, we utilise a Laplacian mesh decomposition technique that maintains a hierarchy of approximations of the surface at various resolutions. Thanks to the Laplacian pyramid representation, we can fuse measurements in a coarse-to-fine and probabilistically sound fashion and maintain a single, consistent representation of the environment.

The method presented in this chapter follows the familiar incremental reconstruction pipeline that builds a detailed model of a scene observed by a single moving camera, *i.e.* we track camera motion and calculate depth maps using multi-view stereo, and we update the model every time a new depth map is available. The proposed method is not restricted to robotics applications and we will consider here also a more general set-up that allows the camera to move freely in 3D space in order to observe the environment and capture depth maps at various scales. We demonstrate the ability of the system to reconstruct scenes with adaptable resolution down to fractions of a millimetre from standard real-time monocular video input, and we will show that our work offers a scalable system that can perform both very fast reconstruction and achieve high quality results.

7.2 Related Work

7.2.1 Level of Detail in Computer Graphics

Our approach relies on the level of detail concept developed in computer graphics. Multi-scale and level of detail object representations play an important role in rendering of complex geometric models, and enable obtaining visually appealing and accurate rendering results while reducing computational load and memory bandwidth. The importance of level of detail methods has led to a plethora of different approaches, that tackle various aspects of LOD representations such as avoiding cracks, or optimality, *i.e.* what is the best lower-resolution representation of a highly detailed mesh. Historically, most LOD computations and pre-processing were first performed on a CPU, and a GPU was only utilised for final rendering. Advances in GPU and programming models led to interest in development of algorithms that best suit the parallel architecture of a modern GPU and now dynamic LOD algorithms can be relatively easily implemented within a standard OpenGL pipeline using tessellation shaders.

Clearly, a comprehensive review of LOD techniques is beyond the scope of this thesis, and we will just highlight the work most relevant to us. One of the most influential pieces of work on multi-scale representations is the paper by [Hoppe, 1996] who introduced the progressive mesh scheme, a continuous-resolution representation of arbitrary triangle meshes. Progressive Meshes allow for a smooth choice of detail level depending on the current view, and were used for high-quality, continuous level of detail rendering in various scenarios [Hoppe, 1997, 1998]. The method produces compelling results, and is designed for arbitrary meshes, but it requires rather complex pre-processing that is not suitable for an incrementally reconstructed mesh.

More relevant and similar to our level of detail approach are methods based on regularly sampled, hierarchical structures such as grid quad-trees [Lindstrom et al., 1996] or Real-time Optimally Adapting Meshes (ROAM) [Duchaineau et al., 1997]. A basic building block of these methods is a patch that represents a small area of the terrain. Each triangle within a patch can be recursively tessellated by binary subdividing its edges until the desired level of detail is reached. Methods based on partitioning of the mesh into patches are very simple and efficient. Another notable example of a LOD method that was specifically designed for large-scale terrain rendering is the geometry clip maps approach [Losasso and Hoppe, 2004], which caches the terrain in a set of nested regular grids centred about the viewer, in a similar way to how texture clipmapping works.

7.2.2 Multi-scale 3D Reconstruction

Explicitly handling scenes with multiple scales within a 3D reconstruction framework is a challenging problem, but it is necessary for obtaining state-of-the-art results, as shown, for example, in [Fuhrmann and Goesele, 2014]. One of the natural choices for multi-resolution representation is to rely on octrees to efficiently partition the space; for example, [Fuhrmann and Goesele, 2011] proposed a method for 3D reconstruction that regularises samples from depth maps at a higher resolution with depth map at coarser scales and fuses them into an octree representation. [Ummenhofer and Brox, 2015] proposed a variational approach for surface reconstruction which includes the scale information directly in the objective function and also solves the optimisation problem on an octree grid. These approaches are off-line methods that globally optimise a batch of images, which makes selection of appropriate scales much easier. Although they have shown remarkable results, they are highly prohibitive for real-time applications where processing should be fast and the reconstruction should be updated incrementally.

In the field of real-time SLAM approaches that operate in an incremental fashion, the main emphasis is usually put on *scaling-up* the reconstruction, rather than obtaining very accurate and detailed models. When designing a large scale, real-time dense reconstruction system, much effort is typically focused on reducing the amount of memory and resources spent on processing "empty" space and therefore these methods are rarely particularly good at dealing with scale changes. As in batch methods, real-time SLAM approaches also often rely on hierarchical data structures [Chen et al., 2013], *e.g.* multi-scale octree representation for TSDF [Steinbrücker et al., 2013]. Kintinuous [Whelan et al., 2012] extracts a dense point cloud from a TSDF volume and incrementally adds the resulting points to a triangular mesh representation of the environment. [Stückler and Behnke, 2014] tackled the problem of multiple scales by representing the environment using a set of surfel maps at different resolutions.

[Nießner et al., 2013] presented a very interesting technique that uses a spatial voxel hashing scheme to compress space, and does not require a hierarchical grid data structure. However, this method was designed to be used mainly with high quality depth data, tends to be quite complex, and involves processing on both CPU and GPU. [Kähler et al., 2016] extended the spatial voxel hashing technique and equipped it with the ability to represent parts of the scene at different resolution levels. Thanks to the adaptive scale selection, the method achieves up to 50% reduction in memory footprint compared to a fixed discretisation grid.

7.3 Dynamic Level of Detail and Multi-scale Surface Representation

7.3.1 Level of Detail

In the most standard approach to level of detail one explicitly maintains different representations of an object at different resolutions. Typically, we start with a very detailed mesh that is gradually simplified and remeshed in order to create a

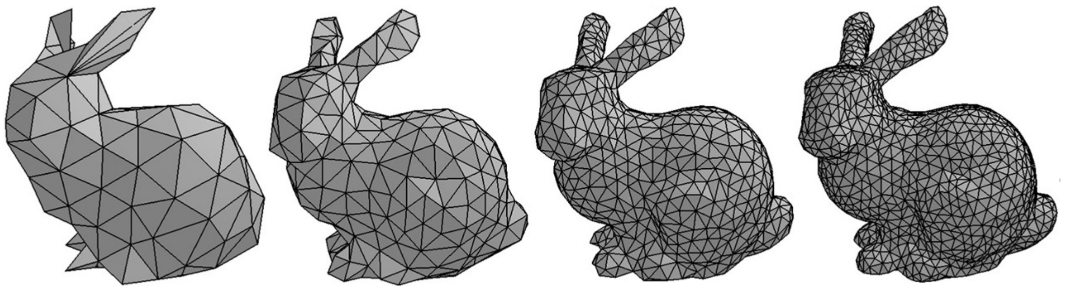


Figure 7.1: A demonstration of the level of detail concept. In the simplest form, one maintains explicit representations of an object with different model complexities (image from [Peyre and Cohen, 2006]).

very coarse model. Remeshing and mesh simplification is typically done off-line as a pre-processing step, and is a very challenging problem widely studied in the literature, for example, in [Peyre and Cohen, 2006]. Fig. 7.1 shows an example of an object represented at four different resolutions. During rendering, based on various criteria, *e.g.* expected on-screen size and other metrics such as object importance, viewpoint-relative speed or position, an algorithm selects on the fly the suitable representations of the object.

7.3.2 Multi-scale Surface Representation based on Laplacian Pyramid Decomposition

The approach that explicitly stores a representation of an object at various resolutions works well for simple rendering, but it is undesirable for incremental reconstruction as it would be very difficult to maintain consistency between the representations of the same object at multiple scales. Alternatively, we can use implicit representations, and there are a few possible underlying data structures and techniques to model a surface at multiple scales. A common choice for multi-resolution mesh representation is wavelets-based [Eck et al., 1995; Lounsbery et al., 1997]. Wavelets offer a very elegant, compact and powerful framework, but when used for fusion, this comes at the cost of significantly increased processing complexity. The dependencies between the wavelet coefficients in a reconstructed signal make it very challenging to maintain and a update full posterior distribution over surface parameterised by wavelet coefficients.

Our surface representation method relies on a Laplacian pyramid decomposition of the mesh on a regular grid [Guskov et al., 1999], also referred to as the Burt-Adelson pyramid scheme [Burt and Adelson, 1983]. In the standard Laplacian pyramid decomposition, one separates a signal (typically an image) into *high-pass* and *low-pass* bands by successive blurring, calculating the difference between a blurred and original signal, and downsampling. A Laplacian pyramid is similar to a wavelet transform, and it allows us to represent properties of the mesh at different scales. However, unlike the wavelet transform, a Laplacian pyramid is a redundant representation of the mesh making it slightly less compact than wavelets. This will turn out to be advantageous, because we will be able to assume that individual levels within the pyramid are independent (*e.g.* the coefficients at scale n are independent of the coefficients at level $n + 1$), which greatly simplifies fusion and enables efficient recursive estimation.

Let us denote by $\mathcal{B}$ the base mesh that captures the coarsest geometry and let us store it using a fixed grid of size $n_B \times n_B$. We will use $\mathcal{D}$ to represent a "detail" mesh that stores only the high-frequency details, not captured by the coarser mesh. As we use an oversampling factor of 2, the size of $\mathcal{D}$ is $(2n_B - 1) \times (2n_B - 1)$; in practice we only need to store the details in subgrids, where they are required, which can help reduce memory consumption. In our system, we allow up to 6 detail levels, $\mathcal{D}_i$ for $i = 1 \ldots 6$, each with increased resolution compared to the previous level, that store only the differences between the higher resolution and the lower resolution meshes.

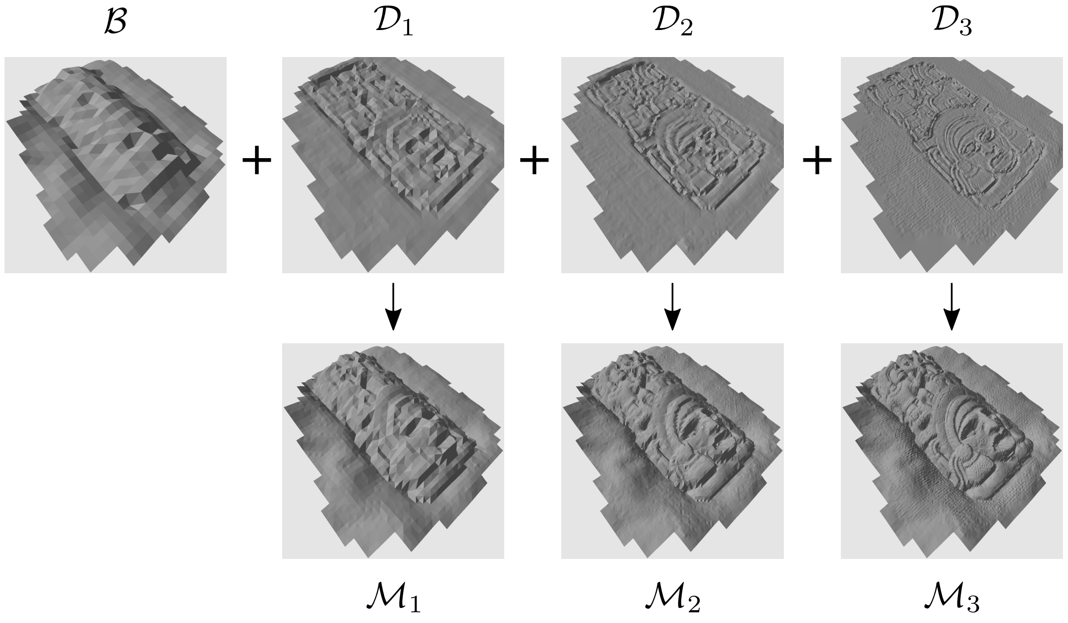


Figure 7.2: Multi-scale mesh representation based on Laplacian pyramid.

As demonstrated in Fig. 7.2, starting with the coarsest mesh $\mathcal{B} = \mathcal{M}_0$ we can generate a sequence of meshes $\mathcal{M}_i$ at increasingly higher resolution, by adding detail coefficients:

$$\mathcal{M}_i = \mathcal{B} + \sum_i \mathcal{D}_i \ . \tag{7.1}$$

Thus, during the rendering process, we perform an operation which is the inverse of Laplacian decomposition, *i.e.* we assemble the mesh using the coarsest level, and add the required details levels.

7.3.3 Tessellation and Vertex Assembly

Generation of a high resolution mesh from the base mesh $\mathcal{B}$ and detail coefficients $\mathcal{D}_i$ requires two steps: tessellation and vertex assembly. During tessellation we subdivide triangles in the base mesh to introduce new vertices, and during vertex assembly we update the positions of newly introduced vertices. This is highly related to the concept of Displacement Mapping, where an existing mesh is subdivided and a displacement map (in our case detail coefficients) is used to displace the vertices of the subdivided mesh. Obviously this process can be repeated recursively, where a newly generated mesh becomes the base mesh for the next level.

Fig. 7.3 shows the tessellation pattern that we can use to increase the resolution of the mesh. We store the base and coefficient meshes on regular grids, thus when going from one level to another, we simply divide each edge of a triangle in half by introducing new vertices and therefore split a triangle into 4 smaller ones. This procedure can be repeated recursively, in total 6 times, and therefore with the base geometry we can create, in total, 7 levels of detail (and achieve a 4096-fold resolution increase).

During vertex assembly, Fig. 7.4, we first predict the position of a newly introduced vertex by interpolating the coarser mesh (we use interpolation based on barycentric coordinates, but many choices are available here, *e.g.* bilinear interpolation). Next, we displace the vertex by adding a “detail” coefficient from the finer resolution. The vertex assembly is more resource consuming compared to using a precomputed mesh at different resolutions, as it requires multiple memory accesses per vertex instead of a single, direct look-up of a vertex value, but it greatly simplify the fusion algorithm as we will see in Section 7.4.

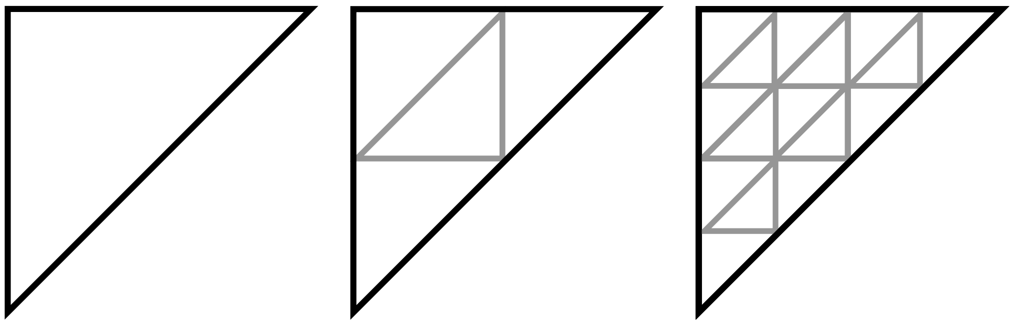


Figure 7.3: When subdiving a triangle, we use a regular tessellation pattern: left is the base triangle, middle and right are two consecutive levels of detail, level 1 and 2, where the triangle is respectively subdivided into 4 and 16 smaller triangles.

7.3.4 Rendering using Dynamic Level of Detail

Thus far we have described a technique that enables us to generate, on the fly, surface representations at different resolutions. However, working with one global resolution all the time might not be practical, for example, when the camera is looking at the surface from far or at an oblique angle, and it typically results in aliasing artifacts when the selected resolution is too high compared to the image resolution. Dynamic LOD algorithms specifically address those issues by adapting the complexity of a 3D object representation based on various criteria and requirements. In our surface representation based on Laplacian pyramid decomposition, we can assemble different parts of the mesh at different resolutions, and this is a requirement for an implementation of a dynamic level of detail algorithm.

There are many different heuristics and techniques in computer graphics for dynamically selecting level of detail; for example, one can use the distance between the rendered object/triangle and camera. For the rendering of surface models and height fields, one often selects the tessellation level based on the length of the edges or area of the triangle.

Our dynamic level of detail algorithm selects the tessellation level for each triangle individually and in such a way that all rendered triangles have approximately the same size. In our implementation we utilise the tessellation unit of a modern rendering pipeline and therefore process as follows: at the first stage we supply the current estimate of the coarse mesh to the tessellation shader. Given the camera pose with respect to the mesh, and camera intrinsics, each triangle in the coarsest mesh $\mathcal{B}$ is projected onto the virtual camera plane and its area is calculated. A parameter

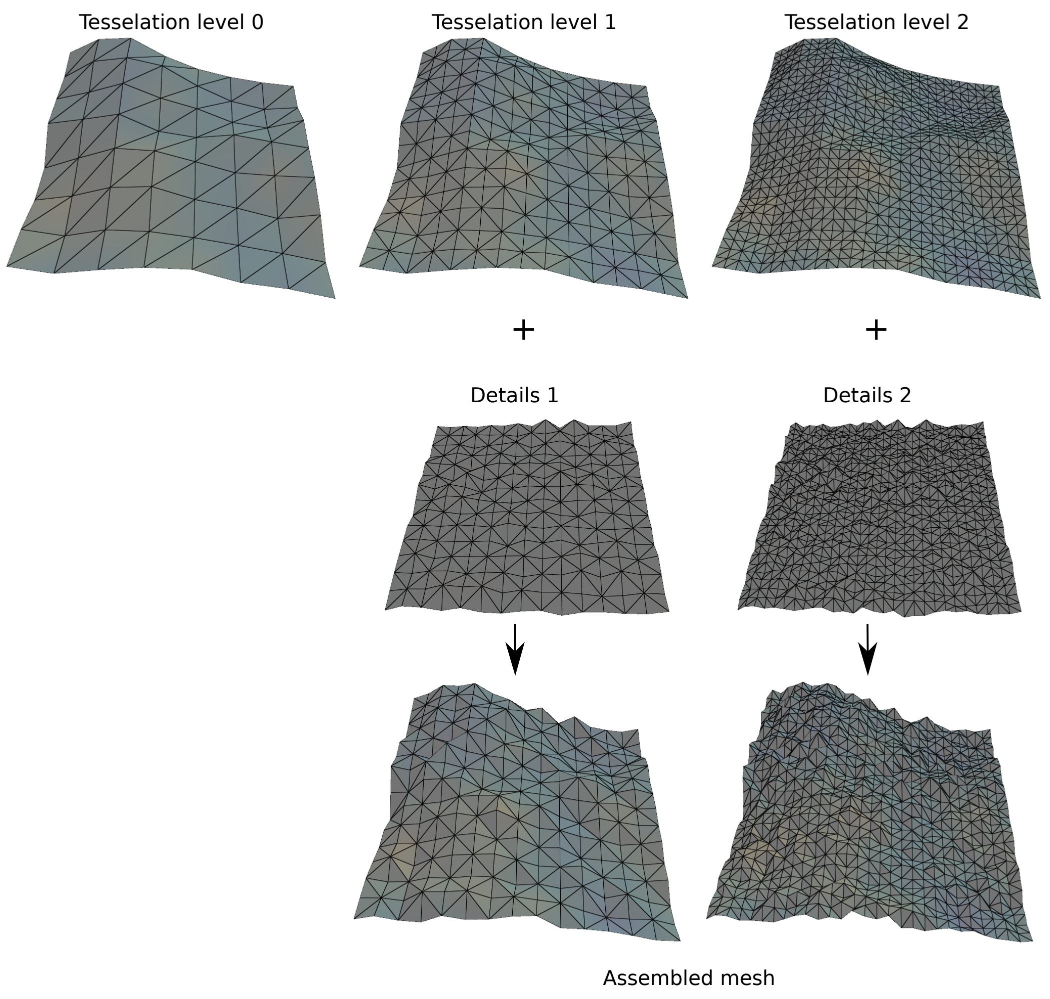
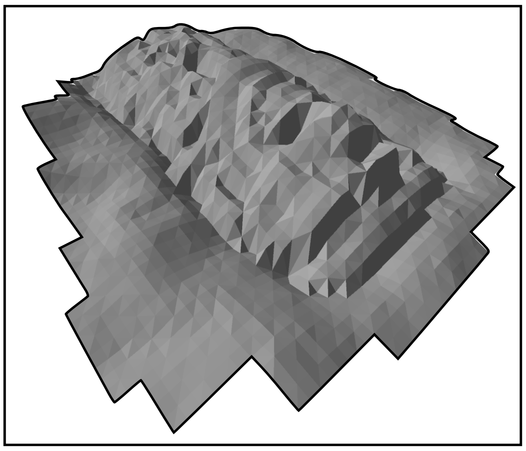


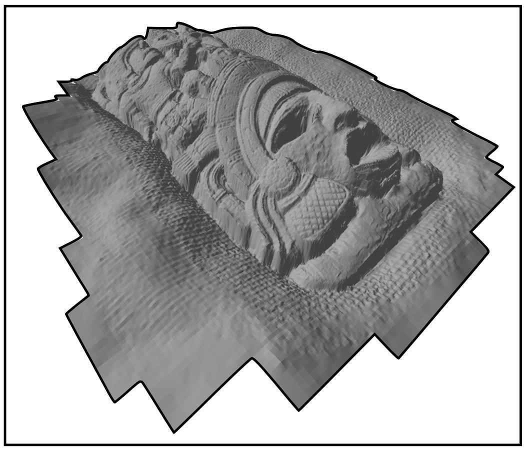
Figure 7.4: An example of tessellation and vertex assembly. In the first step, we introduce new vertices by subdiving triangles in the base mesh and subsequently, update the positions of the vertices on the tessellated mesh using details coefficients. Note that the tessellation pattern shown here is slightly different from Fig. 7.3, but this is an implementation detail: the tessellation shaders we utilise in our implementation exhibit certain limitations in order to achieve the best performance and benefit from hardware acceleration. Nonetheless, the resulting tessellation pattern is fixed and regular and therefore we can still use all the mesh processing and reconstruction machinery introduced so far.

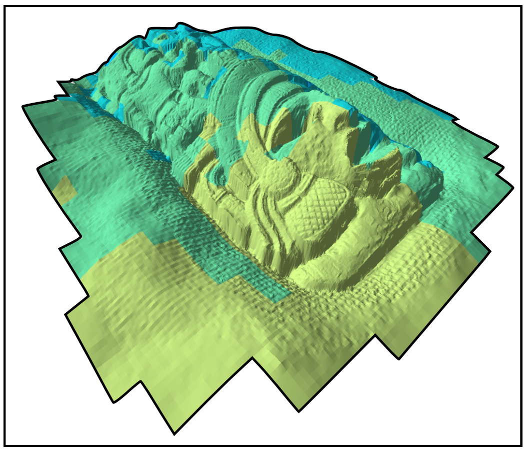
controlling the LOD is the desired triangle area, which tells us how many times the triangle should be divided, and the LOD is calculated using:

$$l = \text{round}\left(\log_2\left(\frac{[\triangle_{\mathcal{B}}]}{a}\right)\right), \tag{7.2}$$

where $[\triangle_{\mathcal{B}}]$ indicates the on-screen area of the base triangle, and a is the desired


(a) Coarse resolution: 2400 fps.


(b) High resolution: 120 fps.


(c) Dynamic resolution: 1300 fps (here colour indicates different LOD). There is no visual difference between high and dynamic resolution.

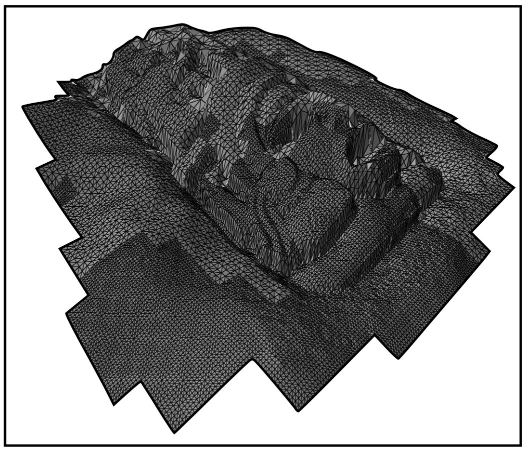

(d) Mesh generated by dynamic tessellation.

Figure 7.5: Rendering using different mesh resolutions: (a) coarse mesh, (framerate approx. 2400 fps); (b) high resolution mesh, (120 fps); (c) rendering using dynamic level of detail (different colour indicates different LOD). In figure (d) we can clearly recognise how different parts of the mesh are generated using different tessellation levels. In general, the closer a base triangle is to the camera, the higher the tessellation level, and more details are added to the mesh.

area (we usually set it to 4 pixels). Once the tessellation level for a triangle has been determined, it is subdivided by introducing new vertices, and subsequently the assembly of vertex values of the newly generated geometry is performed based on the Laplacian pyramid formulation described above. At this stage we can also discard geometry that is clearly not visible in the current frame (*e.g.* is behind the camera) to further improve performance and rendering rate.

Obviously, the proposed strategy is a simple heuristic and it does not guarantee that the tessellated triangles will have the desired size, especially as during this phase we do not take into account the vertex displacements coming from the "details". Also, we select a global tessellation for the whole triangle, which might not be optimal for all areas of the triangle, especially when we are looking at the triangle from an oblique angle. Furthermore, because we are selecting one global LOD per triangle, and not per edge, sometimes neighbouring triangles can have different tessellation levels. When two triangles sharing an edge have different tessellation levels, it can easily result in visible cracks in the mesh. Cracks are quite undesirable for visualisation, but they do not significantly affect the fusion algorithm and are typically small. One further limitation of our approach is the fact that the multi-scale representation on a regular grid is usually suboptimal, *i.e.* the edges in the mesh are unlikely to correspond to natural features of the surface. However, thanks to its simplicity, the proposed method achieves extremely high rendering (and therefore prediction) rates even for complex models. Fig. 7.5 shows the difference between rendering rates for the dynamic and static LOD models.

7.4 Multi-scale Surface Reconstruction

We will now describe our multi-scale fusion approach that combines Laplacian-based surface decomposition, dynamic tessellation and level of detail as described in the previous sections, together with the optimisation-based surface reconstruction of Chapter 5 within a single framework.

The key assumption for our fusion method is that individual levels within the Laplacian pyramid are independent of each other, and that they capture different aspects of an observed "signal" (separate frequency bands). Rather than explicitly estimating representations of the surface at different scales, we directly estimate the coefficients of the Laplacian pyramid. This means that we only maintain the per-level normal equation, $\mathtt{J}_k^\top \mathtt{J}_k$, $\mathtt{J}_k^\top \mathbf{z}_k$, and estimate the parameters within the level by updating and solving it.

More specifically, given a current estimate of the surface model and camera pose, we first perform dynamic level of detail rendering to tessellate each part of the mesh up to the required resolution. We then proceed with fusion in a coarse-to-fine fashion. Starting from the coarsest level a depth measurement is fused into all the levels

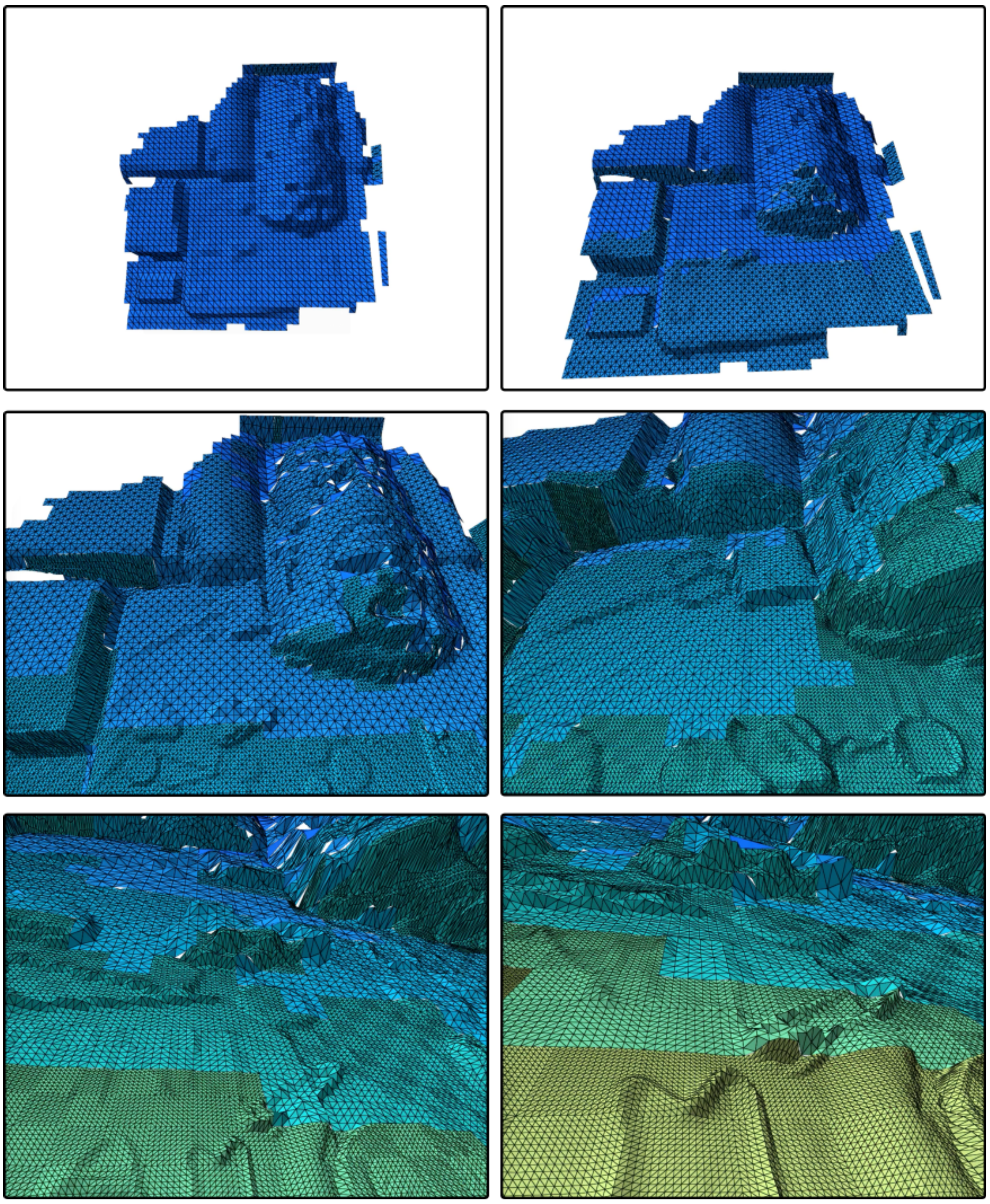


Figure 7.6: Rendering a scene that is dynamically tessellated using our level of detail algorithm. Note how we add more triangles to the model, as we are getting closer to the objects. Different colours indicate different levels of detail.

up to the selected finest one. Our Laplacian surface parameterisation assumes that the levels are independent, and only contain the details/frequencies that were not captured by the previous level. Thus, after a height measurement h_i has been fused into a level k, we first make a prediction of the height at this level, $\hat{h}_i^k$, and in the subsequent level $k+1$ we only fuse the residual between the predicted height and the measured height:

$$r_i^{k+1} = h_i - \hat{h}_i^k \ . \tag{7.3}$$

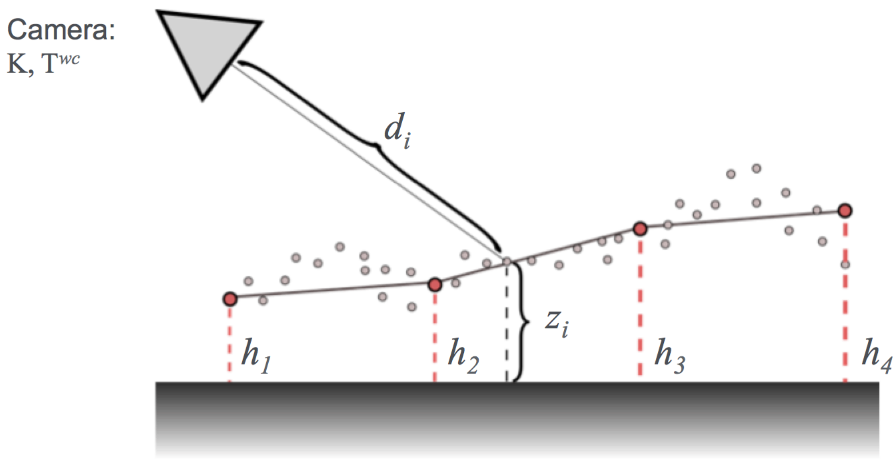
This is repeated recursively for each measurement, until the required level of detail has been reached. Fig. 7.7 demonstrates our approach on an intuitive 2D example. When calculating a residual that is fused into a subsequent level, we also have to calculate its uncertainty. This is relatively straightforward: as we can see in Eq. (7.3) the only transformation a measurement h_i undergoes is a subtraction of a constant, therefore following the rules of propagation of uncertainty it can be shown that the uncertainty of r_i^{k+1} is the same as the uncertainty of the height measurement h_i itself. Put simply, when propagating a measurement across levels, we can reuse the measurement's uncertainty.

Note that before we can calculate the residuals and fuse them into the next level, we have to make sure that the optimisation has converged, *i.e.* that we have solved:

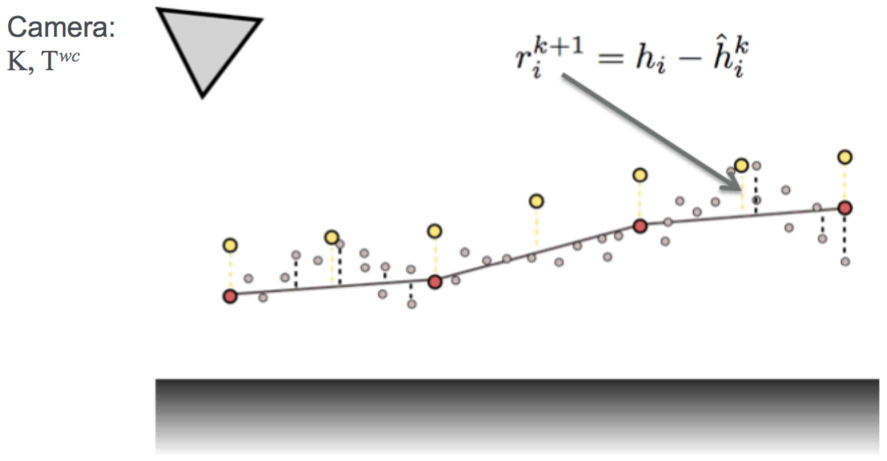
$$\mathsf{J}_k^\top \mathsf{J}_k \mathbf{h}_k = \mathsf{J}_k^\top \mathbf{z}_k \ . \tag{7.4}$$

In practice, we only proceed into the next resolution level after the vertices in the preceding level have reached a certain stability. Here we simply look at the magnitude of the diagonal entries of $\mathsf{J}_k^\top \mathsf{J}_k$ associated with a triangle, which are good proxies for the stability, as they represent per-vertex sums of squared barycentric coordinates from all the measurements thus far. This procedure locks the gauge freedom that would be present if we solved for all heights at different resolutions simultaneously. Although we used $\mathbf{h}_k$ in Eq. (7.4) to denote a quantity we are estimating at each pyramid level, only at the coarsest level of the mesh we actually estimate the height values, and use directly height measurements, $\mathbf{z}_k$. In all subsequent "detail" levels, we estimate the details coefficients using the "residuals" calculated based on Eq. (7.3).

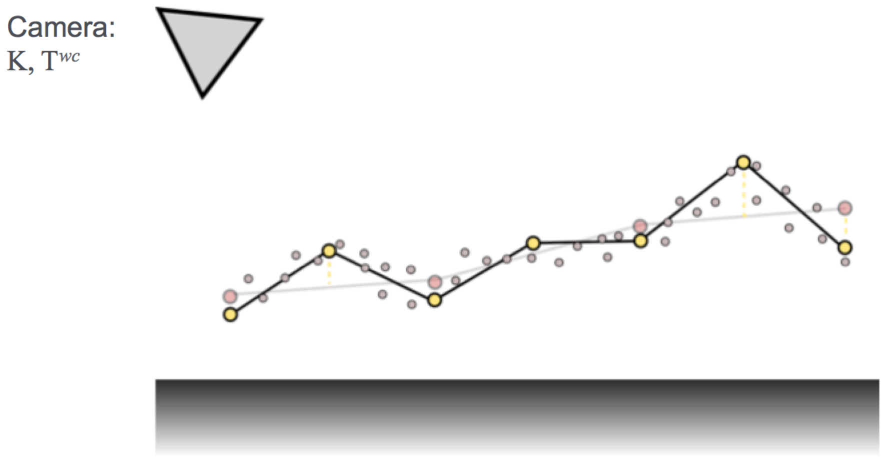
The assumption that pyramid levels are independent of each other allows us to use all the machinery presented already in a straightforward manner, including the principles of recursive estimation from Section 5.7 and the Gauss-Seidel algorithm presented in Section 5.6.2. It is worth noting that the Gauss-Seidel algorithm is well suited and very efficient in solving the systems of linear equations associated with the detail levels. Each detail level captures only a narrow spectrum of high spatial frequencies compared to the grid dimension, and this is the typical setting



(a) We first fuse the measurements into a mesh at the coarsest resolution.



(b) Next, we activate the finer resolution (yellow dots), and calculate the "residuals", *i.e.* the difference between the measurements and predictions from the coarse mesh.



(c) Subsequently, "details" coefficients are estimated based only on the residuals, r_i^{k+1}.

Figure 7.7: A simple 2D example that demonstrates our multi-scale surface reconstruction method.

where "smoothers" and related relaxation methods like Gauss-Seidel are very effective [Trottenberg et al., 2001].

7.5 Implementation Details

To best demonstrate the capability of our method, in particular for obtaining high quality, close-up reconstructions, we did not want to restrict experiments to the typical robot set-up, but decided to test the algorithm in more general settings. Since our fusion method assumes that the camera poses are given, this required a full 6 degrees of freedom camera motion tracking. In our implementation we use ORB-SLAM [Mur-Artal et al., 2015] with its standard settings, but other monocular tracking systems are suitable as well, *e.g.* SVO [Forster et al., 2014] or LSD-SLAM [Engel et al., 2014]. The robust performance of ORB-SLAM and drift-free poses thanks to bundle-adjustment helped us in obtaining consistent reconstructions. We also use the estimated depths of the features detected by ORB-SLAM in the current frame to limit the disparity range searched during stereo matching.

The fusion method is implemented entirely on a GPU and benefits from the hardware accelerated tessellation. The dynamic LOD rendering (Section 7.3.4) utilises the tessellation units of a modern rendering pipeline. Programmable tessellation unit were introduced in OpenGL 4.0, and consist of three stages: tessellation control shader, the fixed-function stage of primitive generation, and a final stage of tessellation evaluation. In our implementation, we supply the coarse mesh to the tessellation control shader, which determines the level of detail for each triangle individually using Eq. (7.2). Based on the output from the control shader, new vertices are generated and the tessellation evaluation shader updates the final positions of the vertices in the mesh. GPU-based tessellation greatly improves the performance and simplicity of our implementation, because all the stages of rendering, including the determination the LOD and vertex assembly, can now be performed fully on a GPU in a single rendering pass/call without any CPU computations. Prior to that, a significant amount of tessellation and computation of LOD were performed on a CPU, resulting in complex interplays between GPU and CPU, and were limited by the CPU/GPU memory bottleneck, whereas now a simple tessellation can be implemented in about 120 lines of code[1]. All remaining computations involving fusion and the solver were implemented in CUDA, and whenever data has to be

[1] `http://codeflow.org/entries/2010/nov/07/opengl-4-tessellation/`

shared between CUDA and OpenGL we use the OpenGL / CUDA interoperability feature of the NVIDIA graphics card. Our implementation (including tracking and depth estimation) achieves real-time performance of 20–25 frames per second on a GTX 680 (most of the time is spent on tracking and depth estimation).

7.6 Experiments

We run a series of experiments on both synthetic and real datasets to demonstrate the practicality and evaluate the performance of our method. We present comparisons with MVE [Fuhrmann et al., 2014], a state-of-the-art off-line, batch-optimisation type method for multi-scale reconstruction, as well as a real-time, point-based method [Whelan et al., 2015] based on the algorithm proposed by [Keller et al., 2013]. We show that our framework can achieve high quality detailed reconstructions but at a run time comparable with Point-based Fusion.

7.6.1 Synthetic Data

The first experiment demonstrates the correctness of our incremental reconstruction method using synthetic data. Fig. 7.8 shows the results of reconstructing a moon-like surface together with surface error obtained using CloudCompare. As a benchmark, we compare the results with an off-line method, MVE [Fuhrmann et al., 2014], which performs global optimisation and considers all images at once. As we can see in Fig. 7.8, our method is capable of obtaining a good quality and correct surface reconstruction while being significantly faster. We should however state here, that the measured timings of the discussed algorithms are not fully compatible as we used different implementations and computational platforms. Whereas our approach was mainly implemented on a GPU (NVIDIA GTX980), MVE is purely CPU-based and in our experiment we used high-end Intel CPU (i7-5930K). This is still in-line with the motivation of our work and highlights the benefits of designing and implementing SLAM algorithms on highly parallel processors.

7.6.2 Real-time Reconstruction

Fig. 7.9 visualises the process of real-time reconstruction of a desk-like environment performed with our method (full video is available at `https://youtu.be/UB_HDJU6LL4`). As the camera is browsing the scene, we can first capture the overall geometry of the scene at a reasonable accuracy. The advantage of our monocular

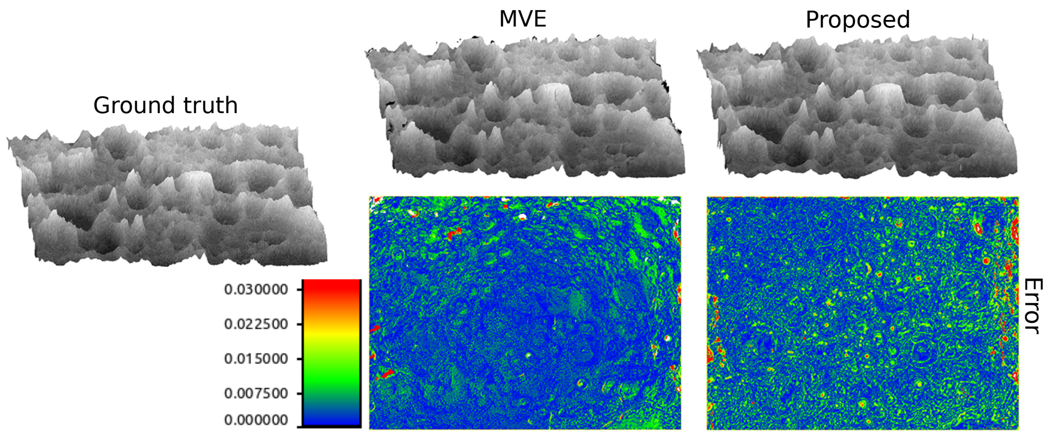


Figure 7.8: Reconstruction of a synthetic moon surface. Left: Ground truth; Middle: Multi-View-Environment (MVE) [Fuhrmann et al., 2014]; Right: Our method. The heat maps below show reconstruction error. Our method achieves reconstruction quality comparable with the batch optimisation method, while running significantly faster (Table 7.1).

	Run time	Avg. error	Std. deviation
Proposed	39 sec.	0.0057	0.034%
MVE	47 min.	0.0037	0.015%

Table 7.1: Run time and reconstruction accuracy of our method compared to the off-line, batch optimisation method (MVE [Fuhrmann et al., 2014]). As expected, the batch optimisation performs better, however, we achieve comparable performance while running our algorithm significantly faster. Note the different computational platforms used in this experiment: our approach was implemented on a GPU (NVIDIA GTX980), whereas MVE is purely CPU-based and we used high-end Intel CPU (i7-5930K).

approach becomes apparent as we start moving the camera closer to objects. Even during significant close-ups we are able to obtain good quality depth maps, and our surface model can automatically adapt its resolution in order to capture small details. Figs. 7.10 and 7.11 present a few highlights of the approach, where within the same framework we can obtain a reconstruction of the whole surface as well as of tiny objects, including coins and paper clips, or even elements on a circuit board.

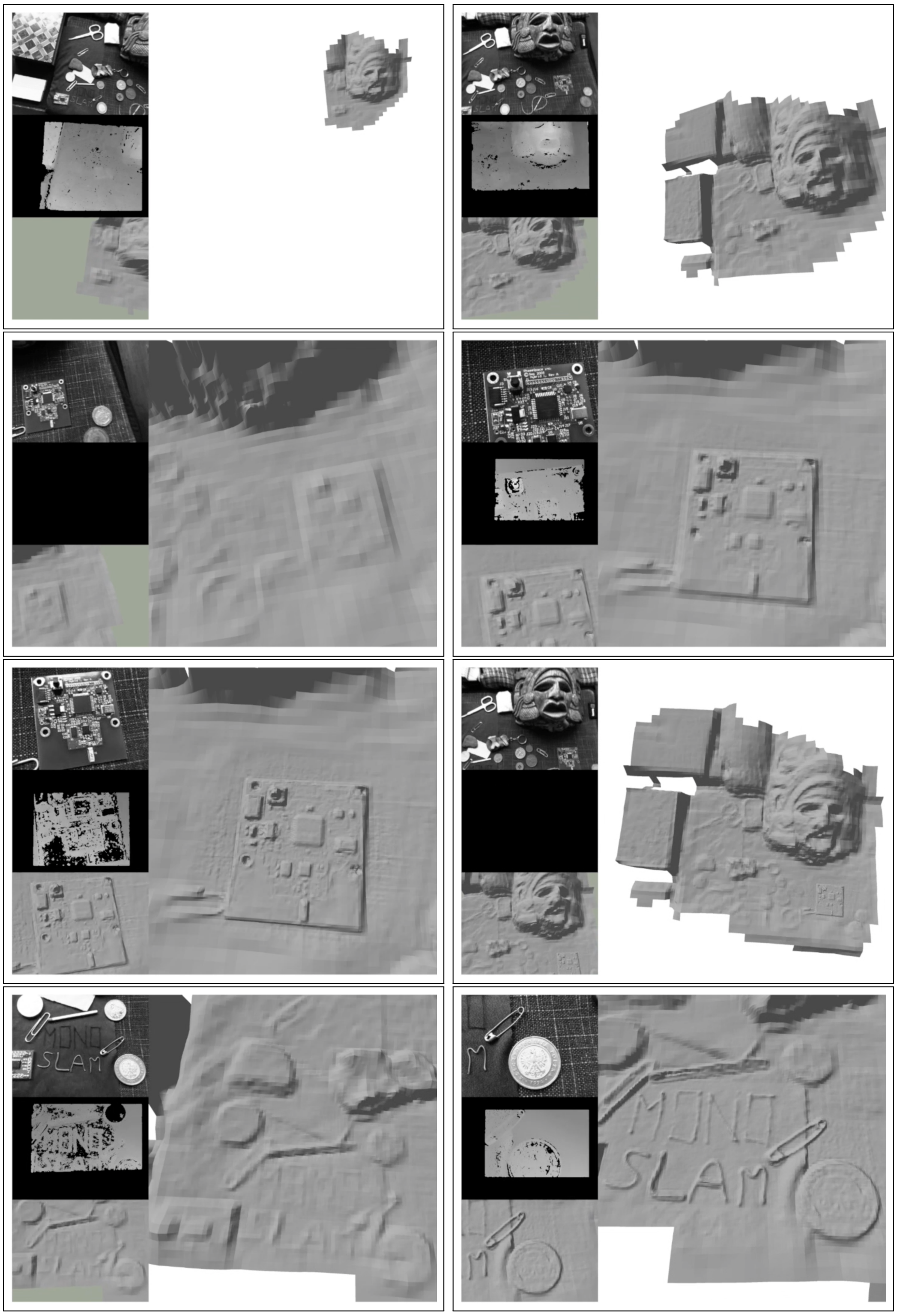


Figure 7.9: An example of real-time reconstruction. Starting from the top left corner: as the camera browses a scene, we first build a model at a relatively course resolution. Next we focus on a small part of the scene to recover details on a circuit board. Subsequently we move the camera to another part of the model where we capture details like coins and a safety pin. Full video available at: https://youtu.be/UB_HDJU6LL4.

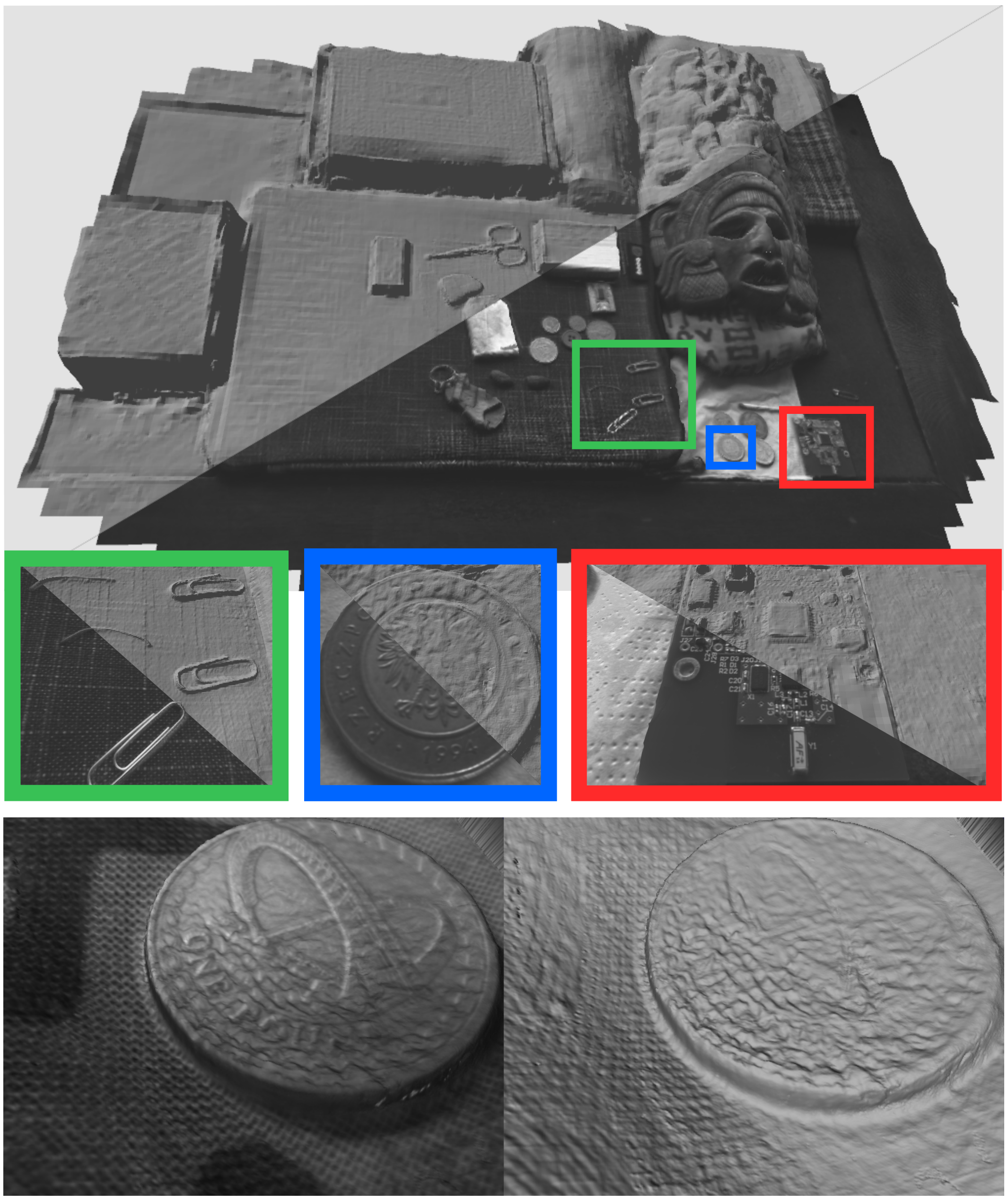


Figure 7.10: Our method efficiently reconstructs a surface model and is capable of creating reconstructions with high quality details. Note that we can even partially recover the very fine texture of a fabric or some details on the coins.

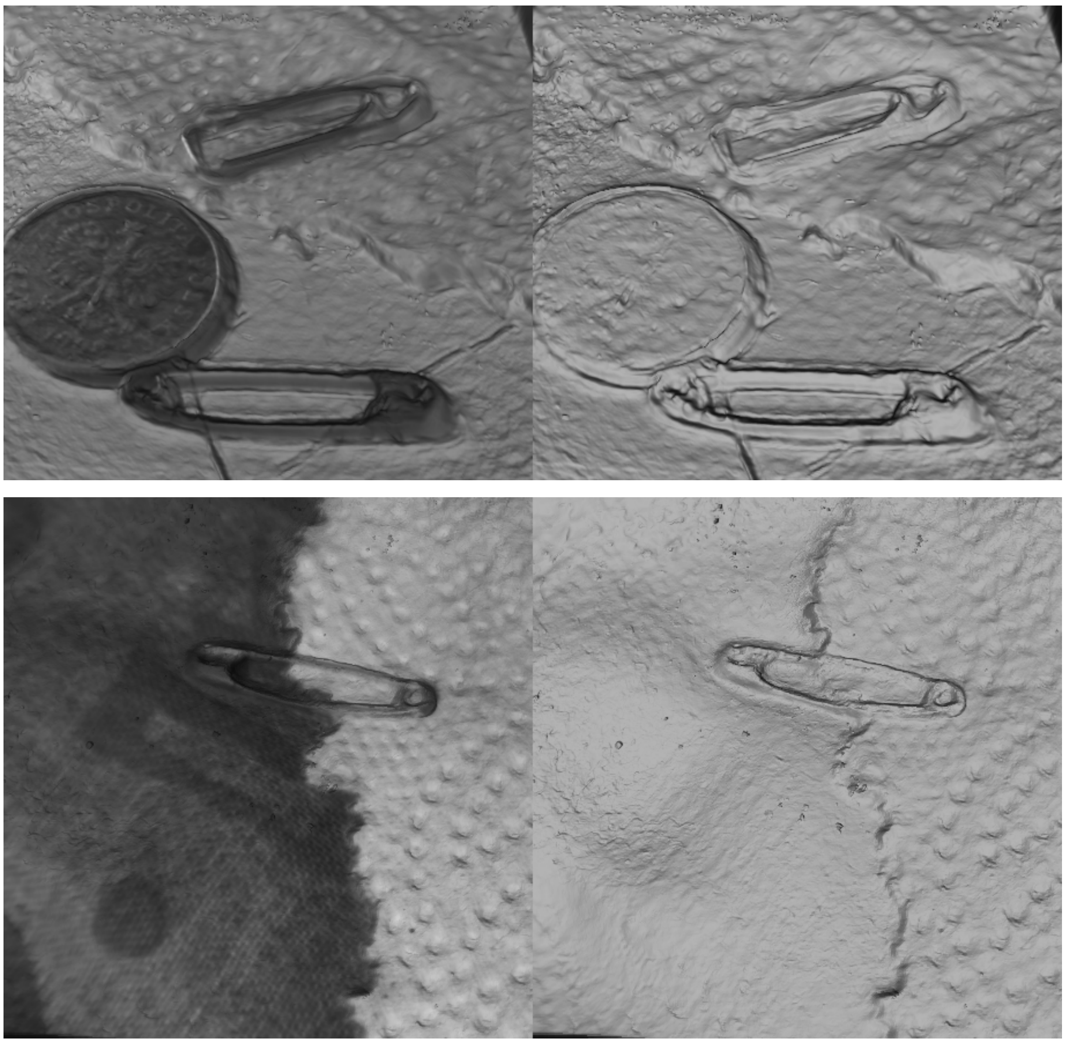


Figure 7.11: Another example of very close-up reconstructions that we are able to obtain using our monocular multi-view stereo pipeline and depth map fusion into the multi-resolution surface model.

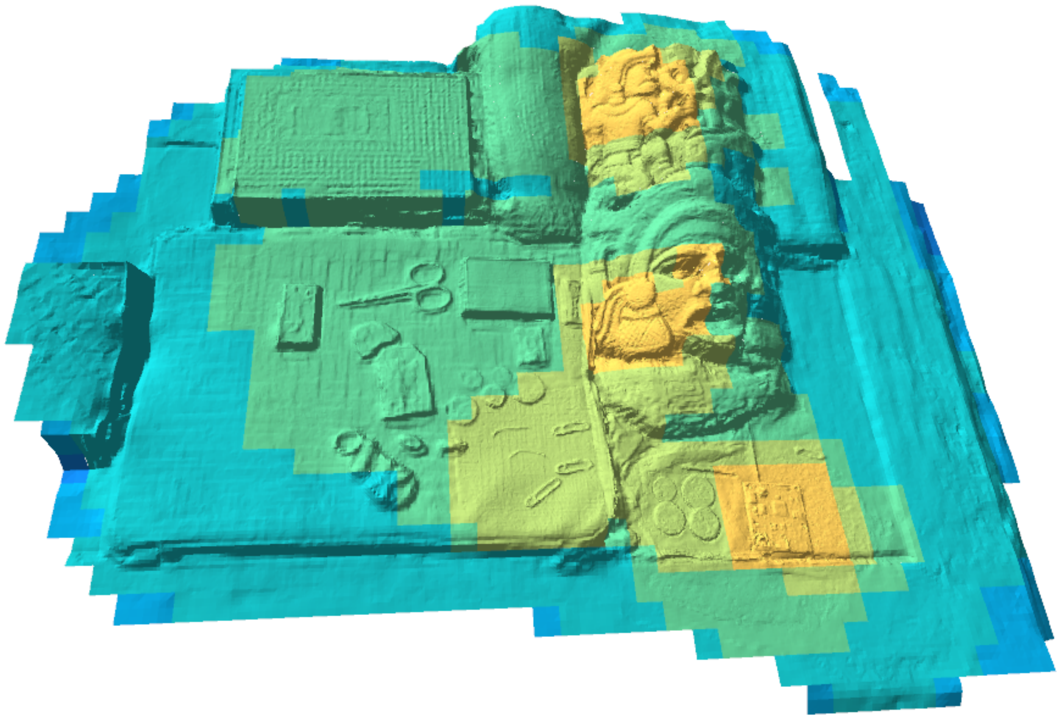


Figure 7.12: In our method we can provide a user with a direct feedback about the quality of reconstruction during the scanning process. Here, different colours represent the resolution that the surface element has been reconstructed to (yellow = high, blue = low). A user can then focus on the areas that need more detail.

An additional advantage of our approach is that it can provide a user with direct feedback about reconstruction quality. In Fig. 7.12 different colours indicate the reconstructed level of detail for every element of the scene. Yellow means that this part of the scene has been captured with a high level of detail, whereas blue represents only the coarsest geometry. In this case, in order to store the model using our adaptive resolution representation, we only need 5.3% of the memory compared to using the full, high resolution mesh.

7.6.3 Comparison against Point-based Fusion

Fig. 7.13 presents additional results and compares our method with the model obtained using Point-based Fusion [Keller et al., 2013], a general real-time reconstruction framework that represents a scene using a set of unordered surfels. For this comparison in both cases we used ORB-SLAM for tracking and our depth estimation method. At the overall scale we obtain qualitatively good results using both approaches, but Point-based Fusion tends to over-smooth the model and cannot

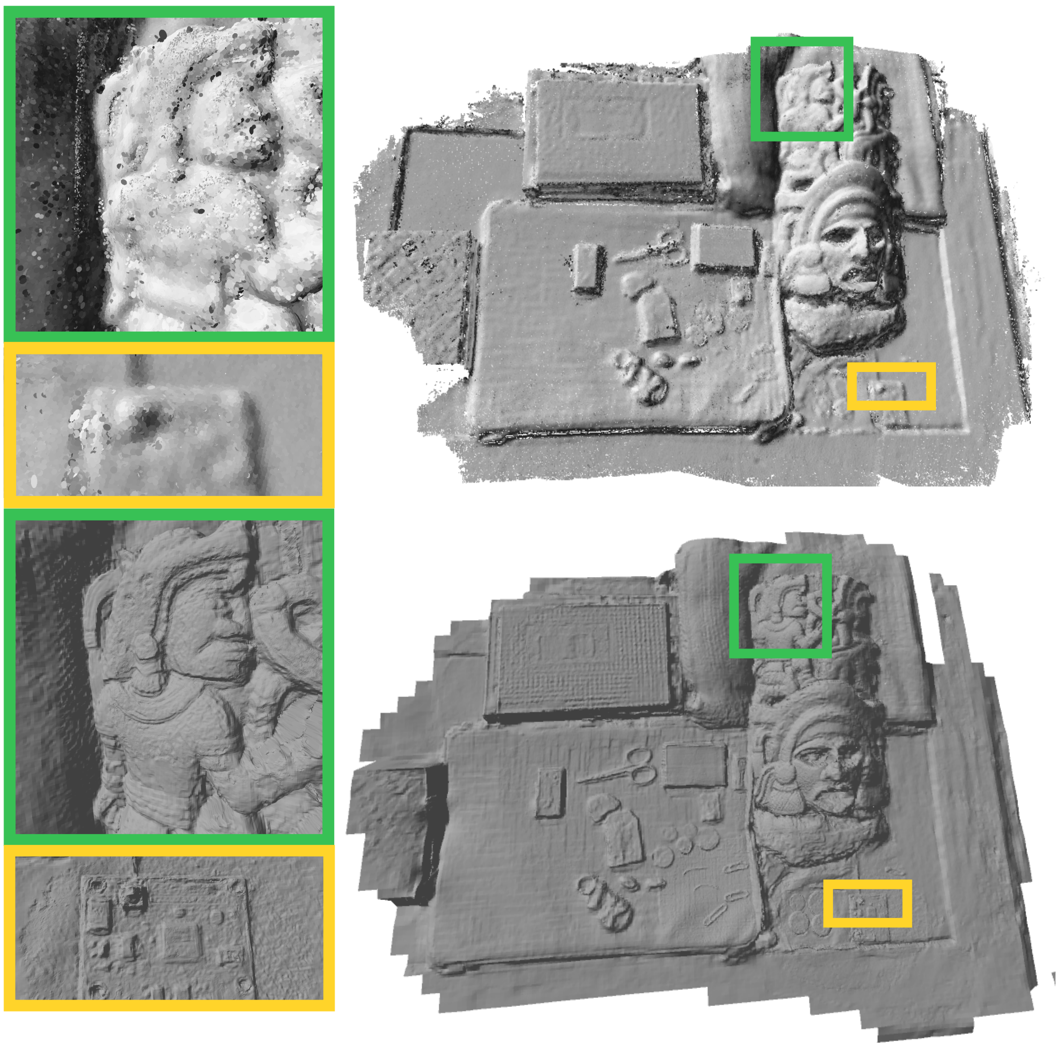


Figure 7.13: Comparison of our proposed method (bottom row) against Point-based Fusion (top row).

correctly handle the significant changes in scale. On the other hand, our method is capable of capturing the overall structure of the scene (although it struggles with sharp vertical edges and cannot handle overhangs properly) while being able to reconstruct tiny details including elements on a circuit board. As shown in Table 7.2 the processing times of the two algorithms are comparable.

7.6.4 Multi-scale Height Map for a Mobile Robot

The multi-scale surface representation and fusion can immediately be used for an environment mapping and representation for a mobile robot, and Fig. 7.14 shows

	Multi-scale Fusion (proposed)	Point-based Fusion
Processing time	8.9 ms (111 fps)	11.1 ms (90 fps)

Table 7.2: Comparison of run times for our Multi-scale Fusion approach and the method based on the Point-based Fusion [Keller et al., 2013]. For both of the algorithms, we used exactly the same camera tracking and depth estimation methods, and only utilised different fusion backends. Both algorithms were run on an NVIDIA GPU GTX980.

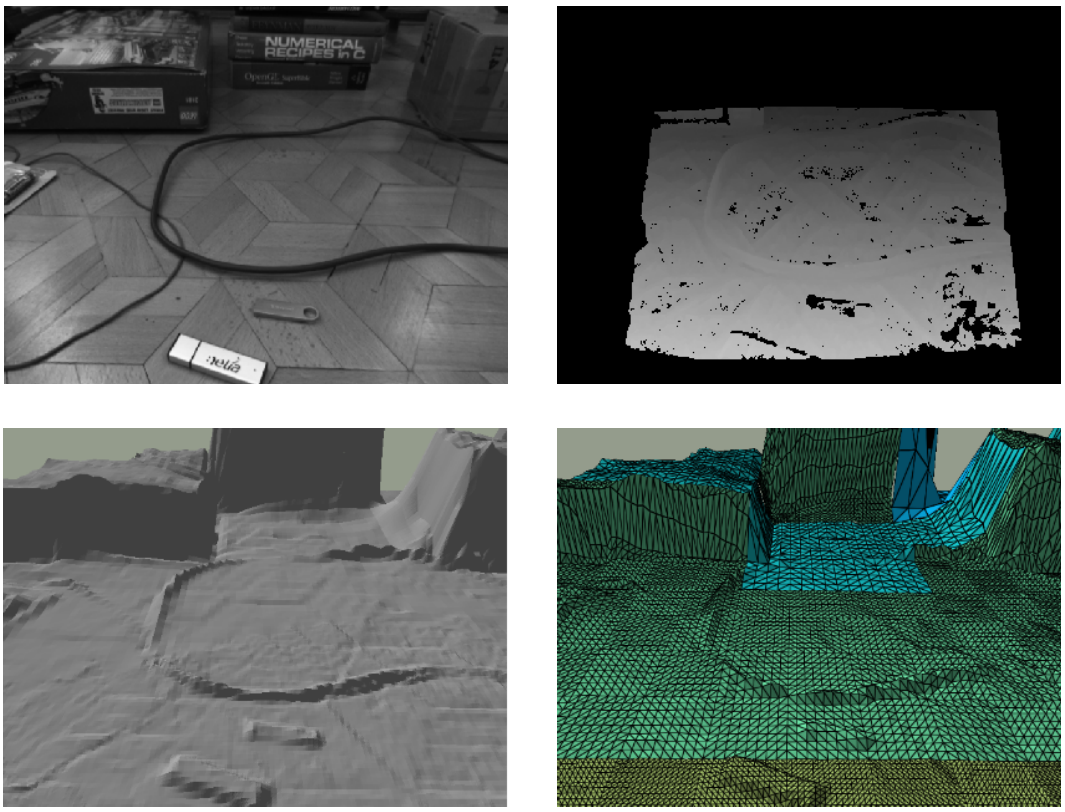


Figure 7.14: Multi-scale fusion is well suited to height mapping from an obliquely angled camera. Top row: a typical input image and depth map. Bottom row: reconstructed scene and the tessellation used for the current frame.

an example of results obtained in a typical set-up we considered in this thesis. As the camera is fixed on the robot, we cannot obtain close-up reconstructions, but the multi-resolution is particularly advantageous for oblique camera angles. It allows us to use high resolution directly in front of the robot and low resolution towards the horizon, and therefore achieve higher performance and better scalability compared to a fixed resolution mesh.

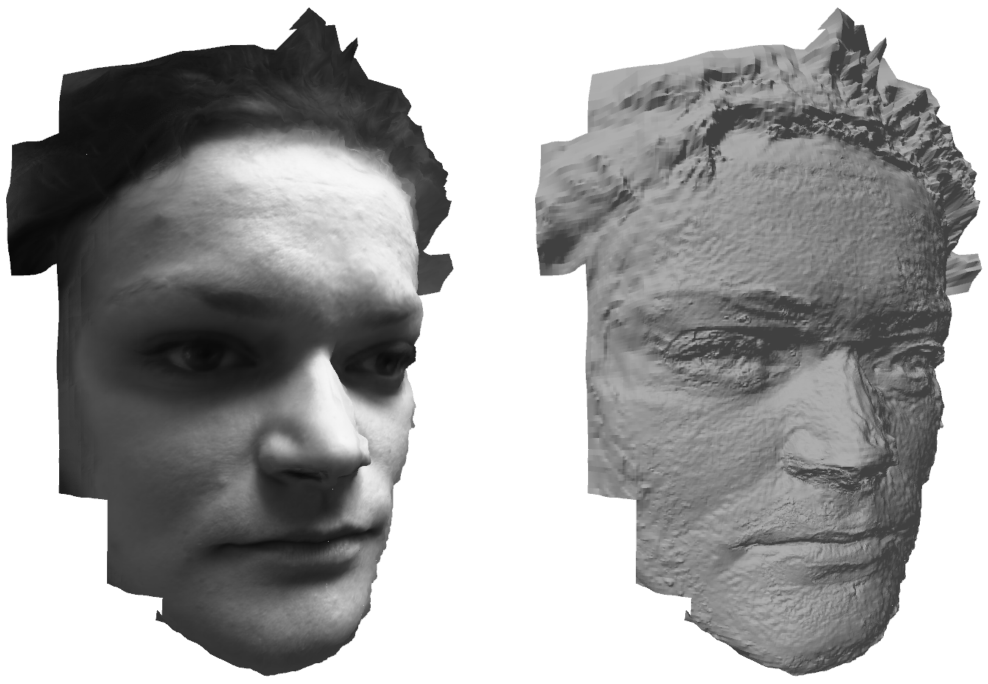


Figure 7.15: Face reconstruction.

7.6.5 Additional Experiments

By using a predefined mesh for depth map fusion our method can be extended to enable easy and robust reconstruction of some common and more general 3D structures (*e.g.* face scanning). In particular, in 3D printing applications using a predefined mesh can be beneficial, as it helps to handle missing data and guarantees that the final model does not contain holes. In fact, a mesh created with our method is directly printable without any additional processing. Fig. 7.15 shows an example of a real face reconstructed using our algorithm.

7.7 Conclusions

In this chapter we have shown how the concept of dynamic level of detail can be used to extend our incremental surface reconstruction framework to handle scenes with multiple scales. Using predictive rendering and on the fly geometry tessellation we adaptively select the best resolution of the model and fuse measurements in an efficient multi-scale mesh representation. We explicitly harness the unique characteristics of a monocular, multi-view stereo pipeline to estimate depth maps at different scales and demonstrate the ability of our system to reconstruct details down to fractions of a millimetre. Our system runs in real-time and achieves high quality reconstructions comparable with much slower off-line methods. In particular, the real-time feedback that our approach can provide to a user can ease the reconstruction process and improve results in many cases.

An obvious limitation of our approach lies in the use of a height map to model a surface. However, we demonstrated that this representation is not only restricted to use in mobile robotics, but can also be applied to other applications, for example face scanning. In the future it might be possible to extend the framework to more general 3D settings and developing a flexible multi-scale fusion method.

Our level of detail technique is relatively simple to implement and very efficient thanks to hardware acceleration in modern GPUs, as well as regular grid structure. However, a fixed topology mesh and the generated tessellation patterns are rarely optimal and do not correspond well to the physical features and geometries of a reconstructed scene. In fact, in our methods we often keep adding vertices into the model, even though they are not required. An interesting improvement could allow an adaptive mesh refinement based on data and quality of reconstruction that also takes into account the complexity of the geometry. Ideally, we would like to represent flat but textured regions with a small number of large coarse triangles but a high resolution texture.

The presented method can be used *as is* for height mapping using the mobile robot considered in this thesis, but we can also imagine several modifications and improvements for this particular scenario. Among other things one could design different metrics for selecting a level of detail, for example, simply based on the distance from the robot, *i.e.* use high resolution in the direct vicinity of the robot (*e.g.* first 20 cm), medium resolution at an intermediate distance (*e.g.* 20 to 50 cm), and for everything beyond that, use a default, coarse resolution. Such a robot-centric, and distance-dependent map could move together with the robot, and would allow us to maintain a coarse map at a large scale, and at the same time provide very detailed local perception.

Chapter 8

Conclusions

8.1 Contributions

We have demonstrated a parallelisable vision approach to mobile robot perception that offers a promising route to truly usable real-time monocular dense SLAM system. Starting from a standard 3D reconstruction pipeline that separates camera tracking, depth estimation and environment mapping we presented a complete system that allows a mobile robot to estimate its ego-motion and build in real-time a detailed, dense map of its environment from a single, monocular camera. By taking a more constrained, application-driven approach and using domain knowledge, our algorithms achieve simplicity, efficiency and robustness, and produce output that is directly suitable for local navigation and obstacle detection.

When designing our system, we put great emphasis on algorithms that can be efficiently implemented on parallel architectures. This quite naturally led to dense computer vision methods that aim at *directly* using information collected from all pixels in an image, and can also produce *dense* reconstructions immediately usable for a mobile robot. In our implementations we did not restrict ourselves to general purpose parallel computing frameworks such as CUDA, but also utilised the elements of a modern rendering pipeline, including the tessellation shaders of OpenGL, to aid the 3D reconstruction process and benefit from hardware acceleration. Our work demonstrated that many well-established concepts and techniques from computer graphics (*e.g.* the concept of dynamic level of detail) can be used to improve performance and address many limitations of existing computer vision algorithms. The presented parallel algorithms also offer excellent scalability in terms of the resolution of input images and environment representation. Our methods operate very comfortably in real-time on a modern PC with GPU, but can also be ported to CPU and give very good prospects for efficient use on embedded parallel processing units in the near future.

As we have already stated, we have proposed an architecture for a dense perception system for a mobile robot based only on a standard, monocular camera. The system is designed to be self-contained, and we explicitly aimed at equipping the robot with all the necessary capabilities for local perception and motion estimation using only a single camera. In particular, in Chapter 3 we showed how a direct image alignment technique derived from the Lucas-Kanade method, together with certain assumptions about the robot motion and its environment, can be used for accurate ego-motion estimation and to augment or even replace standard wheel odometry. The approach from Chapter 4 can greatly facilitate deployment of our computer vision system by providing robust and infrastructure-free extrinsics auto-calibration. In order to calculate the camera orientation on the robot, we extended the parameterisation of a dense image alignment framework to include the roll and pitch angles of the camera and estimated them jointly with inter-frame planar robot motion. The second part of the calibration procedure is more general and can be used to estimate the extrinsics parameters of any sensor capable of estimation of its incremental motion, *e.g.* a laser range finder. The proposed methods, in particular the auto-calibration method based on nonholonomic motion assumptions, can be used as light-weight recalibration and fault detection mechanisms for a wide range of wheeled mobile robots.

Visual odometry is an integral part of a monocular SLAM system for a mobile robot, but precise camera tracking is also a prerequisite for a monocular depth map estimation. In the preliminary chapter, Chapter 2, we presented a multi-view stereo approach that allows us to estimate dense depth maps in real-time, in various settings and at different scales. By utilising the concept of cost volume filtering, we could avoid the computational complexity imposed by variational and optimisation approaches, while still producing useful and robust results.

Camera poses, input frames and depth maps are the inputs to our mapping module. Our core representation for the robot's environment is a simple height map, typically modelled using a triangular mesh, and throughout Chapters 5 to 7 we present different extensions and algorithms for height map estimation. In general, height mapping is formulated as a recursive, least squares 2.5D surface reconstruction problem from stream of depth maps, and every time a new frame is fused we are solving an optimisation problem. Thanks to a carefully chosen fixed topology of the mesh, and a very efficient, parallel iterative solver based on the Gauss-Seidel algorithm (presented in Chapter 5), we can perform real-time estimation even on large scale problems.

Using a generative model of the image and depth map formation process and efficient differentiable rendering, in Chapter 6 we derived a novel, rigorous incremental probabilistic approach to depth map and image fusion. In this method reconstruction is formulated as a recursive nonlinear optimisation problem, where as each new frame arrives we compare it with a generative rendering of our current surface estimate and make an appropriate Bayesian update. We demonstrated that this approach can be used in real-time to create a dense surface model immediately usable for online robot path planning.

A particular highlight of this thesis is the multi-scale surface reconstruction method from Chapter 7, where we use predictive rendering and the concept of level of detail to dynamically select the best resolution of the model, and subsequently fuse the depth and colour measurements into an adaptively tessellated triangular mesh. Consistent, probabilistically-sound and principled fusion is enabled by using an implicit, highly scalable, multi-resolution surface representation based on a Laplacian pyramid. We demonstrate that this method is capable of obtaining high quality, close-up reconstruction at the level of sub-millimetres, and directly harnesses the unique capability of a monocular multi-view stereo pipeline to estimate the geometry of an observed scene at significantly different scales. By directly utilising tessellation shaders, the method is memory and computationally efficient, and can be easily and entirely implemented on a GPU.

8.2 Discussion and Future Research

The approaches presented in this thesis offer great promise for simple monocular sensing which can efficiently and robustly capture comprehensive information about free space and obstacles. In particular, the ability to accurately map even small objects and obstacles, which are invisible to many other sensors, could be beneficial to small robots like robotic vacuum cleaners. There is great room for potential further development and with additional engineering effort, system robustness and run time can be significantly improved. An obvious extension of the method that could help to reduce motion drift is to integrate camera tracking far more closely into the overall reconstruction framework and perform motion estimation directly with respect to the reconstructed model using dense image alignment. It should be also relatively straightforward to extend the auto-calibration approach and by incorporating it into a generative approach, jointly estimate camera intrinsics and other effects such as lens distortions and vignetting. As recently shown in [Engel et al., 2016], by appropriate modelling of the image formation process and by including parameters that affect it (*e.g.* camera gain) into estimation, one can greatly increase the accuracy of direct approaches.

Our method can at the moment provide a robot with only accurate local perception. In the future we would be interested to extending the method and building a full SLAM system that can provide consistent large scale reconstructions by incorporating place recognition and support for loop closures. There are many possibilities in which this can be achieved; for example, one could represent the environment as a collection of small, local height maps that can be aligned or deformed into a globally consistent model. Our probabilistic formulation should enable relatively straightforward fusion of multiple sub-maps into a consistent representation.

We demonstrated that a simple height map model can be a very useful dense representation of a robot's environment. In particular, when modelled as a triangular mesh, height maps are very flexible and allow for many extensions. Our study of surface reconstruction also emphasises the importance of choosing the right parameterisation of the problem. The three different data association strategies we considered when fusing a depth map into a height field resulted in dramatically different complexity of the associated optimisation problem: from a simple averaging when one models a height map as a collection of independent vertices, to an iterative nonlinear optimisation in the case of a generative and differentiable rendering approach.

Our multi-resolution approach to surface reconstruction is a good example of how existing techniques from computer graphics can be very beneficial and adopted to efficiently solve many computer vision problems. In particular, we are interested in extending the dynamic level of detail concept and multi-resolution fusion into more general 3D representations. We believe that reconstruction directly into a triangular mesh can be a viable option in many circumstances, and is not restricted to height map models. Obviously, working directly with unconstrained and general triangular meshes brings with it many non-trivial challenges and complications but also offers interesting avenues for future research.

However, even in its current form, there is great scope for extension of multi-resolution fusion. The level of detail is at the moment only determined based on the expected size of a projected triangle, and the fixed topology and tessellation patterns are rarely optimal, and do not correspond to natural features in the reconstructed scene. Furthermore, our approach often adds geometry where it is not really required. To address this limitation, one could implement certain principles of adaptive mesh refinement techniques. In adaptive mesh refinement one typically starts by solving a problem on a base coarse mesh, and recursively adding more resolution locally, only where it is really needed. The regions that require refinement are identified by monitoring some parameters that characterise the solution. In our case we could develop measures of "goodness-of-fit" and, based on these, adaptively refine and split triangles in the mesh.

In this thesis we followed a passive, monocular approach to robot perception, and demonstrated that indeed we can develop a scalable system that can achieve both accuracy and efficiency. Of course the question remains of whether this is the *right* and the *best* approach. In the particular application area of low-cost robotics this is clearly a promising and viable option, but the exact answer to this question depends on many factors such as performance and computational requirements, as well as cost and overall power consumption. Obviously, passive SLAM approaches have many advantages as demonstrated in this work and they offer interesting scalability and optimisation potential, especially with respect to camera frame-rate and image resolution; however there are also other factors that need to be considered. As small, dual camera modules are becoming increasingly popular in smartphones, adding an additional camera to a robotic platform can potentially increase a system's capabilities and robustness without significantly affecting the cost and computational requirements. Equipping a robot with an active depth sensor, although it will increase the cost and power consumption on the imaging side, could still be beneficial when considering the overall energy budget of sensing *and* computing, as with an active depth sensor one does not have to allocate computational resources to multi-view stereo estimation, and with depth maps camera tracking becomes much easier. A great strength of the approaches presented in this thesis is that, with minimal or no modification, they can be used in conjunction with input from stereo and depth cameras. A particularly promising opportunity for future work is to explore the possibility of combining passive, monocular approaches with other sensing modalities. For example, by running temporal multi-view stereo one can enhance and augment the output from a standard active or passive stereo system. This would allow us to eliminate many drawbacks of passive and active stereo sensors as well as depth cameras, for example restrictions on sensing range or resolution, but at the same time would address certain limitations of monocular, passive vision such as the inability to estimate depth maps in dynamic scenes or to work in darkness. These sensing approaches are complementary and when fused properly can enable very robust and accurate vision systems. Furthermore, multi-sensor fusion can also result in very power efficient solutions, where one adaptively selects the best sensing modality to manage the computational and energy budget for current environmental settings and requirements.

This thesis would not be complete without at least a short reference to deep learning. In our work we have followed a geometric approach to robot perception, where the main objective was to accurately reconstruct a robot's environment, and subsequently use it in robot navigation and as a basis for performing some high-level tasks. With an ever increasing number of methods that try to use deep and reinforcement learning to design an end-to-end system, which goes straight from raw image observations into actions, one can challenge the importance of traditional SLAM methods. Clearly, we anticipate that a wide range of machine learning approaches will play a key role in providing full autonomy to robotic systems, but at the same time we expect that many well-known and established techniques from SLAM will continue to be relevant and, in fact, will be essential for providing consistent and efficient behaviours. We hope that deep learning will be useful in addressing the limitations of current visual SLAM systems and will truly increase their robustness.

Bibliography

Adams, H., Singh, S., and Strelow, D. (2002). An empirical comparison of methods for image-based motion estimation. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*.

Agarwal, S., Snavely, N., Seitz, S. M., and Szeliski, R. (2010). Bundle Adjustment in the Large. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Agarwal, S., Snavely, N., Simon, I., Seitz, S. M., and Szeliski, R. (2009). Building Rome in a Day. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 10

Antonelli, G., Caccavale, F., Grossi, F., and Marino, A. (2010). Simultaneous calibration of odometry and camera for a differential drive mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Ashraf, A. B., Lucey, S., and Chen, T. (2010). Fast image alignment in the Fourier domain. In *CVPR*.

Baker, S. and Matthews, I. (2004). Lucas-Kanade 20 years on: A unifying framework: Part 1. *International Journal of Computer Vision (IJCV)*, 56(3):221–255.

Bao, S. Y., Lin, Y., and Savarese, S. (2013). Dense Object Reconstruction with Semantic Priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 13

Bares, J., Hebert, M., Kanade, T., Krotkov, E., Mitchell, T., Simmons, R., and Whittaker, W. R. L. (1989). Ambler: An Autonomous Rover for Planetary Exploration. *IEEE Computer*, 22(6):18–26. 9, 93

Barnes, C., Shechtman, E., Finkelstein, A., and Goldman, D. (2009). PatchMatch: a randomized correspondence algorithm for structural image editing. In *Proceedings of SIGGRAPH*.

Benhimane, S. and Malis, E. (2004). Real-Time Image-Based Tracking of planes using Efficient Second-order Minimization. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*.

Bhatti, S., Desmaison, A., Miksik, O., Nardelli, N., Siddharth, N., and Torr, P. H. S. (2016). Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint 1612.00380*, abs/1612.00380. 13

Bibby, C. and Reid, I. (2008). Robust Real-Time Visual Tracking using Pixel-Wise Posteriors. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Bleyer, M., Rhemann, C., and Rother, C. (2011a). PatchMatch Stereo — Stereo Matching with Slanted Support Windows. In *Proceedings of the British Machine Vision Conference (BMVC)*.

Bleyer, M., Rother, C., Kohli, P., Scharstein, D., and Sinha, S. (2011b). Object Stereo - Joint Stereo Matching and Object Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3081–3088. 13

Brookshire, J. and Teller, S. (2011). Automatic Calibration of Multiple Coplanar Sensors. In *Proceedings of Robotics: Science and Systems (RSS)*.

Burt, P. and Adelson, E. (1983). The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*, 31(4):532–540.

Bylow, E., Sturm, J., Kerl, C., Kahl, F., and Cremers, D. (2013). Real-time camera tracking and 3d reconstruction using signed distance functions. In *Proceedings of Robotics: Science and Systems (RSS)*. 12

Campbell, J., Sukthankar, R., Nourbakhsh, I., and Pahwa, A. (2005). A Robust Visual Odometry and Precipice Detection System Using Consumergrade Monocular

Vision. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Censi, A., Marchionni, L., and Oriolo, G. (2008). Simultaneous Maximum-Likelihood Calibration of Odometry and Sensor Parameters. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Cernuschi-Frias, B., Belhumeur, P. N., and Cooper, D. B. (1986). 3-D object position estimation and recognition based on parameterized surfaces and multiple views. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Cernuschi-Frias, B., Cooper, D. B., Hung, Y.-P., and Belhumeur, P. N. (1989). Toward a model-based Bayesian theory for estimating and recognizing parameterized 3-D objects using two or more images taken from different positions. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 11(10):1028–1052.

Cheeseman, P., Kanefsky, B., Kraft, R., Stutz, J., and Hanson, R. (1996). Super-Resolved Surface Reconstruction from Multiple Images. In *Maximum Entropy and Bayesian Methods*, volume 62, pages 293–308. Springer Netherlands.

Chen, J., Bautembach, D., and Izadi, S. (2013). Scalable real-time volumetric surface reconstruction. In *Proceedings of SIGGRAPH*.

Comport, A. I., Malis, E., and Rives, P. (2007). Accurate Quadri-focal Tracking for Robust 3D Visual Odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH*.

Dahlkamp, H., Kaehler, A., Stavens, D., Thrun, S., and Bradski, G. (2006). Self-supervised monocular road detection in desert terrain. In *Proceedings of Robotics: Science and Systems (RSS)*. 10, 47, 90, 91

Davison, A. J. (2003). Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 7, 11, 36 de La Gorce, M., Paragios, N., and Fleet, D. J. (2008). Model-based hand tracking with texture, shading and self-occlusions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Delaunoy, A. and Prados, E. (2011). Gradient flows for optimizing triangular mesh-based surfaces: Applications to 3D reconstruction problems dealing with visibility. *International Journal of Computer Vision (IJCV)*, 95(2):100–123.

Delaunoy, A., Prados, E., Gargallo, P., Pons, J.-P., and Sturm, P. (2008). Minimizing the Multi-view Stereo Reprojection Error for Triangular Surface Meshes. In *Proceedings of the British Machine Vision Conference (BMVC)*.

Duchaineau, M., Wolinsky, M., Sigeti, D., Miller, M., Aldrich, C., and Mineev-Weinstein, M. (1997). ROAMing Terrain: Real-time Optimally Adapting Meshes. In *IEEE Conference on Visualization*.

Durrant-Whyte, H. F. (1994). Where am I? A Tutorial on Mobile Vehicle Localization. *Industrial Robot*, 21(2):11–16. 9

Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., and Stuetzle, W. (1995). Multiresolution Analysis of Arbitrary Meshes. In *Proceedings of SIGGRAPH*.

Eigen, D., Puhrsch, C., and Fergus, R. (2014). Depth Map Prediction from a Single Image using a Multi-Scale Deep Network. In *Neural Information Processing Systems (NIPS)*. 10, 24

Elfes, A. (1987). Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3(3):249–265. 9, 89

Engel, J., Koltun, V., and Cremers, D. (2016). Direct Sparse Odometry. *arXiv preprint 1607.02565*. 7, 174

Engel, J., Schoeps, T., and Cremers, D. (2014). LSD-SLAM: Large-scale direct monocular SLAM. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 12, 45, 91, 160

Engel, J., Sturm, J., and Cremers, D. (2013). Semi-dense visual odometry for a monocular camera. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 8, 12, 28, 29, 36

Fankhauser, P., Bloesch, M., Gehring, C., Hutter, M., and Siegwart, R. (2014). Robot-Centric Elevation Mapping with Uncertainty Estimates. In *Proceedings of the International Conference on Climbing and Walking Robots (CLAWAR)*.

Faugeras, O. D. and Lustman, F. (1988). Motion and Structure From Motion in a Piecewise Planar Environment. *International Journal of Pattern Recognition in Artificial Intelligence*, 2(3):485–508.

Fitzgibbon, A. W., Cross, G., and Zisserman, A. (1998). Automatic 3D Model Construction for Turn-Table Sequences. In *Proceedings of the Workshop on Structure from Multiple Images of Large Scale Environments (SMILE), in conjunction with ECCV*. 10

Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). SVO: Fast Semi-Direct Monocular Visual Odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 8, 11, 36, 160

Forster, C., Zhang, Z., Gassner, M., Werlberger, M., and Scaramuzza, D. (2016). SVO : Semi-Direct Visual Odometry for Monocular and Multi-Camera Systems. *IEEE Transactions on Robotics (T-RO)*. 7

Fossati, A. and Fua, P. (2008). Linking pose and motion. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Frahm, J.-M., Georgel, P. F., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y., Dunn, E., Clipp, B., and Lazebnik, S. (2010). Building Rome on a Cloudless Day. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 10

Fuhrmann, S. and Goesele, M. (2011). Fusion of depth maps with multiple scales. In *SIGGRAPH Asia*.

Fuhrmann, S. and Goesele, M. (2014). Floating Scale Surface Reconstruction. In *Proceedings of SIGGRAPH*.

Fuhrmann, S., Langguth, F., and Goesele, M. (2014). MVE — A Multi-View Reconstruction Environment. In *EUROGRAPHICS Workshops on Graphics and Cultural Heritage*. 11, 20, 161, 162

Furgale, P., Barfoot, T. D., and Sibley, G. (2012). Continuous-time batch estimation using temporal basis functions. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Furukawa, Y. and Ponce, J. (2007). Accurate, Dense, and Robust Multi-View Stereopsis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 11

Gallup, D., Frahm, J.-M., Pollefeys, M., and Zuerich, E. (2010a). A Heightmap Model for Efficient 3D Reconstruction from Street-Level Video. In *Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*.

Gallup, D., Pollefeys, M., and Frahm, J. M. (2010b). 3D reconstruction using an n-layer heightmap. In *Proceedings of the DAGM Symposium on Pattern Recognition*.

Gargallo, P., Prados, E., and Sturm, P. (2007). Minimizing the reprojection error in surface reconstruction from images. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Goesele, M., Snavely, N., Curless, B., Hoppe, H., and Seitz, S. M. (2007). Multi-View Stereo for Community Photo Collections. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 11

Graber, G., Pock, T., and Bischof, H. (2011). Online 3D reconstruction using convex optimization. In *Workshop on Live Dense Reconstruction from Moving Cameras at ICCV*. 12

Guskov, I., Sweldens, W., and Schröder, P. (1999). Multiresolution Signal Processing for Meshes. In *Proceedings of SIGGRAPH*.

Hadsell, R., Sermanet, P., Ben, J., Erkan, A., Scoffier, M., Kavukcuoglu, K., Muller, U., and LeCun, Y. (2009). Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics (JFR)*, 26(2):120–144. 13, 47, 91

Handa, A. (2013). *Analysing High Frame-Rate Camera Tracking*. PhD thesis, Imperial College London.

Handa, A., Newcombe, R. A., Angeli, A., and Davison, A. J. (2012). Real-Time Camera Tracking: When is High Frame-Rate Best? In *Proceedings of the European Conference on Computer Vision (ECCV)*. 5

Häne, C., Zach, C., Lim, J., Ranganathan, A., and Pollefeys, M. (2011). Stereo depth map fusion for robot navigation. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*.

Harris, M. (2008). Optimizing Parallel Reduction in CUDA. `http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf`.

Hartley, R. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition.

He, K., Sun, J., and Tang, X. (2010). Guided Image Filtering. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Heise, P., Klose, S., Jensen, B., and Knoll, A. (2013). PM-Huber: PatchMatch with Huber Regularization for Stereo Matching. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2010). RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*. 12

Herbert, M., Caillas, C., Krotkov, E., Kweon, I., and Kanade, T. (1989). Terrain mapping for a roving planetary explorer. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Hernández, C. and Schmitt, F. (2004). Silhouette and stereo fusion for 3D object modeling. *Computer Vision and Image Understanding (CVIU)*, 96(3):367–392. 10

Hirschmüller, H. (2005). Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hirschmüller, H. (2007). Evaluation of Cost Functions for Stereo Matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hirschmüller, H. (2008). Stereo processing by semiglobal matching and mutual information. *PAMI*, 30(2):328–341.

Hoiem, D., Efros, A. A., and Hebert, M. (2005). Geometric context from a single image. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 10

Hoiem, D., Efros, A. A., and Hebert, M. (2008). Putting Objects in Perspective. *International Journal of Computer Vision (IJCV)*, 80(1):3–15. 10

Hoppe, H. (1996). Progressive Meshes. In *Proceedings of SIGGRAPH*.

Hoppe, H. (1997). View-Dependent Refinement of Progressive Meshes. In *Proceedings of SIGGRAPH*.

Hoppe, H. (1998). Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Conference on Visualization*.

Hung, Y.-P. and Cooper, D. B. (1990). Maximum a-posteriori probability 3-D surface reconstruction using multiple intensity images directly. In *Proceedings of SPIE*, volume 1260.

Hung, Y.-P., Cooper, D. B., and Cernuschi-Frias, B. (1988). Bayesian estimation of 3D surfaces from a sequence of images. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Hung, Y.-P., Cooper, D. B., and Cernuschi-Frias, B. (1991). Asymptotic Bayesian surface estimation using an image sequence. *IJCV*, 6(2):105–132.

Irani, M. and Anandan, P. (1999). All About Direct Methods. In *Proceedings of the International Workshop on Vision Algorithms, in association with ICCV*. 6

Jalobeanu, A. (2004). Bayesian Vision for Shape Recovery. In *International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*.

Jalobeanu, A., Kuehnel, F. O., and Stutz, J. C. (2004). Modeling Images of Natural 3D Surfaces: Overview and Potential Applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*.

Kähler, O., Prisacariu, V., Valentin, J., and Murray, D. (2016). Hierarchical Voxel Block Hashing for Efficient Integration of Depth Images. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Kahler, O., Prisacariu, V. A., Ren, C. Y., Sun, X., Torr, P. H. S., and Murray, D. W. (2015). Very High Frame Rate Volumetric Integration of Depth Images on Mobile Device. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 12, 92

Kazhdan, M. and Hoppe, H. (2013). Screened poisson surface reconstruction. *ACM Transactions on Graphics*. 11

Keller, M., Lefloch, D., Lambers, M., Izadi, S., Weyrich, T., and Kolb, A. (2013). Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In *Proc. of Joint 3DIM/3DPVT Conference (3DV)*. 12, 92, 161, 166, 168

Kerl, C., Sturm, J., and Cremers, D. (2013). Dense visual SLAM for RGB-D cameras. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 8

Kitt, B., Rehder, J., Chambers, A., Schonbein, M., Lategahn, H., and Singh, S. (2011). Monocular Visual Odometry using a Planar Road Model to Solve Scale Ambiguity. In *Proceedings of the European Conference on Mobile Robotics (ECMR)*.

Klein, G. and Murray, D. W. (2007). Parallel Tracking and Mapping for Small AR Workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 7, 11, 36, 91

Klose, S., Heise, P., and Knoll, A. (2013). Efficient compositional approaches for real-time robust direct visual odometry from rgb-d data. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*.

Knorr, M., Niehsen, W., Member, S., and Stiller, C. (2013). Online extrinsic multi-camera calibration using ground plane induced homographies. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*.

Kolmogorov, V. and Zabih, R. (2001). Computing Visual Correspondence with Occlusions using Graph Cuts. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Konolige, K., Agrawal, M., Blas, M. R., Bolles, R. C., Gerkey, B., Solà, J., and Sundaresan, A. (2009). Mapping, navigation, and learning for off-road traversal. *Journal of Field Robotics (JFR)*, 26(1):88–113. 13, 91

Konolige, K., Agrawal, M., and Solà, J. (2007). Large Scale Visual Odometry for Rough Terrain. In *Proceedings of the International Symposium on Robotics Research (ISRR)*.

Krotkov, E., Hebert, M., and Simmons, R. (1995). Stereo Perception and Dead-Reckoning for a Prototype Lunar Rover. *Autonomous Robots*, 2(4):313–331. 9

Kümmerle, R., Grisetti, G., and Burgard, W. (2011a). Simultaneous Calibration, Localization, and Mapping. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011b). g^2o: A General Framework for Graph Optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Kundu, A., Li, Y., Daellert, F., Li, F., and Rehg, J. M. (2014). Joint Semantic Segmentation and 3D Reconstruction from Monocular Video. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Kutulakos, K. N. and Seitz, S. M. (2000). A Theory of Shape by Space Carving. *International Journal of Computer Vision (IJCV)*, 38(3):199–218. 10

Kweon, I.-S. and Kanade, T. (1992). High-resolution terrain map from multiple sensor data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):278–292.

Lacroix, S., Mallet, A., Bonnafous, D., Bauzil, G., Fleury, S., Herrb, M., and Chatila, R. (2002). Autonomous Rover Navigation on Unknown Terrains: Functions and Integration. *International Journal of Robotics Research (IJRR)*, 21(10-11):917–942.

Laumond, J.-P., editor (1998). *Robot Motion Planning and Control*, volume 229 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., Fletcher, L., Frazzoli, E., Huang, A., Karaman, S., Koch, O., Kuwata, Y., Moore, D., Olson, E., Peters, S., Teo, J., Truax, R., Walter, M., Barrett, D., Epstein, A., Maheloni, K., Moyer, K., Jones, T., Buckley, R., Antone, M., Galejs, R., Krishnamurthy, S., and Williams, J. (2008). A Perception-Driven Autonomous Urban Vehicle. *Journal of Field Robotics (JFR)*, 25(February):727–774. 9

Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P. (2014). Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, page 0278364914554813. 8, 91

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L. F., Faust, N., and Turner, G. (1996). Real-Time, Continuous Level of Detail Rendering of Height Fields. In *Proceedings of SIGGRAPH*.

Liu, S. and Cooper, D. B. (2010). Ray Markov Random Fields for image-based 3D modeling: Model and efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, S. and Cooper, D. B. (2011). A complete statistical inverse ray tracing approach to multi-view stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, S. and Cooper, D. B. (2014). Statistical Inverse Ray Tracing for Image-Based 3D Modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 36(10).

Loper, M. and Black, M. J. (2014). OpenDR: An Approximate Differentiable Renderer. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Losasso, F. and Hoppe, H. (2004). Geometry Clipmaps: Terrain Rendering using Nested Regular Grids. In *Proceedings of SIGGRAPH*.

Lounsbery, M., DeRose, T. D., and Warren, J. (1997). Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Transactions on Graphics*, 16(1):34–73.

Lovegrove, S., Patron-Perez, A., and Sibley, G. (2013). Spline Fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras. In *Proceedings of the British Machine Vision Conference (BMVC)*.

Lovegrove, S. J. (2011). *Parametric Dense Visual SLAM*. PhD thesis, Imperial College London. 11, 26, 41, 48

Lovegrove, S. J. and Davison, A. J. (2010). Real-Time Spherical Mosaicing using Whole Image Alignment. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Lovegrove, S. J., Davison, A. J., and Ibanez-Guzmán, J. (2011). Accurate Visual Odometry from a Rear Parking Camera. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*.

Lu, J., Shi, K., Min, D., Lin, L., and Do, M. N. (2012). Cross-based local multipoint filtering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lu, J., Yang, H., Min, D., and Do, M. N. (2013). PatchMatch Filter: Efficient Edge-Aware Filtering Meets Randomized Search for Fast Correspondence Field Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lucas, B. D. and Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 6, 37, 45

Lucey, S., Navarathna, R., Ashraf, A. B., and Sridharan, S. (2013). Fourier Lucas-Kanade algorithm. *PAMI*, 35(6):1383–1396.

Luitjens, J. (2014). Faster Parallel Reductions on Kepler. https://devblogs.nvidia.com/parallelforall/faster-parallel-reductions-kepler.

Lukierski, R., Leutenegger, S., and Davison, A. J. (2015). Rapid Free-Space Mapping From a Single Omnidirectional Camera. In *Proceedings of the European Conference on Mobile Robotics (ECMR)*.

Malis, E. (2004). Improving vision-based control using efficient second-order minimization techniques. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Martinelli, A., Scaramuzza, D., and Siegwart, R. (2006). Automatic self-calibration of a vision system during robot motion. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Matthies, L., Kanade, T., and Szeliski, R. (1989). Kalman filter-based algorithms for estimating depth from image sequences. *International Journal of Computer Vision (IJCV)*, 3(3):209–238.

Mei, C., Benhimane, S., Malis, E., and Rives, P. (2008). Efficient Homography-Based Tracking and 3-D Reconstruction for Single-Viewpoint Sensors. *IEEE Transactions on Robotics (T-RO)*, 24(6):1352–1364.

Mei, C., Sibley, G., Cummins, M., Newman, P., and Reid, I. (2009). A Constant Time Efficient Stereo SLAM System. In *Proceedings of the British Machine Vision Conference (BMVC)*.

Merrell, P., Akbarzadeh, A., Wang, L., Mordohai, P., Frahm, J.-M., Yang, R., Nistér, D., and Pollefeys, M. (2007). Real-Time Visibility-Based Fusion of Depth Maps. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 12

Miksch, M., Yang, B., and Zimmerman, K. (2010). Automatic Extrinsic Camera Self-Calibration Based on Homography and Epipolar Geometry. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., and Hadsell, R. (2016). Learning to navigate in complex environments. *arXiv preprint 1611.03673*. 13

Möller, T. and Trumbore, B. (1997). Fast , Minimum Storage Ray / Triangle Intersection. *Journal of Graphics Tools*, 2(1):21–28.

Moravec, H. P. (1977). Towards Automatic Visual Obstacle Avoidance. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, page 584. 8

Moravec, H. P. (1980). *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. PhD thesis, Stanford University. 8

Morris, R., Smelyanskiy, V., and Cheeseman, P. (2001). Matching Images to Models Camera Calibration for 3-D Surface Reconstruction. In *Proceedings of the International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*.

Morris, R. D., Cheeseman, P., Smelyanskiy, V. N., and Maluf, D. A. (1999). A Bayesian approach to high resolution 3D surface reconstruction from multiple images. In *Proceedings of the IEEE Signal Processing Workshop on Higher-Order Statistics*.

Moulon, P., Monasse, P., and Marlet, R. (2013). Global fusion of relative motions for robust, accurate and scalable structure from motion. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 11

Mur-Artal, R., Montiel, J. M. M., and Tardós, J. D. (2015). ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics (T-RO)*, 31(5):1147–1163. 7, 27, 36, 91, 160

Mur-Artal, R. and Tardós, J. D. (2014). ORB-SLAM: Tracking and Mapping Recognizable Features. In *Workshop on Multi View Geometry in Robotics (MVIGRO) - RSS 2014*. 11

Mur-Artal, R. and Tardós, J. D. (2015). Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM. In *Proceedings of Robotics: Science and Systems (RSS)*.

Nayar, S., Watanabe, M., and Noguchi, M. (1995). Real-Time Focus Range Sensor. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 10

Newcombe, R. A. (2012). *Dense Visual SLAM*. PhD thesis, Imperial College London. 2, 48

Newcombe, R. A. and Davison, A. J. (2010). Live Dense Reconstruction with a Single Moving Camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 12, 91

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011a). KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 12, 90, 92, 123

Newcombe, R. A., Lovegrove, S., and Davison, A. J. (2011b). DTAM: Dense Tracking and Mapping in Real-Time. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 2, 12, 20, 24, 26, 28, 30, 36, 92

Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. (2013). Real-time 3D Reconstruction at Scale using Voxel Hashing. In *Proceedings of SIGGRAPH*. 12, 92, 149

Nistér, D., Naroditsky, O., and Bergen, J. (2004). Visual Odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer, second edition.

Nourani-Vatani, N., Roberts, J., and Srinivasan, M. V. (2009). Practical visual odometry for car-like vehicles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Park, Y., Lepetit, V., and Woo, W. (2009). ESM-Blur: Handling & rendering blur in 3D tracking and augmentation. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*.

Parke, F. I. and Waters, K. (2008). *Computer facial animation*. CRC Press.

Peyre, G. and Cohen, L. D. (2006). Geodesic remeshing using front propagation. *International Journal of Computer Vision (IJCV)*, 69(1):145–156.

Pfaff, P., Triebel, R., and Burgard, W. (2007). An Efcient Extension to Elevation Maps for Outdoor Terrain Mapping and Loop Closing. *International Journal of Robotics Research (IJRR)*, 26(2):217–230.

Pfister, H., Zwicker, M., van Baar, J., and Gross, M. H. (2000). Surfels: surface elements as rendering primitives. In *Proceedings of SIGGRAPH*.

Pirchheim, C. and Reitmayr, G. (2011). Homography-based planar mapping and tracking for mobile phones. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*.

Pizzoli, M., Forster, C., and Scaramuzza, D. (2014). REMODE: Probabilistic, monocular dense reconstruction in real time. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 12

Poppe, R. (2007). Vision-based human motion analysis: An overview. *Computer Vision and Image Understanding (CVIU)*.

Pradeep, V., Rhemann, C., Izadi, S., Zach, C., Bleyer, M., and Bathiche, S. (2013). MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 83–88. 12, 20

Prisacariu, V. A., Kähler, O., Cheng, M. M., Ren, C. Y., Valentin, J., Torr, P., Reid, I., and Murray, D. (2014). A Framework for the Volumetric Integration of Depth Images. *arXiv*.

Procopio, M. J., Mulligan, J., and Grudic, G. (2009). Learning terrain segmentation with classifier ensembles for autonomous robot navigation in unstructured environments. *Journal of Field Robotics (JFR)*, 26(2):145–175.

Ranftl, R., Gehrig, S., Pock, T., and Bischof, H. (2012). Pushing the limits of stereo using variational stereo estimation. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*.

Rhemann, C., Hosni, A., Bleyer, M., Rother, C., and Gelautz, M. (2011). Fast cost-volume filtering for visual correspondence and beyond. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Saurer, O., Fraundorfer, F., and Pollefeys, M. (2012). Homography based visual odometry with known vertical direction and weak Manhattan world assumption. In *IEEE/IROS Workshop on Visual Control of Mobile Robots*.

Sawhney, H. S., Zisserman, A., Peleg, S., Szeliski, R., Irani, M., Torr, P., Knight, J., Malik, J., and Anandan, P. (1999). Discussion for Direct versus Features Session. In *Proceedings of the International Workshop on Vision Algorithms*. 7

Saxena, A., Chung, S., and Ng, A. (2005). Learning Depth from Single Monocular Images. In *Neural Information Processing Systems (NIPS)*. 10

Scaramuzza, D., Fraundorfer, F., Pollefeys, M., and Siegwart, R. (2009a). Absolute Scale in Structure from Motion from a Single Vehicle Mounted Camera by Exploiting Nonholonomic Constraints. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Scaramuzza, D., Fraundorfer, F., and Siegwart, R. (2009b). Real-time monocular visual odometry for on-road vehicles with 1-point RANSAC. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Scharstein, D. and Pal, C. (2007). Learning Conditional Random Fields for Stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Scharstein, D. and Szeliski, R. (2001). A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision (IJCV)*, 47:7–42.

Shapira, Y. (2008). *Matrix-based Multigrid: Theory and Applications.* Springer, second edition.

Shewchuk, J. (1994). An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, School of Computer Science, Carnegie Mellon University.

Silveira, G., Malis, E., and Rives, P. (2008). An Efficient Direct Approach to Visual SLAM. *IEEE Transactions on Robotics (T-RO)*, 24(5):969–979.

Smelyanskiy, V., Cheeseman, P., Maluf, D. A., and Morris, R. D. (2000). Bayesian super-resolved surface reconstruction from images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Smelyanskiy, V., Morris, R., Kuehnel, F., Maluf, D. A., and Cheeseman, P. (2002). Dramatic Improvements to Feature Based Stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Smelyanskiy, V. N., Morris, R. D., Maluf, D. A., and Cheeseman, P. (2001). (Almost) Featureless Stereo Calibration and Dense 3D Reconstruction Using Whole Image Operations. Technical report, NASA.

Snavely, N., Seitz, S. M., and Szeliski, R. (2006). Photo tourism: Exploring photo collections in 3D. In *Proceedings of SIGGRAPH*. 10

Steinbrücker, F., Kerl, C., Sturm, J., and Cremers, D. (2013). Large-scale multi-resolution surface reconstruction from RGB-D sequences. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Steinbrücker, F., Sturm, J., and Cremers, D. (2011). Real-Time Visual Odometry from Dense RGB-D Images. In *Workshop on Live Dense Reconstruction from Moving Cameras at ICCV*.

Stückler, J. and Behnke, S. (2014). Multi-resolution surfel maps for efficient dense 3d modeling and tracking. *Journal of Visual Communication and Image Representation*, 25(1):137–147.

Stuehmer, J., Gumhold, S., and Cremers, D. (2010). Real-Time Dense Geometry from a Handheld Camera. In *Proceedings of the DAGM Symposium on Pattern Recognition*. 12

Stutz, J. (2005). Experience With Bayesian Image Based Surface Modeling. Technical report, NASA.

Sun, W. and Yuan, Y.-X. (2006). *Optimization Theory and Methods: Nonlinear Programming*. Springer.

Szeliski, R. (1990). Bayesian modeling of uncertainty in low-level vision. *International Journal of Computer Vision (IJCV)*, 5(3):271–301.

Thrun, S. (2002). Robotic Mapping: A Survey. In Lakemeyer, G. and Nebel, B., editors, *Exploring Artificial Intelligence in the New Millenium*, pages 1–35. Morgan Kauffman.

Thrun, S., Liu, Y., Koller, D., Ng, A. Y., Ghahramani, Z., and Durrant-Whyte, H. (2004). Simultaneous Localization and Mapping with Sparse Extended Information Filters. *International Journal of Robotics Research (IJRR)*, 23(7-8):693–716.

Thrun, S., Montemerlo, M., and Aron, A. (2006a). Probabilistic Terrain Analysis For High-Speed Desert Driving. In *Proceedings of Robotics: Science and Systems (RSS)*.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., and Mahoney, P. (2006b). Stanley : The Robot that Won the DARPA Grand Challenge. *Journal of Field Robotics (JFR)*, 23(April):661–692. 9

Tran, S. and Davis, L. (2006). 3D Surface Reconstruction Using Graph Cuts with Surface Constraints. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Triebel, R., Pfaff, P., and Burgard, W. (2006). Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*.

Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (1999). Bundle Adjustment — A Modern Synthesis. In *Proceedings of the International Workshop on Vision Algorithms, in association with ICCV*. 7

Trottenberg, U., Oosterlee, C., and Schuller, A. (2001). *Multigrid*. Academic Press.

Ummenhofer, B. and Brox, T. (2015). Global, Dense Multiscale Reconstruction for a Billion Points. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Underwood, J. P., Hill, A., Peynot, T., and Scheding, S. J. (2010). Error modeling and calibration of exteroceptive sensors for accurate mapping applications. *Journal of Field Robotics (JFR)*, 27(1):2–20.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M. N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T. M., Kolski, S., Kelly, A., Likhachev, M., McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y.-W., Singh, S., Snider, J., Stentz, A., Whittaker, W. ., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., and Ferguson, D. (2008). Autonomous Driving in Urban Environments : Boss and the Urban Challenge. *Journal of Field Robotics (JFR)*, 25(February):425–466. 9

Žbontar, J. and LeCun, Y. (2014). Computing the Stereo Matching Cost with a Convolutional Neural Network. *arXiv preprint arXiv:1409.4326*.

Weise, T., Wismer, T., Leibe, B., and Van Gool, L. (2009). In-hand scanning with online loop closure. In *Proceedings of the International Conference on Computer Vision Workshops (ICCV Workshops)*.

Wenzel, K., Rothermel, M., Fritsch, D., and Haala, N. (2013). Image acquisition and model selection for multi-view stereo. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.* 11

Whelan, T., Leutenegger, S., Salas-Moreno, R. F., Glocker, B., and Davison, A. J. (2015). ElasticFusion: Dense SLAM without a pose graph. In *Proceedings of Robotics: Science and Systems (RSS)*. 8, 12, 92, 139, 140, 161

Whelan, T., McDonald, J. B., Kaess, M., Fallon, M., Johannsson, H., and Leonard, J. J. (2012). Kintinuous: Spatially Extended KinectFusion. In *Workshop on RGB-D: Advanced Reasoning with Depth Cameras, in conjunction with Robotics: Science and Systems*.

Woodford, O. J., Torr, P. H. S., Reid, I. D., and Fitzgibbon, a. W. (2008). Global stereo reconstruction under second order smoothness priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wu, C. (2013). Towards linear-time incremental structure from motion. In *Proceedings of the International Conference on 3D Vision (3DV)*. 10

Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C., and Burgard, W. (2010). OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems. In *Proceedings of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*.

Yamaguchi, K., Mcallester, D., and Urtasun, R. (2014). Efficient Joint Segmentation, Occlusion Labeling, Stereo and Flow Estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Yoon, K.-J. and Kweon, I.-S. (2005). Locally Adaptive Support-Weight Approach for Visual Correspondence Search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yoon, K.-J. and Kweon, I.-S. (2006). Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

Zabih, R. and Woodfill, J. (1994). Non-parametric Local Transforms for Computing Visual Correspondence. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Zhang, K., Lu, J., and Lafruit, G. (2009). Cross-Based Local Stereo Matching Using Orthogonal Integral Images. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):1073–1079.

Zhang, K., Lu, J., Yang, Q., Lafruit, G., Lauwereins, R., and Van Gool, L. (2011). Real-time and accurate stereo: A scalable approach with bitwise fast voting on CUDA. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(7):867–878.

Zhang, R., Tsai, P.-S., Cryer, J., and Shah, M. (1999). Shape-from-shading: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 21(8):690–706. 10

Zhang, Z. (1999). Flexible camera calibration by viewing a plane from unknown orientations. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Zhou, Q., Miller, S., and Koltun, V. (2013). Elastic Fragments for Dense Scene Reconstruction. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 12

Zienkiewicz, J. and Davison, A. J. (2015). Extrinsics Autocalibration for Dense Planar Visual Odometry. *Journal of Field Robotics (JFR)*, 32(5):803–825. 13

Zienkiewicz, J., Davison, A. J., and Leutenegger, S. (2016a). Real-Time Height-Map Fusion using Differentiable Rendering. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 14

Zienkiewicz, J., Lukierski, R., and Davison, A. J. (2013). Dense, Auto-Calibrating Visual Odometry from a Downward-Looking Camera. In *Proceedings of the British Machine Vision Conference (BMVC)*. 13

Zienkiewicz, J., Tsiotsios, A., Davison, A. J., and Leutenegger, S. (2016b). Monocular, Real-Time Surface Reconstruction using Dynamic Level of Detail. In *Proceedings of the International Conference on 3D Vision (3DV)*. 14