Imperial College London
Department of Computing

Parametric Dense Visual SLAM

Steven Lovegrove

September 2011

Supervised by Dr. Andrew Davison

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.

Abstract

Existing work in the field of monocular Simultaneous Localisation and Mapping (SLAM) has largely centred around sparse feature-based representations of the world. By tracking salient image patches across many frames of video, both the positions of the features and the motion of the camera can be inferred live. Within the visual SLAM community, there has been a focus on both increasing the number of features that can be tracked across an image and efficiently managing and adjusting this map of features in order to improve camera trajectory and feature location accuracy.

Although prior research has looked at augmenting this map with more sophisticated features such as edgelets or planar patches, no incremental real-time system has yet made use of every pixel in the image to maximise camera trajectory estimation accuracy. Moreover, across many practical domains, these feature-based representations of the world fall short. In robotics, sparse feature-based models do not allow a robot to reason about free space and are not so useful for interaction. In augmented reality, sparse models do not allow us to place virtual objects behind real-ones and cannot enable virtual characters to interact with real objects.

In this research we show how a dense surface model offers many advantages and we explore different methods of reasoning about dense surfaces over a sparse feature-based map. We continue by developing different methods for dense tracking and constrained dense SLAM in different applications such as spherical mosaicing. Finally, we show how live dense tracking can be tightly integrated with dense reconstruction to create a 6 DOF monocular live dense SLAM system which outperforms the current state of the art in many respects.

Acknowledgements

First and foremost, I would like to thank my supervisor Andrew Davison for his unwavering encouragement, guidance and support. His patience and generosity has seen me through what I have found to be a tough few years, and he has shown great faith in my work when at times I have felt none for it myself.

I would also like to thank my fantastic lab mates from whom I've drawn a great deal of inspiration. I've enjoyed many a lively discussion and learnt plenty over beers and coffees, particularly from my Lycra clad buddy Richard Newcombe. He never fails to have something interesting or thoughtful to say, and collaborative work with him has been especially fun.

I'm grateful to Kaidi for her love and patience, particularly while I have been writing up. Our recent travels together form my fondest memories, and without those moments and her support I don't know how I would have finished.

I also owe a great deal to my past and present long-suffering flatmates — each has known how to raise my spirits when they were low, be that with tea, desserts, Fußball or a little hacking.

Finally, I would like to thank my family. They have always looked out for me and offered such unfaltering support that it would be all too easy to take them for granted. I owe my sister and Tina a particularly large debt of gratitude for supporting me throughout university and for always seeming to have my back. Most of all, I would like to thank my mother — all qualities that I value in myself can be found within her, and I am so grateful for her time and love over so many years.

Contents

Chapter 1

Introduction

Establishing the positions of a set of cameras and the structure of the scene they observe through their images alone belongs to an area of research known as *structure from motion* (SFM). The related problem of *incrementally* estimating a video camera's motion and the scene structure as images are received, is known as monocular *Simultaneous Localisation and Mapping* (SLAM), or alternatively incremental SFM.

Prior research in the field of monocular SLAM has largely centred around sparse feature-based representations of the world; by tracking the 2D locations of salient image patches across many frames of video, both the 3D positions of these point features and the motion of the camera itself can be estimated. So far, the aim of monocular SLAM systems has primarily been for camera pose estimation, relative to some fixed frame of reference — sparse structure being estimated largely as a necessary by-product of attempting to minimise localisation error relative to a static world-centred coordinate system.

State of the art monocular SLAM systems have improved their tracking accuracy and robustness over time by increasing the number of features which they can successfully match between frames and by finding efficient ways to manage increasingly large sparse feature maps. Many useful applications from robotics to augmented reality certainly require real-time estimation of a camera's pose. However, there is much more that needs to be learned about the world in order to enable other applications where true interaction is possible. Point cloud scene models fall short

when detailed predictive information is needed beyond pure localisation.

Seeing the limitations of the point feature-based visual SLAM systems available at the start of this project, the original focus of this research was to augment feature-based maps with more complete representations that would be more useful across different application areas. The idea was that further information could be extracted from images to 'fill in' the spaces between sparse features in a model, but still that the main framework on which this extra information was hung was essentially the same point feature cloud. We demonstrate some interesting early results based on this idea.

Although it was always clear that a surface model generated in this way could *aid* visual SLAM by offering information regarding expected occlusion and perhaps patch normals, we started from the point of view that SLAM itself was practically solved. However, reconsidering this point from our current perspective, we now see that is is possible to perform visual SLAM in ways which do not require a point feature framework at all; and that taking such an approach which aims at dense scene modelling right from the beginning in fact has such advantages that we might regard the use of point features at all as quite limiting, not only as a useful scene representation but as a part of a video-rate visual SLAM itself.

This view inspired a large proportion of the work presented here, where SLAM systems are constructed from dense whole-image approaches in constrained environments, such as spherical and single planar worlds, breaking away from popular point feature-based methods. We demonstrate that 'expensive' every-pixel methods that had been popular in classic off-line vision literature are particularly well suited to modern computing hardware and are now amenable to real-time implementation and enable very high SLAM performance, in terms of both accuracy and robustness.

Through collaboration, this research concludes by returning to visual SLAM within an unconstrained general 3D scene, showing that dense structure can be estimated and robustly tracked live using whole-image methods. By making use of information from every pixel without extracting point features, we show that significant advantages in accuracy and robustness can be found. Further, the by-product of dense visual SLAM systems is a dense photometric surface model useful for a host of interesting and important applications, particularly when available live.

1.1 Motivating Technologies

Several key technologies have come together in recent years to inspire the direction of this work, and enable it to proceed.

The biggest of these is monocular SLAM itself, whose development has progressed quite rapidly since the first systems were seen. We will give a review of monocular SLAM in Section 1.3. At the start of this project, MonoSLAM [27] was the best system available, but the arrival of PTAM [62] made a big impact, particularly with the availability of the software for research use. PTAM is both more robust and accurate than MonoSLAM, has excellent engineering throughout, and made it very clear what both the advantages and limitations are of a point-based SLAM framework which aims to build consistent maps in real-time. Also important have been 'visual odometry' approaches which showed what is possible when a camera just moves away from its starting point and explores new areas; pioneering methods such as that of Nistér *et al.* [97] showed the surprisingly low level of drift that can be achieved with monocular or stereo vision.

Image alignment based on iterative adjustment to maximise photoconsistency has been the core specific method we have used in the main parts of the thesis. There have been several developments of the original method presented by Lucas and Kanade [77], which was nicely revisited by Baker and Matthews in a series of recent papers [7]. Of particular interest is the related ESM approach [78] which had been used in other recent attempts at SLAM [114].

Highly related to these alignment methods and always a motivation of our work are offline approaches which estimate dense surface geometry via photoconsistency over small planar scene regions and join many of these into full models such as [46] and [41]. These particular papers were our original route into the world of dense multi-view stereo and the amazing dense reconstruction results which had become possible. It was the arrival of live dense reconstruction systems such as that of Newcombe and Davison [93] (also [122]), however, taking advantage of the latest variational optimisation techniques and parallel implementation (as also seen in optical flow estimation such as [135]), which finally allowed the path to open up towards the DTAM system presented at the end of this thesis.

Another major theme running through this work has been the emergence of

Graphics Processing Units (GPU's) as the newly dominant desktop processing resource for computer vision which finally enable real-time processing of every pixel in a video stream. We have closely followed the latest advances in commodity GPU hardware and programming techniques using both graphics APIs and the custom general purpose languages such as CUDA and made design decision in our vision algorithms to best target these architectures. We believe future architectures will rely increasingly on parallelism for power efficiency and performance.

1.2 Potential Applications

Through the accurate real-time estimation of the pose of a moving camera and the dense geometry that it observes, many useful applications will be made possible or enhanced. We expect these applications to fall into one of the following main categories: camera tracking, interactive modelling, spatial awareness, augmented reality or live video compression. We discuss each of these below.

Camera Tracking

Accurate and robust camera tracking itself has some useful applications, where even small improvements in precision and reliability over previous systems offer an advantage. On the road, assisted navigation systems are currently limited by poor localisation normally provided by a single sensor such as GPS. Augmenting existing solutions with accurate camera tracking could offer an inexpensive means to high-quality driver assistance systems. We present such a system in Chapter 5.

Perhaps one of the simplest conceivable applications is in using a camera as an input device. By precisely knowing the absolute pose and orientation of a camera, it could be used as an inexpensive 3D mouse for CAD tool manipulation or within computer games as a means for increasing the non-drifting capabilities of motion controllers without the need for external infrastructure.

Interactive Modelling

Within several industries, the digitisation of physical objects or scenes is very valuable. Frequently, trained visual artists are hired to model objects from scratch as a route to high quality reconstructions, but there is a great demand for automatic technologies with the potential for much-reduced cost as well as increased fidelity.

State of the art *multi-view stereo* (MVS) systems for dense reconstruction are often able to produce very detailed models from images from known camera positions, but many limitations remain. Firstly, they typically take large quantities of computation time to produce any model at all. Since they operate in batch processes, the user has no feedback when capturing data to determine if they have acquired a sufficient number of images from different viewpoints to cover the target. Returning to a scene to capture more data might be expensive, or prone to changing structure or lighting. Real-time dense reconstruction can enable a user to receive feedback with regard to coverage or capture more data in areas with poor model resolution. Later, an offline MVS system may be used to refine the model.

In telerobotics, a human operator sits at a terminal controlling a remote robot receiving visual feedback. In this scenario, an operator's visual perception is limited by the field of view of the camera. Dense visual SLAM might enable increased perceptual awareness by allowing the user to view the history of previous observations within a single, intuitive 3D model.

Spatial Awareness

Despite what science fiction may have promised, in the 21st century robots are still unable to interact in useful ways within general man-made environments. Although there are several obstacles to overcome in realising such a future, surely the biggest is that of achieving timely spatial awareness. At present, robots typically rely on active sensors such as laser scanners which are prohibitively expensive for many areas and have shortcomings in terms of scanning density. New active depth camera technology, such as Primesense / Microsoft Kinect, will no doubt have a large effect on rapid progress in this area and is particularly exciting now it is available at commodity prices that allow many thousands of researchers to enable its possibilities.

However, we still have a strong belief that standard passive vision will only con-

tinue to increase in importance in robotics because we have only begun to scratch the surface of the vast amount of information, both geometrical and semantic, which can be extracted from images. This is particularly true as the ever-growing computational resources we have available enable the real-time uses of imaging technologies already widespread which have much higher resolution, dynamic range, or framerate than those used in this thesis. Visual SLAM systems which currently build only sparse feature maps can help a robot to know where it is but not to explore or interact. Dense surface geometry would enable a robot to explicitly reason about the free space in which it can move and to estimate points of contact for potential interaction. In this sense, dense SLAM can endow a robot with at least some form of spatial awareness.

Augmented Reality

In the area of *augmented reality* (AR), a live video, perhaps from a head-mounted display or broadcast sporting event, is composited with an overlay in which virtual objects or characters generated with computer graphics can be seen to interact with physical ones. In this sense, we aim to augment users' knowledge, or immerse them in an entirely or partially artificial world. These virtual items must exist in a static coordinate system consistent with the real world. For AR to be convincing, the virtual and real must be tightly locked together, and this requires accurate real-time tracking of the moving real camera; visual SLAM based on point features is already the most promising technology for this to be generally achieved. However, dense SLAM would enable much more ambitious applications where virtual items are not just positioned with respect to the real world but can actually fully interact with its geometry. We show some examples of this in Chapter 6.

The potential applications of AR are many; much interest exists in the entertainment industry where we might envisage playing games situated in real environments around the home. There is particular excitement about this technology in the mobile computing area, with many companies already developing AR-based smartphone applications for both assistive tasks and entertainment. Augmented reality is starting to gain ground in television too — broadcasters hope to engage viewers by displaying interesting information in intuitive ways on top of the live video footage.

Live Video Compression

In the modern age, bandwidth has become a valuable resource, but at the same time video streaming and consumer demand for high-definition content has increased. Existing compression schemes typically take advantage of spatial and temporal redundancy of images across video by calculating the motion of pixels over time, an area of research referred to as optic flow. For many parts of a video, where the majority of the scene is static, it is clear that only a few parameters are required to describe how pixels will move between two frames if the structure of the scene is known.

A potential future application of monocular dense SLAM methods may lie in extreme compression of video streams. In current compression algorithms where common pixel motion priors are used, we might expect smooth structure priors and mostly rigid scene priors to greatly enhance compression ratios for comparative image quality.

1.3 A Brief Review of Visual SLAM

In this section we will give a short review of the overall research area of visual SLAM as is relevant to the overall aims of this project. Substantial further review material specific to each of the main chapters will be given in context later in the thesis.

The area of SLAM of course originates in the mobile robotics community, with the goal of providing a robot which is dropped into an environment it has not previously visited with the capability to localise with respect to a continuously growing map it builds itself from on-board sensor measurements. The main assumption nearly always made which makes the whole problem tractable is that the robot is the only moving entity within an otherwise static or rigid environment. Importantly, in robotics it is clearly necessary to achieve SLAM in real-time for it to be generally useful. Also, most researchers in robotic SLAM have been interested in methods which can build in real-time, maps which are *internally consistent*. We will consider the importance of this in the rest of the discussion, but note already that our own aim is also real-time SLAM algorithms which build consistent, drift-free world representations.

For several years the main methodology for maintaining and extending a live map and estimating the current pose of a robot was to use a sequential probabilistic filter whose job was to refine a joint density over estimates of the live robot pose and the positions of the scene landmarks it had observed. The key algorithm in the original real-time SLAM systems (e.g. [96],[25],[16]) was the *extended Kalman filter* (EKF), first proposed for this purpose by Smith *et al.* [117] and Moutarlier and Chatila [91].

These systems implemented SLAM for mobile robots with various sensors, usually those which could measure both depth and bearing directly such as stereo vision or sonar, but invariably also wheel odometry information and strong assumptions about planar robot motion. Monocular SLAM, where we must achieve real-time localisation and mapping from a single camera, perhaps hand-held, is more challenging. A breakthrough monocular SLAM system was Davison's MonoSLAM [26], which adapted the EKF approach successfully to the monocular domain. This work built on earlier but more limited filtering-based sequential approaches to monocular motion estimation and mapping such as the work of Chiuso *et al.* [18] and the much older DROID system [47]; but also the wealth of work in off-line SFM where the problem of estimating the locations of multiple single cameras from image correspondences, or the historical trajectory of a moving video camera, had been studied extensively under different computational constraints (e.g. [40], [104]). In these latter systems, multi-view feature matching and local geometry estimation is used as a front-end to global *bundle adjustment* (BA) of the estimated camera and feature positions, where reprojection error for all features observed in the sequence is jointly minimised.

MonoSLAM was able to estimate the live camera pose and the locations of a sparse set of feature points using an EKF, maintaining full covariance over the complete state vector. However, the EKF scales badly with map size because computational complexity is in general proportional to the cube of the state vector's size (the state vector being composed of stacked robot and feature position estimates). In visual SLAM, this complexity is practically a little better — scaling with the square of the number of features in the total map, since typically observations relate only to a fraction of the vector's state. This means that the size of map which can be created in real-time with an EKF SLAM approach is strictly computation-bounded. MonoSLAM was able to handle only around 50–100 features in the state vector

at 30Hz operation. Over longer trajectories, the EKF is also susceptible to escalating inconsistencies as compared to the optimal bundle adjusted solution, due to propagated linearisation errors. Many authors investigated methods to mitigate both problems, often making an approximation by splitting a large map into several partitions or 'sub-maps' (e.g. [13], [21]).

For large maps, an approach gaining popularity over EKF formulations but dating back to a similar period, was Lu and Milios' method of 'consistent poses' [75]. They were working with 2D laser range-scan data from a moving robot. With this powerful data, it is possible to align two consecutive scans to get an accurate local estimate of the robot's movement from one timestep to the next. Lu and Milios' key observation was that having done so, the local range-scan measurements themselves could be discarded for the purposes of localisation and instead what was stored was a pose-to-pose constraint. As the robot continued to explore, a graph of these constraints was built up. The structure of the map is therefore defined implicitly by just the historical robot poses, rather than by an explicit feature map. The key thing is what happens when the robot re-visits previously seen areas and closes loops in the graph. Lu and Milios attempted loop closures by geometric interpretation of the pose graph, aligning non-consecutive range-scans that overlapped based on their current pose estimates. Once loop closures have been put into the graph, for the poses to be consistent around the loop all of the edges within the loop must compose to form the identity. This was enforced by distributing the error around the loop, weighted by the edge constraint uncertainty.

This 'pose-based SLAM' approach gradually became very influential. The first large scale use of a *consistent pose* method inspired by Lu and Milios was by Gutmann and Konolige, who looked more carefully at loop closure over longer trajectories where correspondence cannot be estimated purely by geometric overlap [44]. Instead, they considered a *topological* graph of connectivity which they took to represent the metric world but not necessarily match it exactly. They ensured only strong loop constraints were enforced and found loop constraints over larger areas. Other excellent later work on this method and efficient strategies for optimising large maps with many constraints was due to Olson *et al.* [99].

Progress continued in feature-based SLAM, and different sorts of filters were investigated. Montemerlo *et al.* demonstrated an alternative to submapping to improve scalability with their FastSLAM algorithm [89]. They described a recursive Bayesian

factorisation of the problem ('Rao-Blackwellization') with an efficient implementation where the posterior distribution was represented by a particle filter over the robot's trajectory, with independent Kalman filters for each landmark attached to each particle. Eade and Drummond somewhat later used this algorithm for a monocular visual SLAM system which was more scalable than EKF-based systems [34], handling hundreds of features in real-time. Previously, Pupilli and Calway had used a more straightforward particle filter method for monocular SLAM, but this was suitable only for very small maps [105].

It wasn't until later that a number of authors showed how these different methods (filtering, bundle adjustment and maximising 'pose consistency') for SFM and SLAM could be seen as different *factorisations* within the full SFM / SLAM *graphical model*. Prominent here were Dellaert and Kaess [28], building on important earlier insights by Thrun *et al.* [126] and Paskin [102]. They demonstrated that many of the graphical methods employed for SLAM had direct equivalents in sparse linear algebra and Bayesian inference, where factorisation can be equivalently seen as marginalisation or variable elimination for instance. This view now supports a unified approach to SLAM, enabling efficient estimation for large maps of both features and robot poses, and supported in particular by efficient graph optimisation libraries such as TORO [43] and g^2o [67].

In monocular SLAM using features, state of the art approaches now commonly use a graph inference approach, the first example of this being due to Eade and Drummond [35]. There continued to be open issues on exactly the best way to spend limited real-time processing resources however in this particularly challenging SLAM domain; and particularly on how a 'front-end' image tracking system should interface with a 'back-end' estimation engine.

In their important paper 'Parallel Tracking and Mapping for Small AR Workspaces', Klein and Murray described their system PTAM, motivating an alternate view to monocular SLAM, that only tracking needs to occur at frame-rate [62]. Rather than integrating feature measurements at every frame to construct their sparse map, they include only wide-baseline *keyframes* which are selected heuristically based on the cameras motion from the set of all video frames observed. These keyframes are globally bundle adjusted in a continuous batch SFM thread. In parallel, at frame rate, the camera's location is computed accurately by matching many hundred of features in the current image to those in the closest keyframes. By bundle adjusting

these key historic poses, they avoid the propagation of linearisation errors inherent in sequential filters such as the EKF.

By avoiding the computational expense of providing an 'optimal' map at each frame, PTAM is able to spend more time on robust tracking, locating to sub-pixel accuracy hundreds of point features within each image. The sheer number of features tracked provides significantly improved accuracy and stability compared to previous systems, supporting greater dynamic camera motions. Later enhancements to PTAM added edge features and an initial, dense $\mathbb{SE}(2)$ based rotation estimation step [63]. Although PTAM did not attack tracking in expansive exploratory sequences, the impact it has made within the research field of monocular SLAM is large.

Taking the view of seeing keyframe based and sequential filtering based SLAM as different factorisations within the graphical methods framework discussed by Dellaert and Kaess, Strasdat *et al.* posed the question, 'Why filter?' [120]. They offered a thorough analysis looking to address the question of how computation should be allocated in a real-time monocular SLAM system so as to maximise tracking accuracy. Is it better to incorporate more video frame measurements over time, or to instead use these processing resources to track more features between sparser video frame measurements? They concluded that it is computationally most efficient to use a minimal number of frames whilst tracking a maximal number of correspondences, except in the case where computational resources are *very* heavily constrained and only a few features can be tracked between frames at all. This analysis very much confirms the design choices of PTAM. The latest developments in point-based monocular SLAM [119] take this on board, and are now aiming at systems which operate like PTAM locally but have a second level which is similar to pose-graph optimisation to enable scaling to much larger workspaces.

Besides these developments in monocular SLAM which have essentially assumed the same basic point feature matching process as the image processing end, there have been a small number of methods where a different approach has been taken. There have been some systems which have used line features (e.g. [116], [33]); and others which have made initial investigations of planar patches as SLAM features (e.g. [88], [115]). These latter methods, and particularly [115] where measurements of planar warps are used directly to give information on camera pose relate most closely to the work we will present later in this thesis and we will revisit them in

context later.

1.4 Contributions

The work in this thesis had the original aim of augmenting point feature-based monocular SLAM with additional representations which would permit dense scene modelling in real-time (preliminary work on this idea is presented in Chapter 3); but over the course of the research conducted has increasingly looked at building SLAM methods which use dense, every-pixel methods throughout and investigates the advantages of these.

Starting in Chapter 4 we look deeply into parametric 'direct' tracking methods, where parts of images are aligned in the Lucas-Kanade [77] style of iterative optimisation of photoconsistency. We apply the latest developments of this class of algorithm such as ESM [78] together with efficient parallel implementation on GPGPU processing hardware to produce a highly effective frame-rate image alignment solution where the parameterisation can be easily changed depending on the targeted application. We demonstrate both a high performance visual gyroscope for a purely rotating camera, and visual odometry for a road vehicle by tracking the road texture observed by a single downward-facing camera (in the latter case, the visual information also being fused with GPS).

In Chapter 5, we move on into methods which are able to make use of this whole image alignment approach within a whole SLAM framework capable of not just drifting motion estimation but real-time consistent mapping. In this chapter, the scenes considered are simplified from full 3D geometry, either to the spherical panorama case or a large planar environment, but we show that there are important practical uses of both of these models. The core architecture of the systems we develop is similar to PTAM, with a backbone of keyframe locations; but now the tracking of live camera pose is achieved with dense whole image alignment, and the optimisation of the whole map via either full joint photoconsistency optimisation or a pose-graph type approach. We demonstrate both real-time consistent spherical panorama generation, with high fidelity real-time rendering of various projections, and real-time planar mapping from an arbitrarily moving camera. In the panorama case, we are able to perform camera intrinsic calibration refinement within the main processing

loop. The planar mapping is demonstrated in various indoor and outdoor scenes, and also in a document scanning application where we show that the quality of tracking permits real-time super-resolution reconstruction.

Finally in Chapter 6, we present the DTAM: Dense Tracking and Mapping system (developed in collaboration with Richard Newcombe) which is a full realisation of a fully dense real-time monocular SLAM system capable of making accurate 3D surface models of a complicated scene browsed only by a hand-held single camera. The results of the previous chapters on alignment-based tracking are transferred to this full 3D domain, where live camera tracking is now achieved by 6DOF alignment of current dense 3D model with the live camera view. We demonstrate the great advantages this offers over point-based tracking, since occlusions can be fully handled; high accuracy is achieved due to the large amount of data considered; and in particular that very rapid camera motion can be tracked due to the multi-scale alignment method which is robust to image blur.

1.5 Publications

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

DTAM: Dense Tracking and Mapping in Real-Time [95]
Richard Newcombe, Steven Lovegrove and Andrew Davison
Proceedings of the International Conference on Computer Vision (ICCV), 2011

Accurate Visual Odometry from a Rear Parking Camera [73]
Steven Lovegrove, Andrew Davison and Javier Ibañez-Guzmán
Proceedings of the IEEE Intelligent Vehicles Symposium (IV), 2011

Real-Time Spherical Mosaicing using Whole Image Alignment [72]
Steven Lovegrove and Andrew Davison
Proceedings of the European Conference on Computer Vision (ECCV), 2010

1.6 Thesis Structure

The structure of the rest of this thesis is as follows. In Chapter 2 we introduce the notation, geometrical models and other preliminaries that we will build the theory of the rest of the thesis on. Chapter 3 presents some general discussion of dense SLAM, and an initial method based on augmenting a standard feature-based SLAM map with surface information. After this we move on to the core of the thesis in Chapters 4 and 5, where we concentrate on real-time tracking using direct methods and then using these methods to create full SLAM systems for reduced spherical and planar domains respectively. In Chapter 6 we present the DTAM (Dense Tracking and Mapping) system for full dense 3D SLAM, which was developed in collaboration with Richard Newcombe and takes many of the ideas in this thesis to their logical conclusion. Finally we conclude in Chapter 7 with some thoughts on potential future work.

Chapter 2

Preliminaries

Within this chapter we will attempt to present the notational convention used throughout this work, as well as outlining some of the important equations which recur when considering multi-view stereo. Also, as graphics hardware has been optimised so heavily over the years for 3D geometry, we will give a brief overview of how to make the most of OpenGL as a vision researcher, describing some of the tools used throughout this work for achieving real-time performance and visualisation.

2.1 Frames of Reference

When we talk about frames of reference, we are referring to local coordinate systems. Making a distinction between one frame of reference and another is often convenient when talking about interacting local systems. Within the field of multi-view stereo, we are interested in the projection of objects into different cameras.

For a camera C_k, we refer to its local frame of reference as just k. In local camera coordinates, camera C_k's optic centre is located at $(0, 0, 0)^\top$. We use a right-handed system in common with OpenGL's convention, with the camera's principal point facing down the negative z-axis and the positive y-axis as the camera's 'up' vector (Figure 2.1). We will use the terms 'camera' and 'frame of reference' interchangeably.

When defining a position vector $\mathbf{x}$ rooted in camera frame k, we label it $\mathbf{x}_k$.

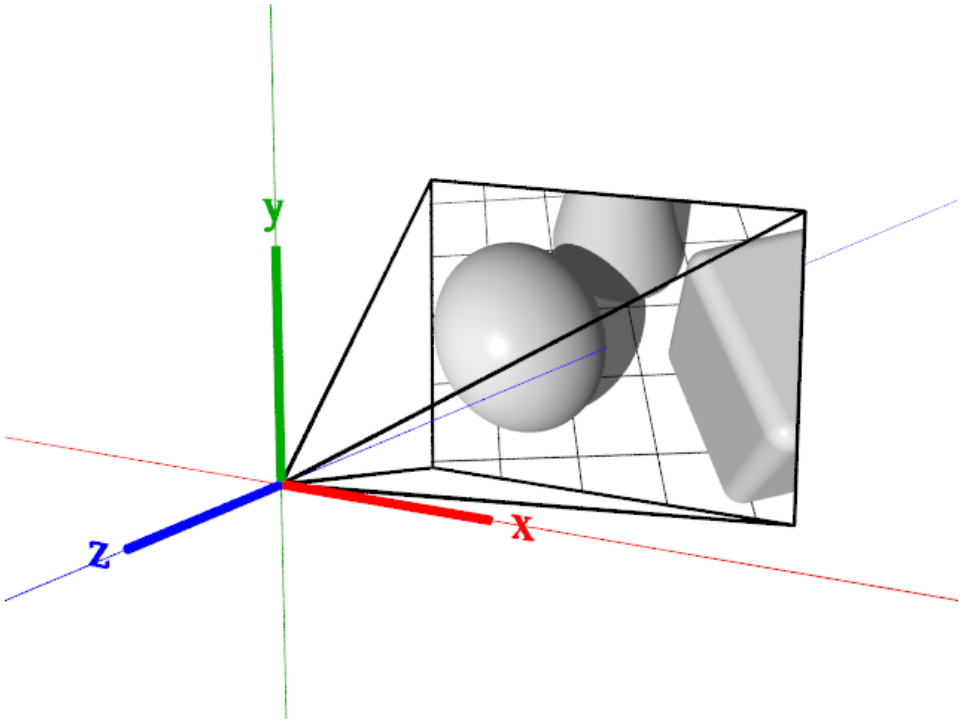


Figure 2.1: Camera Coordinate Convention.

Typically, when no subscript is given, we are implicitly talking about the world frame of reference w — an arbitrary frame of reference in which our cameras are defined. Hence, $\mathbf{x}_w = \mathbf{x}$.

We use homogeneous coordinates to express a greater range of transformations using linear algebra. Using homogeneous vectors increases the size of the vector by one. A vector $\mathbf{x}$ can be extended homogeneously using the dot notation such that $\dot{\mathbf{x}} = \binom{\mathbf{x}}{1}$. The reverse operation which we call homogeneous projection is depicted by π and simply divides through by the last ordinate and truncates by one. This implies that $\dot{\mathbf{x}}, 2\dot{\mathbf{x}}, \alpha\dot{\mathbf{x}}, \ldots$ all represent the same vector $\mathbf{x}$. For a three-vector:

$$\pi \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \end{pmatrix}. \tag{2.1}$$

We represent transformations between cameras by the 4×4 matrix representing homogeneous point transfer between these frames. For example, $\mathtt{T}^{ba}$ represents the matrix which transforms homogeneous points defined in frame a, to the equivalent

points in frame b, such that $\dot{\mathbf{x}}_b \propto \mathtt{T}^{ba}\dot{\mathbf{x}}_a$. We can decompose $\mathtt{T}^{ba}$ as follows:

$$\mathtt{T}^{ba} = \begin{pmatrix} \mathtt{R}^{ba} & \mathbf{a}_b \\ \mathbf{0}^T & 1 \end{pmatrix} = \begin{pmatrix} \mathtt{R}^{ba} & -\mathtt{R}^{ba}\mathbf{b}_a \\ \mathbf{0}^T & 1 \end{pmatrix} . \tag{2.2}$$

Where $\mathtt{R}^{ba}$ is a 3×3 orthonormal rotation matrix representing rotation-only point transfer between frames a and b. Here, $\mathbf{b}_a$ represents the position of the origin of frame b in the frame of reference a.

Our notation allows us to compose transformations very easily; observe how adjacent super-scripted frames of reference match:

$$\mathtt{T}^{ca} = \mathtt{T}^{cb}\mathtt{T}^{ba} . \tag{2.3}$$

If we wish to compute the transformation $\mathtt{R}^{ab}$ from $\mathtt{R}^{ba}$, or $\mathtt{T}^{ab}$ from $\mathtt{T}^{ba}$, we can avoid a general inversion by using our decomposition and the fact that $\mathtt{R}^{ba}$ is orthonormal:

$$\mathtt{R}^{ab} = (\mathtt{R}^{ba})^{-1} = (\mathtt{R}^{ba})^{\top} , \tag{2.4}$$

$$\mathtt{T}^{ab} = (\mathtt{T}^{ba})^{-1} = \begin{pmatrix} (\mathtt{R}^{ba})^{\top} & -(\mathtt{R}^{ba})^{\top}(\mathbf{a}_b) \\ \mathbf{0}^T & 1 \end{pmatrix} . \tag{2.5}$$

Sometimes it is convenient to transform a homogeneous point in one frame of reference to an inhomogeneous point in another. Since the bottom row of our 4×4 transform will always preserve the homogeneous ordinate of a right-multiplied vector, we can equally use any 3D rigid body transformation matrix in its 3×4 form, and we often will;

$$\mathtt{T}^{ba} = \left(\mathtt{R}^{ba} \;\middle|\; \mathbf{a}_b \right) . \tag{2.6}$$

2.2 Projection

If we know the internal and external parameters of a camera C_k we can project points from the world into this camera to find their 2D image coordinates. In multi-view

stereo, the projective action of the camera is normally considered in two stages: a linear component which accurately models a pinhole camera (Figure 2.2) and a non-linear 'lens distortion' component that applies in image space (Figures 2.3 and 2.4). Within this thesis, we typically remove lens distortion to produce an equivalent pinhole image at the start of the processing pipeline. Although it could be argued that data is lost through resampling, this process saves repeated evaluation of the distortion equations, and simplifies subsequent steps significantly.

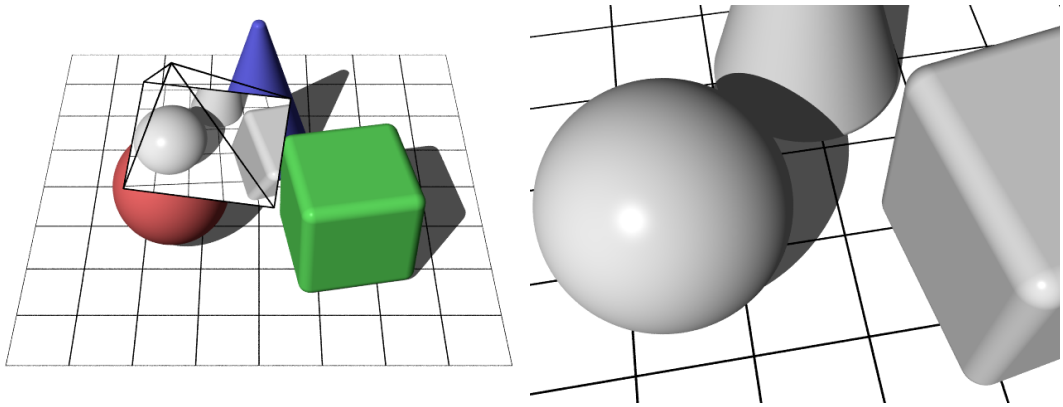


Figure 2.2: Projection of a scene onto a camera's image plane (left) to form a projective image via a pinhole camera (right). Straight lines remain straight.

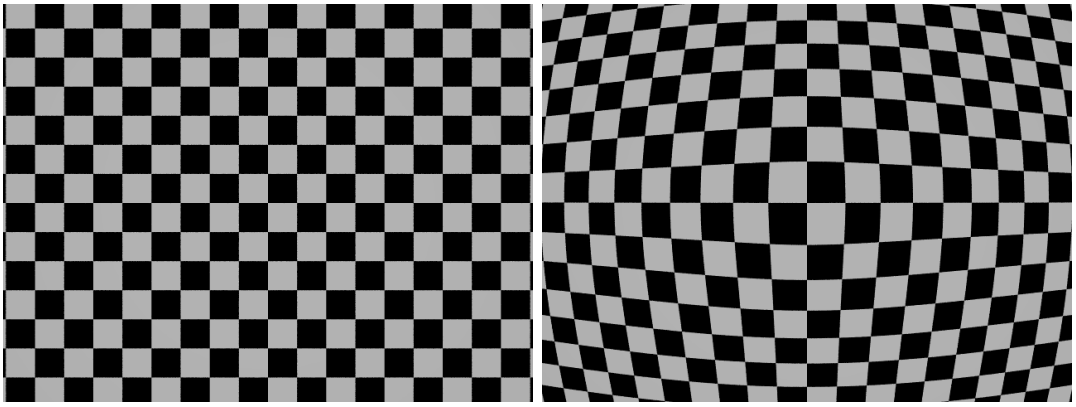


Figure 2.3: Action of Lens distortion on checkerboard pattern. Straight lines appear curved.

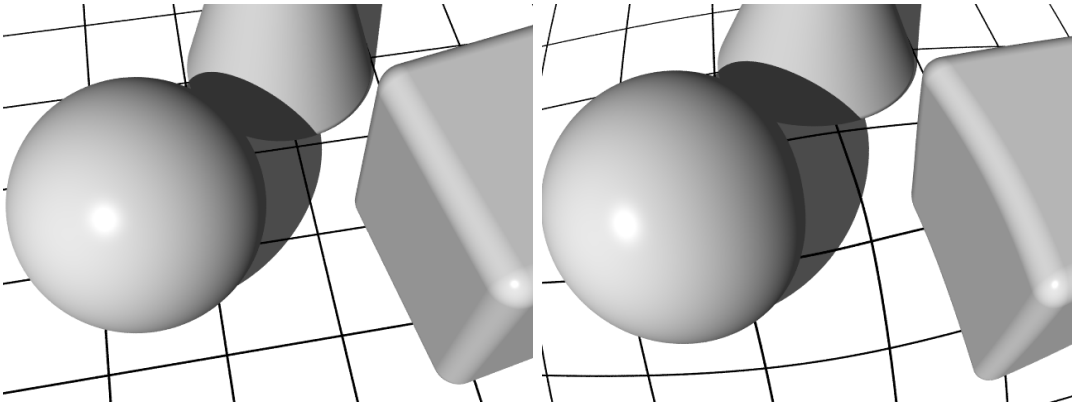


Figure 2.4: Non-linear Lens distortion common to wide angle lenses. Straight lines appear curved.

2.2.1 Pinhole Camera

A 3D scene point $\mathbf{P} = (X, Y, Z)^\top$ expressed in the camera's local frame of reference will project via a pinhole camera to image coordinates $\mathbf{u} = (u, v)^\top$:

$$u = f_u \frac{X}{Z} + u_0, \qquad v = f_v \frac{Y}{Z} + v_0, \tag{2.7}$$

where (u_0, v_0) are the 'principal point' coordinates of the camera in image space, reflecting the centre of projection. f_u, f_v are horizontal and vertical scaling factors, themselves functions of the focal length of the camera f and pixel size, $p_w \times p_h$ (often $p_w \approx p_h$):

$$f_u = \frac{f}{p_w}, \qquad f_v = \frac{f}{p_h}. \tag{2.8}$$

The linear pinhole effect of a camera is often written in matrix form within a single 3×3 matrix $\mathtt{K}$ called the intrinsic / calibration matrix:

$$\mathtt{K} = \begin{pmatrix} f_u & \alpha & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.9}$$

This matrix is parameterised by the camera's horizontal and vertical scaling factors (f_u, f_v), principal point (u_0, v_0), and skew co-efficient α representing any pixel

shear. For most cameras, α is very close to zero and ignored, as we will do for the rest of this thesis. For a 3D point $\mathbf{P}_k$ defined in the frame of reference of camera C_k, by left multiplication the intrinsic matrix takes the point into homogeneous image space coordinates:

$$\dot{\mathbf{u}}_k \propto \mathtt{K}\mathbf{P}_k \ . \tag{2.10}$$

We can use our frame of reference transformations to project points specified in another frame into our image:

$$\dot{\mathbf{u}}_k \propto \mathtt{K}\mathtt{T}^{km}\mathbf{P}_m \ . \tag{2.11}$$

Although the operation of projection has no inverse, in homogeneous coordinates the inverse calibration matrix is still useful for taking direction vectors described by pixel coordinates and transforming them into the camera's frame of reference. Assuming the skew parameter to be equal to 0, the inverse calibration matrix can be defined as:

$$\mathtt{K}^{-1} = \begin{pmatrix} \frac{1}{f_u} & 0 & -\frac{u_0}{f_u} \\ 0 & \frac{1}{f_v} & -\frac{v_0}{f_v} \\ 0 & 0 & 1 \end{pmatrix} \ . \tag{2.12}$$

2.2.2 Non-Linear Lens Distortion

The non-linear component of projection is often considered in image space as a distortion or warp to the linear pinhole projection previously described. Whereas perspective projection maintains straight lines, those lines which are straight in the world may appear curved after non-linear lens distortion, often with the effect of making the centre of the image appear bulbous (barrel distortion), or squashed (pincushion distortion).

Non-linear distortion is scene independent, operating in image space and transforming a standard pinhole projected image $\mathtt{I}^p$ into a distorted image $\mathtt{I}^d$. In computer vision, we are typically interested in how to undo this deformation in order to make use of the projective geometry of the image.

Image distortion is typically considered in two components relative to a centre of distortion $\mathbf{d} = (d_u, d_v)^\top$: *radial distortion* affecting a pixel's distance from the centre of distortion, and *tangential distortion* which applies perpendicular to this direction. For computer vision problems, tangential distortion is frequently disregarded since it is small for most cameras. The centre of distortion is often assumed to be equal to the image's principal point in order to reduce complexity, though this is not always precisely correct.

Considering only radial distortion, we can write down how the coordinates of a pixel in the distorted image $\mathbf{u}_d = (u_d, v_d)^\top$ are transformed into equivalent pinhole image coordinates $\mathbf{u}_p = (u_p, v_p)^\top$ and vice-versa via a radial distortion function, $R : r_p \to r_d$, where $r_p = \|\mathbf{u}_p - \mathbf{d}\|_2$ is the undistorted pinhole distance of a pixel from the centre of distortion, and $r_d = \|\mathbf{u}_d - \mathbf{d}\|_2$ is the distorted distance:

$$\mathbf{u}_d = (\mathbf{u}_p - \mathbf{d})\frac{R(r_p)}{r_p} + \mathbf{d}, \tag{2.13}$$

$$\mathbf{u}_p = (\mathbf{u}_d - \mathbf{d})\frac{R^{-1}(r_d)}{r_d} + \mathbf{d}. \tag{2.14}$$

Choosing an appropriate distortion model for the function R depends partly on the type of camera being used. Camera and lens systems that exhibit little distortion are often adequately modelled by a simple low-order polynomial such as a cubic. For fish-eye lenses, or those with large fields of view, accurately approximating lens distortion may require a higher order polynomial. For stable camera calibration we ideally would like to estimate only a few parameters of a model which can accurately reflect the true distortion. We will briefly describe two distortion models: firstly the popular polynomial model and secondly the field of view model proposed by Devernay and Faugeras which can accurately model wide angle lens distortion with a few parameters [29].

Polynomial Power Series Model

The polynomial power series approximation remains a popular model for radial distortion and is used within several calibration procedures, including the popular method proposed by Zhang based on a planar checkerboard pattern [137]. It is therefore common for vision systems which operate under the assumption of known

intrinsics to use this model. Its polynomial radial distortion function can be written as:

$$r_p = R^{-1}(r_d) = r_d(1 + K_1 r_d^2 + K_2 r_d^4 + \ldots), \tag{2.15}$$

where $K_1, K_2, \ldots$ are the parameters of distortion. We use the smallest order polynomial that adequately reflects the camera's distortion, normally using the first couple of coefficients. For some camera systems, the first order distortion term is sufficient for fractional pixel accuracy. In this case, the inverse distortion function (finding r_d from r_p) can be found by solving a cubic. For higher orders, no closed form solution exists, and instead an iterative method must be used.

FOV Model for Wide Angle Cameras

For very wide angle camera systems, such as those with fish-eye lenses, the 'perfect' projective lens which induces no distortion is actually quite limiting, as angle is disproportionately represented across the image, making objects on the periphery appear very large. For these systems, lenses are typically designed to represent angle more uniformly across image space, and this is the motivation behind Devernay and Faugeras' model for lens distortion [29]. They model the projective geometry of a 'perfect' fish-eye lens:

$$r_d = R(r_p) = \frac{1}{\omega} \arctan\left(2 r_p \tan\frac{\omega}{2}\right), \tag{2.16}$$

$$r_p = R^{-1}(r_d) = \frac{\tan(r_d \omega)}{2 \tan\frac{\omega}{2}}, \tag{2.17}$$

where ω is the single parameter of the model and reflects a physical property of the camera, its *field of view* (FOV). A large advantage of this model is that it has a simple analytical inverse. They comment that this formulation can be combined with a polynomial correction to improve accuracy further whilst limiting the number of parameters required. Devernay and Faugeras also present a calibration method for radial distortion based on this model and compare it to the polynomial power series formulation. They demonstrate that they can achieve decreased mean reprojection error from their formulation over the power series formulation with the same number of parameters.

2.3 Planes

In this thesis we consider planar regions at various times — whether for efficient rendering or as a simplifying surface representation for reconstruction / tracking. We choose to define a plane in the following way:

$$\hat{\mathbf{n}} \bullet \mathbf{P} + d = 0, \tag{2.18}$$

for a point $\mathbf{P} = (X, Y, Z)^\top$ on a plane where $\hat{\mathbf{n}}$ is the outward unit normal of the plane and d is the distance of closest approach to the origin from the plane (strictly positive). This can be written homogeneously as:

$$\mathbf{N} = \begin{pmatrix} \hat{\mathbf{n}} \\ d \end{pmatrix}, \tag{2.19}$$

$$\mathbf{N} \bullet \dot{\mathbf{P}} = 0 \ . \tag{2.20}$$

Following from the homogeneous definition of $\mathbf{N}$, we can parameterise a plane minimally by the scaled normal vector, which we can write using the projection function:

$$\mathbf{n} = \pi(\mathbf{N}) = \frac{\hat{\mathbf{n}}}{d} \ . \tag{2.21}$$

The parameterisation expressed in Equation 2.21 is degenerate for planes passing through the centre of the coordinate system, but when considering camera-centric frames of reference, planes passing through the optic centre of the camera needn't be considered since they cannot be imaged.

Note that the solution space of Equation 2.20 is satisfied regardless of the scaling on $\mathbf{N}$, so the following condition defines the same plane (though the homogenisation is important):

$$\dot{\mathbf{n}} \bullet \dot{\mathbf{P}} = 0 \ . \tag{2.22}$$

2.3.1 Planes Between Frames

As with vectors, we use a subscript to denote the frame in which a plane is defined. $\mathbf{N}_a$ or $\mathbf{n}_a$ for example would depict the plane expressed in frame of reference a. We can express a plane in a different frame of reference as follows:

$$\mathbf{N}_b = \left(\mathtt{T}^{ab}\right)^\top \mathbf{N}_a, \qquad \dot{\mathbf{n}}_b = \left(\mathtt{T}^{ab}\right)^\top \dot{\mathbf{n}}_a. \tag{2.23}$$

Notice that contrary to frame transformations between vectors, for planes we take the transpose of the matrix $\mathtt{T}^{ab}$ and not the inverse for Equation 2.23.

2.3.2 Pixel-Plane Intersection

The ray formed from a camera pixel can be expressed in the camera's local frame of reference as follows:

$$\mathbf{P} = Z.\mathtt{K}^{-1}\dot{\mathbf{u}}, \tag{2.24}$$

where Z, the depth along the z-axis, is a free parameter that also forms the last ordinate of $\mathbf{P} = (X, Y, Z)^\top$, a point along the ray.

For intersection with a plane $\mathbf{N} = (\hat{\mathbf{n}}, d)^\top$, we need only equate $\mathbf{P}$ in Equations 2.18 and 2.24 and rearrange for Z:

$$Z = \frac{-d}{\hat{\mathbf{n}} \bullet (\mathtt{K}^{-1}\dot{\mathbf{u}})} = \frac{-1}{\mathbf{n}^\top \mathtt{K}^{-1}\dot{\mathbf{u}}}. \tag{2.25}$$

Finally, we can substitute this back into Equation 2.24 equating Z to obtain the intersection point, $\mathbf{P}$:

$$\mathbf{P} = \frac{-\mathtt{K}^{-1}\dot{\mathbf{u}}}{\mathbf{n}^\top \mathtt{K}^{-1}\dot{\mathbf{u}}}\ . \tag{2.26}$$

2.3.3 Plane-Induced Homographies

A *plane-induced homography* describes the transformation in image space between pixels in two cameras observing a common plane. The plane-induced homography is a projective transformation, in that straight lines remain straight. It can be

represented homogeneously as a 3×3 matrix $\mathtt{H}^{ba}$ taking pixel coordinates $\mathbf{u}_a$ from image $\mathtt{I}^a$ into pixel coordinates $\mathbf{u}_b$ in image $\mathtt{I}^b$, such that $\dot{\mathbf{u}}_b \propto \mathtt{H}^{ba}\dot{\mathbf{u}}_a$.

We can derive an expression for the homography induced by a plane by considering the ray formed from a pixel in one camera, which we intersect with a scene plane and project into our second camera after changing the frame of reference. We do that by expressing the intersection of a pixel ray and a plane (Equation 2.26), and taking the 3×4 representation of our frame transform (Equation 2.6):

$$\dot{\mathbf{u}}_b \propto \mathtt{K}\left(\mathtt{R}^{ba} \;\middle|\; \mathbf{a}_b\right)\begin{pmatrix} -\mathtt{K}^{-1}\dot{\mathbf{u}}_a \\ \mathbf{n}_a^\top \mathtt{K}^{-1}\dot{\mathbf{u}}_a \end{pmatrix}. \tag{2.27}$$

Tidying up, we can write:

$$\mathtt{H}^{ba} = \mathtt{K}\left(\mathtt{R}^{ba} \;\middle|\; \mathbf{a}_b\right)\left(\mathtt{I} \;\middle|\; -\mathbf{n}_a\right)^\top \mathtt{K}^{-1}. \tag{2.28}$$

Or, as is more commonly seen:

$$\mathtt{H}^{ba} = \mathtt{K}\left(\mathtt{R}^{ba} - \mathbf{a}_b\mathbf{n}_a^\top\right)\mathtt{K}^{-1}. \tag{2.29}$$

In the literature, $\mathbf{a}_b = -\mathtt{R}^{ba}\mathbf{b}_a$ is frequently referred to as $\mathbf{t}$. Here we try to be more explicit.

Pure Rotational Homographies

For two cameras which share the same optic centre — two cameras with no translation between them — the plane-induced homography is simplified. Since $\mathbf{a}_b = \mathbf{b}_a = \mathbf{0}$, the term $\mathbf{a}_b\mathbf{n}_a^\top = \mathbf{0}$. In fact, the plane-induced homography becomes independent of the plane altogether:

$$\mathtt{H}^{ba} = \mathtt{K}\mathtt{R}^{ba}\mathtt{K}^{-1}. \tag{2.30}$$

We note that for a purely rotating camera, no parallax is observed, and we can describe the pixel transformation between frames by a homography regardless of the scene being viewed.

2.3.4 Homographic Image Warp

Considering an image $\mathtt{I}$ as a continuous function, we define $\mathtt{I}(\mathbf{u})$ as a mapping from two-dimensional image coordinates $\mathbf{u} = (u, v)^\top$ to the corresponding pixel intensities within the image $\mathtt{I}$. Sub-pixel values can be taken implicitly using bilinear or bi-cubic interpolation for example.

Given a reference image $\mathtt{I}^r$, we can generate a new synthetic image $\mathtt{I}^s$ by warping $\mathtt{I}^r$ by a homography $\mathtt{H}^{sr}$. We do so functionally by describing image $\mathtt{I}^s$ through the transfer of its pixels $\mathbf{u}_s \in \mathtt{I}^s$:

$$\mathtt{I}^s(\mathbf{u}_s) = \mathtt{I}^r\left(\pi\left(\mathtt{H}^{rs}\dot{\mathbf{u}}_s\right)\right), \tag{2.31}$$

where π is the projection function which performs homogeneous division (Equation 2.1).

One scheme for actually building $\mathtt{I}^s$ is to enumerate each pixel and pass it through Equation 2.31 to find its value. This scheme can be described as backward warping since we start with a pixel in the synthetic warped image and calculate the sub-pixel coordinates of the corresponding pixel in the original image. Though inexpensive to compute, this simple warping scheme can generate visual artefacts since data is not necessarily re-sampled correctly. Consider the homography corresponding to 'zooming out', $\mathtt{H}^{rs} = \left(\begin{smallmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right)$; clearly, pixels in the synthetic image will be generated by sampling the reference image sparsely. Treating pixels as rays in this way does not accurately model the image formation process and can cause aliasing in the output warped image, though it is acceptable for many uses.

One solution to improve the quality of the warped image is to apply an appropriate filter to the source image $\mathtt{I}^r$ before use, such as a Gaussian convolution. Mipmapping is an efficient way to select which pre-filtered image to use in a precomputed image pyramid, and is often used in computer graphics [132].

2.4 Lie Groups and Their Algebra

So far we have introduced several transformations that allow us to refer to various 3D motions and 2D projections. We have largely expressed these transformations

using matrices manipulating homogeneous coordinates. Later on, when we come to look at various minimisations which hope to refine these transformations, we will be interested in incremental parameterisations which are both efficient and well behaved in various ways.

Considering for example 3D rotation, the pose of a camera c which can only rotate has three degrees of freedom (3 DOF), since a minimum of three parameters are required to define a camera's orientation. As we saw in Section 2.1, this orientation can be expressed by a 3×3 orthonormal rotation matrix $\mathtt{R}^{wc}$ with 9 elements. We will frequently use this matrix parameterisation as it offers a simple, unified way of considering different transformations, and allows us to use the standard and powerful tools of linear algebra.

When optimising over the parameters of a 3D rotation, however — to compute an angular velocity between video images for instance — the matrix form is not so appropriate. The 9 parameters of the matrix are *over-parameterised*, in that a 3D rotation only really has 3 DOF. Worse still, not all combinations of parameters will create a valid rotation matrix.

Several different parameterisations for 3D rotation exist, such as Euler angles, quaternions, normalised quaternions and axis-angle, and any of them can be converted to and from rotation matrices. In this thesis however, we follow the trend of an increasing number of authors (particularly since the work of Drummond and Cipolla [31]) who make use of Lie groups and their algebras for incremental transformations over several different spaces. In mathematics, a *group* consists of any set G and operation $\bullet$ that satisfy some simple properties:

Closure for all $a, b \in G,\ a \bullet b \in G$.

Associativity for all $a, b, c \in G,\ (a \bullet b) \bullet c = a \bullet (b \bullet c)$.

Identity there exists an element $a \in G$, such that for all $b \in G,\ a \bullet b = b \bullet a = b$. This element is written as 1_G.

Inverse for every element $a \in G$, there exists an element $b \in G$ such that $a \bullet b = b \bullet a = 1_G$.

A Lie group is any group which is also a finite dimensional smooth manifold, where the group operations of multiplication and inversion are smooth maps. A

number of transformation matrices that we have already seen form Lie groups under multiplication. 3D rotation matrices belong to the *special orthogonal* Lie group $\mathbb{SO}(3)$, 3D rigid body transformation matrices belong to the *special Euclidean* Lie group $\mathbb{SE}(3)$, and the 3×3 homographic transform $\mathtt{H}$ belongs to the *special linear* group $\mathbb{SL}(3)$.

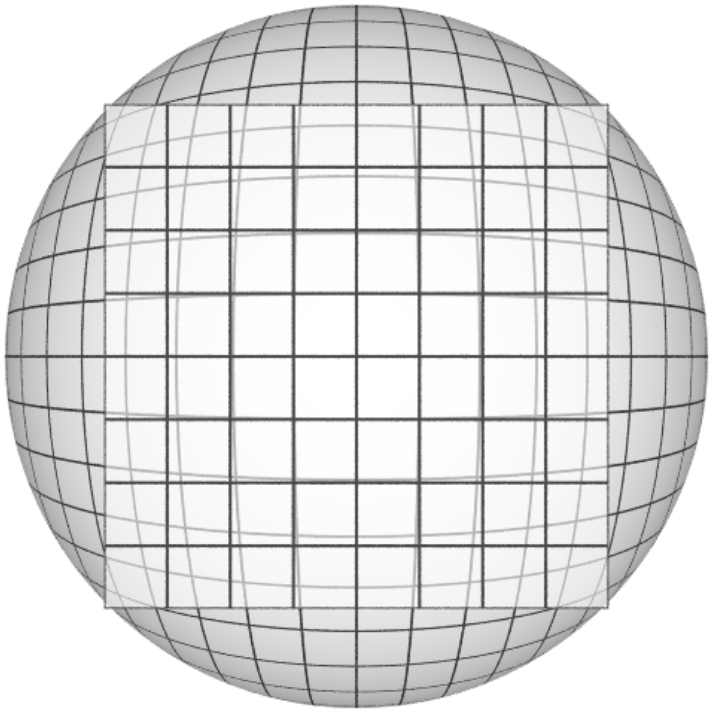


Figure 2.5: Illustrative Tangent Space (plane) projected onto manifold (sphere)

What is interesting about Lie groups for us is that each is associated with a Lie algebra, a tangential vector space around the group's identity element. Any element within the group can be *lifted* to a unique element within the algebra. Though not technically representing the lifting of a real Lie algebra, Figure 2.5 illustrates the projection of a tangent plane onto the manifold of a sphere. Similar to concepts in cartography, the tangent space represents a locally Euclidean space on the surface of the manifold, as illustrated by the approximate alignment of grid lines around the identity element where the plane and manifold touch.

In optimisation, we can parameterise incremental transformations which belong to a Lie group by their algebra, which represents a minimal and smooth differentiable linear space about the group's identity element. This parameterisation is ideal for

optimisations in which derivatives are considered, since the tangent space in which the algebra resides reflects the derivative of the group's manifold.

The transformations $\mathtt{R}^{ba} \in \mathbb{SO}(3)$, $\mathtt{T}^{ba} \in \mathbb{SE}(3)$, and $\mathtt{H}^{ba} \in \mathbb{SL}(3)$ can be parameterised by $x \in \mathbb{R}^3$ belonging to $\mathfrak{so}_3$, $x \in \mathbb{R}^6$ belonging to $\mathfrak{se}_3$, and $x \in \mathbb{R}^8$ belonging to $\mathfrak{sl}_3$ respectively. Elements of a Lie algebra are related to a Lie group via the matrix exponential map:

$$\mathtt{R}(\mathbf{x}) = \exp\left(\sum_{i=1}^{3} \mathbf{x}_i \underset{\mathbb{SO}(3)}{\mathrm{gen}_i}\right), \ \mathbf{x} \in \mathfrak{so}_3, \tag{2.32}$$

$$\mathtt{T}(\mathbf{x}) = \exp\left(\sum_{i=1}^{6} \mathbf{x}_i \underset{\mathbb{SE}(3)}{\mathrm{gen}_i}\right), \ \mathbf{x} \in \mathfrak{se}_3, \tag{2.33}$$

$$\mathtt{H}(\mathbf{x}) = \exp\left(\sum_{i=1}^{8} \mathbf{x}_i \underset{\mathbb{SL}(3)}{\mathrm{gen}_i}\right), \ \mathbf{x} \in \mathfrak{sl}_3, \tag{2.34}$$

where $\underset{G}{\mathrm{gen}_i}, i \in 1..N$ are the Lie group generators for group G.

The partial derivatives of a Lie group element with respect to its algebra about $\mathbf{0}$ are trivially formed from the group generators themselves:

$$\frac{\partial \mathtt{R}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}=\mathbf{0}} = \underset{\mathbb{SO}(3)}{\mathrm{gen}_i}, \qquad \frac{\partial \mathtt{T}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}=\mathbf{0}} = \underset{\mathbb{SE}(3)}{\mathrm{gen}_i}, \qquad \frac{\partial \mathtt{H}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}=\mathbf{0}} = \underset{\mathbb{SL}(3)}{\mathrm{gen}_i}\ . \tag{2.35}$$

For a list of Lie group generators commonly found in computer vision and used within this thesis, please refer to Appendix A.

2.5 OpenGL for Vision

An under-appreciated complexity in computer vision research is in the visualisation of multi-view stereo data. Increasingly, and certainly within this thesis, the power of hardware designed to accelerate computer graphics is also directly used for computer vision. Computation of generative photometric models, for example, can be accelerated many-fold using commodity hardware found in nearly every PC. As the sophistication of computer graphics has increased, so too has graphics hardware, offering general purpose programmability that even enables data-parallel computation

unrelated to graphics to be performed many times faster than on a modern desktop CPU. Equally, the performance of massively parallel architectures such as that of graphics processors is increasing at a much faster rate than of serial processors. It becomes clear that computation on massively parallel architectures is not a fad and will only grow with time; in order to take advantage of advances in computer hardware, computer scientists will increasingly need to exploit parallel algorithms.

Within this section we will outline some useful computer graphics topics and describe how they relate to concepts in computer vision. Our aim is to unify notation and terminology. We target OpenGL since it is a modern and certainly the most prevalent graphics library available. If you are new to OpenGL, this section should be read in conjunction with an OpenGL code tutorial.

2.5.1 The OpenGL Rendering Pipeline

OpenGL is a *rasterisation* engine. Instead of considering the path of photons between light sources and the camera as a ray tracer might, a rasteriser acts by processing simple geometric primitives which make up the scene, such as points, lines, triangles and polygons to determine how they would project in to the virtual camera to form pixels. We call the image plane of the virtual camera into which we form the rendered image the *framebuffer*. In addition to a colour image, we typically also associate a depth buffer of equal size to the framebuffer to hold the z-depth at each pixel. Surfaces can be rendered by rasterising a tessellation of small piecewise planar elements, often triangles, which approximate the surface.

Geometric primitives in OpenGL are defined by their *vertices*. How a primitive is rendered is decided by properties set at each vertex. Within the fixed OpenGL pipeline, these properties are interpolated across *fragments* belonging to the primitive to determine how they will be rendered. A fragment represents a would-be pixel or pixel contribution belonging to a single primitive, a square element that aligns exactly with a pixel in the final rendered image but may not fully describe it. To understand this differentiation, we must consider what happens when more than one geometric primitive projects to the same area in the framebuffer. Each primitive is processed sequentially into a collection of fragments by the rasteriser. A fragment contributes to a pixel only if it passes some tests. Typically, it will pass if the depth of the fragment is closer to the camera than the value stored in the

depth buffer. When a fragment passes, not only will the pixel be updated with the fragment's colour (or potentially a blend of colour), but the depth buffer will also be updated with the fragment's z-depth. This process allows primitives to be processed in any order and is called the z-buffer algorithm, ensuring that hidden surfaces do not occlude those that should be visible. Alternative methods such as the painters algorithm require that primitives are ordered from far to near, which is costly and not always possible.

When rendering surfaces, three important properties that can be set at vertices are the colour of the surface at the vertex, the normal of the surface at the vertex, and the coordinates within a texture (an image stored on the graphics card) that the vertex projects into. The latter two properties are optional. Within the fixed OpenGL pipeline, these properties are interpolated across the fragments that make up the primitive. A user may also specify active lighting and assign an active texture whilst rendering a primitive. The combination of active lights, active textures, fragment colour, fragment normal, and fragment texture coordinates determine the final colour of a fragment. When texture mapping is enabled, the pixel within the active texture described by a fragments texture coordinate is used to modulate the colour of the fragment.

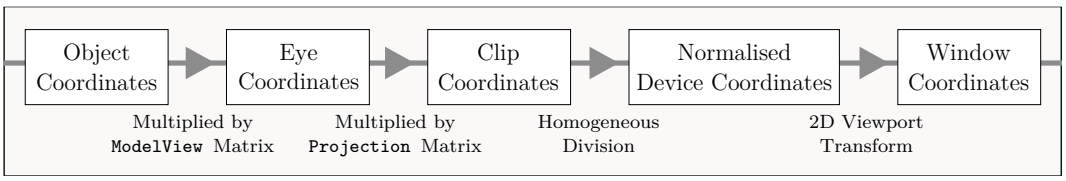


Figure 2.6: Illustration of how OpenGL's fixed vertex pipeline transforms vertices for projection within an OpenGL window.

Figure 2.6 introduces the traditional, fixed vertex pipeline of OpenGL; we can see it mirrors the basic principles of camera projection that we have already established. OpenGL accepts from the user geometric primitives such as lines and triangles defined by their vertices in object coordinates o, an arbitrary frame of reference relative to the object being defined. The object coordinates are transformed into camera coordinates c (Figure 2.1) via the `ModelView` matrix, a 4×4 transformation matrix $\mathtt{T}^{co} \in \mathbb{SL}(3)$. The object coordinate system can be changed between vertices by changing the `ModelView` matrix. It is convenient to consider the `ModelView` matrix as a combination of a model matrix $\mathtt{T}^{wo}$ and view matrix $\mathtt{T}^{cw}$ such that $\mathtt{T}^{co} = \mathtt{T}^{cw}\mathtt{T}^{wo}$.

In this way we can render views of an object from different cameras situated in the same global frame of reference by keeping vertex data the same but changing the view matrix (and hence OpenGL's `ModelView` matrix).

From the camera-centred coordinate system c (Eye Coordinates), primitives are further transformed into Clip Coordinates via the `Projection` matrix, $\mathtt{K}_{gl} \in \mathbb{R}^{4\times4}$. This matrix is similar in spirit to the calibration matrix from computer vision, but where the depth of a projected vertex is additionally offset and scaled so that it can be represented during rasterisation within the depth buffer. In this coordinate system, vertices that fall outside of a unit cube are discarded (clipped), as they represent unobservable geometry outside of the camera's frustum, lie closer than a near clipping plane, or lie further than a far clipping plane.

Defining n to be the z-axis value of the near clipping plane in camera coordinates, and left (l), right (r), bottom (b) and top (t) to be its x and y extent, $\mathtt{K}_{gl}$ can be specified in terms of these and the far clipping plane z-axis value (f) [3]:

$$\mathtt{K}_{gl} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}. \tag{2.36}$$

Within computer vision, we frequently want to relate the OpenGL camera to a physical device with known intrinsics. Although non-linear lens effects cannot be set up directly within the OpenGL fixed rendering pipeline, we can write down the parameters of $\mathtt{K}_{gl}$ in terms of our regular linear camera's intrinsic parameters:

$$l = u_0 \frac{n}{f_u}, \qquad r = (u_0 - w)\frac{n}{f_u}, \qquad t = v_0 \frac{n}{f_v}, \qquad b = (v_0 - h)\frac{n}{f_v}, \tag{2.37}$$

where u_0, v_0, f_u and f_v are standard intrinsic parameters (Section 2.2.1) and w, h represent the images width and height in pixels respectively. The near and far clipping planes, n and f, do not directly alter the projection, only which vertices get discarded and how depths are quantised within the depth buffer when primitives are rasterised.

With this information, we are ready to render images that mimic a real calibrated camera. Code listing 2.1 demonstrates the main steps in rendering a simple triangle

Listing 2.1: Using OpenGL to Display a 3D triangle patch

```
...
// Specify column-major OpenGL projection matrix
float Kgl[16] = {...};

// Specify column-major world-to-camera transform
float T_cw[16] = {...};

// Load Projection and ModelView matrices
glMatrixMode(GL_PROJECTION);
glLoadMatrixf(Kgl);
glMatrixMode(GL_MODELVIEW);
glLoadMatrixf(T_cw);

// Clear Framebuffer's colour and depth values
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// Draw triangle by specifying its vertices
// in world coordinates
glBegin(GL_TRIANGLES);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
glEnd();
...
```

defined in world coordinates into a camera c. `Kgl` and `T_cw` represent the 4×4 matrices $\mathtt{K}_{gl}$ and $\mathtt{T}^{cw}$ respectively, laid out in column-major order. Refer to an OpenGL code tutorial for information on how to set up an OpenGL window and viewport.

2.5.2 Projective Textures

Primitives such as triangles are texture mapped by defining texture coordinates at each vertex. The process of defining these coordinates is sometimes referred to as UV-mapping, relating as they do geometry to surface texture. As we discussed briefly early, per-vertex properties including texture coordinates are interpolated across fragments during rasterisation in order to determine contributions from texture images.

Given a photo of a simple structure such as a cuboid skyscraper and a 3D triangular model of it, we might like to texture the sides of the model visible in the

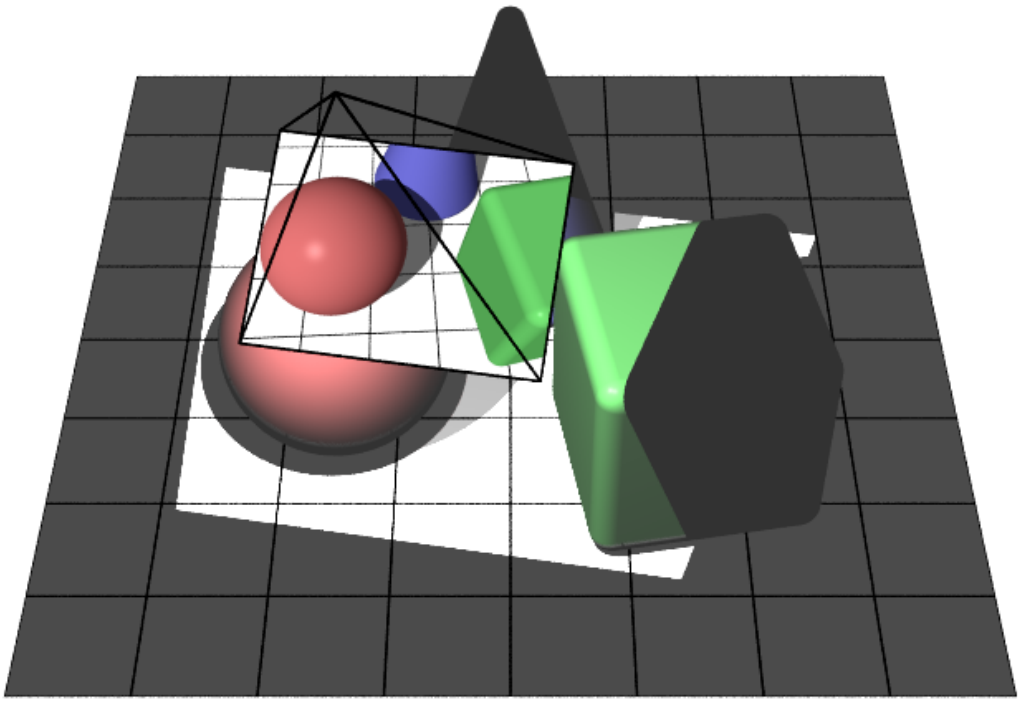


Figure 2.7: Projective texturing applied to a synthetic scene. Colour is given to a colourless world via projection.

photo. The texture coordinates of the cuboid's vertices can easily be established by reading the UV coordinates of the projections of the building's corners in the photo. Unfortunately, this isn't enough to correctly map the photo's texture to the cuboid. OpenGL operates by default under the assumption that textures represent orthographic projections of the model's surface. Although during rasterisation, interpolation of texture coordinates is performed projectively relative to the virtual camera, the projective action of the camera that made the texture must also be considered. This subtlety is commonly overlooked.

Projective texturing is the term given to texturing a geometric model from an image by considering the projective warp of coordinates in the texture map as it is applied; this is incredibly useful for visualising computational vision. Its use within OpenGL is enabled by specifying texture coordinates in a higher dimensional homogeneous space, allowing interpolation across fragments to be performed correctly. Projective texturing is described in detail by Everitt who also demonstrates the

problems of incorrect texture mapping [36]. Figure 2.7 illustrates the power of projective texturing; geometry of an accurately aligned projector casting colour onto a colourless world. It quite naturally has the same projective geometry as a camera capturing the same image.

The simplest way to set up projective texturing for a projector p with intrinsics $\mathtt{K}_{gl}$ is to tell OpenGL to automatically assign to each vertex texture coordinates that equal the camera frame coordinates of that vertex. Parameters passed to `glTexGen` can be used to define a matrix describing how these coordinates should be transformed into the given texture. This matrix can be used much like a combination of the `ModelView` and `Projection` matrices, and we place into it the object-to-texture transformation $\mathtt{T}^{to}$:

$$\mathtt{T}^{to} = \begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathtt{K}_{gl}\mathtt{T}^{po} \tag{2.38}$$

where $\mathtt{T}^{po}$ is the object-to-projector coordinate transformation. The constant matrix at the front takes normalised device coordinates into standard texture coordinates. One tricky point is that OpenGL will internally multiply this matrix by the inverse `ModelView` matrix itself in order to compute the transform $\mathtt{T}^{tc} = \mathtt{T}^{to}\,(\mathtt{T}^{co})^{-1}$. This is needed given that camera frame coordinates will be assigned as texture coordinates; it removes the need to invert the `ModelView` matrix.

Listing 2.2 illustrates how projective texturing with automatic texture coordinate generation can be initialised, where `T_to` represents the matrix $\mathtt{T}^{to}$ in row-major ordering. First, automatic texture coordinate generation for the ordinates `GL_S`, `GL_T`, `GL_R` and `GL_Q` are set to `GL_EYE_LINEAR`, telling OpenGL to apply texture coordinates based on the camera frame position of a vertex. Since OpenGL texture coordinate generation is quite flexible, setting the texture transformation is slightly obfuscated, but essentially $\mathtt{T}^{to}$ is loaded row-by-row as vectors for each ordinates `GL_EYE_PLANE`. Finally, texture generation is enabled before primitives are rendered — it should be disabled again after we are done.

Listing 2.2: Setting up Projective Texturing

```
...

// Define camera-to-texture Transformation matrix
float T_to[16] = {...};

// Automatically generate texture coordinates that equal
// a vertices position in camera (eye) coordinates.
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);

// Set Texture Transformation matrix
glTexGenfv(GL_S, GL_EYE_PLANE, T_to);
glTexGenfv(GL_T, GL_EYE_PLANE, T_to + 4);
glTexGenfv(GL_R, GL_EYE_PLANE, T_to + 8);
glTexGenfv(GL_Q, GL_EYE_PLANE, T_to + 12);

// Enable texturing and automatic texture coordinates
glEnable(GL_TEXTURE_2D);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);
glEnable(GL_TEXTURE_GEN_Q);

// Render geometry
...
```

2.5.3 Buffer Objects and Framebuffers

When rendering large quantities of primitives on the graphics card, significant gains in throughput can be achieved by uploading vertex, pixel, normal and other data as a contiguous block of memory, called buffer objects, rather than using the classic API for drawing elements with `glBegin` and `glEnd` which might be too slow.

In Chapter 6 we use both *vertex buffer objects* (VBOs) and *pixel buffer objects* (PBOs) in order to efficiently display large numbers of textured depth maps. Using languages such as CUDA, buffer objects can be modified, manipulated and processed, all whilst they are still in graphics memory, without ever having to transfer them between the host and graphics card.

Figure 2.8 illustrates how we can represent a scene by storing a depth image within a VBO, associated with an historic camera position. The stored structure resides in

memory on the graphics card and can be efficiently rendered without transferring geometric data between host and device. Colour and normal data can also be stored in buffer objects and associated with vertex data during rendering.

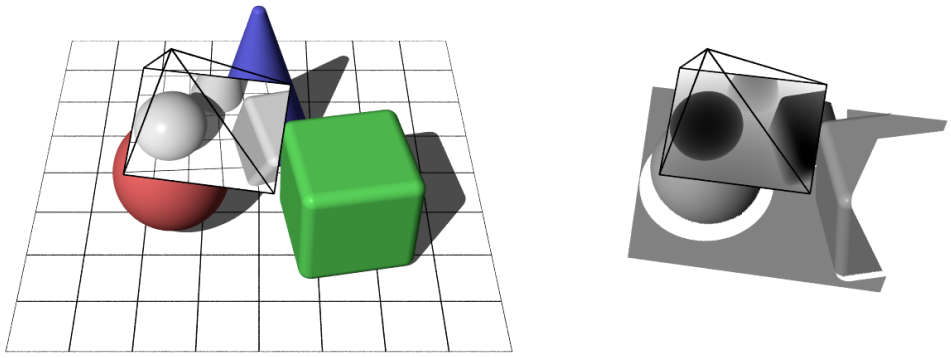


Figure 2.8: Vertex Buffer Objects can be used to efficiently store and render depth maps to represent surface geometry. Left: a camera records the projective image of a scene onto its image plane. Right: Depth data associated with each pixel can be used to render visible portions of the same scene.

Whereas buffer objects represent source data used for rendering, in OpenGL the *framebuffer* represents the target portions of memory to render in to. By default, the framebuffer is set up to output to the screen, often on some kind of display window. OpenGL also allows you to create your own off-screen framebuffers in order to render graphics for purposes other than display, or perhaps to be used for later rendering. We won't detail how to set up buffer objects and framebuffers as they are well documented elsewhere (see for example songho.ca [3]), but they are both enabling technologies for systems described later in this thesis.

2.5.4 Shaders

Shaders offer a programmatic means of adjusting per-vertex and per-fragment properties within the graphics rendering pipeline. They were introduced to enhance the previous fixed pipeline of vertex transformation followed by primitive rasterisation. Shaders enable the graphics pipeline to be customised in order to perform advanced rendering techniques or to produce special surface effects or interesting material properties. We will use shaders later on in order to efficiently pass information through the rendering pipeline.

Shaders are written using a shader language and are generally compiled from source at runtime to target the running graphics hardware. There are three types; vertex shaders, fragment shaders and geometry shaders, though we will only discuss the first two.

Vertex shaders can manipulate the attributes of vertices including position, normal, colour and texture coordinates through code by taking vertex attributes and external parameters as input.

Pixel-wise *fragment shaders* can modify fragments. *Fragments* are the elements of rasterisation which are accumulated to form pixels; they are composed with existing image pixels based on depth and other tests such as stencils, and blended based on OpenGL attributes. A fragment shader may modify the colour of a fragment, including its transparency, by transforming properties that are inherited by the fragment and by performing texture lookups. A fragment shader may even discard the fragment all together, preventing the possibility that it will contribute to a pixel value.

Vertex Map Example

In computer vision, if we have some model of the scene represented by primitives within the OpenGL framework, we can very efficiently synthesise a novel view of the scene by rendering it into a virtual camera with parameters mirroring that of our real camera.

There may be times when it is useful to project not only colour information but also geometric information per pixel, such as scene depth. An image comprised of 3D scene points corresponding to the location that a pixel intersects a scene is termed a *vertex map*. We gave an example of a vertex map stored inside of a VBO earlier (Figure 2.8). Using shaders we can efficiently create a vertex map to represent a scene that we render.

In order to render a vertex map and colour image simultaneously, we must define a vertex shader and a fragment shader which operate on two *colour attachments* within our *framebuffer* — in other words our framebuffer will contain two images that we can render in just one pass using some shaders. We choose to generate a vertex map this way rather than extracting the depth buffer data because it allows

Listing 2.3: Cg Vertex Map Shaders

```
// Vertex Map - Vertex Shader
void vVertexMap(
  float4 in_color        : COLOR,
  float4 in_pos          : POSITION,
  out float4 out_color   : COLOR,
  out float4 out_proj    : POSITION,
  out float4 out_pos_cam : TEXCOORD0,
  uniform float4x4 mvpm  : state.matrix.mvp,
  uniform float4x4 mvm   : state.matrix.modelview
){
  // Vertex Color
  out_color = in_color;

  // Vertex projection
  out_proj = mul(mvpm,in_pos);

  // Vertex coordinates in camera frame
  out_pos_cam = mul(mvm, in_pos);
}

// Vertex Map - Fragment Shader
void fVertexMap(
  float4 in_color        : COLOR,
  float4 in_pos_cam      : TEXCOORD0,
  out float4 out_color0  : COLOR0,
  out float4 out_color1  : COLOR1
){
  // COLOR0 contains regular scene rendering
  out_color0 = in_color;

  // COLOR1 contains colour coded vertex positions
  out_color1 = in_pos_cam;
}
```

us to have more control over how the data is processed, and makes it easier to work with later.

Listing 2.3 gives an example of a vertex and fragment shader written using the shader language Cg. When activated together, the two shaders will cause OpenGL drawing operations to output an ordinary image and a vertex map simultaneously within a framebuffer with two attached colour channels.

Cg is a C styled language with a few additions; the code listings demonstrate how parameters are annotated with *semantics*, coloured red. These bind named input and output variables to predefined quantities available to the shader at runtime,

such as input colour and texture coordinates that have been applied. Parameters labelled as *out* represent properties that the shader will set, whilst those labelled *uniform* receive their value from the main CPU program.

In our example, the vertex shader outputs the vertex colour unchanged and transforms the vertex position through the `ModelView` and `Projection` matrices (combined within `state.matrix.mvp`) as the fixed pipeline would do. In addition, the shader also assigns to the texture coordinate property the value for the vertex position, transformed into the frame of reference of the camera. In the fragment shader, the properties for a fragment (produced by interpolating vertex properties) are used to set the colour for two colour attachments, `COLOR0` and `COLOR1`. Without the shaders, only the colour data would reach the default colour attachment `COLOR0`. More sophisticated logic and filtering can be included within shaders — what makes them useful is that they sit in the middle of he OpenGL pipeline and are massively data parallelisable.

2.6 Software

This thesis details and develops a number of software systems, the requirements of which have changed throughout the research. Starting in Chapter 3, a largely point-feature centric approach motivated the use of Davison's SceneLib for visual SLAM. Foundation methods for video input and image processing have been largely enabled by the high quality and open source library libCVD, supported by the maths library TooN.

As we have moved toward dense approaches starting in Chapter 4, it has been more important to utilise graphics hardware to achieve real-time performance. In early work, this was enabled largely through the (mis)use of OpenGL shaders, but the emergence of CUDA for general purpose programming on graphics hardware has really opened the way to make best use of the cards that lie in many PC's.

Data parallel programming in CUDA really is quite natural for many image processing tasks, and we found it largely replaced the use of image processing libraries such as libCVD. We found that it was desirable to avoid copying data between host and device and that it was easier to keep and process all images entirely on graphics hardware.

The only common software library used throughout has been OpenGL itself for display. We have built on top of OpenGL to produce the MIT licensed open source software library Pangolin. It is a light weight rapid prototyping utility library for vision researchers to help visualise data (through OpenGL) and receive input (through keyboard / mouse / camera) easily and efficiently.

Finally, work in Chapter 6 would not have been possible without the open source software PTAM described by Klein and Murray [62]. It was used not only in comparisons with our system, but helped a great deal in enabling us to develop it.

2.7 Summary

In this chapter we have presented the mathematical tools and geometrical models behind the main methods explored in the thesis, and also given details on the important role that graphics processing hardware and the OpenGL API can provide in real-time computer vision systems aiming at dense SLAM. We will start using all of these methods in earnest in Chapter 4, but first in Chapter 3 present some preliminary work on dense reconstruction in general and how the feature-based visual SLAM systems available at the start of this project can be augmented to produce surface meshes.

Chapter 3

Augmenting Feature-Based SLAM for Live Modelling

3.1 Introduction

By tracking the 2D locations of salient image patches across many frames of video, both the 3D positions of these point features and the motion of the camera itself can be estimated. For drift-free camera tracking, feature-based monocular SLAM systems maintain the 3D locations of such a set of features as landmarks within a map, maintained and updated as the camera moves.

Each consistent feature point within a visual SLAM map represents a 3D point on the surface of an object. Since only salient image patches are tracked through video in current visual SLAM systems, only a sparse set of 3D point features is maintained in the map representing the world. Assuming that the point features which exist in the map have been well estimated and are not erroneous, we know that the true scene's surface must pass through these points.

In this chapter, we will look at how real-time 3D modelling of static scenes might be enabled by using a visual SLAM system's map of sparse features as a skeleton for estimating the dense surface geometry of the world. Pan *et al.*'s system ProFORMA described in [100] has much in common with the methods described in this chapter and was independently developed during the same period as our own work. Pro-

FORMA is a more fully realised system than that which we present and validates several of the concepts explored in this chapter, such as the use of tetrahedralisation.

Chapter 6 describes a different system with the same goals as this one, building from developments that became available later on in the period of this thesis, such as commodity programmable graphics hardware and the increased accessibility of real-time dense multi-view stereo.

3.2 Background

In multi-view stereo (MVS), multiple images taken from cameras with known pose and internal parameters are considered in order to generate a 3D model of an unconstrained scene. State of the art MVS methods such as that of Furukawa and Ponce are able to produce highly detailed photo realistic reconstructions [41]. They approximate the surface of the scene by overlapping oriented planar patches within an incremental method that expands currently reconstructed regions through photo-consistent additions at the models boundary. Offline visual reconstruction techniques such as this have been the focus of much research, but typically have taken minutes if not hours to complete for modest numbers of images. Attempting to construct such models in real-time has yet to receive much attention.

In order to compute the pose of cameras used as input to multi-view reconstruction systems, sparse features are typically extracted from images, matched and then bundle adjusted as a batch process in order to reduce the reprojection error of 3D features into observed images. In order to perform real-time 3D reconstruction, an obvious starting position is to replace offline bundle adjustment with a real-time visual SLAM system that operates on sparse point features, which is the route that we will take within this chapter.

It is challenging to move from a map of sparse point features towards a mesh or other representation which densely represents geometry. For estimating the live pose of the camera, we used Davison's sequential probabilistic sparse feature visual SLAM system, MonoSLAM [26]. It is quite conservative when choosing which features it should use to track, which can make the map very sparse. A further difficulty in constructing a surface from such a probabilistic SLAM map is that it can change over time as observations of points are made, each correlated to the cameras uncertain

location.

Although methods such as Kazhdan *et al.*'s popular Poisson Surface Reconstruction can generate surfaces through dense 3D point features [58], there remain significant and unresolved challenges when the density of features is too low, as in a sparse map. Whereas on the very local scale we might expect surfaces to obey notions of smoothness related to curvature, this doesn't hold on the coarser scale. Neighbouring sparse features within a map may be meters apart, and the resulting topology of the implied surface may be highly ambiguous.

For maps whose real surface topology is very constrained, a simple approach can be to carefully choose a lower two-dimensional space upon which we can project our 3D point features. We can then triangulate features in 2D within this space (using a Delaunay triangulation for example) to obtain connectivity information for our final mesh back in 3D. This will generate meshes who's topology is correct only if we can find a 2D space on to which the projection of all real-world surfaces have no overlap.

For scenes in which all features remain unoccluded from a single known location $\mathbf{P}$, features can be projected onto a unit sphere centred at $\mathbf{P}$ and triangulated in 2D by simply consider their 2D spherical coordinates. Even for such constrained surfaces, the generated mesh loses its desirable Delaunay property — adequate weighting is not given in the direction of the projection which can result in skinny triangles. For simple scenes and by choosing an appropriate space in which to project, acceptable results can be obtained.

A straight-forward example of this is the work of Beardsley *et al.* who choose to triangulate in the lower dimensional space of the image itself (a projection on to the image plane) [9]. They ensure topology is maintained correctly by including only point features which are currently observed within the live image.

For scenes belonging loosely to a plane, such as one consisting of a single side of a corridor, there may not be a single finite point from which all features are visible, but instead one set at infinity in some direction. In this case we can project points orthographicaly onto the plane for triangulation. This formulation is common to many aerial ground mosaicing systems such as that of Jung and Lacroix [57].

The more common and general case is that not all features in a scene are visible

from a single point, and that we are unable to simplify the problem so straightforwardly. Additionally, we would like to avoid committing to finding such a point or to placing one that may later be invalidated by an addition to our model. The main part of this chapter will therefore be concerned with visibility reasoning for general 3D scenes and the development of a tetrahedral meshing approach which can automatically infer the locations of surfaces in the world as the interface between solid and empty regions.

3.3 Minimum Energy Surfaces

Given a sparse set of 3D features lying on the surface of unknown scene geometry, can we generate an approximate model of the surface from the sparse features alone? To answer this question, let us first consider a lower-dimensional version of the problem: that of fitting a boundary over sparse points to form a two-dimensional plan consistent with the world. Figure 3.1 illustrates a minimum energy (shortest length through all features) approach to fitting an approximate surface through sparse features which lie on the true boundary. We can see that the boundary misrepresents the true surface of the real-world in a catastrophic way, placing the doorway in the wrong place. The result is a model that is not a useful approximation of the world.

In this simple example, the minimum energy formulation breaks down because over sparse features there exists no real notion of smoothness. It is not the case that nearby sparse point features necessarily lie on the same surface. Moreover, a minimal energy formulation as suggested can only represent a single continuous surface. To separate distinct objects we must first cluster points into separate sets which contain them. However, there are further cues available than we have used here, since a visual SLAM system produces not just a raw point cloud but also visibility evidence which can help to improve reconstruction.

3.4 Visibility

In the interpretation of surfaces, boundaries are important and are frequently detectable based on what they occlude rather than by their texture. One way in

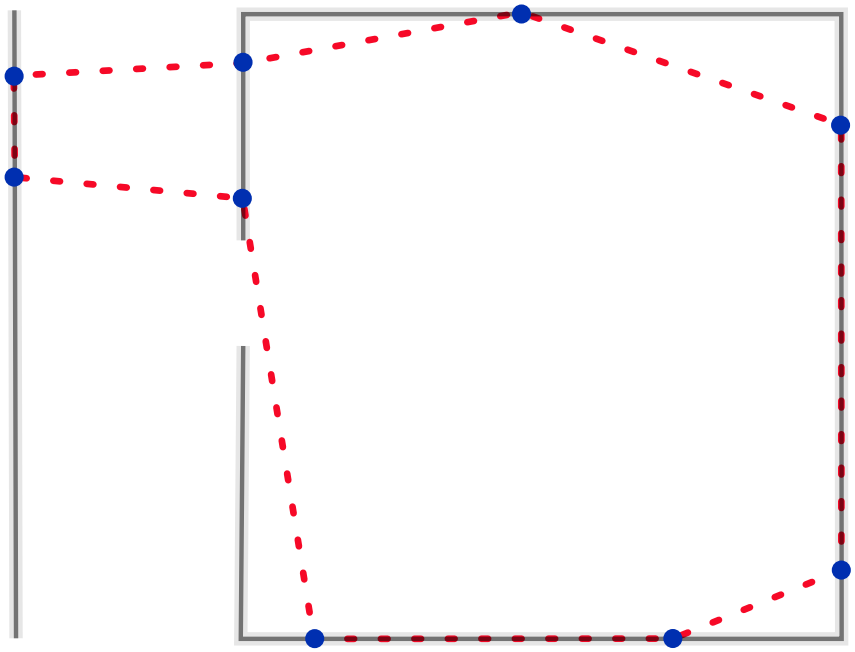


Figure 3.1: An overhead plan of a simple room and corridor connected by an open doorway. A minimum energy boundary (red dotted) is estimated through detected feature points (blue circles).

which we reason about the structure of the world is by considering what we can and cannot see as we move through it. We can do something analogous by placing the sparse features in context, recording from what position they have been observed in addition to where we estimate them to be.

In the context of a real-time SLAM system, these observations of features are readily available. The camera's pose when observing a feature provides context which can help to disambiguate potential surface structure. Specifically, we do not expect a feature that we are observing to be occluded by another surface. Figure 3.2 applies context to our two-dimensional boundary generation example. Here, the two features observed by our system in the corridor are grounded in the context of the current camera location. We can reason that since the features are in view, there cannot be a surface in front of them.

Using this simple notion of context, as the camera navigates the room the model's surface can be pushed in and out as features are observed. Continuing with our two-dimensional analogy from Figure 3.2, imagine that the camera were to travel through the doorway and into the corridor (Figure 3.3). If we continue to use contextual

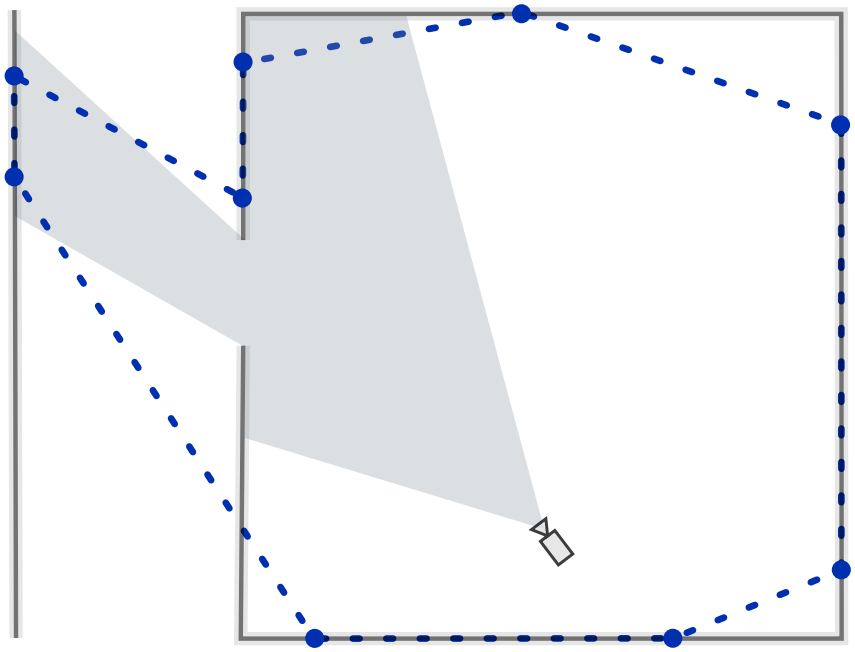


Figure 3.2: Using Context to Infer Correct Surface Topology

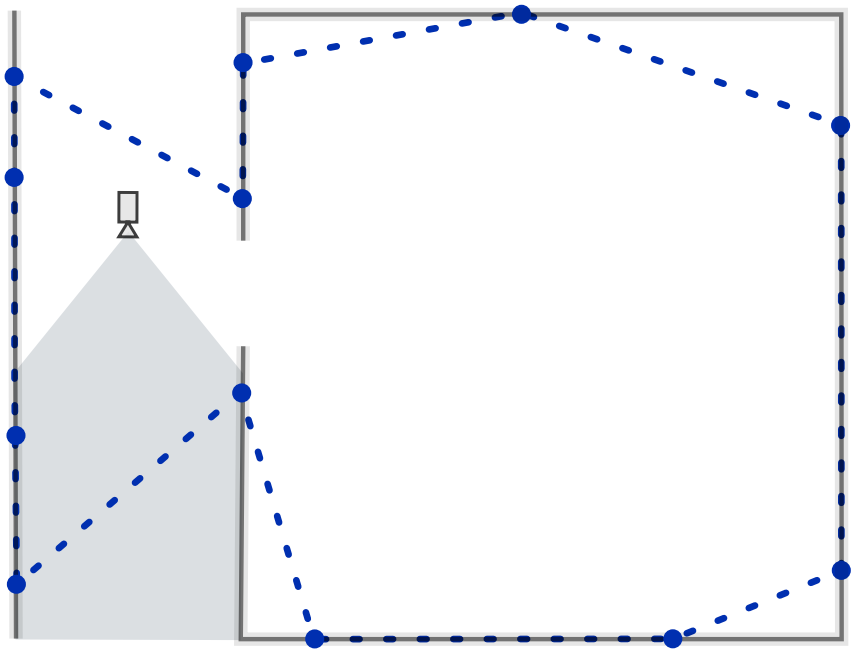


Figure 3.3: Using Context to Explore Complex Geometry

visibility to make surface updates, a much better result is obtained. Changes to map topology occur locally within the space that is currently in view.

3.5 Using Visibility to Define Volumes

3.5.1 Visibility Constraints

As a camera moves through a scene, each feature that is observed places a constraint on free space; we can say explicitly that the line segment between the camera and the feature is not occluded and passes through no surfaces. Free space constraints have been used before in robotics where range sensor measurements are integrated into evidence grids, a regular metric grid covering the area in which the robot moves and whose elements maintain the likelihood of them being free space based on observations [80]. In multi-view stereo, voxel-based volumetric reconstructions have been used for fusing multiple depth maps by equivalent free space reasoning [103]. In these cases however, the measurements have been dense — carving out volumes of space with each one. For our sparse constraints, it would be hard to pick an appropriate regular grid in which to accumulate constraint statistics. Moreover, grid-based methods are inherently inflexible and cannot be easily updated as corrections are made within a sparse feature-based map.

Figure 3.4 takes us again to two dimensions, where a single camera observes features whilst travelling through a room. The visibility constraints represent lines which we know to be free space.

In a dynamic system where features' position estimates may update and improve, such constraints are still valid, since the meaning assigned to the constraint still holds, i.e. that a feature was visible from a certain camera location. We should note that as estimates of the locations of features improve, the main component of change will be along the visibility constraint line itself.

Although we gather more visibility constraints every frame, the information that we gain from including them depends on a number of factors. We will discuss pruning of these constraints later (Section 3.5.5).

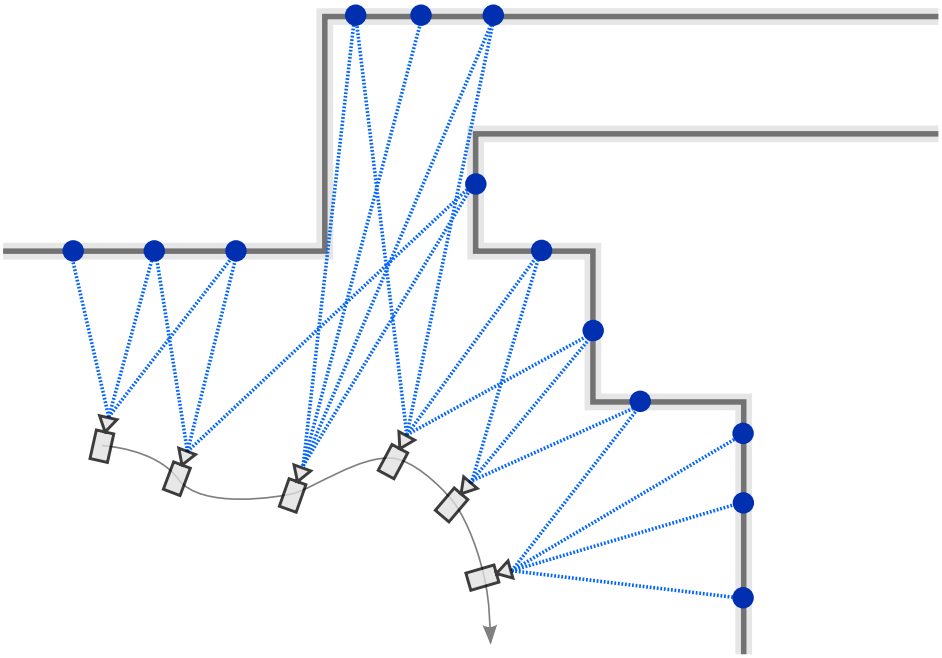


Figure 3.4: A camera moves through a scene observing point features (blue circles) which reside on surfaces within the world. For each feature observed from a camera, the line segment between the feature and camera marks free space (dotted blue line).

3.5.2 Visibility Volumes

Looking at Figure 3.4, we can see that the hard constraints introduced by this concept create a cross-shaded region, which reflects an estimate of free-space in the world (Figure 3.5). The boundary of this free-space region is contained within the set of possible surfaces defined by the triangulation of the feature points and cameras' centres. Notice that those triangles shaded as free space correspond to those intersected by visibility constraints.

Given a monocular SLAM system's sparse feature map, we can partition space into a parameterised tetrahedralisation by taking the features and a set of saved camera locations as vertices within a three-dimensional Delaunay tetrahedralisation. We label each of the component tetrahedra as solid or free space based on whether one or more visibility constraints intersect its volume. In three dimensions, the various visibility constraints will not necessarily touch, but the set of constraints is dense enough to eliminate those tetrahedra which belong to free space in the world (Figure 3.6). The estimated surface runs along faces which lie between free and

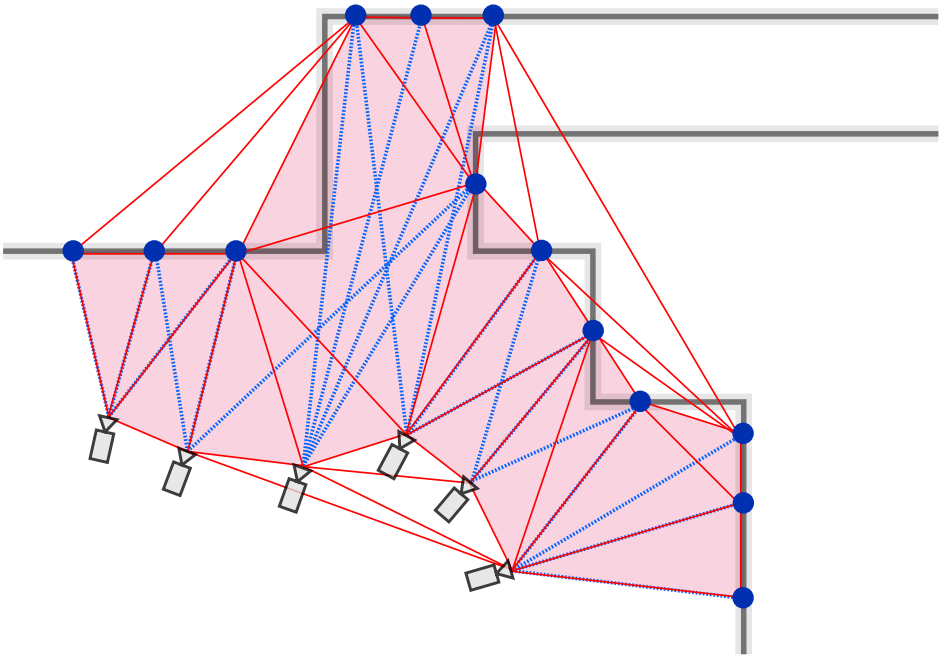


Figure 3.5: The red shaded region described by the crossing visibility constraints (blue dotted lines) forms a coarse approximation to free space within the scene.

occupied tetrahedra.

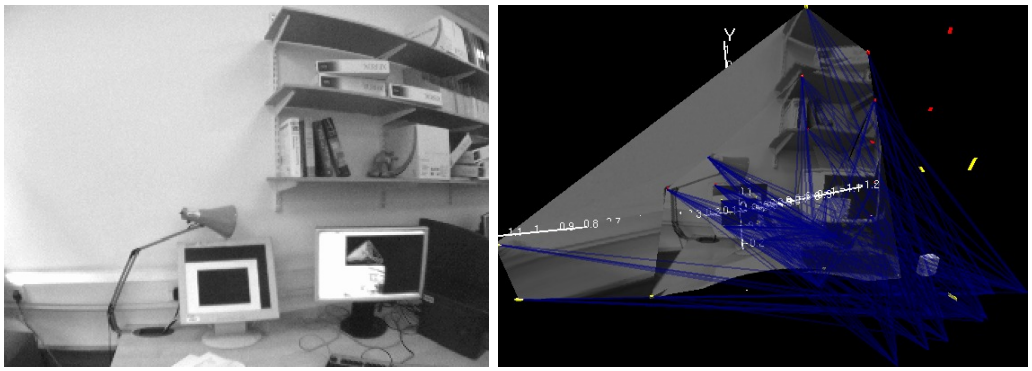


Figure 3.6: Left: sample frame from video. Right: visibility constraints (blue) for a real scene.

We include historic camera locations as vertices within the tetrahedralisation since we are modelling the volume representing free space, not directly that of the object. We pose the problem this way to enable us to model any scene directly, including planar expanses which will cause problems for tetrahedralisation based only on the point features themselves. This is in contrast to ProFORMA [100] which is only

demonstrated on closed objects. For thin volumes, the Delauney property of the tetrahedralisation is meaningless, generating poorly formed primitives.

3.5.3 Occupancy Cost

The hard visibility constraints so far described are binary criteria, but the tetrahedral regions we are reasoning about are potentially quite large and our constraints are only represented by rays. Moreover, the locations of the rays contain uncertainty and they may only narrowly intersects a tetrahedron within the volume.

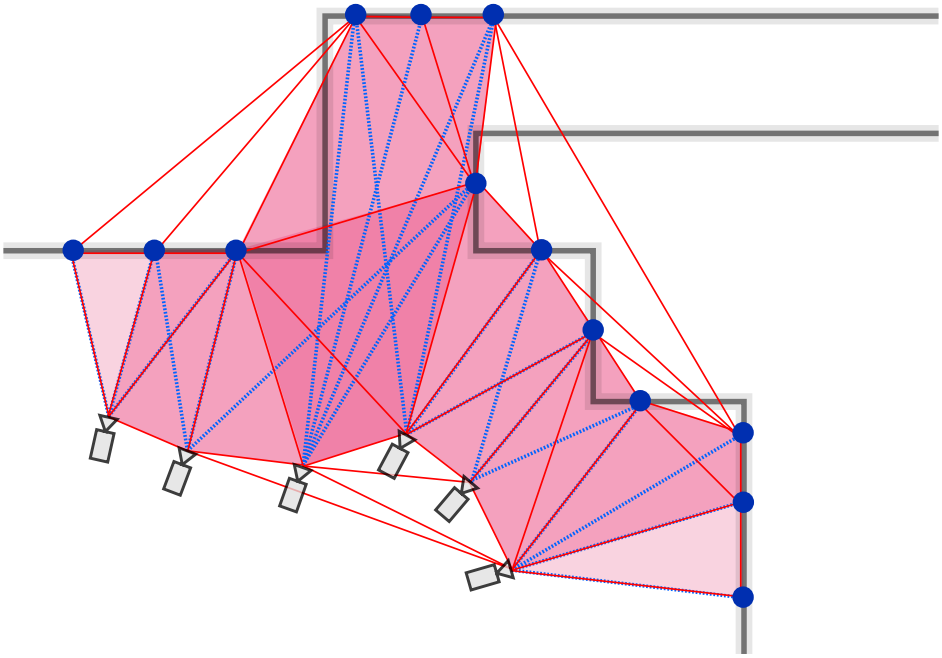


Figure 3.7: By recording the length and number of rays that pass through the volume, we can assign a cost of occupancy to a cell which increases as more constraints intersect it. Deeper red reflects higher cost, and thus a cell which is more likely to represent free space.

Given a tetrahedron in our space and a visibility constraint that passes through it, we can assign an occupancy cost (in principle related directly to occupancy probability) based on a number of potential metrics. Some important factors are the number of visibility constraints that violate the volume, their certainty, and the length of the intersecting segment, as well as the size of the volume itself (Figure 3.7).

We choose to construct a simple occupancy cost based on the total length of

visibility constraints that pass through a tetrahedra normalised by its volume. We define a line segment l as a tuple described by its endpoints; $l = (P_1, P_2),\ P_1, P_2 \in \mathbb{R}^3$. A tetrahedron is defined by a set of four vertices, $ABCD$ where $A, B, C, D \in \mathbb{R}^3$. We define $ABCD \cap l$ to denote the intersection of tetrahedra $ABCD$ with line segment l, resulting in a new shorter line segment or $\emptyset$, the empty set. $ABCD \cap l$ can be computed by successively chopping the interval defined by the segment with four half spaces corresponding to the tetrahedron faces. We define the norm of a line segment to be equal to its length, $|l| = |A - B|,\ l = (A, B)$. We define V to be the the set of all visibility constraints, as defined by their respective line segments. Finally, we can define the occupancy cost O for tetrahedron $ABCD$ as:

$$O(ABCD) = \sum_{l \in V} \frac{|ABCD \cap l|}{TV(ABCD)}, \tag{3.1}$$

where $TV\big(ABCD\big)$ is the signed volume of a tetrahedron, as defined by the scalar triple product:

$$TV\big(ABCD\big) = \Big((B - A) \times (C - A)\Big) \bullet (D - A) \tag{3.2}$$

From this definition, each tetrahedron can be assigned an occupancy cost — those which have been intersected by zero constraints will have zero cost and represent solid space. A free/empty labelling can be assigned to each tetrahedron based on a threshold of this value, or as a ratio of the highest cost tetrahedron within the volume.

3.5.4 Updating Geometry and Occupancy Scores

Since the vertices of the tetrahedralisation are formed from probabilistic point features, it is quite possible that they will move or get deleted, and new points will be added. Although tetrahedralising under a 100 sparse feature points will be quite fast, we do not want surface structure to have the appearance that it is always changing. Additionally, by keeping at least some tetrahedra which do not change, we can use them to accumulate information for that space. Specifically, we store the cost of occupancy.

For each new visibility constraint which is added to the system, we update the cost of occupancy for those tetrahedra which it intersects. We do this efficiently

by tetrahedra space skipping, moving from the start of the segment to the end by travelling through tetrahedra. From the surface of a face of one tetrahedron, we know that the next face encountered will belong to the adjacent tetrahedron. This means that when transitioning along the ray, only three planes need to be considered.

In order to make sure that a tetrahedron hasn't been caused to turn in on itself as features are moved by MonoSLAM, we monitor the sign of the signed volume for each tetrahedron, $TV\big(ABCD\big)$; if it changes, we delete the tetrahedra and retriangulate that space. The cost of occupancy can be recalculated directly from Equation 3.1.

3.5.5 Controlling Complexity

As our camera navigates the world, observing features, a naive implementation of our visible volumes mechanism would result in the number of visibility constraints growing very rapidly with the number of frames processed and thus growing with time. This would have very negative effects on both tetrahedralisation, which includes camera poses as vertices, and on intersection calculations. Since our goal is to define an approximate volume by these constraints, it is clear that some will be more important that others. When exploring new space, the camera end-points of these constraints may be important but when roaming inside our existing volume they are less important.

While a more sophisticated strategy might be possible to reduce the rate of growth in the number of constraints, here we use some heuristics to significantly reduce the number of visibility constraints stored in our representation. Firstly, it is important to add constraints that include a newly added feature in order to include these features within the context of the visibility volume. We can also join very close camera end-nodes which helps to reduce stored camera history states and factor out common observations with cameras. Further, we can add constraints at regular intervals in unexplored space. Via these measures, the number of constraints will grow in space and not in time.

The complexity of performing a Delaunay tetrahedralising of N points from scratch using the convex hull method in four-dimensions is $\mathcal{O}(N^2)$ [32]. Inserting a vertex into a tetrahedralisation can be performed by local flips at $\mathcal{O}(1)$ complexity provided the containing tetrahedron is known. Since updates to the tetrahedralisa-

tion occur largely when visibility constraints are added, containing tetrahedra can be efficiently computed by space skipping where rays are traversed through faces. The number of faces within a tetrahedralisation grows linearly with the number of points, meaning a worst case face-traversal complexity of $\mathcal{O}(N)$, but having a much lower expected complexity for general tetrahedralisations. For a probabilistic map where vertices are subject to move, checking tetrahedralisation consistency requires that we monitor each tetrahedra — a maximum of N^2 may exist. As more space is explored, it is this consistency check which dominates in computational cost, though it doesn't necessarily have to proceed at frame-rate.

3.6 Results

We present some examples of the method in operation in a cluttered indoor scene (Figure 3.8). The MonoSLAM system [27] was run in standard configuration on 30Hz video from a hand-held web-cam in order to generate point feature maps containing on the order of 50–100 features in a desktop area or the corner of a room.

3.7 Evaluation

The method is promising, but the reconstructions are currently of very rough quality due to the sparsity of the point cloud framework. It would be interesting to try it on much more dense point clouds; ProFORMA gives some evidence that it would be effective for highly-textured objects. However, what really became clear was that a feature-based map is not necessarily a good starting point for dense scene modelling, and that we should go back and consider methods which aim to use information from every image pixel.

3.8 Summary

In this chapter we have presented a preliminary investigation of how dense 3D SLAM might be achieved by reasoning about occupancy in a tetrahedralised volume with vertices at the estimated locations of point landmarks and historic camera poses in feature-based SLAM. Visibility reasoning based on the recorded successful mea-

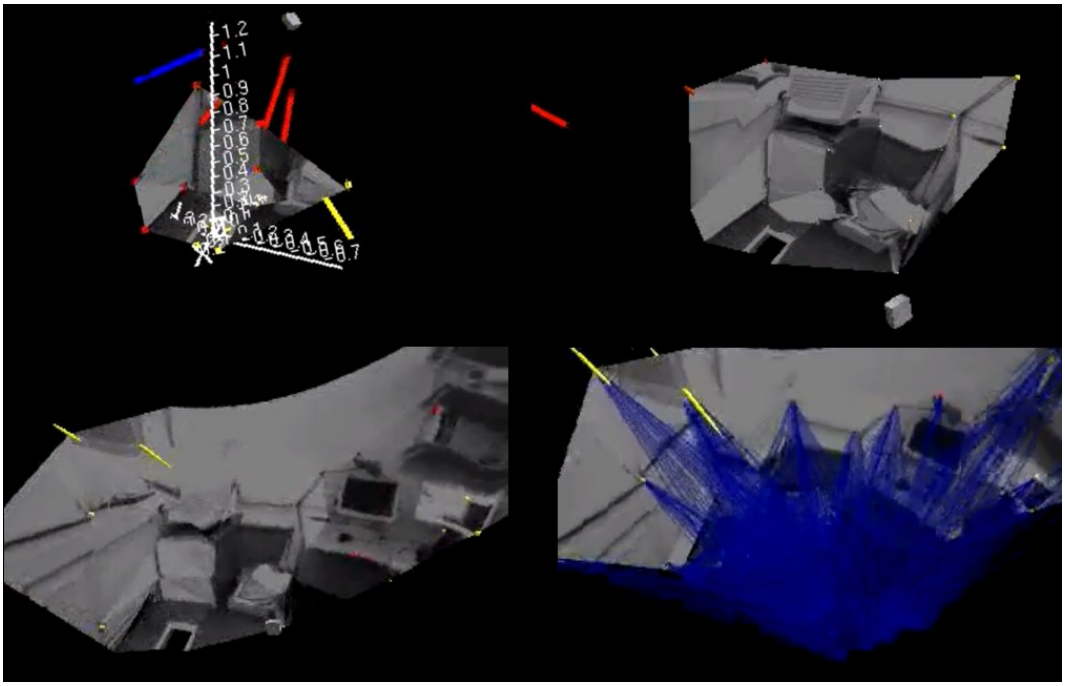


Figure 3.8: Novel snapshots of a live reconstruction as the camera browses the scene. Uncertainty ellipses for for sparse point features are shown in yellow, those still being initialised are in red. Bottom Right: the mesh of visibility constraints are overlaid with the scene and depicted as as blue lines

surements of features as the camera moves, and a straightforward occupancy cost measure, are shown together to be effective in determining the occupancy of tetrahedra and in defining reconstructed surfaces of arbitrary topology.

The reconstructions achieved, however, are notably blocky and it is clear that the sparsity of the point cloud used as a framework is a major limiting factor in the quality of models that can be achieved in real-world scenes. This insight prompted a change in direction and the start of a serious investigation of direct registration methods which have the potential to enable tracking and mapping using information from every pixel in a video stream rather than abstracted features. In the next chapter we study tracking methods, before moving on to the ways in which such techniques can be used for real-time SLAM in Chapters 5 and 6.

Chapter 4

Direct Parametric Visual Tracking

4.1 Introduction

Given a sequence of images from a video camera observing data represented by some model, visual tracking is concerned with determining the parameters of the model as it evolves over time, via analysis of the images alone. We are particularly motivated by visual ego-motion where we track an entire scene and wish to infer the motion of the camera from the perceived motion of the scene's structure.

Within this chapter we will look at *direct* methods for visual tracking based on the minimisation of a *dense* every-pixel cost function. The term 'direct' refers to how the parameters of the model are calculated *directly* from the way in which they relate to observed pixel intensities between video frames. The key component behind direct methods is a generative model which lets us predict how the live frame should look from previous ones, given the true values of the model parameters. Given that these actual values are unknown in advance, we estimate them by finding those which, when passed through our generative model, predict an image that most closely corresponds to live video frame. The photometric cost function which we minimise then, is simply the sum of squared difference between the live image and the predicted.

An efficient and effective approach for estimating the true model parameters for a visual tracking problem is to consider the gradient of the photometric cost function relating predicted and real views with respect to changing parameters. Within an iterative framework, the residual between each pixel in the video image and the prediction directly forms a linear constraint on a refined estimate to the model parameters. To find the true parameters, we must have a good estimate from which to start; this is often the case over video where parameters typically vary smoothly over time. As a guess for the current frame we can use our previous parameters or those based on some motion model. Attempting to estimate model parameters via gradient methods when our initial guess is too far from the true solution can lead us to a local minimum, and potentially catastrophic tracking failure. This hinders the straightforward applicability of direct methods for unordered images and for wide baseline matching without inter-frames. Direct methods, however, when combined with an accurate model can offer unrivalled visual tracking quality through video by considering data from all of the pixels.

Making use of every pixel maximises the information that can be extracted from image data and can lead to very high precision visual tracking. Clearly, some pixels are not as informative as others — those with small gradient magnitude in textureless regions for instance do not help constrain parameters within the cost-space as well as pixels with interesting gradients in one or more directions, like edges and corners. Nonetheless, each pixel is still informative and requires no effort to classify or select, as opposed to abstractions such as point and edge features.

A major argument against direct methods for real-time tracking is that they are expensive or not as robust as feature-based methods. The reality is that the robustness of a direct method will depend on how well the model can describe the image data and for gradient methods whether or not an initialisation within the convex basin of the true solution can be reliably made. Typically, since the parameters of the model are highly over-parameterised with respect to pixel observations, direct methods degrade gracefully with deteriorating image quality and can operate at low image resolutions. Estimation of parameters at low resolutions can frequently provide accurate results with improved convergence properties. These estimates can be further used to initialise estimates at higher resolutions within a coarse-to-fine strategy, helping to improve convergence properties and computational efficiency.

4.2 Background

In 1981, Lucas and Kanade proposed an efficient method for image registration using the spatial intensity gradient to accelerate search [77]. Assuming brightness constancy — that pixels in correspondence hold the same value — they observed that error in alignment as a function of alignment parameters, formed a smooth cost-space with meaningful gradients which could guide iterative estimates for registration. They showed that the location of the minimum within this cost space could be found by linearising and taking first-order steps 'downhill' which amount to a Newton-Raphson iteration.

In their original paper, Lucas and Kanade motivated their approach to 2D patch matching for different domains such as solving dense stereo correspondence. The parameter space for this problem was just one-dimensional in pixel disparity, with each patch around a pixel minimised independently. They also considered higher dimensional problems such as dense optic flow where each patch is optimised with respect to $u - v$ displacement.

Unlike their counterparts Horn and Schunck, who were also working on dense optic flow [49], Lucas and Kanade considered no explicit regularisation between neighbouring pixels. Horn and Schunck's work led to a whole literature of variational methods which balance noisy observed data with a regularisation which reflects some prior. In his thesis, Lucas contrasts his work with that of Horn and Schunck noting that whereas their early variational approach contained explicit regularisation, the image patch size in Lucas' work was a form of implicit regularisation [76]. He also demonstrated how general two-dimensional linear transforms such as rotation and shearing could be modelled.

Later, Tomasi and Kanade proposed an algorithm for sparse feature tracking through video; this was based on this same method for pixel correspondence but seeded at sparse salient image locations [127]. This came to be known as the KLT-tracker, which is prevalent still as a means for obtaining sub-pixel accurate sparse correspondence over video frames.

In the years that followed, direct methods were applied to different problems with more complex models. Irani and Anandan explored frame to frame alignment of video from a rotating camera and of a camera observing planar structure for

different mosaicing applications [54]. Related work looked at augmenting parametric structure with depth [55] and using pixel cost measures based on cross correlation and mutual information as opposed to absolute difference [52].

A thorough review was undertaken by Baker *et al.* which attempted to unify and categorise modifications to the original Lucas Kanade method [7]. A number of uses reversed the meaning of the image and template and so were dubbed inverse, allowing certain derivatives to be pre-computed for the template image which in many applications is fixed constant.

In 1999, Torr and Zisserman wrote a technical overview of feature-based methods for structure and motion, pushing for their use over direct methods [128]. They argued that the abstraction of features offered many advantages, including superior photometric invariance, such as changes to illumination and projective viewpoint changes. A companion paper by Irani and Anandan released simultaneously provided an overview of direct methods offering their alternate opinion [53]. They detailed methods to robustify the cost measure frequently found in direct methods in order to offer some of the illumination invariance enjoyed by feature-based techniques. An alternative to introducing an invariant cost metric is to model the changes directly and estimate these parameters jointly, such as in work by Silveira and Malis [114].

As far as viewpoint invariance is concerned, we must differentiate live tracking from generic wide baseline matching when contrasting feature-based and direct methods. Across consecutive frames of video, viewpoint typically changes little and quite predictably. Image patches which do not trivially match over these different views are a strong argument *for* direct methods, since it goes to show that it is the geometric information present in these regions which will constrain the solution. Generally, feature abstractions discard this information.

Up to the present, feature-based methods (mostly point feature-based) have dominated in practical structure from motion applications and have been the de-facto approach for real-time monocular SLAM [26, 34, 64]. As the robustness of monocular SLAM systems has increased, so too have the number of features that are tracked. Tracking more features offers resilience to mismatches which might represent a sizeable quantity of correspondences. Accurately tracking greater numbers of features also improves localisation accuracy, albeit with diminishing returns [120]. More ac-

curate localisation can also lead to better pose prediction via a motion model, which in turn can simplify and improve data association / initialisation for the next video image. In the limit, to achieve the highest tracking accuracy obtainable, we will be looking to consider contributions from every image pixel, and we must return once again to direct methods.

In [11], Benhimane and Malis demonstrated a real-time direct procedure for accurately tracking planar regions up to 150×150 pixels in size by estimating the homography which relates them. They posed the problem of alignment in terms of a Lucas-Kanade style approach named *efficient second-order minimisation* (ESM), first described by Malis for vision-based control [78]. Although their implementation was not able to consider every pixel of the image, their method was implemented for a standard desktop processor in 2004.

Comport *et al.* describe a system which uses every pixel within an image for highly accurate 6 DOF stereo odometry [22]. Dense stereo methods allow approximate structure to be associated with each pixel in the stereo pairs. ESM is again used within a direct method to align the projection of the previous textured depth map, parameterised by motion, to the current live stereo pair. Considering every pixel of each 759×280 resolution video frame, their method proceeds at $10s$/image-pair in Matlab. Their stated accuracy of 0.004% positional drift over 360m through a car mounted sequence in moving traffic represents a staggering increase in accuracy over previous visual odometry methods such as that of Nistér [98] based on stereo feature matching. The recent odometry system described by Konolige *et al.* operates from feature matching and a sliding window bundle adjustment in hard real-time, but it is hard to compare to the system of Comport *et al.* as it is designed for rough environments [66].

Lucas Kanade formulations such as ESM are inherently and trivially parallelisable and are ideally suited for implementation on data parallel architectures such as commodity graphics processors; this means that there is plenty of opportunity to achieve real-time every-pixel results with these methods. In [69], the performance of ESM is compared to other feature-based tracking methods, yielding greater accuracy when it is able to converge.

Given the potential benefits of real-time direct tracking methods, we will continue by describing them in more detail and relating them to real-time tracking

applications.

4.3 Methodology

The general approach that we will take for direct visual tracking, is to define a parametric *generative model* for how we expect the appearance of our tracked image data to change over time. Then, given an estimate of the current model parameters, we define a *photometric cost function* relating to how well our model matches observed data. From here, we will attempt to refine the model parameters to reduce the photometric cost associated with the current estimate — we can do this in an efficient way without cause for an exhaustive search of the parameter space.

4.3.1 Generative Model

The key to the Lucas-Kanade approach is to consider tracking as image alignment over some continuous space of possible transformations. In the case of tracking an image patch over a video sequence, we might choose to define the parameter space as rotation θ and displacement x, y horizontally and vertically. For a particular patch within a video frame, the values for x, y and θ describe the space of possible transformations into the next frame. These transformations offer a *generative model* for how the patch might look in a space of possible appearances.

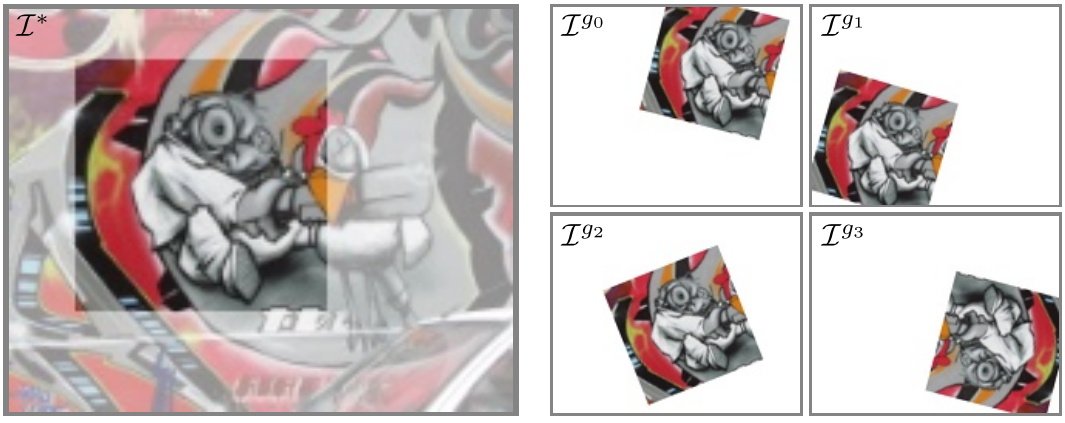


Figure 4.1: Original image $\mathtt{I}^*$ and four synthetic images ($\mathtt{I}^{g_0}$ to $\mathtt{I}^{g_3}$) formed by transforming the target patch with specific parameters of the generative model.

Figure 4.1 illustrates four samples ($\mathtt{I}^{g_0}$ through $\mathtt{I}^{g_3}$) from the continuous three-dimensional parameter space for how *every* pixel of the square patch might look after some motion has occurred. We would like to choose a parameterisation which supports all the patch appearances we might expect to observe, whilst finding a form which is not over-parameterised. Our (x, y, θ) model supports two-dimensional rigid body transformation in the image, but we can imagine different transformations such as affine or planar projective. We will look at some more interesting models later.

Treating our images as continuous functions, we define $\mathtt{I}(\mathbf{u})$ as a mapping from two-dimensional image coordinates $\mathbf{u} = (u, v)^\top$ to the corresponding pixel intensities within the image $\mathtt{I}$. From a reference image, $\mathtt{I}^*$, we can formalise our generative model by defining a parametric, generative image: $\mathtt{I}^g$.

From our two-dimensional rigid body tracking example, we define the value at a pixel $\mathbf{u} = (u, v)^\top$ in the generated image $\mathtt{I}^g$ by:

$$\mathtt{I}^g \begin{pmatrix} u \\ v \end{pmatrix} = \mathtt{I}^* \begin{pmatrix} u\cos(\theta) - v\sin(\theta) + x \\ u\sin(\theta) + v\cos(\theta) + y \end{pmatrix} \tag{4.1}$$

Or in matrix form we can define a homography $\mathtt{H}(\mathbf{x})$ parameterised by $\mathbf{x} = (x, y, \theta)^\top$ which reflects the same set of transformations:

$$\mathtt{I}^g \begin{pmatrix} u \\ v \end{pmatrix} = \mathtt{I}^* \left(\pi \left(\mathtt{H}(\mathbf{x}) \, (u, v, 1)^\top \right) \right). \tag{4.2}$$

$$\mathtt{H}(\mathbf{x}) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{pmatrix}, \tag{4.3}$$

where $\pi : \mathbb{R}^3 \to \mathbb{R}^2$ is the lowering function which performs homogeneous division (Equation 2.1).

This functional description of pixels in the generated image can be used to compose the image, as described in Section 2.3.4. In the following two chapters, we will generally be looking at homographic warps as above. In Chapter 6 however, we will

consider a more general function taking the form of a parametric warp W, which describes a parametric warp field:

$$\mathtt{I}^g\begin{pmatrix} u \\ v \end{pmatrix} = \mathtt{I}^*\Big(W\Big(\mathbf{x}; (u,v)^\top\Big)\Big), \tag{4.4}$$

4.3.2 Photometric Cost Function

Given a generative model that can predict the appearance of a portion of an image from a source image and a set of transformation parameters, we define a photometric cost function which represents how well the predicted image matches some observed image. In the context of tracking, we might want to match between consecutive video frames, choosing the set of motion parameters which minimises the match score.

For a generative model parameterised by the vector $\mathbf{x} \in \mathbb{R}^N$, we will define the match cost between a reference image $\mathtt{I}^r$ and a transformed live image $\mathtt{I}^l$ to be $F(\mathbf{x})$:

$$F(\mathbf{x}) = \frac{1}{2} \sum_{\mathbf{u}_r \in \Omega_r} \Big(\mathtt{I}^l\Big(W\left(\mathbf{x}; \mathbf{u}_r\right)\Big) - \mathtt{I}^r\left(\mathbf{u}_r\right)\Big)^2. \tag{4.5}$$

For each pixel $\mathbf{u}_r = (u,v)^\top$ in the reference image, point transfer through the warp function $W(\mathbf{x})$ forms a global data association estimate, placing $\mathtt{I}^r(\mathbf{u}_r)$ and $\mathtt{I}^l(W(\mathbf{x}; \mathbf{u}_r))$ in correspondence. Wishing to penalise intensity differences between these pixels, we choose an L_2^2 cost metric, evaluating the sum of squared differences between pixels. This sum is taken for every pixel $\mathbf{u}_r$ in image $\mathtt{I}^r$, belonging to the region $\Omega_r \subseteq \mathtt{I}^r \subset \mathbb{R}^2$. For greyscale images, the difference $\mathtt{I}^r(\mathbf{u}_r) - \mathtt{I}^l(W(\mathbf{x}; \mathbf{u}_r))$ will be scalar. For colour images, this difference is a vector, but the square can be taken just the same to yield a scalar. This is equal to the sum of the squared difference between each of the channels.

Having defined a cost function relating match quality to model parameters, we wish to find the set of parameters which minimise this cost, thus establishing the highest quality alignment. Of course, this may not reflect the true motion parameters — image noise and outliers or perceptual ambiguity such as the aperture problem may lead to an incorrect minimum, or a set of equally valid minima. These effects

vary by scene and can be avoided in many situations. We label the true minimiser of $F(\mathbf{x})$, $\mathbf{x}^\circ$:

$$\mathbf{x}^\circ = \underset{\mathbf{x}\in\mathbb{R}^N}{\arg\min}\ F(\mathbf{x}) \tag{4.6}$$

One simple scheme to find $\hat{\mathbf{x}} \approx \mathbf{x}^\circ$ is to discretise a likely interval within the space of possible model parameters and search this space exhaustively to find the smallest $F(\mathbf{x})$. With N parameters and q quantisations per parameter, computational complexity grows quickly, $O\left(q^N\right)$. Choosing a suitable discretisation is difficult as we would like to balance computational cost with precision. With too coarse a discretisation, the basin of the true minimum may be missed altogether.

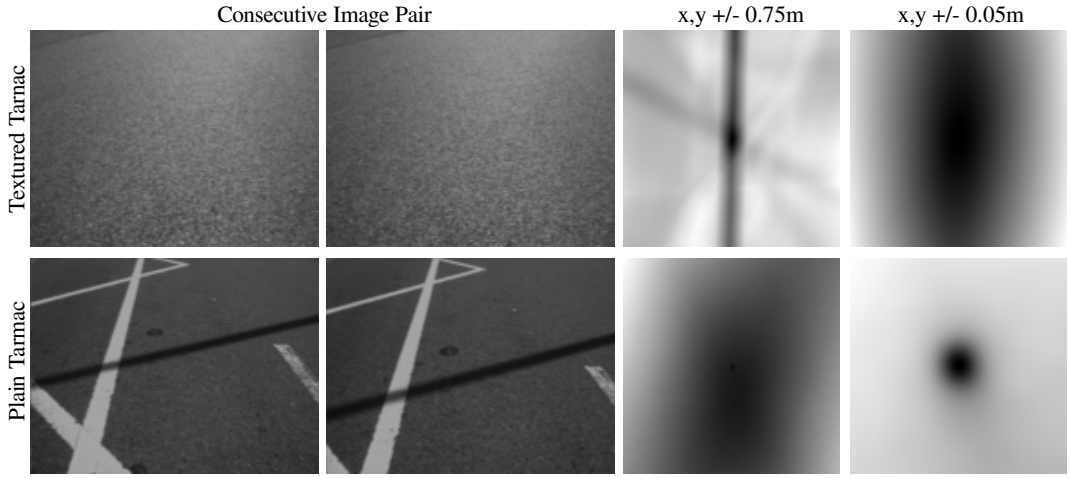


Figure 4.2: Sample image pairs (left) with associated parameter space cost plot (middle) and enlargement (right), centred around minimum cost parameterisation. A pixel in a cost plot represents the cost associated with the given sideways (x), and forward-backward (y) parameters. Cost plots have been normalised such that black represents lowest observed cost and white represents highest observed cost.

Figure 4.2 visualises the cost of alignment, $F(\mathbf{x})$, between two consecutive video frames for a car-mounted, downward facing parking camera. For illustration, the model is parameterised by motion over a plane in two variables, left-right (x) and forward-backward (y). This scenario is discussed later in Section 4.7. Within a large neighbourhood, you can see that there is a clear global minimum in these two examples, though the size of the basin varies. For the plain tarmac with high frequency texture, the basin is small and steep, with a relatively flat surrounding area. The textured tarmac with large varying regions demonstrates structure in the

cost space, with clearer localisation in x than in y due to the forward-backward oriented road markings.

4.3.3 Incremental Minimisation

For a given cost function $F(\mathbf{x})$, we have already described a scheme that could approximate the minimiser $\mathbf{x}^\circ$ of the function $F(\mathbf{x})$ via discretisation and exhaustive search; this method however is costly. Lucas and Kanade noted that for many problems the cost function $F(\mathbf{x})$ is quite smooth and they suggested an iterative scheme, computing the derivatives of this function with respect to the model parameters at a current estimate, and taking linear steps 'downhill' towards a minimum via gradient decent to refine the estimate. Provided the starting estimate is within the convex basin of the global minimum, it will be found — otherwise we may get stuck in a local minimum.

Figure 4.3 illustrates the cost landscapes from Figure 4.2, this time as a height field. Starting from an initial guess (white and red boxes), steps are taken 'downhill' along the dashed line by iteratively approximating the location of a downhill stationary point. This is typically achieved by linearising the function — calculating the derivative of $F(\mathbf{x})$ at the current estimate with respect to the parameters $\mathbf{x} = (x, y)^\top$. The global minimum in the centre of the plot may (white) or may not (red) be reached with this method. For the plain tarmac, we can see that the very sharp basin itself resides on a hill, showing that our initial estimate must be contained within this region to guarantee convergence to the global minimum. Although the textured tarmac landscape looks complicated in the large scale, we can see its global minimum actually has a very wide and smooth basin, as can be observed from the enlargement.

One factor that can influence the performance of the minimisation is the iterative linear approximation of $F(\mathbf{x})$ about the current estimate. How smooth this function is will affect how far from the current estimate the linear approximation is valid. The choice of parameterisations $\mathbf{x}$ of $F(\mathbf{x})$ can play a large role in making this space more linear. This is particularly true of three-dimensional transforms where many common parameterisations are highly non-linear and may even contain singularities. Euler angles for example can enter a state of *gimbal lock* which reduces the effective dimensionality of the space in certain regions.

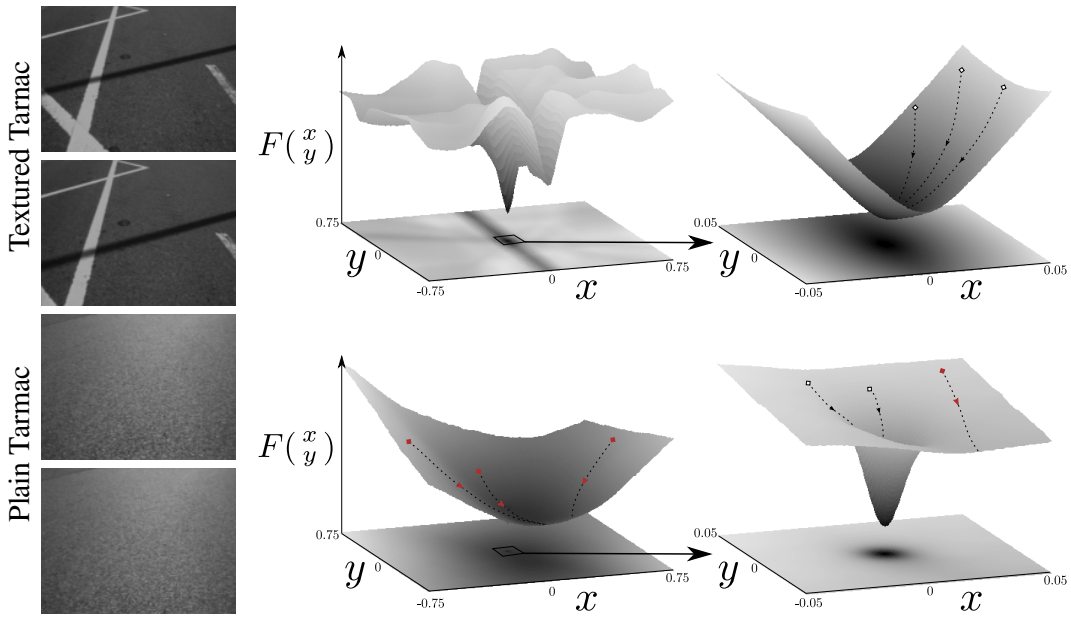


Figure 4.3: Height map cost landscape plots (centre) over the x, y parameter space for the two image pairs shown in Figure 4.2. Textured tarmac (top), plain tarmac (bottom), enlargements (right). Gradient methods initialised at different locations (squares) iterate taking steps 'downhill' toward local minimum. Those initialised within the true convex basin (white) will achieve the true solution, others (red) will not. The textured tarmac cost landscape is highly structured with a very clear minimum, and its enlarged central region is wide and smooth. The plain tarmac cost function is quite flat, with a minimum which is hard to see in the larger scale. An enlargement around the true minimum of this cost function shows that a clear basin exists, but it is comparatively narrow.

In line with a growing body of literature [31, 78, 82], where possible we will use the Lie algebra discussed in Section 2.4 to parameterise incremental transformations. The matrices used to express rotation and general rigid-body transformations in two or more dimensions belong to their own Lie groups. The matrix group of rigid-body transforms in three-dimensions for instance is named special Euclidean, $\mathbb{SE}(3)$. These groups represent smooth differentiable manifolds, each with an associated algebra representing a tangent space to the manifold around the identity element. This tangent space reflects the degrees of freedom of the transformation, and can be mapped back onto the Lie group via the matrix exponential map (Section 2.4).

For illustration then, consider the 3×3 matrix $\mathtt{H}$ representing a homography belonging to the special linear Lie group $\mathbb{SL}(3)$. Given an estimate $\hat{\mathtt{H}}$ of $\mathtt{H}$, we can

parameterise an incremental update $\mathtt{H}(\mathbf{x})$ to $\hat{\mathtt{H}}$ by its corresponding algebra, $\mathbf{x} \in \mathfrak{sl}_3$, where $\mathfrak{sl}_3$ is an eight-dimensional space. The matrix $\mathtt{H}(\mathbf{x})$ can be calculated as a function of $\mathbf{x}$ through the generators for $\mathbb{SL}(3)$ and the exponential map (Section 2.4). We can now write a photometric cost function using this pixel relation, parameterised by the vector $\mathbf{x}$:

$$F(\mathbf{x}) = \frac{1}{2} \sum_{\mathbf{u}_r \in \Omega_r} \left(\mathtt{I}^l \left(\pi \left(\hat{\mathtt{H}}^{lr} \mathtt{H}(\mathbf{x}) \dot{\mathbf{u}}_r \right) \right) - \mathtt{I}^r \left(\mathbf{u}_r \right) \right)^2. \tag{4.7}$$

Iteratively then, we are interested in finding an estimate $\hat{\mathbf{x}}$ of the minimiser $\mathbf{x}^\circ = \arg\min_{\mathbf{x}} F(\mathbf{x})$. Since we will find the approximate minimiser through linearisation, and because when using the Lie algebra, linearisations to $\mathtt{H}(\mathbf{x})$ are more accurate at $\mathbf{x} = \vec{0}$, estimates $\hat{\mathbf{x}}$ are incorporated into the current estimate $\hat{\mathtt{H}}^{lr}$ via the update:

$$\hat{\mathtt{H}}^{lr} \leftarrow \hat{\mathtt{H}}^{lr} \mathtt{H}(\hat{\mathbf{x}}). \tag{4.8}$$

We can formulate different cost functions similarly, choosing suitable update rules. There are different ways to obtain $\hat{\mathbf{x}}$ at each iteration, and we will proceed by describing some common methods, a detailed discussion of which can be found in the review of Lucas-Kanade methods and applications by Baker and Matthews [7].

4.3.4 Forward-Compositional Alignment

The cost function $F(\mathbf{x})$ parameterised by $\mathbf{x} \in \mathbb{R}^N$ is formed from the sum of squared differences between pixels in correspondence given the current estimate. We define the individual difference for a reference pixel $\mathbf{u}_r$ and its associated pixel in the live image to be $f_{\mathbf{u}_r}(\mathbf{x})$. For greyscale images, $f_{\mathbf{u}_r}(\mathbf{x})$ corresponds to a scalar, whereas for colour images, it is a vector. We can stack these per-pixel differences into a big column vector $f(\mathbf{x})$ — for colour images, each pixel contributes three elements, one for each colour channel. Writing P = (num pixels) × (num colour channels), $f(\mathbf{x}) \in \mathbb{R}^P$. We can now rewrite $F(\mathbf{x})$ as the L_2 norm of a vector of differences, squared:

$$F(\mathbf{x}) = \frac{1}{2} \sum_{\mathbf{u}_r \in \Omega_r} \left(f_{\mathbf{u}_r} \left(\mathbf{x} \right) \right)^2 = \frac{1}{2} \| f(\mathbf{x}) \|_2^2, \tag{4.9}$$

$$f_{\mathbf{u}_r}(\mathbf{x}) = \mathtt{I}^l\left(\pi\left(\hat{\mathtt{H}}^{lr}\mathtt{H}(\mathbf{x})\dot{\mathbf{u}}_r\right)\right) - \mathtt{I}^r(\mathbf{u}_r). \tag{4.10}$$

Defining $\boldsymbol{\nabla} f(\mathbf{y}) \in \mathbb{R}^{P\times N}$ to be the row vector of partial derivatives of $f(\mathbf{x})$ evaluated at $\mathbf{y}$, we start by defining a first order approximation $\hat{f}(\mathbf{x}) \approx f(\mathbf{x})$ and $\hat{F}(\mathbf{x}) \approx F(\mathbf{x})$ about $\mathbf{0}$ using a first order Taylor series expansion:

$$\hat{f}(\mathbf{x}) = f(\mathbf{0}) + \boldsymbol{\nabla} f(\mathbf{0})\mathbf{x}, \tag{4.11}$$

$$\hat{F}(\mathbf{x}) = \frac{1}{2}\|\hat{f}(\mathbf{x})\|^2 = \frac{1}{2}\hat{f}(\mathbf{x})^\top \hat{f}(\mathbf{x}). \tag{4.12}$$

A simple application of the product rule allows us to approximate $\boldsymbol{\nabla} F(\mathbf{x})$ from Equation 4.12:

$$\boldsymbol{\nabla} F(\mathbf{x}) \approx \boldsymbol{\nabla}\hat{f}(\mathbf{x})^\top \hat{f}(\mathbf{x}). \tag{4.13}$$

By definition, the local minimum $\mathbf{x}^\circ$ of $F(\mathbf{x})$ represents a stationary point of that function, and we make the assumption that such a minimum exists:

$$\boldsymbol{\nabla} F(\mathbf{x}^\circ) = \mathbf{0}. \tag{4.14}$$

It follows from Equations 4.11, 4.13 and 4.14 that we can find the approximate local minimiser $\hat{\mathbf{x}} \approx \mathbf{x}^\circ$ of $F(\mathbf{x})$ by solving:

$$\boldsymbol{\nabla}\hat{f}(\hat{\mathbf{x}})^\top \left(f(\mathbf{0}) + \boldsymbol{\nabla} f(\mathbf{0})\hat{\mathbf{x}}\right) = \mathbf{0}. \tag{4.15}$$

Writing the Jacobian $\mathtt{J} = \boldsymbol{\nabla} f(\mathbf{0})$, $\mathtt{J} \in \mathbb{R}^{P\times N}$, solving Equation 4.15 is equivalent to solving Equation 4.16.

$$\mathtt{J}\hat{\mathbf{x}} = -f(\mathbf{0}). \tag{4.16}$$

We can instead solve this overdetermined system in the linear least squares sense by considering its normal equations:

$$\mathtt{J}^\top \mathtt{J}\hat{\mathbf{x}} = -\mathtt{J}^\top f(\mathbf{0}). \tag{4.17}$$

Which we can rearrange to yield $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} = -\left(\mathtt{J}^\top \mathtt{J}\right)^{-1} \mathtt{J}^\top f(\mathbf{0}). \tag{4.18}$$

$\mathtt{J}^\top \mathtt{J}$ is the Gauss-Newton approximation to the Hessian, and Equation 4.18 can be seen as a single iterative non-linear least squares Gauss-Newton step. Practically, we would never construct $\mathtt{J}$ explicitly, instead building $\mathtt{J}^\top \mathtt{J}$ and $\mathtt{J}^\top f(\mathbf{0})$, an $N \times N$ and $N \times 1$ matrix respectively, by summing contributions from each pixel. Remembering that $\mathtt{J} = \boldsymbol{\nabla} f(\mathbf{0})$, we can write:

$$\mathtt{J}^\top \mathtt{J} = \sum_{\mathbf{u}_r \in \Omega} \left(\boldsymbol{\nabla} f_{\mathbf{u}_r}(\mathbf{0})^\top \ \boldsymbol{\nabla} f_{\mathbf{u}_r}(\mathbf{0})\right) \tag{4.19}$$

$$\mathtt{J}^\top f(\mathbf{0}) = \sum_{\mathbf{u}_r \in \Omega} \left(\boldsymbol{\nabla} f_{\mathbf{u}_r}(\mathbf{0})^\top \ f_{\mathbf{u}_r}(\mathbf{0})\right), \tag{4.20}$$

where:

$$\boldsymbol{\nabla} f_{\mathbf{u}_r}(\mathbf{a}) = \left(\frac{\partial f_{\mathbf{u}_r}(\mathbf{x})}{\partial \mathbf{x}_1}\Big|_{\mathbf{x}=\mathbf{a}}, \ \ldots, \ \frac{\partial f_{\mathbf{u}_r}(\mathbf{x})}{\partial \mathbf{x}_N}\Big|_{\mathbf{x}=\mathbf{a}} \right). \tag{4.21}$$

The partial derivatives $\boldsymbol{\nabla} f_{\mathbf{u}_r}(\mathbf{0})$ are of size $\mathbb{R}^{1\times N}$ for greyscale images, $\mathbb{R}^{3\times N}$ for colour images. Using index notation we can write them down directly:

$$\frac{\partial f_{\mathbf{u}_r}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}=\mathbf{0}} = \frac{\partial \mathtt{I}^l(\mathbf{a})}{\partial \mathbf{a}}\Big|_{\mathbf{a}=\pi\left(\hat{\mathtt{H}}^{lr}\dot{\mathbf{u}}_r\right)} \frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\Big|_{\mathbf{b}=\hat{\mathtt{H}}^{lr}\dot{\mathbf{u}}_r} \hat{\mathtt{H}}^{lr} \frac{\partial \mathtt{H}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}=\mathbf{0}} \dot{\mathbf{u}}_r \ . \tag{4.22}$$

If $\mathtt{H}(\mathbf{x})$ is parameterised by the Lie algebra $\mathbf{x} \in \mathfrak{sl}_3$, then the partial derivatives of $\mathtt{H}(\mathbf{x}) \in \mathbb{SL}(3)$ with respect to the parameters $\mathbf{x}$ about $\mathbf{0}$ are simply equal to the group generators for $\mathbb{SL}(3)$ (Section 2.4). We typically compute image derivatives $\frac{\partial \mathtt{I}(\mathbf{a})}{\partial \mathbf{a}} \in \mathbb{R}^{1\times 2}$ ($\mathbb{R}^{3\times 2}$ for colour image) by central difference, making sure pixels at

the boundary where this derivative is undefined are excluded in the sum. These derivatives and those of the homogeneous projection function $\frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}} \in \mathbb{R}^{2\times 3}$ are defined as follows:

$$\frac{\partial \mathtt{I}\left(\begin{smallmatrix} u \\ v \end{smallmatrix}\right)}{\partial u} = \frac{\mathtt{I}\left(\begin{smallmatrix} u+1 \\ v \end{smallmatrix}\right) - \mathtt{I}\left(\begin{smallmatrix} u-1 \\ v \end{smallmatrix}\right)}{2}, \qquad \frac{\partial \mathtt{I}\left(\begin{smallmatrix} u \\ v \end{smallmatrix}\right)}{\partial v} = \frac{\mathtt{I}\left(\begin{smallmatrix} u \\ v+1 \end{smallmatrix}\right) - \mathtt{I}\left(\begin{smallmatrix} u \\ v-1 \end{smallmatrix}\right)}{2},$$

$$\frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}} = \begin{pmatrix} \frac{1}{b_2} & 0 & \frac{-b_0}{b_2^2} \\ 0 & \frac{1}{b_2} & \frac{-b_1}{b_2^2} \end{pmatrix}.$$

4.3.5 Inverse-Compositional Alignment

The forward-compositional approach requires that the Gauss-Newton approximation to the Hessian $\mathtt{J}^\top \mathtt{J}$ be recomputed at each iteration since it is a function of the current estimated transformation, $\hat{\mathtt{H}}^{lr}$. The inverse-compositional formulation reverses the role of the images to keep the term containing the update parameter $\mathbf{x}$ constant when taking the derivative at $\mathbf{0}$. The updated per-pixel cost becomes:

$$f_{\dot{\mathbf{u}}_r}(\mathbf{x}) = \mathtt{I}^l\left(\pi\left(\hat{\mathtt{H}}^{lr}\dot{\mathbf{u}}_r\right)\right) - \mathtt{I}^r\Big(\pi\left(\mathtt{H}(\mathbf{x})\dot{\mathbf{u}}_r\right)\Big). \tag{4.23}$$

All other steps remain the same, except that we now use the inverse of the update transform during the update step, $\hat{\mathtt{H}}^{lr} \leftarrow \hat{\mathtt{H}}^{lr}\mathtt{H}(\mathbf{x})^{-1}$. $\mathtt{J}^\top \mathtt{J}$ is now constant through iterations whilst the reference image remains unchanged:

$$\frac{\partial f_{\dot{\mathbf{u}}_r}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}=\mathbf{0}} = \frac{\partial \mathtt{I}^r(\mathbf{a})}{\partial \mathbf{a}}\Big|_{\mathbf{a}=\mathbf{u}_r} \frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\Big|_{\mathbf{b}=\dot{\mathbf{u}}_r} \frac{\partial \mathtt{H}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}=\mathbf{0}} \dot{\mathbf{u}}_r\ . \tag{4.24}$$

4.3.6 Efficient Second-Order Minimisation

The method of *efficient second-order minimisation* (ESM) can be seen as a combination of both forward and inverse compositional methods [14]. Where the forward-compositional formulation takes image derivatives in the live image for its linear system and the inverse-compositional method takes image derivatives in the reference image, ESM forms a system from the average of both. We start by expressing

our original pixel-wise cost $f(\mathbf{x})$ (Equation 4.10) by its Taylor series expansion, and doing the same for $\boldsymbol{\nabla} f(\mathbf{x})$:

$$f(\mathbf{x}) = f(\mathbf{0}) + \boldsymbol{\nabla} f(\mathbf{0})\mathbf{x} + \frac{1}{2}\mathbf{x}^\top \boldsymbol{\nabla}\boldsymbol{\nabla} f(\mathbf{0})\mathbf{x} + \dots \tag{4.25}$$

$$\boldsymbol{\nabla} f(\mathbf{x}) = \boldsymbol{\nabla} f(\mathbf{0}) + \mathbf{x}^\top \boldsymbol{\nabla}\boldsymbol{\nabla} f(\mathbf{0}) + \dots \tag{4.26}$$

Equating $\mathbf{x}^\top \boldsymbol{\nabla}\boldsymbol{\nabla} f(\mathbf{0})$ in Equations 4.25 and 4.26, ESM applies a novel second order approximation $\hat{f}(\mathbf{x})$ of $f(\mathbf{x})$ from only first order terms:

$$\hat{f}(\mathbf{x}) = f(\mathbf{0}) + \frac{1}{2}\left(\boldsymbol{\nabla} f(\mathbf{0}) + \boldsymbol{\nabla} f(\mathbf{x})\right)\mathbf{x} \tag{4.27}$$

It follows from Equations 4.27, 4.13 and 4.14 that we can find the approximate local minimiser $\hat{\mathbf{x}} \approx \mathbf{x}^\circ$ of $F(\mathbf{x})$ by solving:

$$\boldsymbol{\nabla} \hat{f}(\hat{\mathbf{x}})^\top \left(f(\mathbf{0}) + \frac{1}{2}\left(\boldsymbol{\nabla} f(\mathbf{0}) + \boldsymbol{\nabla} f(\hat{\mathbf{x}})\right)\hat{\mathbf{x}} \right) = \mathbf{0} \tag{4.28}$$

Writing $\mathtt{J} = \frac{1}{2}\left(\boldsymbol{\nabla} f(\mathbf{0}) + \boldsymbol{\nabla} f(\hat{\mathbf{x}})\right)$, Equation 4.28 can be solved once again through the following expression:

$$\hat{\mathbf{x}} = -\left(\mathtt{J}^\top \mathtt{J}\right)^{-1} \mathtt{J}^\top f(\mathbf{0}) \tag{4.29}$$

In order to find the solution $\hat{\mathbf{x}}$ in Equation 4.29, we are required to evaluate the partial derivatives of the cost function at $\mathbf{0}$ and at the solution $\hat{\mathbf{x}}$. We have already seen these derivatives taken around $\mathbf{0}$ for the forward-compositional case (Equation 4.22). Taken around $\hat{\mathbf{x}}$ we get:

$$\frac{\partial f_{\mathbf{u}_r}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial \mathtt{I}^l(\mathbf{a})}{\partial \mathbf{a}}\Big|_{\mathbf{a}=\pi(\hat{\mathtt{H}}^{lr}\mathtt{H}(\hat{\mathbf{x}})\dot{\mathbf{u}}_r)} \frac{\partial \pi(\mathbf{b})}{\partial \mathbf{b}}\Big|_{\mathbf{b}=\hat{\mathtt{H}}^{lr}\mathtt{H}(\hat{\mathbf{x}})\dot{\mathbf{u}}_r} \hat{\mathtt{H}}^{lr} \frac{\partial \mathtt{H}^{lr}(\mathbf{x})}{\partial \mathbf{x}_i}\dot{\mathbf{u}}_r \tag{4.30}$$

Evaluating this cost directly is not possible since $\hat{\mathbf{x}}$ is unknown in advance. The method of ESM, however, observes that at the true solution, $\hat{\mathbf{x}} = \mathbf{x}^\circ$, the warped live image should in principle *exactly* equal the reference image. We can therefore

express the live image derivatives in terms of the reference image derivatives, taking care to adjust the basis:

$$\mathtt{I}^{l}\left(\mathbf{u}_{l}\right)=\mathtt{I}^{r}\left(\mathbf{u}_{r}\right),\quad \mathbf{u}_{l}=\pi\left(\hat{\mathtt{H}}^{lr}\mathtt{H}\left(\mathbf{x}^{\circ}\right)\dot{\mathbf{u}}_{r}\right) \tag{4.31}$$

$$\frac{\partial \mathtt{I}^{l}(\mathbf{u}_{l})}{\partial \mathbf{u}_{l}}\Big|_{\mathbf{u}_{l}=\pi(\hat{\mathtt{H}}^{lr}\mathtt{H}(\mathbf{x}^{\circ})\dot{\mathbf{u}}_{r})}=\frac{\partial \mathtt{I}^{r}(\mathbf{a})}{\partial \mathbf{a}}\Big|_{\mathbf{a}=\mathbf{u}_{r}}\frac{\partial \pi\left(\mathbf{b}\right)}{\partial \mathbf{b}}\Big|_{\mathbf{b}=\dot{\mathbf{u}}_{r}}\mathtt{H}(-\mathbf{x}^{\circ})\hat{\mathtt{H}}^{rl}. \tag{4.32}$$

Although Equation 4.32 is still a function of $\mathbf{x}^{\circ}$, it depends on it in a less significant way. Since the method is iterative, we assume $\mathbf{x}^{\circ}$ is small and evaluate it as $\mathbf{0}$ in the remainder of Equations 4.30 and 4.32.

The second-order approximation within the formulation can help to improve the speed of convergence, so although we cannot precompute the same Jacobian terms as for the inverse-compositional formulation, we are likely to reach the solution sooner [78].

4.4 Coarse-to-Fine Warping

As we saw in Section 4.3.3, the ability of a gradient-based method to find the true global minimum of a cost function relies on having a good initial estimate which lies within the convex basin of the true solution. One way in which we can potentially widen this basin is to define a related but smoother cost function to minimise. A common approach is to modify the input images by applying a Gaussian blur, removing higher frequencies and hence simplifying the cost landscape. Having converged to a minimum within this smoother function, we can return to the original image starting with our estimate from the smoother function. This can be repeated several times in a *pyramid* of blurring known as *coarse-to-fine* refinement [12]. By blurring the images, we can also sub-sample them when considering the cost function without loosing information, meaning that we save computation time too.

Figure 4.4 illustrates a typical Gaussian power of two image pyramid for a simple scene. As resolution is decreased, we notice that objects remain discernible over many reductions. Fine details such as the lines along the floor become less visible, and boundaries are blurred.

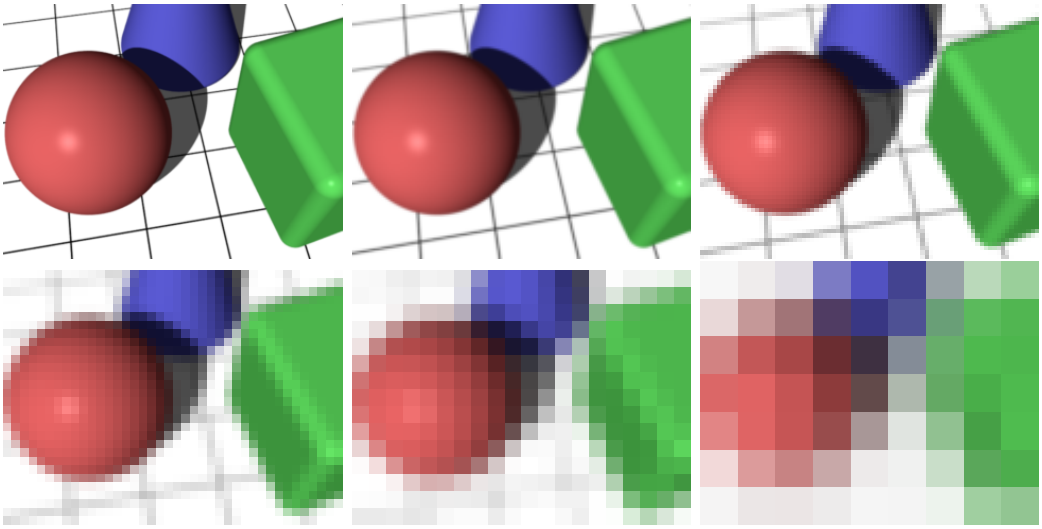


Figure 4.4: Gaussian Power of Two Image Pyramid: in which the original image (top left) is iteratively blurred and downsampled to create progressively lower resolution images.

For simple parameterisations, the cost of solving registration via the methods described in the previous section is dominated by the number of pixels considered. If we define the original image to form the 0^{th} level of the pyramid, an image at the n^{th} level has $\frac{1}{2^{2n}}$ the number of pixels. Iterations performed over the lowest resolution image of Figure 4.4 for example will take around 1000 times more operations than at the highest. On parallel architectures, where contributions from pixels can be computed in parallel, some of this cost can be hidden, though summation of per-pixel contributions must be partly serialised in a log reduction.

In order to compute the Gaussian blurred image $\mathtt{I}^*$ from the original $\mathtt{I}$, we convolve $\mathtt{I}$ with a Gaussian kernel. Gaussian kernels are separable, and we are able to apply the equivalent of a 2D convolution kernel to the image by instead performing two separate 1D convolutions. First, either a vertical or horizontal 1D kernel is applied to create an intermediate image, and then the transposed kernel is applied to yield the final blurred result. For a simple and fast 3×3 Gaussian convolution kernel, we can use the following coefficients, taking care to correctly reweight at image

boundaries:

$$\begin{pmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ \frac{2}{4} \\ \frac{1}{4} \end{pmatrix} \times \begin{pmatrix} \frac{1}{4} & \frac{2}{4} & \frac{1}{4} \end{pmatrix}. \tag{4.33}$$

When working with images of reduced dimensions, care must be taken to appropriately adjust the intrinsic parameters used, especially if we are working with image coordinates that start in the center of the first pixel. For this reason it can be convenient to use a coordinate system which starts at the corner of the first pixel, or even to work in normalised coordinates which are scale independent. An example of such a system can be found in OpenGL texture coordinates and normalised device coordinates.

4.5 Iteratively Reweighted Least Squares

Earlier in this chapter, we introduced several related methods for dense tracking, where motion parameters were estimated by minimising a per-pixel photometric cost function $F(\mathbf{x})$ with respect to the parameters $\mathbf{x}$:

$$F(\mathbf{x}) = \frac{1}{2} \sum_{\mathbf{u} \in \Omega} \left(f_{\mathbf{u}}(\mathbf{x}) \right)^2, \tag{4.34}$$

where $f_{\mathbf{u}}(\mathbf{x})$ is the residual for pixel $\mathbf{u}$ induced by parameters $\mathbf{x}$.

Finding the minimum $\mathbf{x}^\circ = \arg\min_{\mathbf{x}} F(\mathbf{x})$ represents the least squares solution and the *most likely estimator* or M-estimator of $F(\mathbf{x})$ assuming that the observed pixel differences $f_{\mathbf{u}}(\mathbf{x})$ are normally distributed about zero [136]. This will not be the case for pixels which do not belong to our model, or where the implicit brightness constancy assumption that we make has been broken by changes in illumination. These outliers can corrupt the data and have a disproportionate effect on the solutions result [48].

We can instead reformulate Equation 4.34 in terms of a different norm $\rho(r)$ over

the residual error r. For least squares, $\rho(r) = \frac{1}{2}r^2$:

$$F(\mathbf{x}) = \sum_{\mathbf{u}\in\Omega} \rho\left(f_{\mathbf{u}}(\mathbf{x})\right). \tag{4.35}$$

The minimiser of this expression forms the M-estimator for our choice of $\rho(r)$. Instead of minimising this new function directly, we can instead solve an iteratively reweighted least squares problem, which takes the form of our original least squares formulation (Equation 4.34), but where the contributions from each pixel are reweighted [136, 118]:

$$F(\mathbf{x}) = \frac{1}{2}\sum_{\mathbf{u}\in\Omega} \omega\left(f_{\mathbf{u}}(\mathbf{x})\right)^2, \tag{4.36}$$

where the pixel-wise *weight function* ω and *influence function* ψ are defined to be:

$$\omega(r) = \frac{\psi(r)}{r}, \qquad \psi(r) = \frac{\partial\rho(r)}{\partial r}. \tag{4.37}$$

For least squares, the influence function $\psi = r$, which tells us that the influence of a measurement on the solution increases linearly with the size of its residual. This implies that $\omega = 1$, and we can see that Equation 4.36 becomes equivalent to Equation 4.34. For other M-estimators where the weight is not equal to 1, we can simply scale each pixels corresponding contributions to the normal equations $(\mathtt{J}_{\mathbf{u}})^\top(\mathtt{J}_{\mathbf{u}})$ and $(\mathtt{J}_{\mathbf{u}})^\top f_{\mathbf{u}}(\mathbf{0})$ by ω, and solve as we would ordinarily.

Type	$\rho(r)$	$\psi(r)$	$\omega(x)$
L_2	$\frac{r^2}{2}$	r	1
L_1	$\lvert r\rvert$	$\operatorname{sign}(r)$	$\frac{1}{\lvert r\rvert}$
Huber $\begin{cases} \lvert r\rvert \le c \\ \text{else} \end{cases}$	$\begin{cases} \frac{r^2}{2} \\ c\left(\lvert r\rvert - \frac{c}{2}\right) \end{cases}$	$\begin{cases} r \\ c\operatorname{sign}(r) \end{cases}$	$\begin{cases} 1 \\ \frac{c}{\lvert r\rvert} \end{cases}$
Tukey $\begin{cases} \lvert r\rvert \le c \\ \text{else} \end{cases}$	$\begin{cases} \frac{c^2}{6}\left(1-\left(1-\left(\frac{r}{c}\right)^2\right)^3\right) \\ \frac{c^2}{6} \end{cases}$	$\begin{cases} r\left(1-\left(\frac{r}{c}\right)^2\right)^2 \\ 0 \end{cases}$	$\begin{cases} \left(1-\left(\frac{r}{c}\right)^2\right)^2 \\ 0 \end{cases}$

Table 4.1: Influence functions that give rise to useful M-Estimators within iteratively reweighted least squares [136].

So which influence function should we choose? Table 4.5 summarises some of the most common influence functions used in computer vision, as described by Huber [50] and Zhang [136]. The L_1 norm is a popular choice for having constant influence irrespective of the residual error size. The Huber norm is a combination of the L_2 norm for residuals below the constant c and the L_1 norm those above it. This smooth basin can lead to a nicer cost landscape whilst still being quite robust and convex.

In this thesis, we often use the non-convex Tukey influence function to find a very robust M-estimator for given data. It corresponds to a Gaussian inlier model and uniform outlier model, giving no influence to outliers above the tuning parameter. Although very robust norms can lead to more accurate results in the presence of outliers, they can also effect the rate of convergence toward the solution and introduce local minima into the cost landscape. One solution proposed by Huber is to start with a convex norm and then apply a few iterations of a super-robust non-convex norm after, mitigating the effect of large errors [50]. Our very similar solution is to vary the Tukey tuning parameter c as we converge on the solution in order to achieve a balance between convergence and robustness.

4.6 Visual Gyroscope, Rotational Odometry

Odometry is the term given to the estimated position formed from the integration of velocity measurements over time. Since each measure of velocity has some associated error, the sum of these measurements has an increasing absolute error which we call *drift*. Through a video sequence, if we are able to measure incremental motion between frames, we can integrate these estimates to produce *visual odometry*. The first kind of odometry we will consider is the special case of a camera which can only rotate.

Several offline direct video mosaicing papers such as [54, 107] relate pairwise consecutive images to one another or to a reference image using Lucas-Kanade style methods as we have introduced earlier and will use here. An alternative is to track salient features between images and estimate some transformation between them by minimising the transfer error induced by the correspondences between frames, as can be found in feature-based mosaicing literature [124].

Whereas it is common in both direct and feature-based literature to track between frames by parameterising pixel motion in terms of a general projective transform, we assume camera intrinsics are known in advance and parameterise pixel motion by the motion of the camera itself. This is in common with the EKF feature-based mosaicing system of Civera *et al.* described in [20]. This leads to a direct estimate of angular velocity and reduces the degrees of freedom of the optimisation from eight to just three.

Where previously it might have been seen as prohibitively expensive to use direct methods for real-time estimation of motion from every pixel, work such as that of Benhimane and Malis have shown that reasonable numbers of pixels can be considered at frame rate on desktop machines for tracking homographies [11]. They suggest ESM is an enabling technology for accelerating direct methods. In Klein and Murray's PTAM, a direct whole image based approach is used on small images in order to estimate rotational velocity at each frame to help locate point features [64].

Klein and Drummond showed that the rotational axis and magnitude of rotational velocity could be estimated in real-time from just a single frame by considering only the observed image's motion blur [61]. Unfortunately, a sign ambiguity exists which makes it impossible to establish which direction around the axis of rotation the camera is moving without some prior.

In what follows, we will take an ESM approach (Section 4.3.6), as inspired by work such as [11].

4.6.1 Formulation

In Section 2.3.3 we saw how the motion of pixels between frames for such a video sequence can be described by a rotational homography. We can represent the motion of the camera from the previous frame p to the live one l by a rotation matrix $\mathtt{R}^{lp} \in \mathbb{SO}(3)$. Following from the previous section, we can define a photometric cost function parameterised by $\boldsymbol{\omega} \in \mathfrak{so}_3$ as follows:

$$F(\boldsymbol{\omega}) = \frac{1}{2} \sum_{\mathbf{u}_p \in \Omega_p} \left(\mathtt{I}^l \left(\pi \left(\mathtt{K}\hat{\mathtt{R}}^{lp} \mathtt{R}(\boldsymbol{\omega}) \mathtt{K}^{-1} \dot{\mathbf{u}}_p \right) \right) - \mathtt{I}^p \left(\mathbf{u}_p \right) \right)^2 . \tag{4.38}$$

We solve this cost iteratively using ESM (Section 4.3.6) enabling us to estimate rotational velocity very accurately and in real-time; we have a visual gyroscope. Through coarse-to-fine warping, we can cope with large rotational velocities and high rotational accelerations. Through the use of direct methods, our estimate is remarkably robust to image degradation such as motion blur and camera defocus.

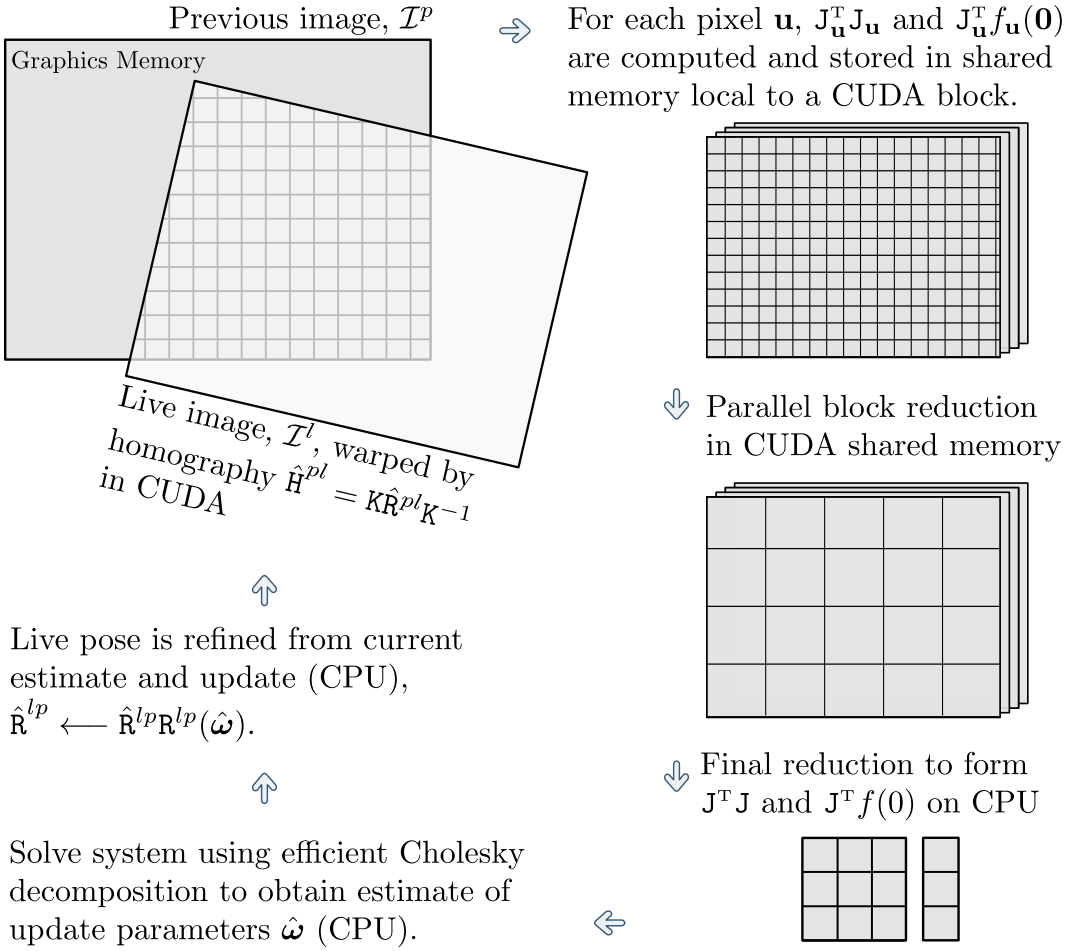


Figure 4.5: Iterative Rotation Estimation.

Figure 4.5 shows the main steps in calculating the accurate inter-frame rotation estimate of the camera. Live 30Hz operation is enabled through the inherent parallelisability of the method and the power and general programmability of modern graphics cards. Given a current estimate of the rotation, $\hat{\mathtt{R}}^{lp}$ (which at first may be the identity or calculated from the previous known velocity), CUDA threads operate massively in parallel per pixel to compute $\mathtt{J}_\mathbf{u}^\top\mathtt{J}_\mathbf{u}$ and $\mathtt{J}_\mathbf{u}^\top f_\mathbf{u}(\mathbf{0})$, storing the results in fast memory shared between blocks of threads. These are summed effi-

ciently in a parallel reduction within this memory to create a much smaller array of systems. These are finally summed to form $\mathtt{J}^\top\mathtt{J}$ and $\mathtt{J}^\top f(\mathbf{0})$ on the CPU where the parallel advantage stops becoming so great. Finding $\hat{\boldsymbol{\omega}} = -\left(\mathtt{J}^\top\mathtt{J}\right)^{-1}\mathtt{J}^\top f(\mathbf{0})$ by inverting $\mathtt{J}^\top\mathtt{J}$ with an efficient Cholesky decomposition, we apply the update rule $\hat{\mathtt{R}}^{lp} = \hat{\mathtt{R}}^{lp}\mathtt{R}(\hat{\boldsymbol{\omega}})$ and repeat until the update $\hat{\boldsymbol{\omega}}$ is small.

4.6.2 Comparison to Gyroscope

To test the ability of our method to track dynamic local motion, we have compared the angular velocity output of our method against a solid state Xsens MTi gyroscope (rated up to $300°/s$) fixed rigidly to the rear of the camera. The camera and gyroscope were mounted on a tripod and oscillated to produce increasingly rapid motion (up to around 5 oscillations per second) about each of its axes in turn. We compare the live camera image to a single reference image taken before motion started (rather than using the previous frame). This helps us to evaluate potential performance within a keyframe type system as we will explore later in Chapter 5. In this way, a potentially motion-blurred live image is being compared with a clean reference image. Alignment of the camera and gyroscope frames of reference have been performed physically, and are potentially subject to small error.

Figure 4.6 illustrates sample plots of angular velocity plotted against time for both our visual gyroscope and the physical gyroscope, including a sample failure where alignment to the reference image is lost. We limited per frame ESM iterations to 48 at the 5^{th} level of the pyramid, 16 at the 4^{th}, 8 at the 3^{rd}, 4 at the 2^{nd}, and 2 at the 1^{st} and using any remaining time to perform iterations at the 0^{th} level corresponding to the original image. The characteristics of estimation are somewhat different depending on the axis of rotation, though in each case our visual gyroscope follows the physical gyroscope quite closely. The small systematic errors on the axis are likely due to a combination of camera miss-calibration and slight misalignment between camera and gyroscope frames of reference.

We found that angular velocity about the z-axis (cyclotorsion) is estimated most consistently; in the sample plot notice that the truncated peaks of the gyroscope data show that the tracking limits of the device were exceeded while visual tracking still continued — our system was able to maintain fidelity about this axis in excess of 7 $rads^{-1}$. This performance might be explained by the pixel flow induced by such

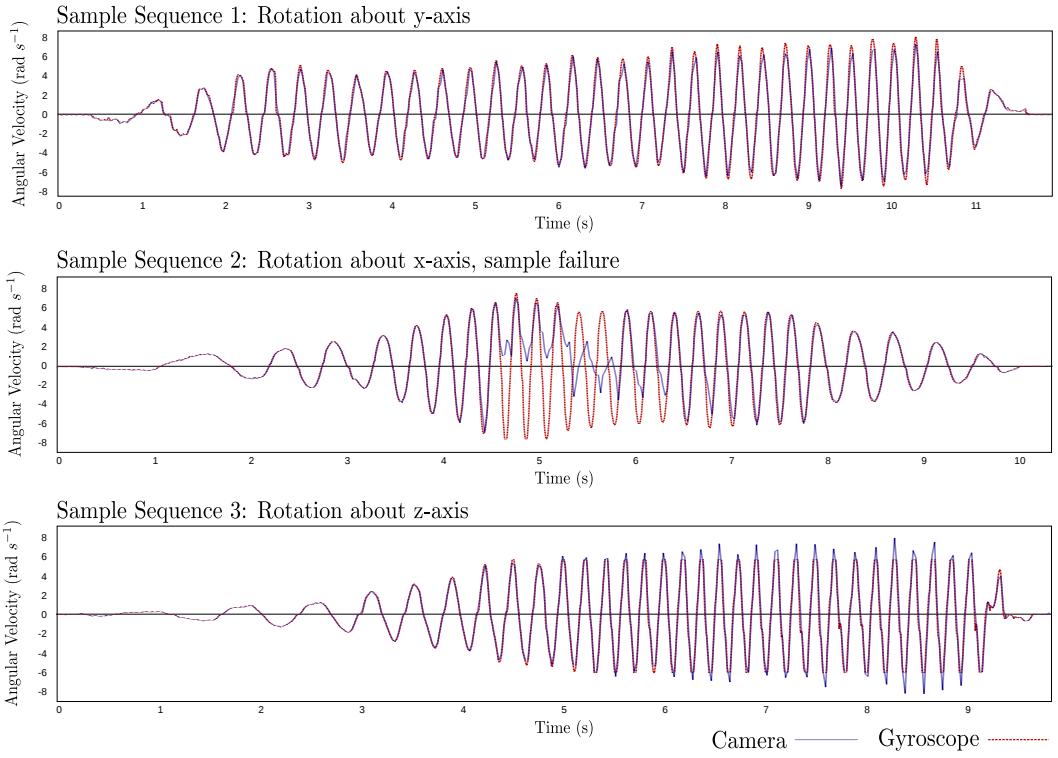


Figure 4.6: Sample output from runs of our visual gyroscope illustrating high dynamic tracking performance; the plots show angular velocity estimates from our vision system compared with the output from a gyroscope as the camera was vigorously oscillated about each of the three camera-oriented axes (y axis pan; x elevation, z cyclotorsion).

motion, with flow vectors ranging from quite small (about the centre of rotation) to very large at the outside. Smaller image displacements between predicted and measured pixels in correspondence make for a more meaningful linearisation, and we could argue that *some* pixels with good approximate association can guide larger errors at the image edges. With this motion, the number of pixels that remain common to both the live and reference image also remain high.

Through experimentation, we find that it is easier to brake live tracking against a reference keyframe by rotations around the x or y axes, though it is still relatively hard. The second plot in Figure 4.6 shows an example of such a failure; the tracking under-shoots, and by-chance re-acquires correspondence with the reference image against which it is tracking a few oscillations later. We find the system to be least stable about the x-axis, though only by a minor factor. We suggest that this is due

to the narrower vertical field of view.

Such high angular velocities can introduce significant motion blur within an image; its extent will depend on the duration during which the camera holds open its shutter. Matching a blurred live image against a non-blurred reference image can give rise to an ambiguous and wide minima during minimisation and therefore constrains estimated motion less strongly. By reducing the shutter time of the camera to reduce motion blur, images instead appear darker and are subject to increased pixel noise which can also harm estimation. To increase resilience to motion blur, Park *et al.* include blurring within their generative model by considering the image formation process [101]. Since consecutive frames will be similarly blurred, we could instead estimate inter-frame rotation directly as described earlier (like a true gyroscope), but this has the disadvantage of being subject to drift.

4.7 Visual Odometry From a Parking Camera

This research was supported by Renault, leading to the publication: 'Accurate Visual Odometry from a Rear Parking Camera' by Steven Lovegrove, Andrew Davison and Javier Ibañez-Guzmán [73]

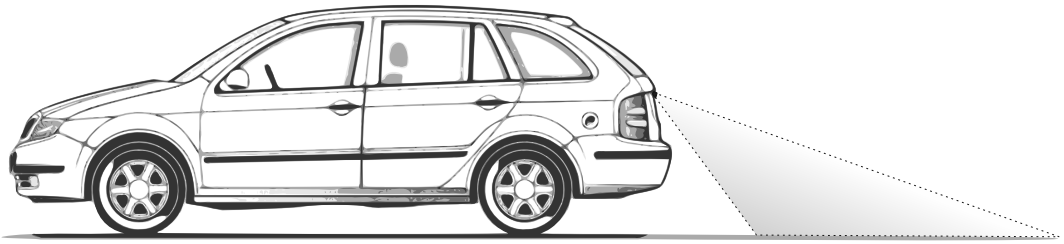


Figure 4.7: A rear parking camera, as fitted to many current vehicles, views the road surface directly behind the car during normal driving.

Through analysis of the video stream from a rear parking camera, we show that it is possible to estimate the motion of a vehicle travelling along the ground in real-time. This camera, a standard feature of many current models, views the road surface directly behind the vehicle during normal driving (Figure 4.7). The road surface consists of mostly high frequency, self-similar, speckly texture (Figure 4.8).

Although many point features can be detected in these images, they are individually highly ambiguous. Matching each part of the image successfully requires the support of the whole frame.

Figure 4.8: Four video frame samples taken from a parking camera in our test sequence.

In large scale urban driving experiments with a full ground-truth comparison, we show that our trajectory estimates from pure visual odometry are locally very accurate and smooth, though subject to inevitable drift over the long distances travelled in our full experiment. We go on to perform a live filtered fusion between our parking camera visual odometry estimates and an automotive GPS signal, and show that this combination of commodity level sensing available on many standard vehicles gives a quality of trajectory estimate comparable to an expensive PHINS system in robustness and accuracy.

Processing VGA images, we have a CPU-only implementation that operates at 30 fps on an Intel i7 920 and a GPU accelerated implementation that achieves in excess of 300 fps with a single NVidia GTX 480 GPU.

4.7.1 Related Work

In the motor industry, there has been much interest in using cameras to add functions related to safety and autonomous driving, with a particular emphasis on forward-looking stereo camera rigs to detect and estimate the locations of pedestrians or other vehicles. There have recently been attempts to use these cameras for estimation of the ego-motion of the vehicle itself. These approaches (e.g. [59, 6]) have mainly followed the methods for visual odometry developed in the computer vision and robotics literature over recent years, based on point feature matching and geometry estimation over a sliding window. The first convincing stereo visual odometry system of this type was due to Nistér *et al.* [97], and state of the art performance is well

represented by the recent work of Mei *et al.* [84]. Comport *et al.* describe a direct stereo odometry system which makes use of every pixel within the image in an approach which has similarities with out own [22]. They demonstrate highly accurate 3D odometry without extracting point features. Napier *et al.* tackled drift in their forward-looking stereo system by aligning frames to overhead satellite images [92].

General monocular visual odometry systems still lag somewhat in performance behind stereo systems, particularly due to the extra difficultly in reliably tracking enough high quality features to constrain motion, but good results can be achieved from video streams with high texture levels (e.g. [121]). In general, monocular odometry systems suffer not only from increasing absolute pose uncertainties just as stereo systems do, but also of drifting scale, since an objects absolute size cannot be determined from monocular vision alone. Scaramuzza *et al.* describe how on many moving vehicles, the nonholonomic constraint can be used to break indefinite scale drift [110]. Assuming that the camera is fixed rigidly to the vehicle and not directly over its non-steering axle, whilst the vehicle turns, the scale of the scene can be related to the distance from this fixed axle, which we know does not change. In this work, we use a much simpler constraint — that the camera's height from the ground is approximately fixed.

The video stream available from a commodity vehicle parking camera is clearly not amenable to a general feature-based approach due to the lack of distinctive texture in most frames. However, here we have the strong advantage of the assumptions that can be made about the constrained planar scene shape and vehicle motion, and this motivates our choice of a whole image alignment method.

To our knowledge, we present the first convincing results for trajectory estimation based on a parking camera which views only the road surface. Azuma *et al.* [5] recently presented a method for estimating the motion of a car using a single forward-looking camera, which combined a standard feature-based approach in the top half of the image viewing street-side objects with an approach in principle similar to our own using homography estimation for the lower part of the image observing the road. However, many parts of their method were ad hoc, and the results ultimately presented were limited (only one corner of real road). This was presumably because the robustness of the method was low. Our large scale results demonstrating both high accuracy and robustness go much further in proving the validity of estimating vehicle motion from road texture alignment and have particular value due to the

low-cost, software-only solution permitted by the use of an existing parking camera.

4.7.2 Method

The dense, direct methods introduced earlier in this chapter enforce global alignment which can overcome the difficulties in local matching. We again use an ESM formulation, explicitly taking advantage of the locally planar road surface. Thereby, we are able to make use of all of the texture present to robustly measure the image warp from one video frame to the next. This in turn leads to an accurate estimate of the vehicle's motion over the ground plane.

For two consecutive video frames, the image warp that transforms the visible portion of one frame to the other is uniquely described by the video camera's motion and the structure of the scene being observed. For a rear-mounted parking camera, video frames recorded whilst driving will largely image the ground upon which the vehicle is moving. We treat the ground as a smooth, locally planar manifold, and produce odometry relative to that manifold.

Our simple model assumes that the car moves perfectly parallel to the ground and inter-frame vehicle motion can be parameterised by the vector $(\Delta x, \Delta y, \Delta\theta)^\top$. The constrained dynamics of a vehicle allow us to model motion with a constant velocity prior. This helps us in our optimisation to find the true minima over other local minima which may exist, even though pixels travel very quickly across the frame.

4.7.3 Parameterisation

The camera's motion is a function of the vehicle's motion and thus we consider two important frames of reference, that of the camera and that of the car (Figure 4.9).

We describe the two-dimensional location of the vehicle with respect to the world whilst taking image $\mathtt{I}^i$ by $\mathtt{T}^{wv_i}$, a homogeneous point transfer matrix belonging to the Lie group of rigid body transforms in two-dimensions, $\mathbb{SE}(2)$. For a live image, $\mathtt{I}^l$ and the previous one, $\mathtt{I}^p$ with associated positions $\mathtt{T}^{wv_l}$ and $\mathtt{T}^{wv_p}$, we can calculate the vehicle transformation between them by matrix composition, $\mathtt{T}^{v_l v_p} = (\mathtt{T}^{wv_l})^{-1}\,\mathtt{T}^{wv_p} = \mathtt{T}^{v_l w}\mathtt{T}^{wv_p}$.

Since the parking camera captures images of the world projectively in three-

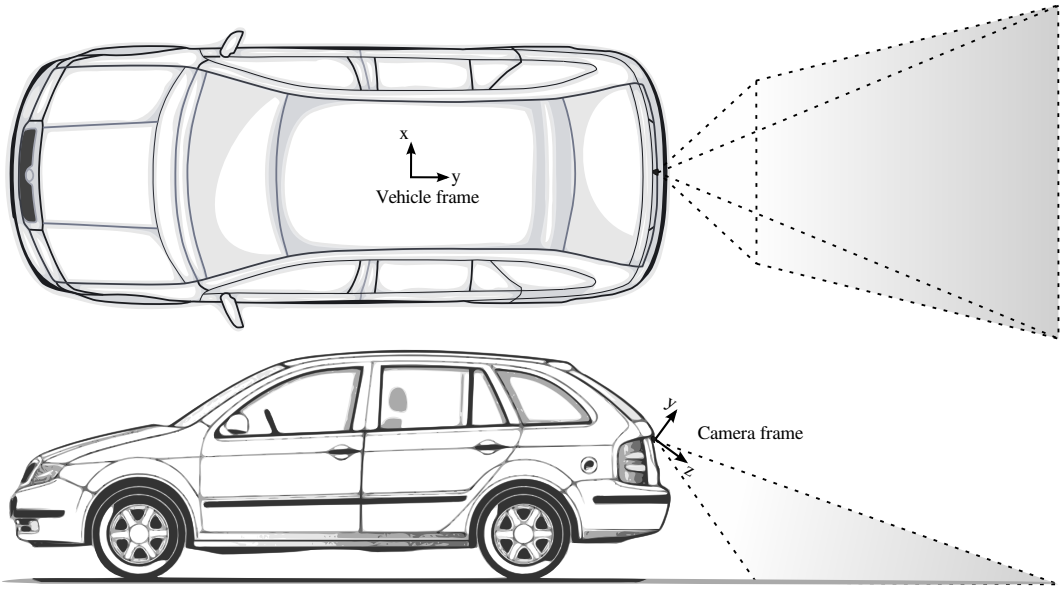


Figure 4.9: Car centric and camera centric frames of reference, calibrated to allow for simple parameterisation.

dimensions, we also describe the pose of the camera whilst taking image frame $\mathtt{I}^i$, by $\mathtt{T}^{wc_i}$, a homogeneous point transfer matrix belonging to the Lie group of rigid body transforms in three-dimensions, $\mathbb{SE}(3)$. Although $\mathtt{T}^{wv_i}$ is a two-dimensional transform, we can arbitrarily refer to it in three-dimensions by 'raising' it into $\mathbb{SE}(3)$ by enforcing that the vehicle lives on the X-Y plane. We do so implicitly for notational convenience.

As the camera is fixed rigidly to the vehicle, there is a constant transform $\mathtt{T}^{vc} = (\mathtt{T}^{wv_i})^{-1}\,\mathtt{T}^{wc_i}$ which relates the vehicle frame of reference to that of the camera, for all i. This allows us to write the three-dimensional transformation of the camera between images $\mathtt{I}^p$ and $\mathtt{I}^l$ as a function of the vehicle motion.

$$\mathtt{T}^{c_l c_p} = (\mathtt{T}^{vc})^{-1}\,\mathtt{T}^{v_l v_p}\mathtt{T}^{vc} \tag{4.39}$$

4.7.4 Formulation

Observations of a plane via a projective camera are related to one another via a plane-induced homography, as discussed in Section 2.3.3:

$$\mathtt{H}^{lp} = \mathtt{K}\mathtt{T}^{c_l c_p} \left(I | - \mathbf{n}_{d_c} \right)^{\top} \mathtt{K}^{-1}. \tag{4.40}$$

Here, $\mathtt{H}^{lp} \in \mathbb{R}^{3\times3}$ describes the homogeneous transformation that takes pixels in the 'previous' camera image, $\mathtt{I}^p$, to those in the 'live' image, $\mathtt{I}^l$. $\mathtt{K} \in \mathbb{R}^{3\times3}$ is referred to as the intrinsics matrix which projects points in three-dimensions into the camera. $\mathbf{n}_{d_c} = \frac{\hat{\mathbf{n}}_c}{d_c}$ describes the pose and height of the camera relative to the ground, where $\hat{\mathbf{n}}_c$ is the unit normal of the plane in the camera frame of reference c and d_c is the distance of closest approach to the plane.

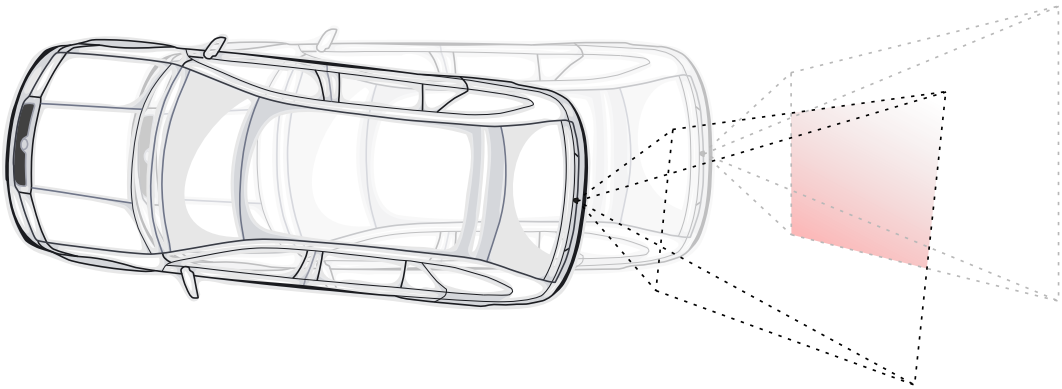


Figure 4.10: A co-visible region (shaded red) exists between consecutive camera images. A linear mapping described by a plane-induced homography lets us transfer pixels from one image to another as the vehicle moves.

Between two consecutive image frames $\mathtt{I}^l$ and $\mathtt{I}^p$ (Figure 4.10), if we were to know the parameters of $\mathtt{K}$, $\mathtt{T}^{vc}$, $\mathtt{T}^{v_l v_p}$ and $\mathbf{n}_{d_c}$, we could synthesize one frame from the other by transferring the pixels via $\mathtt{H}^{lp}$ (Equation 4.40). For any given synthesis, we can compare how similar it is to the true image by comparing how similar their pixels are. Given that the parking camera is fixed rigidly to the vehicle, and following from the assumption that it is travelling along a locally planar surface, none of $\mathtt{K}$, $\mathtt{T}^{vc}$, or $\mathbf{n}_{d_c}$ will change; therefore, they can be easily estimated via off the shelf camera calibration packages. This leaves the motion of the vehicle $\mathtt{T}^{v_l v_p}$ unknown and is, in fact, the quantity we wish to estimate.

We rewrite Equation 4.40 using Equation 4.39 and define a parametric cost function:

$$\mathtt{H}^{lp} = \mathtt{K}\mathtt{T}^{cv}\mathtt{T}^{v_l v_p}\mathtt{T}^{vc}\left(I|-\mathbf{n}_{d_c}\right)^{\top}\mathtt{K}^{-1}. \tag{4.41}$$

$$F(\mathbf{x}) = \frac{1}{2}\sum_{\mathbf{u}_p\in\Omega_p}\left(\mathtt{I}^{l}\Big(\pi\left(\mathtt{K}\mathtt{T}^{cv}\hat{\mathtt{T}}^{v_l v_p}\mathtt{T}(\mathbf{x})\mathtt{T}^{vc}\mathbf{u}_p\right)\Big) - \mathtt{I}^{p}\Big(\pi\left(\mathbf{u}_p\right)\Big)\right)^2. \tag{4.42}$$

$\mathtt{T}(\mathbf{x})$ represents a small change to the estimate $\hat{\mathtt{T}}^{v_l v_p}$. We choose to parameterise our update matrix $\mathtt{T}(\mathbf{x})$ by $\mathbf{x}\in\mathbb{R}^3$ belonging to $\mathfrak{se}_2$, the Lie algebra of the Lie group $\mathbb{SE}(2)$. This parameterisation is equivalent to the vector $(\Delta x, \Delta y, \Delta\theta)^{\top}$. For many vehicles, the nonholonomic motion constraint could allow us to reduce the dimension of our parameterisation even further if we were to assume no slippage occurs between the road surface and vehicle tyres. We did not test if this was the case. The Lie algebra $\mathfrak{se}_2$ is related to the Lie group $\mathbb{SE}(2)$ by:

$$\mathtt{T}(\mathbf{x}) = \exp\left(\sum_{i=1}^{3}\mathbf{x}_i\mathtt{A}_i\right), \tag{4.43}$$

where $\mathtt{A}_{0,1,2}\in\mathbb{R}^{3\times3}$ are the group generators for $\mathbb{SE}(2)$ [60], listed for convenience in Appendix A.

As we have seen previously, we are interested in finding $\mathbf{x}^{\circ} = \arg\min_{\mathbf{x}} F(\mathbf{x})$ which we do iteratively via ESM, computing $\hat{\mathbf{x}}\approx\mathbf{x}^{\circ}$ and applying the update:

$$\hat{\mathtt{T}}^{v_l v_r} \leftarrow \hat{\mathtt{T}}^{v_l v_r}\mathtt{T}(\hat{\mathbf{x}}). \tag{4.44}$$

This process is repeated until convergence, which may be detected based on a threshold on the magnitude of the update $\hat{\mathbf{x}}$.

4.7.5 Results

Experimental Setup

Our evaluation is against data obtained during an extended experiment where a multi-purpose Renault Espace passenger vehicle conducted a 2.5km run through an urban setting, capturing images continuously at 30fps while travelling at speeds of up to 45km/h. The rear view camera is a Fire-i digital camera having VGA resolution.

On board was also a ground truth system capable of estimating with high accuracy the vehicle's position and orientation. This PHotonic Inertial Navigation System (PHINS) consists of a tactical level 'inertial measurement unit' made of 3 fibre optic gyroscopes and 3 pendulum-type accelerometers, a bi-frequency GPS receiver and the vehicle wheel odometry [56]. Despite occlusion of GPS signals or multi-paths, the ground truth estimations remain precise with positioning error below half a meter.

The intrinsic parameters $\mathtt{K}$ of the parking camera used were calibrated using a standard calibration grid technique. The camera's extrinsic location relative to the vehicle frame $\mathtt{T}^{cv}$ was initially hand-measured, but this estimate was then refined by visual means in a process similar to that developed by Miksch *et al.* [87].

Drifting Odometry

Our visual odometry system estimates inter-frame motion at frame rate (30Hz). This measurement is most useful to determine the velocity of the vehicle. Figure 4.11 shows the linear and angular velocity as measured by our system compared to that of the ground truth PHINS system on a five and a half minute road sequence. Visual odometry is plotted at 30Hz whereas the PHINS system is plotted at 1Hz. The bottom plot shows the system's confidence as measured by the sum of squared pixel error between aligned frames. We can see that the visual odometry measurement follows the ground truth system closely. There are four short sections in the sequence where tracking could not provide an estimate and here constant velocity is assumed (Annotation 2).

We can integrate our measured velocity over time to produce a trajectory in some

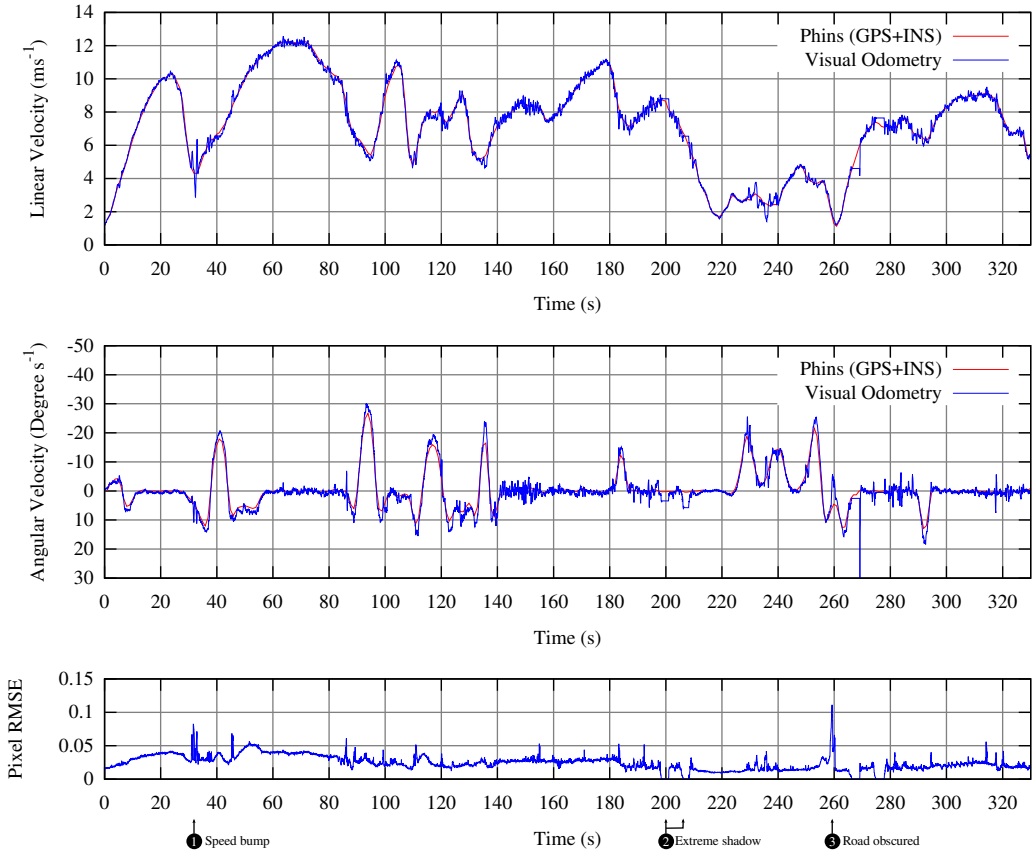


Figure 4.11: Linear and angular velocity of visual odometry compared with ground truth over time for 2.5km sample sequence.

common frame of reference. This, like *any* form of motion estimation based only on relative local sensing such as vision, will of course drift away from the frame of reference over time. Figure 4.12 shows stretches of trajectory estimate, each one minute long, aligned in each case to ground truth at the first frame. This helps us to visualise the rate at which the odometry drifts from an absolute map.

Figure 4.13 shows four ten second stretches of trajectory, again each aligned on the first frame to ground truth. We additionally show standard automotive GPS overlaid for comparison. We can see that our visual odometry system is much smoother locally than the GPS measurements and is sampled at a higher frequency.

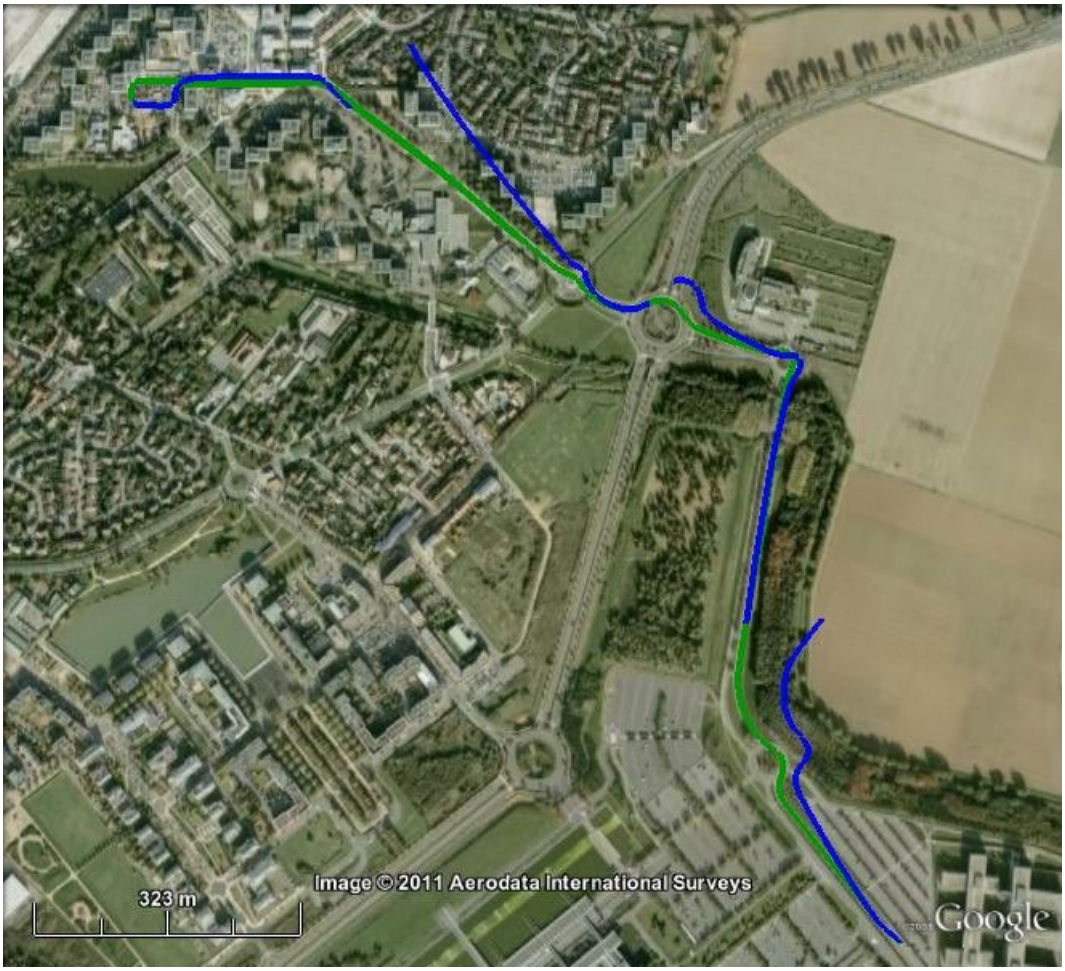


Figure 4.12: Four one-minute-long sequences of integrated visual odometry (blue) against ground truth (green).

4.8 Localisation From a Parking Camera and GPS

The benefits of visual odometry and GPS are complimentary. Visual odometry offers a robust, accurate and smooth local estimate of motion whereas GPS provides coarse measurements fixed to a global map. Although it is common to fuse multiple measurements such as these within a probabilistic filter such as an EKF, we chose to perform a real-time sliding window optimisation.

We used g^2o [67], a general framework for graph optimisation, in order to fuse absolute GPS measurements and relative visual odometry measurements into a combined real-time accurate estimation of pose relative to a non-drifting reference frame.

Figure 4.13: Four 10 second sequences of integrated visual odometry (blue) against GPS (red) and ground truth (green).

Within this pose graph (figure 4.14), vertices correspond to the poses of the car at the moment in time when each video frame was taken. Edges connecting consecutive video frames represent relative pose measurements from visual odometry. GPS measurements are represented as unary edges reflecting absolute constraints, occurring approximately once in every thirty frames of video.

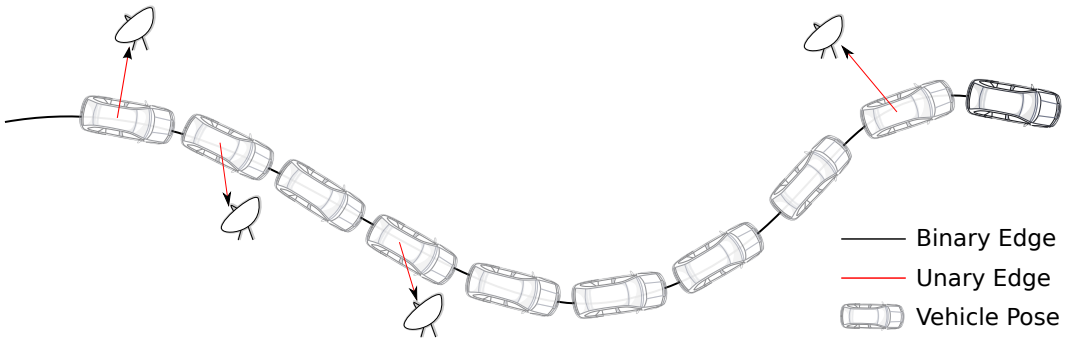


Figure 4.14: The absolute pose of the vehicle with respect to a static world reference frame is established via a sliding window pose graph formulation. It consists of relative binary edges from visual odometry, and unary edges reflecting absolute constraints from GPS measurements.

Within this pose graph formulation, since non-consecutive poses are never related to one another, the influence of new measurements on existing ones will diminish

quickly with pose age. This is particularly true since the absolute GPS measurements constrain the overall shape of the car trajectory. Practically speaking, poses over the age of around 10 seconds receive next to no contributions from new measurements, and can be considered fixed. By fixing the oldest vertex in the graph each time the total number of unfixed vertices reaches above a threshold, we can maintain constant time operation with a fixed-sliding window estimation. In a graph consisting of such simple structure and of relatively small size, g^2o is able to incorporate new measurements very quickly, and well within real-time constraints.

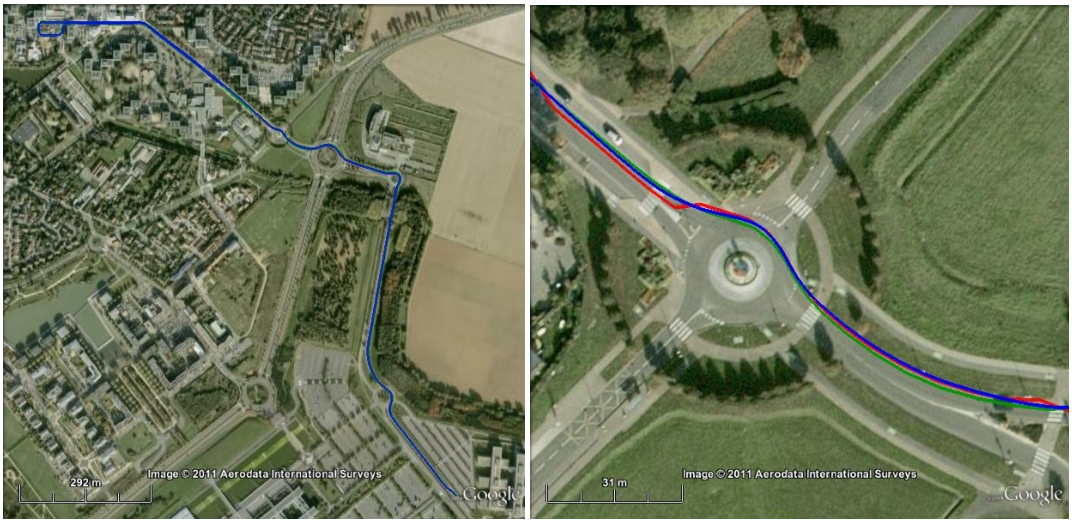


Figure 4.15: Fused visual odometry and GPS (blue) against ground truth (green). The right hand image is an enlargement of the roundabout from the left hand image, additionally displaying the GPS trajectory (red). Notice that the combination of visual odometry and GPS more accurately reflects the ground truth trajectory than GPS alone.

Figure 4.15 shows the resulting trajectory after sensor fusion. We can see that the resulting system deviates only a small amount from the ground truth system and it is hard at the scale of the image on the left to differentiate them. In the enlargement which includes also the automotive GPS trajectory, we can see that compared to GPS, the combination of GPS and visual odometry is much smoother, whilst accurately capturing the path of the vehicle through the roundabout.

4.8.1 Failure Modes and Future Work

There are two main failure modes in our system (Figure 4.16). The first comes from lighting which exceeds the dynamic range of the camera; there are four short sections in the sequence which are also identifiable in Figure 4.11 where the video frames are completely saturated black. Clearly, no software solution could do any better than relying on motion priors or extra sensors such as wheel odometry. Such a situation is clearly detectable by measuring the image variance. We assume constant velocity motion when image variance passes below a set-once threshold.

Using a light to illuminate the area directly behind the vehicle remains a potential solution in dark areas, but care must be taken to consider the non-uniform effect it will have on the image. Since the illumination source would be fixed to the vehicle, any pattern in the lights projection onto the road will appear static in the video, thus breaking the critical brightness constancy assumption, as road travelling under the pattern will change in brightness over time.

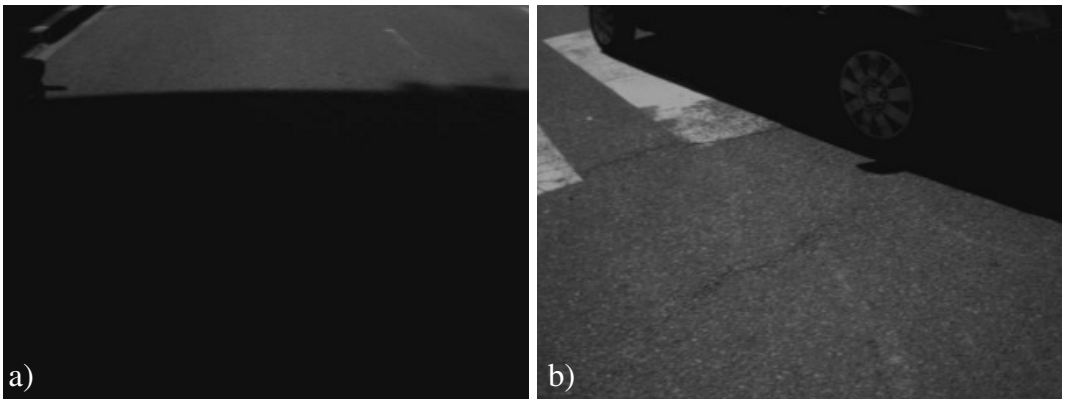


Figure 4.16: a) The vehicle passes under heavy shadow causing the image frame to become completely saturated by pure black. b) Another vehicle obscures a large amount of the road, and induces incorrect motion.

The second failure mode comes when our basic scene assumption is violated. Annotation 1 in Figure 4.11 shows the velocity measurement error induced by a non-planar speed bump and the associated increase in mean squared pixel error. Annotation 3 highlights this assumption being broken again when the road is part obscured by a turning vehicle (Figure 4.16b). These situations can be considered during sensor fusion by making use of the visual odometry's confidence output, mitigating their overall influence.

In order to establish the true validity of visual odometry from a rear parking sensor, further evaluation is required on new datasets — specifically, the robustness of the method with respect to different environmental conditions such as weather, illumination and road surfaces remains an open question. Matching as we do only over consecutive video frames, we expect the effect of global changing illumination to be negligible, though reflective road surfaces in wet conditions, and lighting from moving targets may cause some problems.

4.9 Summary

In this chapter, we have motivated the use of direct methods for real-time frame to frame video tracking problems, and shown that they can be implemented efficiently on graphics hardware to process each pixel of every image at frame-rate to achieve accurate alignment.

For rotational odometry, this every pixel formulation offers significant resilience to motion blur, allowing high rotational velocity frames to be tracked accurately. The coarse-to-fine refinement strategy makes it simple to track densely whilst avoiding non global solutions and has the effect of reducing computation time. With such a wide basin of convergence, highly dynamic camera motions which exhibit large rotational accelerations can be tracked reliably. In comparisons against an industrial gyroscope, fidelity in certain axis were shown to exceed the maximum rate of the device, and at other times followed the gyroscope data closely.

Our system for visual odometry based only on video frames from a rear parking camera was shown to generate smooth and accurate motion estimates. The road texture recorded by the parking camera would pose significant problems for systems based on feature matching techniques. However, by considering pixels jointly across the entire image, local correspondence is achieved through global support. In this way, the problems of incorrect data association and diminishing support are avoided. Although we have provided evaluation on a 2.5km sequence, more thorough evaluation of this method across varying road surfaces, environmental conditions, times of day and road speeds remains future work. We suspect the method will perform badly at night, and it is an open question as to whether a wet surface for example remains approximately diffuse to fit with the underlying brightness constancy

assumption.

Chapter 5

Direct Parametric SLAM

5.1 Introduction

Whilst tracking camera motion through a video sequence, we typically make the assumption that individual frames are captured instantaneously, giving us image measurements at discrete instances in time which correspond to evolving camera poses. Making the rigid world assumption, we expect these measurements to be consistent with one another within the bounds of our sensor's accuracy.

Chapter 4 introduced the Lucas-Kanade method and showed how a pair of images from video can be aligned to one another through minimisation of a photometric cost function, defined via an appropriate generative model. Often though, we are interested in more than *odometry*, and wish to localise ourselves *without drift* within a fixed frame of reference. We may also be interested in the structure of the world in which the camera exists, abstracted in some form by a *map*, or perhaps we are only interested in this map.

Drift-free localisation requires either that we can in some way directly measure our absolute position, or that we can summarise and store where we have been and recognise when we revisit a place from our past. For general outdoors motion, sensors such as GPS can sometimes help in the former case by providing coarse but absolute measurements (as we saw in Section 4.8). For rotational estimation, a compass and inertial sensors can offer this absolute measurement. If we are without

any of these things, we must build a map of the world as we go and simultaneously use it to locate where we are; a process referred to as simultaneous localisation and mapping (SLAM). For a short review of SLAM, please refer to Section 1.3.

Taking the video sequence in its entirety, off-line methods that estimate the poses of the camera and the scene jointly are dubbed structure from motion (SFM). Typically, an SFM pipeline will start by extracting features and attempting *data association* between detected features in different video frames — finding out which 2D image measurements refer to the *same* physical 3D landmark. Given this fixed data association which we assume is correct, the optimal configuration of cameras and 3D points is defined to be that which minimises the sum of squared error between the projection of these points into each video image and their measured locations. This measure is minimised in a process called *bundle adjustment* which requires that we solve a large non-linear system of equations. This stage can be computationally expensive, scaling poorly with the number of camera poses and landmarks considered. It has itself been the focus of much research to improve performance with those most successful taking advantage of the system's sparsity (e.g. [71, 65]).

In the live setting, things can be a little more difficult. Bounded by finite resources between incoming video frames, our aim is typically similar to SFM, except we would like the most accurate pose and map incrementally at each time instance. The differentiation between SLAM and odometry is that in SLAM we wish to reduce the rate at which error is accumulated in our pose to be approximately constant for trajectories which revisit previous areas often. In both SFM and SLAM, we prevent endlessly accumulating error by relating non-consecutive elements in our map together, a process we call *loop-closure*.

Whilst processing video, we receive megabytes of data every second — 1582 MB per minute for a standard 24 bit colour VGA stream. In chapter 4, we motivated every-pixel approaches to aligning video frames for greater accuracy and robustness; of course, we would ideally also like to apply this philosophy to use *all* of the data from each of our video frames in order to construct the best map possible. In practice however, we face hard time constraints; we are left with the decision of how best to sparsify the largely redundant data whilst maximising our goals — perhaps maximising tracking or mapping accuracy within bounded time constraints.

5.2 Real-Time Direct Spherical Mosaicing / SLAM

The simplest SLAM problem that we will consider is that of real-time spherical mosaicing from a rotating camera with fixed intrinsics. A spherical mosaic is created by stitching multiple images together taken from cameras with a common optic centre in order to form a new image with increased field of view, potentially covering an entire sphere. Not only does such a mosaicing system contend with only three degrees of freedom compared to six for general motion in three-dimensions, but the space of exploration is limited to that of a simple view-sphere, meaning the size of our complete map is bounded.

5.2.1 Background

There is a great deal of literature concerned with building mosaics from images and video; a tutorial by Szeliski provides a good description of the different components which are often required and includes an overview of modern methods for implementing them [124]. He points out that generally two types of alignment are important: some local frame to frame / mosaic alignment, and global alignment to ensure all images of the mosaic are globally consistent. Szeliski's tutorial pays little attention to the matter of computation time though, and this reflects a broader pattern throughout the mosaicing literature.

It is clear that feature-based methods have enabled real-time frame to frame tracking for a while (e.g. [90]), and increasingly, direct methods are also able to densely track planar regions over increasing numbers of pixels at frame-rate (e.g. [11]). Building globally consistent mosaics interactively, however, requires that we can integrate frames into a mosaic non-destructively and close loops at moderate speeds; this gives feedback for a user to decide when they can stop collecting images or if they need to sample from an unexplored area.

Early results in video mosaicing came largely from direct methods, aligning consecutive frames locally and then integrating into a common coordinate system via composition [123, 54]. The first system to consider globally consistent registration was due to Sawhney *et al.* [108]. They first estimate global alignment by chaining local direct frame to frame alignment, and then find local registration between overlapping neighbours. Finally, they minimise a global error formed from the transfer

of image corners based on the locally computed frame alignments.

As methods associated with feature tracking and matching matured, they offered some advantages for mosaicing over direct methods. Work by Lowe on scale invariant feature transform (SIFT) made it possible to match features robustly over a wide range of transformations without any initial guess [74]. Finding the alignment of out of order still images become more reliable through the use of these methods, and a shift can be seen toward these 'more efficient' feature centric approaches where computation time is spent in areas of high image texture; the typical pipeline of which starts by matching sparse correspondences between frames by either tracking through video (using for example the KLT tracker [127]) or using some descriptor, such as SIFT, to find close image patches in the feature space. From this approximate *data association*, inliers and outliers are typically separated via a technique such as RANdom SAmple Consensus (RANSAC) [39] , and finally all camera poses and features are bundle adjusted [81] to minimise the transfer error between determined and measured feature locations. A modern example of such a system is described by Brown and Lowe [15].

In attempting to register a series of images over 360° loops, misestimation of camera intrinsic parameters can make it hard to align the first and last overlapping frames by predicted position alone. Once they have been matched, the local orientations may not sum to 360°, and this difference forms an error which must be propagated globally throughout the trajectory. With an estimated focal length which is too long or too short, this redistribution of error will make local registration worse and can cause images to bunch up or be spread out. One thing that we can do is allow camera intrinsics to change within the global alignment process to absorb error.

Estimating camera parameters from unknown structure is referred to as *self calibration*, as described in the general 3D setting by work such as [37]. Civera *et al.* incorporated self calibration into their real-time EKF SLAM system by including them in the state vector as static variables with uncertainty [19]. In the case of a camera which can only rotate, self calibration is particularly well posed, and it is possible to very accurately estimate internal camera parameters without any knowledge of the scene [1]. In self calibration, lens distortion is frequently ignored and assumed known, though it too can greatly affect mosaic quality. One of the few works to consider the estimation of lens distortion from a rotating camera is that of

Sawhney and Kumar [109].

In our system, the expected performance of calibration refinement is further enhanced by our ability to match images automatically around full 360° panoramas, giving the potential for accurate calibration even for cameras with a narrow field of view. Including camera intrinsics in our optimisation enables us to achieve large loop closures over 360° without explicit loop closure detection. By refining camera intrinsics quickly before large loop closures are made, we can be more confident that alignment error with respect to the fixed mosaic frame of reference is small and within the basin of convergence of the true global solution. Unlike general SLAM problems, a camera moving within a spherical mosaic can only move so far before observing again a previously mapped area of the scene, so the maximum expected drift is bounded.

The first and only known globally consistent mosaicing system able to track motion at frame-rate was described by Civera *et al.*— a SLAM system based on point feature matching and sequential filtering using an EKF [20]. As discussed previously, the computational complexity of the EKF means that it scales badly with the number of features tracked. For their mosaicing system, Civera *et al.* used only around 10-20 features (matched using 11×11 pixel patches) per frame; all but around 3% of every image was ignored for the purposes of image alignment. Of course, some pixels are more informative for alignment than others, but it is hard to argue that considering so few will set limits on the mosaicing quality which can be achieved.

5.2.2 Overview

Inspired by Klein and Murray's PTAM [62] and the analysis of Strasdat *et al.* [120], for our rotational SLAM problem, we will consider a keyframe-oriented factorisation where tracking and map maintenance are decoupled into interleaved / parallel tasks.

Our algorithm is split into two tasks which can either run as parallel threads on a multi-core PC or be interleaved on a GPU by: a) tracking from a known map, and b) global map maintenance and optimisation (see Figure 5.1). For local tracking, we use the direct, whole image second order minimisation method ESM, described in Section 4.6, which allows us to align the live frame relative to stored historic keyframes. We implement this alignment on graphics hardware for high-quality

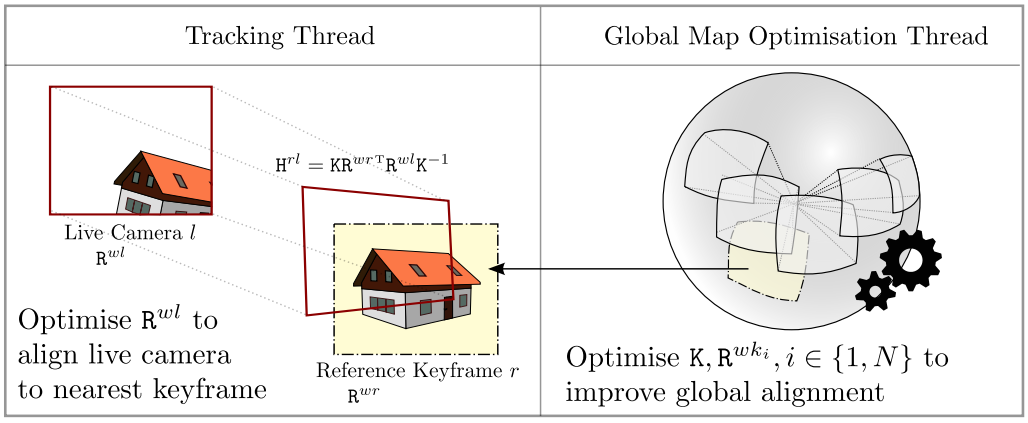


Figure 5.1: Parallel Tracking and Mosaic Optimisation. Local alignment relative to the mosaic is performed in hard real-time, whilst accurate global map adjustment is performed at interactive speeds slower than frame-rate.

real-time tracking relative to our map. In order to maintain our map, we formulate a dense every-pixel cost function which refines not only the poses of each constituent keyframe, but also the global intrinsic parameters which help enable accurate loop closure and global registration. Despite the large amount of data such a global joint optimisation must process, we show it can converge at interactive rates on modern hardware to produce high quality, globally consistent mosaics live, such as the office panorama shown in Figure 5.2.

We remove radial distortion from all live frames as they enter our system and deal only with perspective images from then onwards. Distortion parameters are estimated once per camera and lens; we use PTAM's calibration tool [62]. Additionally, we describe an automatic method for relocalisation if tracking should fail, allowing the current mosaic to be re-joined without corruption.

Keyframe Map

Our mosaic map is formed from a collection of keyframes — key historic camera poses with associated image data and a power of two image pyramid. Keyframes within our map are related to one another by a 3 DOF rotation. We store the current estimate of a keyframe's pose as a rotation matrix $\mathtt{R}^{wk}$ relating the camera's local frame of reference, k, to that of an arbitrary world frame of reference, w.

Figure 5.2: Full hemisphere mosaic shown spherically projected about two different frames of reference.

Tracking

When tracking commences, we set the first live image to be our first keyframe, k_0 with pose $\mathtt{R}^{wk_0}$ set to the identity matrix. For each subsequent live frame, we use the previous live pose to select the closest keyframe from our map (Figure 5.3). We estimate the current pose by considering the image warp between this keyframe and the current image, which in turn allows us to estimate the relative motion.

Exploration

As tracking continues, we create new keyframes and add them to the map if the overlap between our current image and closest keyframe becomes too small and falls below a threshold. Keyframes which we add inherit the pose of the live camera at that time (Figure 5.4).

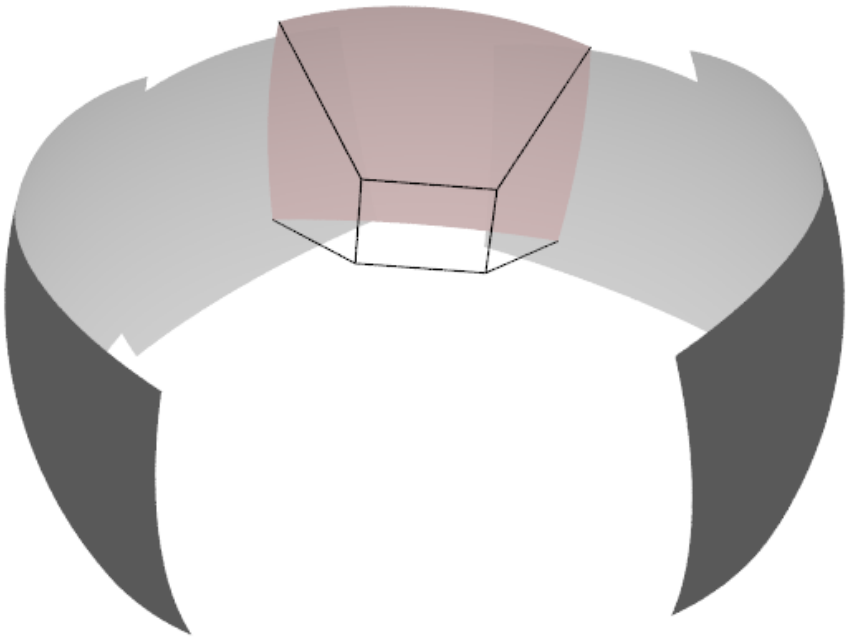


Figure 5.3: A keyframe is selected from the mosaic map based on proximity to our estimated location. Local alignment is performed between the keyframe and the live camera image in order to estimate the live camera pose.

5.2.3 Direct Joint Global Optimisation

Joint global optimisation of all keyframes of the map and camera intrinsics occurs either interleaved with tracking on the graphics card after tracking is complete, or in parallel in a second thread on the CPU. We approach global optimisation within the same framework as tracking: first defining an objective function which relates our complete state vector directly to the error induced by each overlapping pair of image pixels We then take an ESM formulation to iteratively reduce the error within a Gauss-Newton minimisation.

We parameterise updates to keyframe poses $\mathtt{R}^{wi} \in \mathbb{SO}(3)$ through the Lie algebra $\mathfrak{so}_3$ as we do for local frame to frame rotation tracking. Assuming that our camera has square pixels, $f_u = f_v$, we write the intrinsics matrix $\mathtt{K}$ as a function of a three vector $(f, u_0, v_0)^\top$. If we wish to allow the aspect ratio of pixels to vary, or to

Figure 5.4: A mosaic is constructed incrementally, estimating camera pose live but saving sparse historic video frames when the overlap with the current mosaic drops below a threshold. Keyframes are shown outlined in red. Loop closure can be seen to tighten the map after a full 360 degree trajectory has been closed.

estimate pixel skew, we may write $\mathtt{K}$ as a function of a four or five vector instead. We will progress with the three vector parameterisation and formulate updates to the camera's intrinsic parameters by $k \in \mathbb{R}^3$, through exponentiation. This formulation allows us to work within the forward-compositional Lucas-Kanade framework by defining a compositional update $\mathtt{K}(k)$ on $\hat{\mathtt{K}}$, shown in Equations 5.1 and 5.2. Notice $k = \mathbf{0}$ represents no change to the intrinsics.

$$\mathtt{K}(k) = \begin{pmatrix} e^{k_0} & 0 & e^{k_1} \\ 0 & e^{k_0} & e^{k_2} \\ 0 & 0 & 1 \end{pmatrix}, \tag{5.1}$$

$$\hat{\mathtt{K}} \leftarrow \hat{\mathtt{K}} \circ \mathtt{K}(k), \tag{5.2}$$

where $\mathtt{A} \circ \mathtt{B}$ represents the entry-wise scalar product (Hadamard product) between matrices $\mathtt{A}$ and $\mathtt{B}$ of equal dimension.

For N keyframes, our update vector x can be decomposed into rotation parameters, $r_i \in \mathfrak{so}_3, i \in \{1..(N-1)\}$, and intrinsic parameters k, such that $x = (k, r_1, r_2, ...r_{(N-1)})$. Notice that for N keyframes we have $N-1$ poses to optimise, with the first fixed to prevent parameter drift. The objective function which we now wish to minimise considers all pixels between all pairs of overlapping keyframes, where Ω_j^i is the set of all pixels in image $\mathtt{I}^j$ which are currently predicted to reside within image $\mathtt{I}^i$:

$$F(x) = \frac{1}{2} \sum_{j=0}^{N-1} \sum_{i=j+1}^{N-1} \sum_{p_j \in \Omega_j^i} \left[\mathtt{I}^i\left(\mathtt{H}^{ij}(x) p_j\right) - \mathtt{I}^j\left(p_j\right) \right]^2, \tag{5.3}$$

$$\mathtt{H}^{ij}(x) = \hat{\mathtt{K}} \circ \mathtt{K}(k) \mathtt{R}^{ij}(r_i, r_j) (\hat{\mathtt{K}} \circ \mathtt{K}(k))^{-1}, \tag{5.4}$$

$$\mathtt{R}^{ij}(r_i, r_j) = (\hat{\mathtt{R}}^{wi} \mathtt{R}^{wi}(r_i))^\top \hat{\mathtt{R}}^{wj} \mathtt{R}^{wj}(r_j). \tag{5.5}$$

Notice how the relative position of two keyframes depends on the update parameters of both the keyframes. The induced homography is a function of these and the update to the camera intrinsics.

We calculate the incremental minimiser $\mathbf{x}^\circ = \arg\min_{\mathbf{x}} F(\mathbf{x})$ of Equation 5.3 using the ESM machinery from Section 4.3.6. We have two implementations of this minimisation, one which runs on the CPU and another which uses the GPU. The CPU version is able to run continuously in a parallel thread to maintain the map with iterations which can take a significant period of time. The most costly aspect of estimating $\mathbf{x}^\circ$ is in constructing the normal equations which must be solved. On machines with more that two cores, this system can be trivially parallelised further, computing the contributions from each keyframe pair in separate threads.

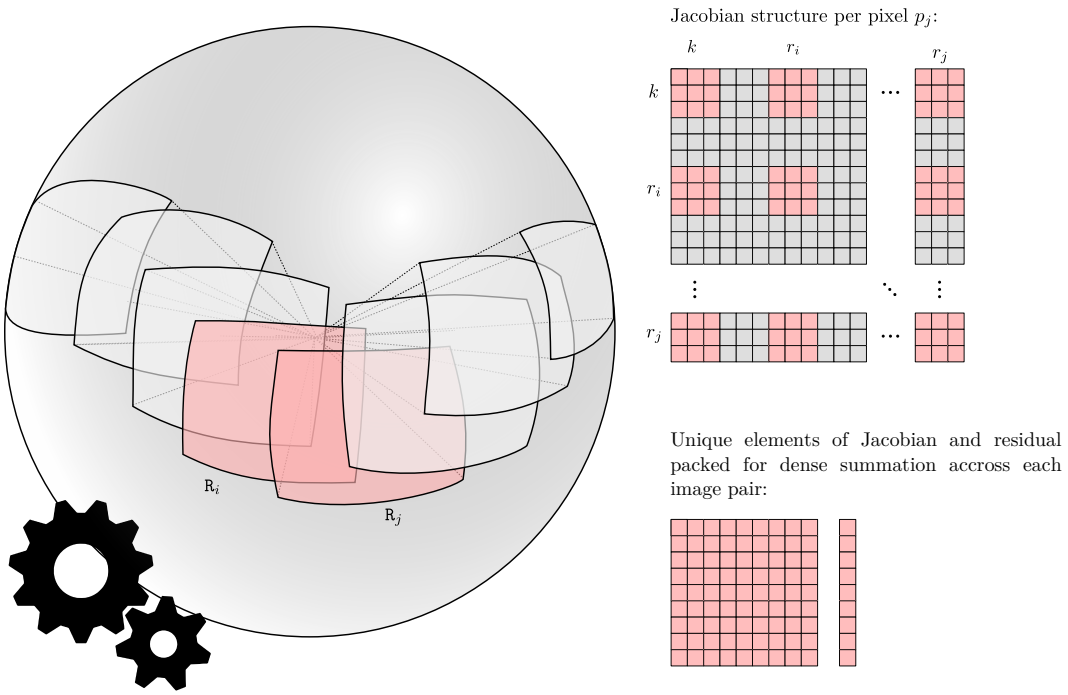


Figure 5.5: Joint global optimisation of the entire map requires that we calculate the matrix $\mathtt{J}^\top\mathtt{J}$, of size $(4 + (N-1) * 3)^2$, for a mosaic of N keyframes. For efficient computation, we must consider the sparsity pattern when evaluating the contribution for a pair of keyframes; non-zero elements are shown in red.

Due to the cost of context switching on current graphics hardware, our GPU implementation must be interleaved with tracking, which also runs on the GPU, for machines with a single graphics card. Since tracking must complete at frame-rate, a single iteration of the global optimisation must be fast in order to be interleaved in this way, running as many iterations as time permits before the next video frame arrives. In order to permit efficient computation on the graphics card, we must consider the sparsity of contributions to the normal equations per image pair. Taking

a single overlapping pair of images at a time, we compute the contributions per pixel to $\mathtt{J}^\top \mathtt{J}$ and $\mathtt{J}^\top f(\mathbf{0})$ as a compacted 9×9 and 9×1 block respectively corresponding to the update parameters for camera intrinsics (which are common across all keyframes) and the update parameters for the pose of the two keyframes (Figure 5.5). For our GPU implementation, taking the linear system's sparsity into consideration lets us place the per-pixel contributions into extremely fast GPU shared memory — these can be summed very quickly within a parallel reduction since the size of the compacted system is small and the memory fast.

5.2.4 Recovery from Tracking Loss

Regardless of how well a tracking system performs, there are frequently times when assumptions are badly violated and tracking cannot continue. Detecting when tracking has failed can be tricky, but it is important in order to prevent corruption of the map and to inform the user that action needs to be taken in order to continue.

Examples of where tracking might fail include fast motion which may degrade the image through motion blur so severely that images cannot be matched accurately to one another. Rapid camera acceleration can also break smooth motion assumptions, perhaps preventing keyframes from being added before the current mapping area is left or such that localisation falls into a local minima. The kidnapped robot problem may also be an issue — for mosaicing, the camera may be covered temporarily before again observing the scene.

For detecting when we are lost, we record image pixel variance for each keyframe which we use to normalise the mean absolute photometric error when tracking with respect to this keyframe. Two thresholds with respect to this metric determine when tracking is poor, and no new keyframes should be added if it can be avoided, and when tracking is lost.

The feature-based relocalisation method of Williams *et al.* [131] takes each observable salient image patch in the current lost frame, and generates a set of potential corresponding features from the map using *randomised trees* [68]. As new features are added to the map, a set of warped patches reflecting possible homographic viewpoint changes are generated, associated with the original map feature and then integrated into the randomised trees as training data. This quite expensive training

stage enables fast lookup times per feature when lost. From the match hypotheses generated from the randomised trees, RANSAC is used to look for a geometrically consistent set.

In the related SLAM problem of loop closure detection, Angeli *et al.* [4] describe a fast method instead based on *bags of visual words*, summarising groups of observable features which are stored as the system explores. New frames generate new bags of words, which can be efficiently compared to previous frames using a reverse index. The probability distributions over occurrences of visual words are stored in order to accurately reflect the probability that two frames are actually the same. Cummins and Newman describe a similar scheme in their paper, FABMAP [23], generalising match likelihood to probabilities of co-occurrence.

We have provided our SLAM system with a straightforward relocalisation capability similar in spirit to the 'small blurry image' method of PTAM [62] but which directly takes advantage of the main ESM pose estimate algorithm. If the camera becomes lost then we aim to recover a pose estimate simply by attempting ESM pose estimation from a number of seed locations visible in our current mosaic, starting at the smallest image size in an image pyramid (Figure 5.6). Of the estimated warp parameters obtained, we refine the most photo-consistent estimate by performing more ESM iterations at higher resolutions in the pyramid. We use the poses of our keyframes as seed locations, but indeed any regular sample would be equally valid.

Figure 5.6: 'Small blurry image' relocalisation. Small images can be used to quickly test hypothesised locations given a generative model.

Computation time for relocalisation is proportional to the number of seed locations. For spherical mosaics, relocalisation need not be costly. When we determine

that we're lost, we run the relocalisation procedure on one in ten frames. This method operates well in environments with low perceptual aliasing.

5.2.5 Visualisation

We will describe some of the common approaches for visualising rotational mosaics, including projective, spherical, cylindrical, polar and cube projections. These different methods describe various surfaces with which to consider intersecting pixel rays originating from the optic centre of the camera. Given a projection surface, we must also specify how this is unwrapped into a two-dimensional image via a second (not necessarily perspective) projection.

Certain projection surfaces, such as planes and cylinders can be trivially unrolled into image space. Perhaps the most intuitive projective surface, the sphere, is quite difficult to unroll, since it is not possible to do so without introducing distortions, such as spatially varying scales of size or varying absolute directions. In the mosaicing literature, a spherical mosaic typically refers to the projection of the scene onto a sphere and the unrolling of this sphere by latitude and longitude. In cartography, map makers have long studied many such projections from sphere to plane in order to visualise the surface of the earth on paper.

Since we aim to create mosaics interactively at frame-rate, it is also important to render them at frame-rate. We make use of Cg shaders (Section 2.5.4) to enable us to visualise the full quality, blended mosaic live, and for correctly sampling from the constituent keyframes.

Interactive Perspective Panorama

For rendering an interactive perspective panorama, where a user is free to navigate a scene by rotating in a fixed position, we treat our virtual (OpenGL) camera much like a keyframe, positioned at the origin and parameterised by the camera to world transform $\mathtt{R}^{wc}$. We can map image space coordinates from our OpenGL viewport to a keyframe k by composing the homography $\mathtt{H}^{kc} = \mathtt{K}\mathtt{R}^{wk\top}\mathtt{R}^{wc}\mathtt{K}^{-1}$.

We use a shader which we invoke once for each keyframe within the field of view of the virtual camera, passing in as a parameter the homography $\mathtt{H}^{kc}$ which enables

us to place the keyframe within the viewport. This shader, operating per pixel in the output image, simply adds the keyframe's colour value to the colour already in the framebuffer associated with the viewport. Additionally, it adds 1.0 to the alpha channel for the pixel which serves as a counter.

Finally, we invoke another normalisation shader, which simply divides the Red, Green and Blue channels by the alpha channel. The result is a panorama where each keyframe is displayed blended with equal weight. One of the nice aspects of this method is that image fusion occurs in the space of the viewport. This means that each keyframe, whose pixel data is not sampled to the same 'grid' in viewport space, gets mixed and can form an image of higher resolution than the constituent images, with less pixel noise and reduced aliasing artefacts.

Cylindrical Panorama

Cylindrical panoramas are formed by intersecting the keyframe image pixels with the surface of a cylinder centred at the capturing camera's optic centre (Figure 5.7). We again employ vertex shaders to render the cylindrical panorama live, letting the user adjust the view and explore interactively.

Within the shader, the u and v viewport coordinates are interpreted as yaw ($\psi \in [-\pi, +\pi]$) and height ($h \in \mathbb{R}$, adjustable range), defining a unique point on the surface of a cylinder.

For each keyframe, we invoke the shader, where, for each pixel in the destination panorama we compute the desired image ray described by the unit vector $\hat{\mathbf{r}}_{\text{cyl}}$,

$$\hat{\mathbf{r}}_{\text{cyl}}(\psi, h) = (\cos\psi, h, \sin\psi)^{\top} . \tag{5.6}$$

This ray is transferred into the frame of reference of the keyframe using the virtual camera to keyframe rotation matrix, $\mathtt{R}^{kc}$, which is uploaded as a parameter to the shader. Finally, the camera intrinsic matrix can be used to map this to keyframe image-space coordinates. Given this correspondence, we proceed as with

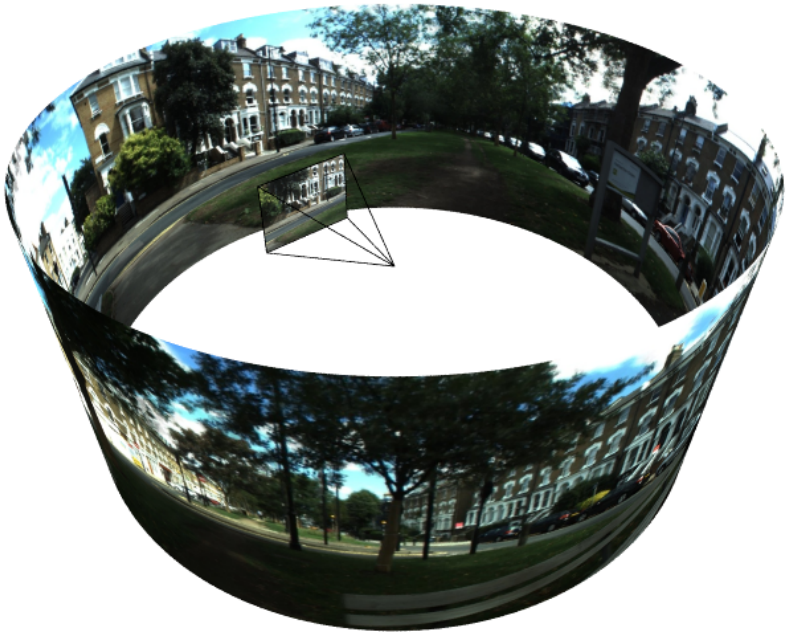


Figure 5.7: Virtual camera in centre of cylindrical mosaic.

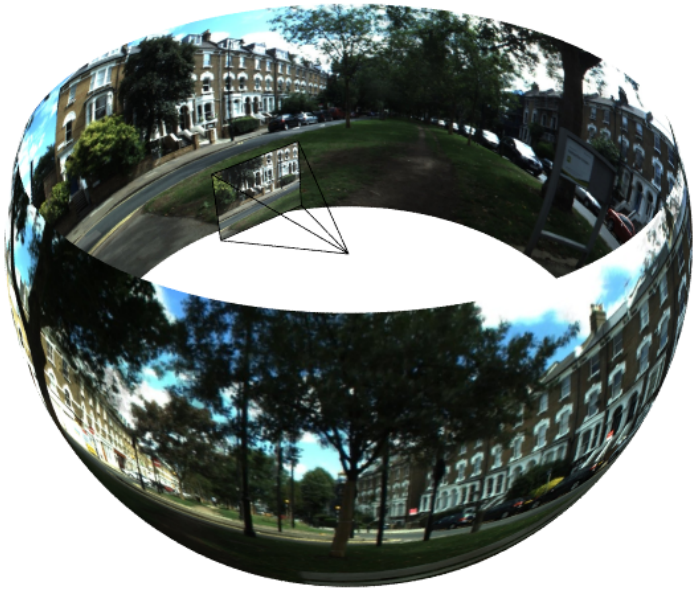


Figure 5.8: Virtual camera in centre of spherical mosaic.

the interactive projective panorama:

$$\mathtt{I}^{\text{cyl}}(\psi, h) = \frac{1}{N} \sum_{k}^{1..N} \mathtt{I}^{k} \left(\pi \left(\mathtt{KR}^{kc} \hat{\mathbf{r}}_{\text{cyl}}(\psi, h) \right) \right). \tag{5.7}$$

Although simple, a disadvantage of this projection is that the area directly above and directly below the cylinder cannot be represented, and those parts of the world around these points occupy a disproportionate amount of space within the final mosaic. We must also decide on an appropriate value for h. For this reason, cylindrical projections are typically only appropriate for panoramas generated from rotations about a single axis.

Spherical Panorama

To create spherical panoramas, we intersect the keyframe image pixels with the surface of a sphere using similar machinery as for cylindrical panoramas (Figure 5.8). Within the shader, the u and v viewport coordinates are interpreted as yaw ($\psi \in [-\pi, +\pi]$) and pitch ($\theta \in [-\frac{\pi}{2}, +\frac{\pi}{2}]$) respectively.

For each keyframe, we invoke the shader, where, for each pixel we then compute the desired image ray described by the unit vector $\hat{\mathbf{r}}_{\text{sph}}$,

$$\hat{\mathbf{r}}_{\text{sph}} = (\cos\theta \cos\psi, \sin\theta, \cos\theta \sin\psi)^{\top}. \tag{5.8}$$

Once again, this ray allows us to transfer spherical coordinates to keyframe image coordinates. We sum the contributions per pixel from each keyframe and normalise by the alpha component as before:

$$\mathtt{I}^{\text{sph}}(\psi, \theta) = \frac{1}{N} \sum_{k}^{1..N} \mathtt{I}^{k} \left(\pi \left(\mathtt{KR}^{ks} \hat{\mathbf{r}}_{\text{sph}}(\psi, h) \right) \right). \tag{5.9}$$

Using the spherical projection enables full hemisphere and complete sphere mosaics to be generated and visualised with finite image size (e.g. Figure 5.2). This is in contrast to cylindrical projections which require infinite height to represent north and south poles. Spherical projections contain no singularities, though image data

at the poles is still given a disproportional area due to the parameterisation which corresponds to the unravelling and flattening of the shell of a sphere.

Polar Panorama

Polar image coordinates can be combined with a spherical mapping to produce interesting panoramas (Figure 5.9). Taking the output mosaic of size $w \times h$ pixels, the $(u, v)^\top$ coordinates of a pixel can instead be represented in polar form with origin in the image centre by $(\varphi, r)^\top$:

$$\begin{pmatrix} \varphi \\ r \end{pmatrix} = \begin{pmatrix} atan2\left(u_c, v_c\right) \\ \sqrt{u_c^2 + v_c^2} \end{pmatrix}, \qquad \begin{pmatrix} u_c \\ v_c \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} - \begin{pmatrix} \frac{w}{2} \\ \frac{h}{2} \end{pmatrix} \tag{5.10}$$

After scaling to fit the appropriate interval, we can interpret φ as longitude and r as latitude; $\psi = \varphi,\ \theta = \pi r - \frac{\pi}{2}$. Finally, we continue as with the spherical projection using Equations 5.8 and 5.9. Notice that a straight line from the mosaic image centre represents a vertical sweep from south to north pole, such that equal inclinations form concentric circles in the image — in outdoor sequences the horizon is often the most noticeable and creates quite distinct 'tiny planet' effect images.

Cube Mapping

In computer graphics, a cube map is a popular way to represent full panoramas as they can be efficiently rendered live using the existing fixed graphics pipeline. A cube map consists of six different square perspective images taken orthogonally to one another capturing the projection of the world onto the size faces of a cube. With each image having a 90° field of view, the relative size of pixels in terms of the solid angle they represent remains fairly uniform, with those at the edges representing around 1.4 times the solid angle as those in the centre. This is fairer than any of the previous mappings described for full sphere coverage.

If each square face has dimension $S \times S$ pixels, the camera intrinsics matrix for

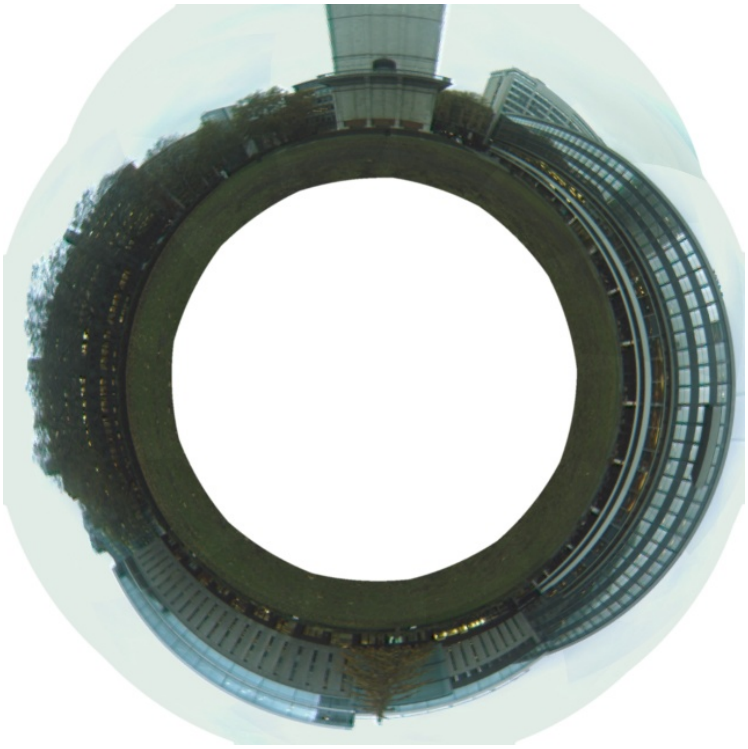


Figure 5.9: Applying a spherical projection in polar image coordinates produces an interesting effect.

the synthetic views can be written:

$$\mathtt{K}_{\text{cube}} = \begin{pmatrix} \frac{S}{2} & 0 & \frac{S}{2} \\ 0 & \frac{S}{2} & \frac{S}{2} \\ 0 & 0 & 1 \end{pmatrix}. \tag{5.11}$$

As for the interactive projective projection, we can write down the homography relating pixels in each face of the cube to those in each keyframe:

$$\mathtt{H}^{kf} = \mathtt{K}\mathtt{R}^{wk\top}\mathtt{R}^{wf}\mathtt{K}_{\text{cube}}^{-1}. \tag{5.12}$$

These homographies relating keyframe images to a cube face allow us to generate a component of the cube map through blending as described previously.

Cube maps can be trivially used in OpenGL as environment maps for special lighting effects. Figure 5.10 shows the OpenGL teapot rendered with a reflective

metallic surface from within an office environment.

Figure 5.10: Environment Mapped Teapot rendered by supplying a cube map to OpenGL.

5.2.6 Global Consistency and Intrinsics Refinement

For evaluation of global registration, we present several spherically projected 360° panoramas (Figures 5.11, 5.12, 5.15) captured with two different cameras, and with two different lenses for each camera. They are constructed by blending every keyframe of the map with equal weight, as described in Section 5.2.5, enabling us to easily visualise the quality of their alignment.

For areas of the mosaic formed from multiple images, pixel noise is significantly reduced, and the mosaic appears smoother. The different sampling pattern of keyframes and sub-pixel accuracy we achieve in alignment combine to create a supersampling, efficiently rendered in real-time on the graphics card.

Figures 5.13 and 5.14 demonstrates the importance of our joint estimation of camera intrinsic parameters, even for pre-calibrated cameras. Starting with intrinsics

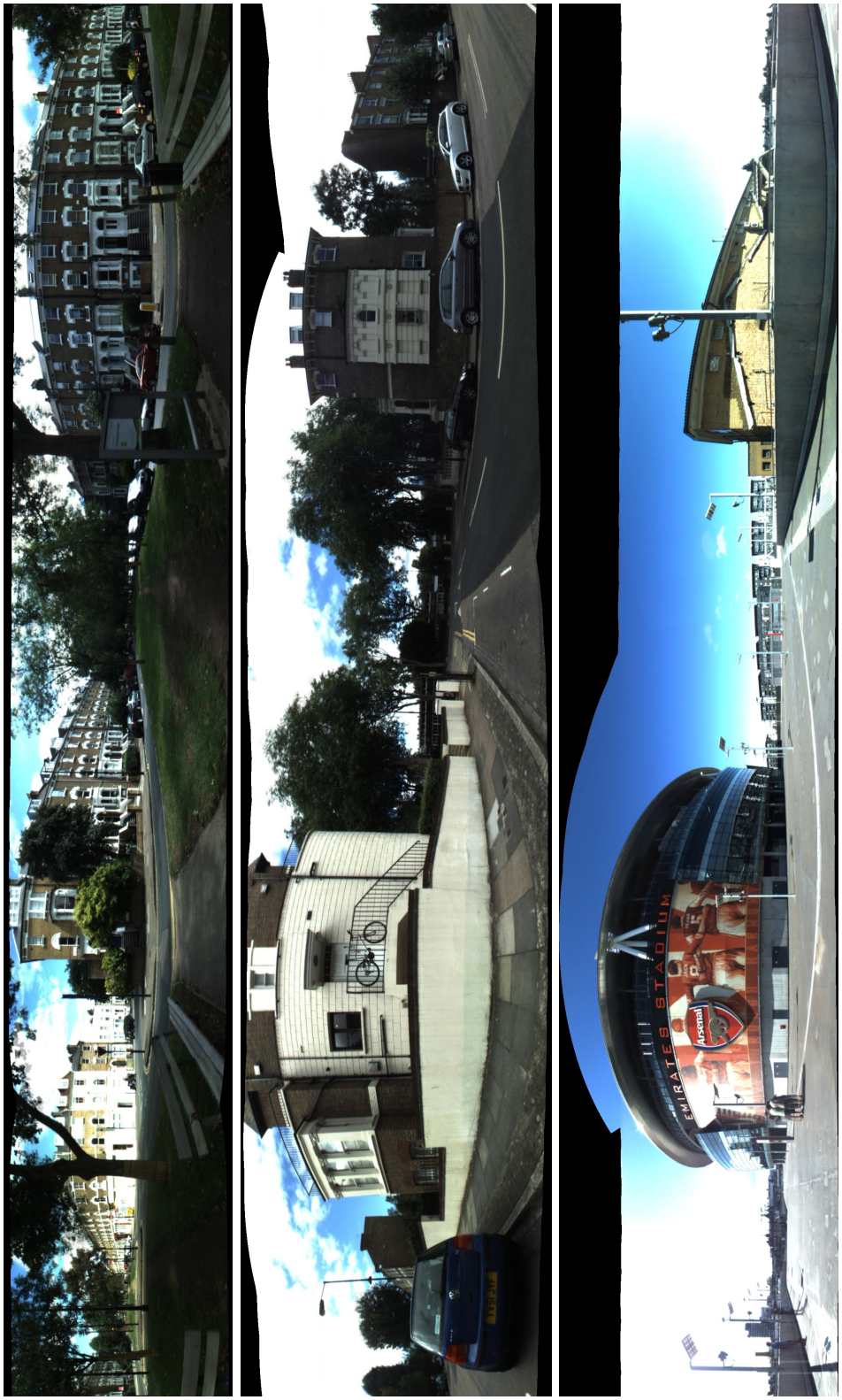


Figure 5.11: Outdoor spherical mosaics.

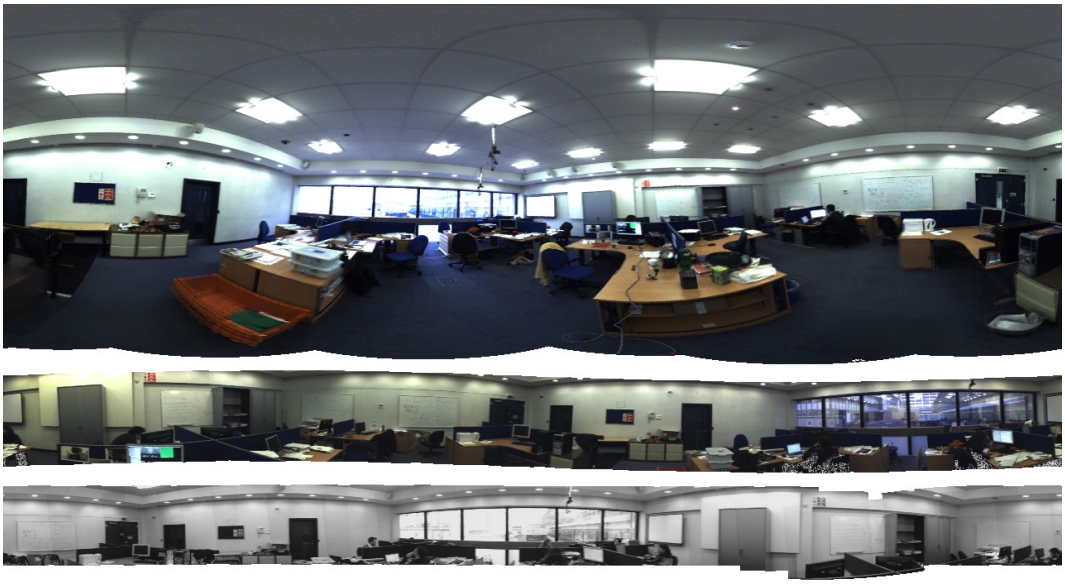


Figure 5.12: 360° spherically-projected panoramas for three indoor sequences, taken with different lenses. Point Grey Flea2, 70° FOV wide angle (top, close to full sphere including full hemispherical upward coverage, 27 keyframes), 50° FOV TV Lens (middle, single horizontal loop trajectory, 17 keyframes), and Unibrain 45° FOV Standard lens (bottom, single horizontal loop trajectory, 19 keyframes).

estimated from a third party camera calibration tool, and continuing with no intrinsics optimisation, the first mosaic in this figure appears fuzzy. Upon inspection we can see that the estimated loop length is longer than the actual length (in pixels), causing the images to bunch up (the enlargement of the whiteboard helps to convey this point). This is caused by intrinsic parameters which are wider than the actual camera. The second mosaic in this figure is the result of allowing our algorithm to optimise intrinsics as well as pose parameters (from the starting point of the first mosaic).

The mosaics in Figure 5.12 were generated from three different lenses, all at 640×480 resolution, and initialised with generic intrinsic calibration (nearest 10° FOV and central principal point). Table 5.2.6 shows the initial horizontal field of view, which was based on our knowledge of the lens, and the converged field of view estimate after a full loop was completed for these sequences.

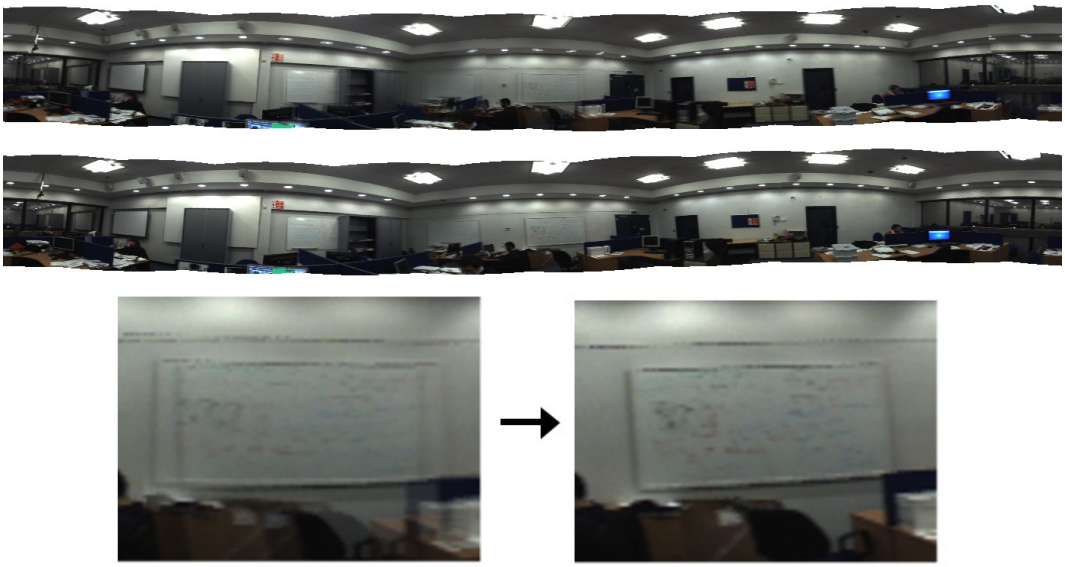


Figure 5.13: Mosaicing with fixed intrinsics estimated from a third party calibration tool (top), compared against enabling live intrinsics estimation (middle). An enlargement of the whiteboard from the two mosaics, emphasising improvement in alignment, is shown at the bottom. The whiteboard is representative of several areas of the mosaic.

Camera	Lens	Lens Quality	Stated FOV	Refined FOV
PtGrey Flea2	Wide	Good	70°	69.42°
PtGrey Flea2	TV Lens	Fair	50°	51.43°
Unibrain Fire-i	Standard	Poor	50°	45.56°

Table 5.1: Calibration Refinement results for Different Cameras and Lenses. Calibration initialised from Quoted Horizontal Field of View (FOV), and refined by mosaicing cylindrical loops from 640×480 indoor sequences.

5.2.7 Convergence to Global Minimum

The results from mosaicing based on poor initial intrinsics (Figure 5.12) help to motivate that our system has useful convergence properties. By including intrinsics in our optimisation, we help to enable loop closure by increasing the accuracy of our pose estimate when we come to complete a loop. By completing a loop too soon, or too early, we are more likely to fall into local minima — especially if perceptual aliasing in this area is high.

Figure 5.15 shows an outdoor mosaic generated from rapid hand-held motion of a Unibrain webcam with a wide angle lens. Note that in this experiment the pure

Figure 5.14: Mosaicing with (middle) and without (top) live intrinsic refinement. Enlargements of these mosaics are shown (bottom) highlighting the improved registration as error is propagated more accurately throughout the 360 degree trajectory.

rotation assumption was approximately satisfied without a tripod due to the large distance to the scene. This scene contains high perceptual aliasing in the windows and building pillars, making loop closure difficult. For this sequence, we were unable to converge to a globally consistent mosaic from our generic 80° FOV calibration parameters. Instead, we started from the parameters estimated from a third party calibration tool.

Figure 5.15: 360° Tower panorama from 21 keyframes (live hand held Unibrain webcam, 320×240 resolution), shown in horizontally and vertically-oriented cylindrical projection. Note the vertical hole due to poor texture and cloud movement in the sky.

Time to convergence is another important evaluation criterion. Each iteration in our global minimisation is costly — forming the linear system from image data dominates computational time. Actually solving this system is cheap since spherical mosaics require only a relatively small number of keyframes. For this reason, computation time scales linearly with the number of pairs of overlapping pixels. For N keyframes, depending on keyframe alignment, this has a worst case complexity of $\mathrm{O}(N^2)$. In practice, our system achieves convergence within time on the order of seconds after completing a loop; often less than one second when a wide angle lens means that the number of keyframes to span a loop is low.

Figure 5.16 shows some stills from the construction of the outdoor 'Tower' mosaic during loop closure, visualised with a perspective camera rendering. This sequence demonstrates the quality of local image alignment, loop closing, and global relaxation. The left image of the three shows the final moment before the 360° path of the camera was completed, and the first and last keyframes were bridged by another keyframe (middle image). The loop is not closed immediately; this only happens when the system is confident (last image). Prior to loop closure, we can see the image misalignment by examining the building pillars. Measuring the distance between one pillar in the constituent keyframes demonstrates that rotational drift prior to

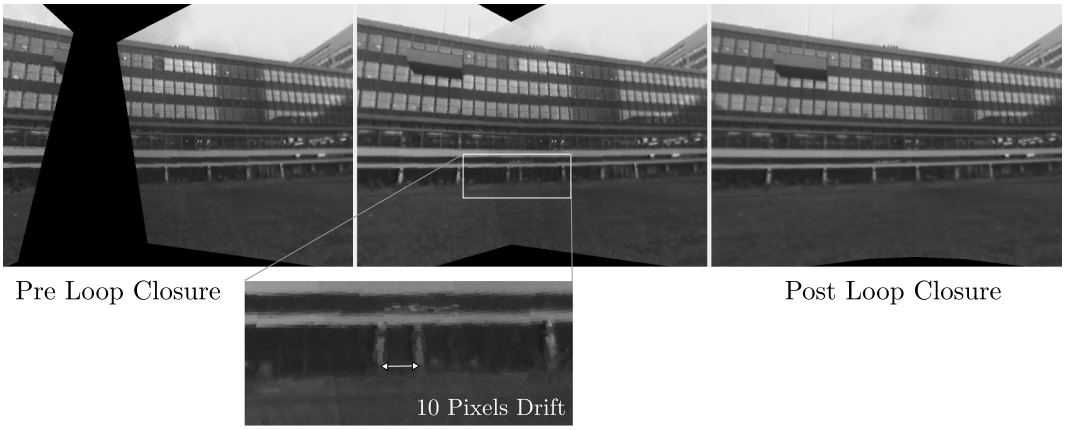


Figure 5.16: Frames immediately before (left, middle) and after closure (right) from a 360° sequence containing high perceptual aliasing. Prior to loop closure, the first and last keyframes were misaligned by 10 pixels (just under 3°).

loop closure is just 10 pixels ($< 3°$) for this sequence.

5.3 Real-time Planar Mosaicing / SLAM

Within this section, we will look at real-time pose estimation and globally consistent dense mosaicing from a single plane with infinite extent. We will consider a camera that can move freely in three dimensions with six degrees of freedom whilst observing the plane, and we will allow the video sequence to begin from any orientation.

In this setting, as with rotational mosaicing, each frame from the video is related to another via a plane-induced homography. For rotational mosaicing, this homography is a function of rotation and camera intrinsics; the scene is simply a 'plane at infinity' with no parameters. For planar mosaicing, the frames are additionally related via the global plane parameters.

5.3.1 Background

One of the simplest methods for creating planar mosaics from video is to estimate a planar homography between consecutive video frames and then warp each image into a common local image coordinate system. As discussed previously when considering spherical mosaicing, Irani *et al.* described a system which did just this, estimating

inter-frame homography parameters via a direct, every-pixel method [54]. This composition of image transformations however will be subject to accumulating image space transfer error. One simple method to reduce this error is to always register the current video frame with respect to an accumulated model, consisting of some combination of all previous video frames. Following registration, the frame is also included in the model, and tracking continues. If the global model consists purely of a composited image, taking contributions from each frame sequentially, we are not guaranteed a globally optimal mosaic since the positions of constituent images can only be aligned with respect to previous frames and not future ones. Since this model is inherently rigid, it also means that local corrections cannot be made during loop closure, though global image space transformations such as scaling or sheering can.

In his thesis concerned with visual SLAM through planar tracking, Unnikrishnan described a pose graph formulation to globally optimise planar observations [130]. He assumes however that the angle of the camera relative to the plane and the distance from the plane is known a-priori, parameterising motion in the plane's coordinate system with three degrees of freedom. This is a significant simplification since the removal of any common global plane parameter means video images are related via just a simple rigid body transformation, and standard pose-graph optimisations such as those based on [75] can be employed.

Sawhney *et al.* described a method for planar mosaicing which is able to generate globally consistent mosaics within a framework which is in principle quite similar to our own [108]. Relating image pairs locally via homographies which they estimate using a direct approach, they define a separate global point wise transfer minimisation of image corners and mid-points to refine the local inter-image homographies. This optimisation does not require feature data-association and correspondence, but rather takes the sum of transfer error induced by transforming fixed image points through different but similar homographies.

We will describe a system for determining the 6 DOF pose of a camera relative to a planar surface live, whilst constructing a textured mosaic of this surface. This is in contrast to existing systems which either do not create self-consistent mosaics, or produce a globally consistent mosaic as a batch process.

The method of Sawhney [108] could be posed incrementally, splitting live frame

registration and global optimisation, but the cost of estimating full homographies at frame-rate is higher than our approach, and live camera motion cannot be obtained directly. Their global optimisation is also over parameterised and does not enable trivial planar orthographic projections of the planar surface, since the plane parameters remain unknown.

Perhaps the most relevant work is that of Silveira *et al.*, who describe a real-time system for 3D visual SLAM which uses the direct image-based method of ESM to jointly estimate the live drifting camera pose and the parameters of small planar regions within the scene to obtain visual odometry [115]. Although their aim is clearly to use a large percentage of the image, real-time constraints only let them use around half of the pixels. We take large influence from their work, and in particular use their planar parameterisation.

5.3.2 Overview

As for rotational mosaicing, we will decouple tracking and map maintenance into parallel tasks which can be executed simultaneously on multiple processing cores. We will parameterise our *map* by the poses of distinct *keyframes* and by the parameters of a single plane of infinite extent.

By maintaining the global map promptly and offering a *best estimate* at any time, we can actually simplify camera tracking by considering the global plane parameters and reference keyframe fixed, minimising only the error with respect to the parameters of motion from one frame to the next. This is in contrast to previous methods which require the estimation of a full homography between frames, and an increase from six parameters to eight. This live map with globally adjusted plane parameters also allows us to provide globally adjusted camera poses live, with real-time rendering of the mosaic in the true planar orthographic frame of reference.

5.3.3 Optimising Pose

Given a nearby keyframe which we will call the *reference keyframe*, r, with plane parameters $\mathbf{n}_r$ defined in that frame of reference, we can compute our current *live* pose $\mathtt{T}^{wl}$ by finding a the relative transform $\mathtt{T}^{lr}$.

Using a now very familiar methodology, we wish to minimise the photometric error induced by our current estimated motion parameters $\hat{\mathtt{T}}^{lr}$ by iteratively considering an incremental update $\mathtt{T}(\boldsymbol{\psi})$, as is done in [10]. We define the following cost function describing the error we wish to minimise.

$$F(\boldsymbol{\psi}) = \frac{1}{2} \sum_{\mathbf{u}_r \in \Omega_r} \left(\mathtt{I}^l \left(\pi \left(\mathtt{H}^{lr}(\boldsymbol{\psi}) \dot{\mathbf{u}}_r \right) \right) - \mathtt{I}^r \left(\mathbf{u}_r \right) \right)^2, \tag{5.13}$$

where:

$$\mathtt{H}^{lr}(\boldsymbol{\psi}) = \mathtt{K} \hat{\mathtt{T}}^{lr} \mathtt{T}(\boldsymbol{\psi}) \left(\mathtt{I}| - \mathbf{n}_r \right) \mathtt{K}^{-1}. \tag{5.14}$$

We solve this system using ESM as detailed in Section 4.3.6. The partial derivatives specific to this cost function are shown below.

$$\frac{\partial \mathtt{H}^{lr}(\boldsymbol{\psi})}{\partial \boldsymbol{\psi}_i} = \mathtt{K} \hat{\mathtt{T}}^{lr} \frac{\partial \mathtt{T}(\boldsymbol{\psi})}{\partial \boldsymbol{\psi}_i} \left(\mathtt{I}| - \mathbf{n}_r \right) \mathtt{K}^{-1}. \tag{5.15}$$

$$\frac{\partial \mathtt{T}(\boldsymbol{\psi})}{\partial \boldsymbol{\psi}_i} \Big|_{\boldsymbol{\psi}=\mathbf{0}} = \underset{\mathbb{SE}(3)}{\mathrm{gen}_i}. \tag{5.16}$$

5.3.4 Optimising Pose and Plane

Estimating the pose of the camera directly is possible when the structure of the planar scene is known, as we just described. When the system is first started however and we have no data, the scene structure is unknown. In practice, it is often possible to start by optimising only pose assuming fronto-parallel structure, adding keyframes as with a fully bootstrapped system. Global optimisation of these frames can then proceed to correct this poor initialisation. Provided this correction takes place promptly, the tracking system can continue before the baseline is large, where the error induced by the incorrect model is magnified.

In cases where we require more accurate pose estimation right from the start, or where the plane is viewed from such an oblique angle that fronto-parallel initialisation causes tracking to fail immediately, we can optimise the pose and plane jointly to correctly model the observed image warping between frames.

Between two projective images of a plane, a general homography, $\mathtt{H} \in \mathbb{SL}(3)$ can describe the pixel mapping from one image to the other as we have seen before. This homography can be parameterised minimally by its Lie algebra $\mathfrak{sl}_3 \in \mathbb{R}^8$, but for our planar SLAM setting, we are also interested in the live pose of the camera, which is difficult to extract from the homography itself. Extracting camera pose can be achieved by decomposition of $\mathtt{H}$, but results suffer in accuracy [38][138][79].

We use the minimal plane parameterisation suggested by Silveira *et al.* [115] which can implicitly enforce cheirality and allows us to use a compositional cost function as we have presented previously, and solve it using ESM. Alternative parameterisations such as that used by Habbecke and Kobbelt require a forward-compositional formulation [45][46]. We therefore iteratively solve for incremental updates to $\mathbf{x} = (\boldsymbol{\psi}, \mathbf{y})^\top$, consisting of both pose ($\boldsymbol{\psi}$) and plane ($\mathbf{y}$) parameters, with the following cost function:

$$F(\mathbf{x}) = \frac{1}{2} \sum_{\mathbf{u}_r \in \Omega_r} \left(\mathtt{I}^l \left(\pi \left(\mathtt{H}^{lr}(\mathbf{x}) \dot{\mathbf{u}}_r \right) \right) - \mathtt{I}^r \left(\mathbf{u}_r \right) \right)^2, \tag{5.17}$$

where:

$$\mathtt{H}^{lr}(\mathbf{x}) = \mathtt{K} \left(\hat{\mathtt{R}}^{lr} \;\middle|\; \hat{\mathbf{r}}_l \right) \mathtt{T}(\boldsymbol{\psi}) \left(\mathtt{I} \;\middle|\; -\mathbf{n}_r(\mathbf{y}) \right)^\top \mathtt{K}^{-1}. \tag{5.18}$$

Considering a pixel $\mathbf{u}$ intersecting a plane $\mathbf{n} = \frac{\hat{\mathbf{n}}}{d}$ at point $\mathbf{P} = (X, Y, Z)^\top$, the z-axis depth of pixel $\mathbf{u}$ is Z, as defined earlier in Equation 2.25.

Instead of parameterising a plane in terms of $\mathbf{N}$ or $\mathbf{n}$, Silveira *et al.* propose to parameterise the plane in terms of the inverse depth value ($\frac{1}{Z}$) at three fixed non-collinear image locations.

$$\frac{1}{Z} = -\mathbf{n}^\top \mathtt{K}^{-1} \dot{\mathbf{u}}. \tag{5.19}$$

We write the vector of inverse depths for each image location as $\mathbf{z}$, and define our homogeneous set of non-collinear points $\dot{\mathbf{u}}_0, \dot{\mathbf{u}}_1, \dot{\mathbf{u}}_2$, based on image corners, within a matrix $\mathtt{U} \in \mathbb{R}^{3\times3}$:

$$\mathbf{z} = \left(\frac{1}{Z_0}, \frac{1}{Z_1}, \frac{1}{Z_2} \right)^\top, \tag{5.20}$$

$$\mathtt{U} = (\dot{\mathbf{u}}_0, \dot{\mathbf{u}}_1, \dot{\mathbf{u}}_2) = \begin{pmatrix} w, 0, w \\ 0, h, h \\ 1, 1, 1 \end{pmatrix}. \tag{5.21}$$

We can therefore express the vector of inverse depths ($\mathbf{z}$) and the scaled normal ($\mathbf{n}$) parameterisations in terms of one another and a fixed, pre-computable matrix $\mathtt{Q}$. From Equations 5.19, 5.20 and 5.21, we can write:

$$\mathbf{z} = (-\mathbf{n}^\top \mathtt{K}^{-1}\mathtt{U})^\top. \tag{5.22}$$

Writing the constant part as:

$$\mathtt{Q} = -\left(\mathtt{K}^{-1}\mathtt{U}\right)^\top, \tag{5.23}$$

allows us define:

$$\mathbf{z} = \mathtt{Q}\mathbf{n}, \qquad\qquad \mathbf{n} = \mathtt{Q}^{-1}\mathbf{z} \tag{5.24}$$

In order to enable compositional updates on this parameterisation, and enforcing cheirality, an increment $\mathbf{z}(\mathbf{y})$ to our current estimate of the plane parameters $\hat{\mathbf{z}}$, is parameterised further by $\mathbf{y} \in \mathbb{R}^3$ through exponentiation:

$$\mathbf{z}(\mathbf{y}) = (e^{\mathbf{y}_0}, e^{\mathbf{y}_1}, e^{\mathbf{y}_2})^\top \tag{5.25}$$

This allows us to write the incremental version of our plane parameterisation (Equation 5.26) with the appropriate update rule (Equation 5.27).

$$\mathbf{n}(\mathbf{y}) = \mathtt{Q}^{-1}\hat{\mathbf{z}} \circ \mathbf{z}(\mathbf{y}), \tag{5.26}$$

$$\hat{\mathbf{z}} \leftarrow \hat{\mathbf{z}} \circ \mathbf{z}(\mathbf{y}), \tag{5.27}$$

where $\mathtt{A} \circ \mathtt{B}$ represents the entry-wise scalar product (Hadamard product) between matrices $\mathtt{A}$ and $\mathtt{B}$ of equal dimension. This allows us to expand Equation 5.18 and write it more explicitly in terms of our parameterisation:

$$\mathtt{H}^{lr}(\mathbf{x}) = \mathtt{K}\left(\hat{\mathtt{R}}^{lr} \;\middle|\; \hat{\mathbf{r}}_l\right) \mathtt{T}(\boldsymbol{\psi}) \left(\mathtt{I} \;\middle|\; -\mathtt{Q}^{-1}\hat{\mathbf{z}} \circ \mathbf{z}(\mathbf{y})\right)^{\top} \mathtt{K}^{-1}. \tag{5.28}$$

Minimising Equation 5.17 using ESM given this parameterised homography requires the following partial derivatives:

$$\frac{\partial \mathtt{H}^{lr}(\mathbf{x})}{\partial \boldsymbol{\psi}_i} = \mathtt{K}\left(\hat{\mathtt{R}}^{lr} \;\middle|\; \hat{\mathbf{r}}_l\right) \frac{\partial \mathtt{T}(\mathbf{x})}{\partial \boldsymbol{\psi}_i} \left(\mathtt{I} \;\middle|\; -\mathtt{Q}^{-1}\hat{\mathbf{z}}\right)^{\top} \mathtt{K}^{-1}, \tag{5.29}$$

$$\frac{\partial \mathtt{H}^{lr}(\mathbf{x})}{\partial \mathbf{y}_i} = \mathtt{K}\left(\hat{\mathtt{R}}^{lr} \;\middle|\; \hat{\mathbf{r}}_l\right) \left(\mathbf{0} \;\middle|\; -\mathtt{Q}^{-1}\hat{\mathbf{z}}\frac{\partial \mathbf{z}(\mathbf{y})}{\mathbf{y}_i}\right)^{\top} \mathtt{K}^{-1}, \tag{5.30}$$

$$\frac{\partial \mathbf{z}(\mathbf{y})}{\mathbf{y}_0} = (1,0,0)^{\top}, \qquad \frac{\partial \mathbf{z}(\mathbf{y})}{\mathbf{y}_1} = (0,1,0)^{\top}, \qquad \frac{\partial \mathbf{z}(\mathbf{y})}{\mathbf{y}_1} = (0,0,1)^{\top}. \tag{5.31}$$

5.3.5 Efficient Keyframes and Plane Joint Optimisation

In Section 5.2.3 we described a photometric full joint optimisation for spherical mosaicing. This consisted of minimising a cost function over the sum of every pixel-wise difference between overlapping images, parameterised by the pose of each constituent keyframe and by the global camera intrinsics. We showed that this optimisation could be performed fast enough to help close loops metrically and offer interactive mosaic building. We attempted a similar formulation for planar mosaicing, but found real-time constraints a problem — the global estimation of the common plane parameters occurs too slowly at the start, leading to poor tracking performance and potential tracking failures.

For planar SLAM, estimating only pose for the live camera, drifting measurements are amplified over time since inaccurate pose induces incorrect plane parameters in that frame of reference, leading to even worse pose estimation in the subsequent frame. For stability, global correction needs to occur very quickly. This is in contrast to rotational mosaicing, where local alignment depends only on local motion

estimation and camera intrinsics (which are not themselves a function of pose). This leads to a relatively constant level of drift when integrating measurements. For offline planar mosaicing methods which are not interested in the actual pose of the camera, it is enough to relate frames locally by a homography which can reflect the true pixel transformation regardless of any global parameters. Minimising a global error parameterised by unrestricted homographies however leads to an over-parameterisation which does not strictly enforce a single plane across the entire mosaic and perspective distortions may occur.

Deviating from the approach we proposed for spherical mosaicing, we detail a new global minimisation which is based on minimising a transfer error between stored keyframes given a poses and plane parameterisation. This method could equally be adapted for spherical mosaicing to improve the speed performance of that system.

Homographic Transfer Error

In our planar mosaicing scenario, considering that two overlapping video frames are related locally by a homography, the composition of homographies over a chain of images leading back to the original would form the identity matrix if these could be computed accurately enough. Taking estimated homographies between video images as measurements, we can write down an image space *transfer error* E^{cr} between a reference image r and comparison image c based on the integral of pixel displacements between this measured homography, $\mathtt{H}^{cr}$, and the homography induced by plane and camera parameters, $\hat{\mathtt{H}}^{rc}$ (Equation 5.32). Figure 5.17 illustrates the image space transfer error for a single coordinate $\mathbf{u}_r$ in image $\mathtt{I}^r$.

$$E^{cr} = \iint\limits_{\mathbf{u}_r=\binom{u}{v}\in\Omega_r} \left\| \pi\left(\mathtt{H}^{cr}\dot{\mathbf{u}}_r\right) - \pi\left(\hat{\mathtt{H}}^{cr}\dot{\mathbf{u}}_r\right)\right\|_2^2 \mathrm{d}u\,\mathrm{d}v. \tag{5.32}$$

Global Transfer Error Cost Function

Even over a rectangular interval, Equation 5.32's integral is quite complicated to compute analytically, with multiple cases for evaluation. To ease computation, we instead approximate the integral with point-wise samples and formulate with respect to an incremental update on the parameters of motion $\boldsymbol{\psi}_i \in \mathfrak{se}_3, i \in [1 \dots N]$

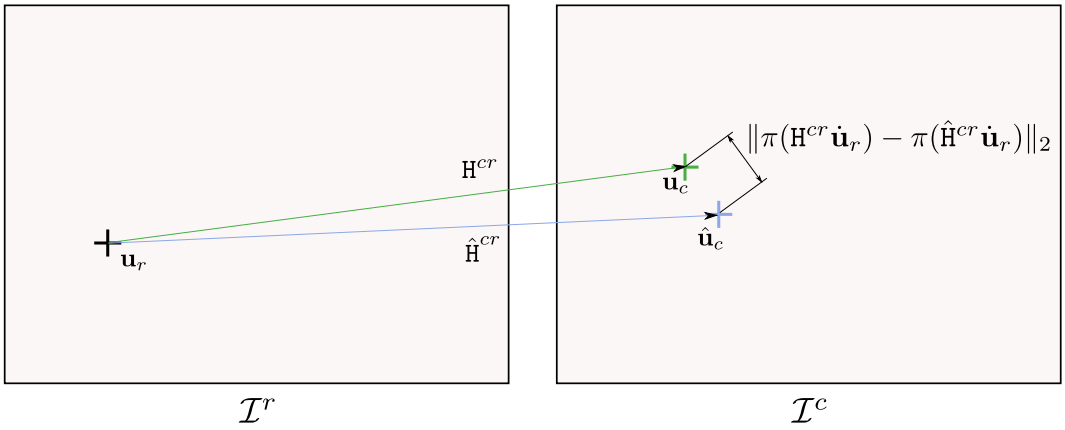


Figure 5.17: Illustration of measured and induced homographies $\mathtt{H}^{cr}$ and $\hat{\mathtt{H}}^{cr}$ respectively, taking an image point from $\mathbf{u}_r$ in $\mathtt{I}^r$ to their corresponding locations in $\mathtt{I}^c$. The image space distance (transfer error) induced in $\mathtt{I}^c$ by the two homographies is labelled.

and those of the plane $\mathbf{y} \in \mathbb{R}^3$. The cost function $F(\mathbf{x})$ is written in terms of a concatenated vector $\mathbf{x} = \left(\boldsymbol{\psi}_1^\top, \ldots, \boldsymbol{\psi}_N^\top, \mathbf{y}^\top\right)^\top$:

$$F(\mathbf{x}) = \frac{1}{2} \sum_j \sum_i \sum_{\mathbf{u}_j \in \Omega_j} \left[\pi\left(\mathtt{H}^{ij}\dot{\mathbf{u}}_j\right) - \pi\left(\hat{\mathtt{H}}^{ij}(\mathbf{x})\dot{\mathbf{u}}_j\right) \right]^2, \tag{5.33}$$

$$\hat{\mathtt{H}}^{ij}(\mathbf{x}) = \mathtt{K}\underset{3\times4}{\mathtt{T}}^{ij}(\boldsymbol{\psi}_i, \boldsymbol{\psi}_j) \left(\mathtt{I} \;\middle|\; -\mathbf{n}_j(\boldsymbol{\psi}_j, \mathbf{y}) \right)^\top \mathtt{K}^{-1}, \tag{5.34}$$

$$\mathtt{T}^{ij}(\boldsymbol{\psi}_i, \boldsymbol{\psi}_j) = \left(\mathtt{T}^{wi}\mathtt{T}(\boldsymbol{\psi}_i)\right)^{-1} \mathtt{T}^{wj}\mathtt{T}(\boldsymbol{\psi}_j), \tag{5.35}$$

$$\mathbf{n}_j(\boldsymbol{\psi}_j, \mathbf{y}) = \pi\left(\left(\hat{\mathtt{T}}^{wj}\mathtt{T}(\boldsymbol{\psi}_j)\right)^{-\top} \begin{pmatrix} \mathtt{Q}^{-1}\hat{\mathbf{z}} \circ \mathbf{z}(\mathbf{y}) \\ 1 \end{pmatrix} \right), \tag{5.36}$$

where the first keyframe $\mathtt{T}^{w0} = \mathtt{I}$ is fixed equal to the world coordinates, and the plane parameters $\mathbf{y}$ are defined in this frame of reference. To fix the system's scale and prevent parameter drift, we additionally keep fixed one of the plane parameters, $\mathbf{y}_0$. Fixing scale in this way is less simple with the homogeneous vector plane parameterisation, where scale is associated with the norm. Our approach is also significantly simpler than fixing the plane and attempting to hold the relevant degrees

of freedom of the first camera, which also cannot be isolated to a single Lie algebra parameter.

Minimisation of this cost function is performed within a simple Iterative Reweighted Least Squares scheme using the Tukey influence function for robust and stable optimisation. This way, poor edges in the graph (formed for example when matching in ambiguous regions of texture) do not destroy the map.

5.3.6 Maintaining a Keyframe Map

Our map is composed of keyframes which form vertices within a graph. Each keyframe stores its associated video image and colour derivative image data, along with its estimated pose.

Live video tracking proceeds by selecting a keyframe from which to track by choosing that which has the greatest symmetric overlap with its own live image, based on the last estimated camera pose and current plane and keyframe pose estimates. We define overlap to be the image-space area visible when transforming the contents of one image to another via a homography parameterised by the images relative motion and plane. We define symmetric overlap to be the average of this area going forward and backward, helping to reflect the number of pixels jointly considered.

Measured homographies between keyframes which we determine overlap (based on their estimated pose and plane parameters) form edges between keyframes in this graph. As the live pose of the camera is tracked relative to its reference keyframe, a new keyframe is added when the percentage of symmetric overlap with this reference keyframe falls below some threshold. This new keyframe inherits the pose of the live camera and automatically becomes the new reference keyframe for tracking. A new edge is added to the graph between this keyframe and the last reference keyframe, initialised with the homography induced by the current plane parameters and estimated pose. The live camera pose $\mathtt{T}^{wl}$ is never explicitly stored, but instead computed through composition as a relative measure from the reference keyframe. This allows tracking to continue smoothly throughout global adjustments.

After each video frame arrives, $\mathtt{T}^{lr}$ is refined to update the live camera pose. Once complete, the remaining time is spent increasing the accuracy of measured homographies by refining them without imposing any constraints. A few iterations of

this 8 DOF estimation occur within each cycle on the GPU interleaved with tracking. Given the keyframe graph, the next edge selected for refinement is the one which has received the least number of iterations, excluding those which have fully converged (with an update vector size below some epsilon). This naturally prioritises the most recently added edges, corresponding to those areas which have most recently been explored and to new loop closures. Estimating these homographies accurately enables the global optimisation to maintain accurate poses of those keyframes near to the current live camera, helping live pose estimation to continue accurately and minimising drift.

5.3.7 Results

Consistent maps

Figure 5.18 demonstrates the incremental live tracking and floor reconstruction from a camera moving generally over a planar surface. Local homographic measurements estimated between keyframes are shown as blue edges. When there exists overlap between non-consecutive keyframes, new relations are added to the graph. These iterations are refined iteratively, and only included in the global estimation if they have an acceptable normalised error. As these edges are included, they introduce loop closures which pull the mosaic into global agreement.

Estimation of Plane Parameters

Given our efficient parameterisation based on keyframe poses and plane parameters, we can use use this information at any time without further computation for rendering or other analysis. We can render the planar mosaic from any point of view and in particular, we are also able to create a mosaic in planar coordinates, equivalent to an orthographic projection from a camera positioned fronto-parallel to the plane. This would not be be so trivial if each frame were only related via a homography.

Following a similar approach to that described in Section 5.2.5, we can employ Cg shaders to 'backward warp' pixels from the final mosaic into constituent keyframes in order to find their contributions and blend pixels. We define a planar frame-of-reference $\mathtt{T}^{wp}$ which is oriented such that $Z = 0$ lies along the surface, and whose

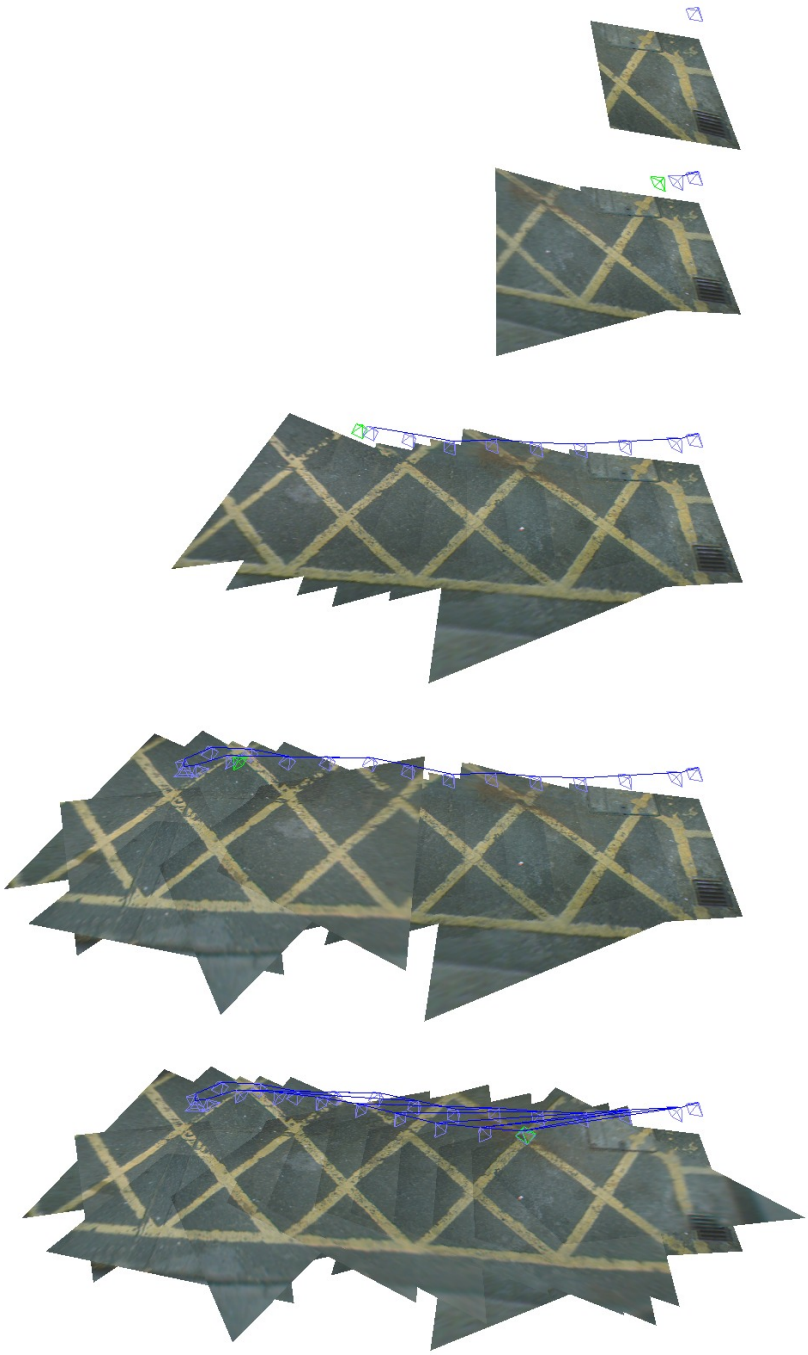


Figure 5.18: Incremental construction of a planar keyframe map. Current live pose shown as a green frustum, historic keyframes as blue frustums. Measured homographies relating historic keyframes are displayed as blue edges within the graph. Notice that the map is brought into global agreement (bottom) as edges are added between non-consecutive poses.

centre of coordinates is at the intersection of the plane with the central pixel of the first keyframe. We additionally take positive Y to corresponds with *up* in this keyframe. Taking the pose of the first keyframe, $\mathtt{T}^{w0} = \mathtt{I}$ to be aligned with the world frame of reference, and taking the plane parameters $\mathbf{n} = \frac{\hat{\mathbf{n}}}{d}$ to be defined in this frame of reference, we can write down the mapping $\mathtt{M}^i(\mathbf{u})$ from $\mathbf{u} = (u, v)^\top$ in plane coordinates to a particular keyframe i's image by considering a plane oriented basis and Z offset:

$$\mathbf{r} = \hat{\mathbf{n}} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} , \tag{5.37}$$

$$\mathbf{up} = \mathbf{r} \times \hat{\mathbf{n}} , \tag{5.38}$$

$$Z_w = \frac{-1}{\mathbf{n} \bullet (0, 0, 1)^\top} , \tag{5.39}$$

which can be composed into:

$$\mathtt{T}^{wp} = \left(\begin{array}{ccc|c} \mathbf{r} & \mathbf{up} & \hat{\mathbf{n}} & \begin{matrix} 0 \\ 0 \\ Z_w \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) , \tag{5.40}$$

$$\mathtt{M}^i(\mathbf{u}) = \pi \left(\mathtt{K} \mathtt{T}^{iw} \mathtt{T}^{wp} \begin{pmatrix} \mathbf{u} \\ 0 \\ 1 \end{pmatrix} \right) . \tag{5.41}$$

Figure 5.19 shows several keyframes from a planar mosaic created by moving a camera interactively over an arm-span baseline to fill in the ceiling. The image is rendered in planar coordinates where pixels have been blended with equal weight. Figure 5.20 contains an enlargement overlaid with a square grid to demonstrate that perspective distortions are small and that planar parameters have been accurately estimated.

Non-Planarities

Within our method, we strictly assume that all video pixels are observing the same plane of infinite extent. In practice, this is generally not precisely true, and large deviations from this assumption might exist.

Figure 5.21 illustrates an example of a planar mosaic of the ceiling where a projector post protrudes from the surface. Despite violating our assumption, employing

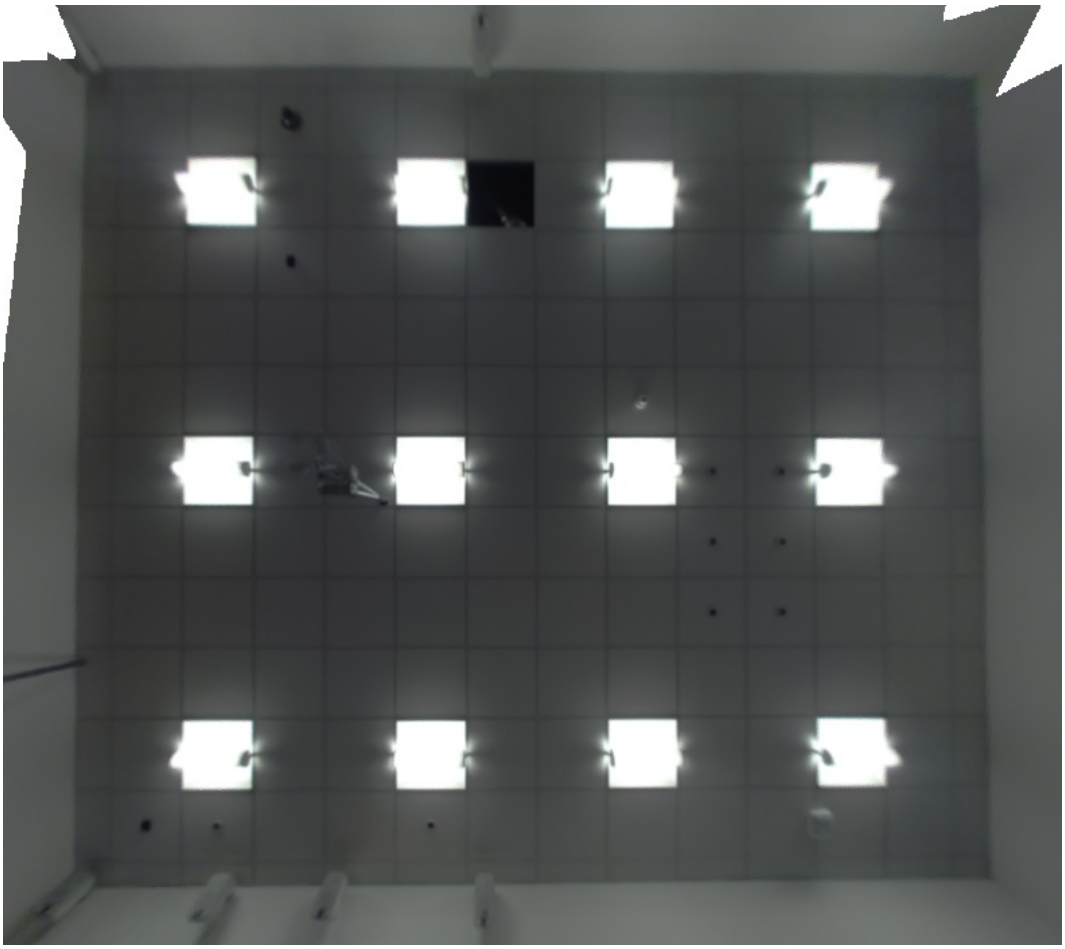


Figure 5.19: Office ceiling fronto-parallel orthographic projection of a planar mosaic consisting of 31 keyframes blended with equal weight. The mosaic was constructed from a stationary vantage point with hand-waved camera motion. Images taken by a VGA PointGrey Flea2 with 70° field of view representing approximately 5×4 tiles at any time.

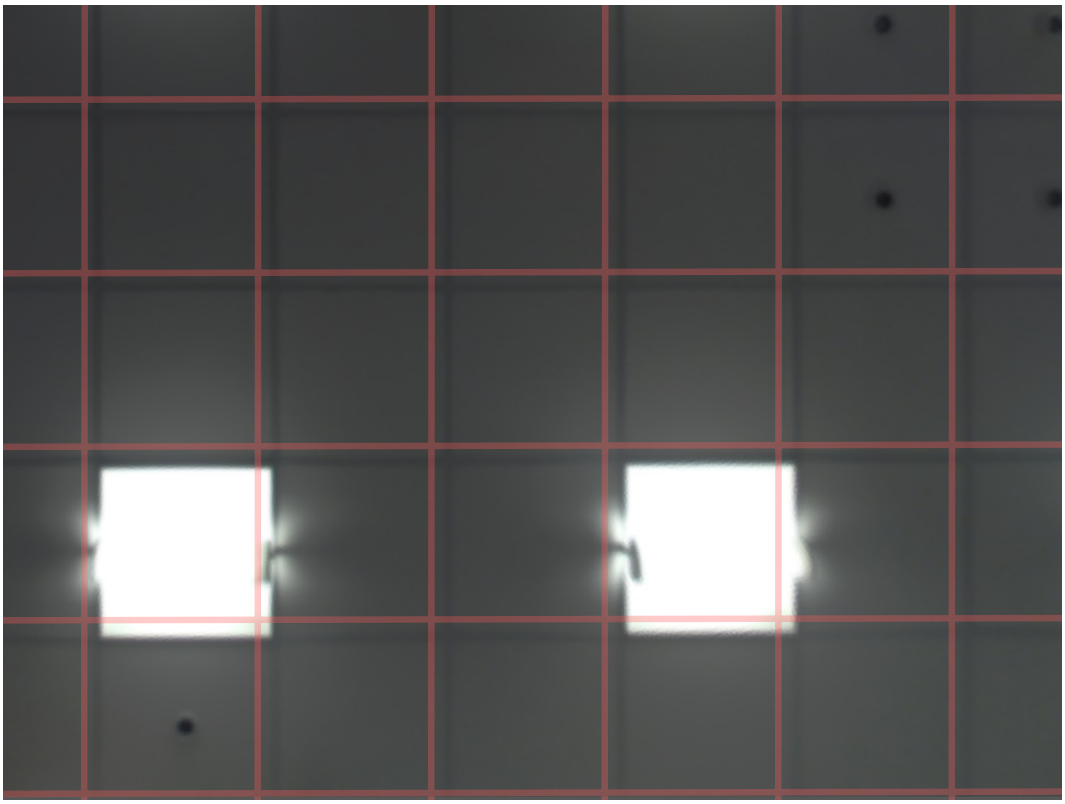


Figure 5.20: Office ceiling fronto-parallel orthographic projection enlargement overlaid with square grid to demonstrate projective distortion free rendering.

Figure 5.21: Planar Mosaic (right) generated from multiple keyframes (e.g. left, middle) where planar assumption is not strictly true. Mosaic generated by blending constituent keyframes with equal weight.

robust estimators in both the local and global optimisations means that it has little influence in the mosaic.

Despite the rest of the plane being registered correctly, objects which violate our planar assumption will cause our final mosaic to be smeared where parallax renders the homographies invalid. We do not attempt to address this issue, but it has been considered extensively in the literature. Instead of blending pixels from all keyframes, we could select pixels from individual keyframes (e.g. [24][2]). For small deviations, we could additionally employ optic flow to calculate how to displace pixels between frames before blending [113]. Further still, we could explicitly attempt to estimate depth across the planar surface, modelling observed parallax directly [107]. All of these methods require added computation and it is not clear that they can be achieved at interactive rates. In any case, we believe that averaging creates mosaics of high enough quality for interactive use — other compositing techniques might be applied as a post step depending on the application.

A more extreme example of a scene which is largely planar but with many off-plane areas is shown in Figure 5.22. Here, the plane orthographic projections show the differing quality of plane parameter estimation with standard least squares, and with iteratively reweighted least squares using the Tukey influence function (Section 4.5). With robust estimation the partition forming the dominant plane remains sharp, whereas in the least squares solution it is blurred and skewed reflecting the misestimation of image registration and plane parameters.

Document Scanning

Given that we can estimate plane and pose parameters accurately, a simple application of our method is for interactively scanning documents without projective distortions. The only previous real-time document scanning work to our knowledge is that of Akihiko *et al.* who use a point-feature based visual SLAM system to interactively create a sparse feature map, which is then used in an offline stage to process the final document scan [51]. Without considering euclidean geometry, other planar mosaicing systems operating in image space are prone to projective distortion since plane parameters are not estimated.

Figure 5.23 shows the graph of a complete mosaic consisting of a number of

Figure 5.22: Planar mosaicing from 8 keyframes without (top) and with (middle) robust estimation (Tukey M-Estimator $c = 255 \rightarrow 2.5$). Sample keyframes (bottom) demonstrate observed disparity. Using robust estimation, plane parameters are sufficiently estimated, as demonstrated by the right angles seen in the partition from this orthographic projection.

keyframes with dense connectivity representing large overlap between a number of the images. You can see that the camera was moved at different distances from the plane to capture different details of the document.

Figure 5.24 shows the orthographic, fronto-parallel projection of the same mosaic. Whereas Figure 5.23 is rendered using projective texturing without keyframe blending where we can clearly see shadows cast by the user holding the moving camera, these view dependant shadows average out in Figure 5.24.

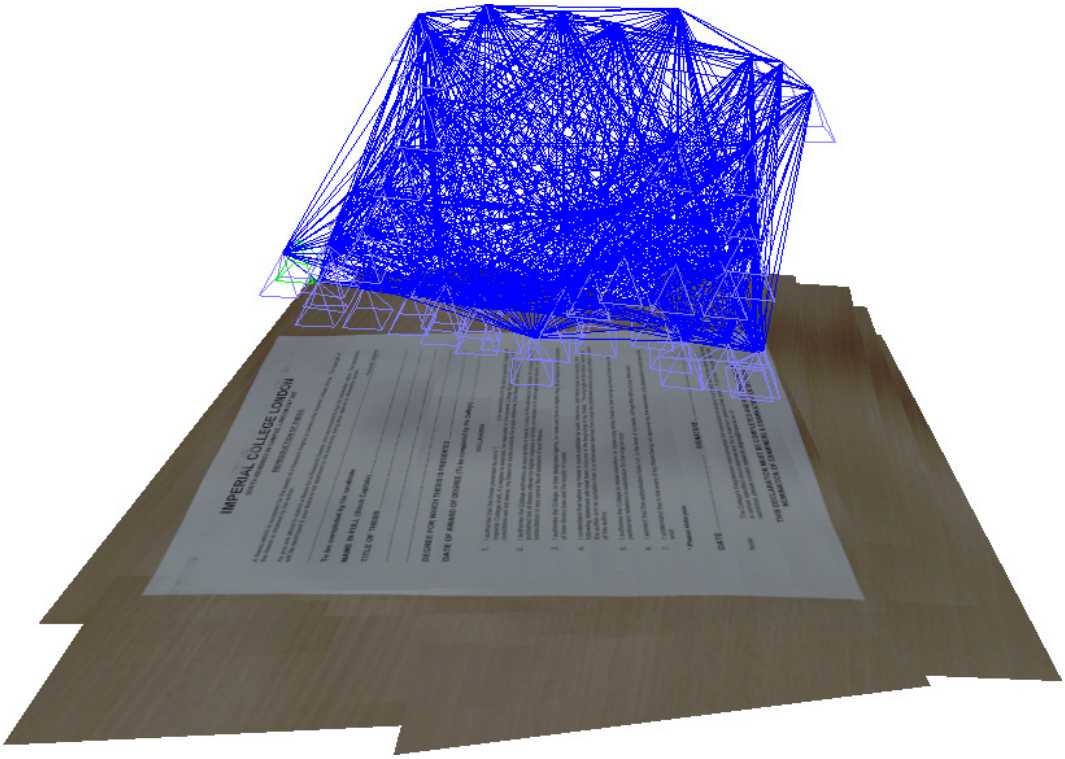


Figure 5.23: Planar mosaic of a text document consisting of several keyframes, rendered by overlaying each keyframe consecutively (without blending). Notice that the camera's shadow appears in the individual keyframes.

Live, Super Resolved Images

Making use of all image data and using robust estimators, overlapping keyframes within the mosaic can be registered with high precision within a fraction of a pixel. Global adjustment can increase this accuracy further provided camera intrinsic parameters are well estimated and hold over extended regions where we assume them

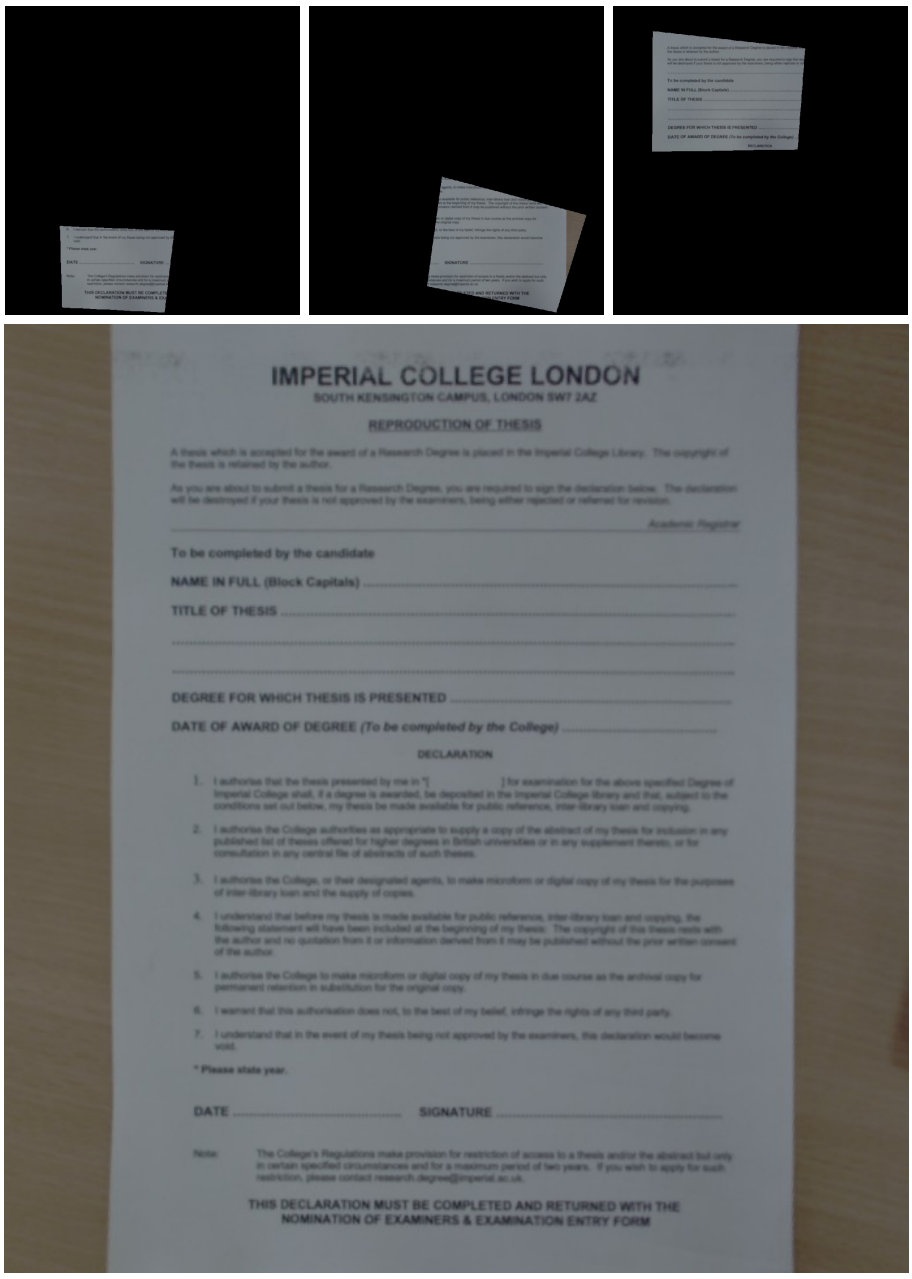


Figure 5.24: Several keyframes are transformed into the planar coordinate system (top row) and blended with equal weight to form the final mosaic (bottom). Low frequency lighting variations caused by shadowing average out.

to be constant.

Each keyframe's image samples the scene in a regular grid, the combination of all keyframes forming a super-sampling much finer than a single image. Whilst rendering the full mosaic we are free to choose a new grid in which to resample our data. Through backward warping, each pixel (grid element) in the output image will receive contributions from all keyframes sampling this piece of surface. Multiple observations of a textured region reduce noise over a single measurement and so our final mosaic can have a much improved signal to noise ratio. Figure 5.25 considers a small area of Figure 5.19 displayed using a surface plot to demonstrate how image noise is reduced by combining multiple registered images through averaging.

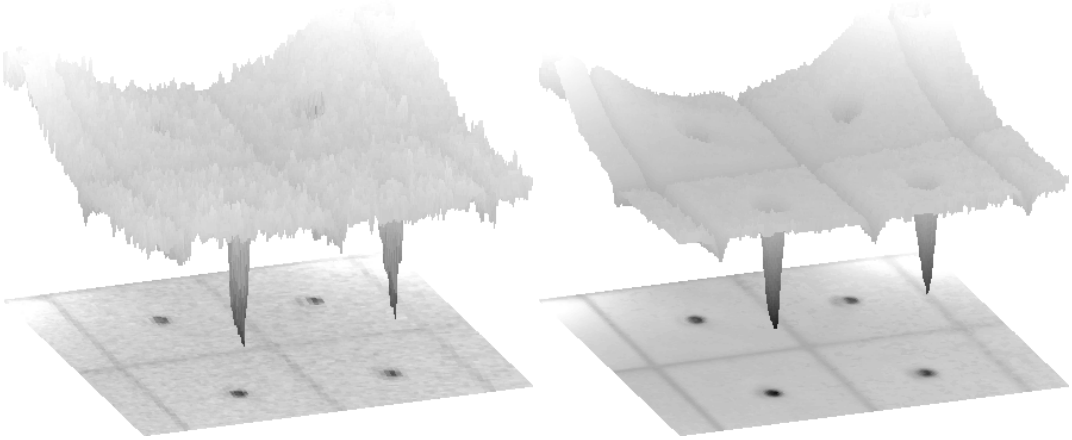


Figure 5.25: Taking a small section from a planar mosaic of some ceiling tiles, we show surface plots for the pixel intensities of a a single raw keyframe from the mosaic (left), verses the average in that area of all aligned keyframes. Notice how pixel noise can be reduced whilst maintaining sharp image features.

Since individual CCD sensor elements on the focal plane of a digital camera are not packed tightly against one another, images captured often contain some degree of aliasing if the optical path itself is not completely limiting the resolution of the camera. Aliasing can be seen in the form of 'jaggies' where neighbouring pixels do not vary smoothly in intensity. Aliasing can also introduce interesting Moiré patterns — the interference introduced by sampling a regular texture by a misaligned or differing size grid.

Assuming a Gaussian distribution on pixel intensities, multiple samples accurately registered can be averaged within a sub-pixel interval to improve spatial resolution to

small degrees. More sophisticated multi-image super-resolution techniques consider the effects of different imaging stages such as lens blur to obtain significantly higher quality super-resolved images at additional computational cost (e.g. [129]).

Figure 5.26 illustrates the modest gains in resolution that can be achieved by using our system to mosaic a text document over a narrow baseline from several constituent keyframes. Rendered interactively, the user receives feedback as to the quality of the final mosaic. In contrast to our earlier document mosaicing, we do not zoom in or change viewpoint heavily, so the sample image keyframe crops are representative of the input resolution and aliasing present.

5.4 Summary

Within this chapter, we have demonstrated that high quality, globally consistent spherical mosaics can be generated in real-time by combining the precise and very robust tracking methods that were described in Chapter 4, with simple global optimisations also inspired by these approaches, and based on the sampling of keyframes.

Through the very accurate estimation of local motion, global adjustment is made much simpler, and we do not explicitly consider loop closure, rather, geometric loop closures are estimated well enough that they can proceed automatically.

In order to mosaic a plane in real-time as we move generally over it, we were required to refine the brute force approach which we had shown worked well for rotational mosaicing. Instead, we formulated an efficient global cost which approximates the integral of model transfer error given the precisely measured homographies between pairs of keyframes.

Unlike most of the mosaicing literature, our model is parameterised by keyframe poses and plane parameters, rather than by the homographies which relate frames themselves. This formulation is more efficient, and enables us to track in real-time camera pose by placing structure estimation in a lower priority procedure.

For planar mosaicing, since 3D keyframe poses and plane parameters are estimated live, we can also apply texture directly on to the plane in planar coordinates, allowing us to avoid projective distortions common to other techniques. Finally, we demonstrate the accuracy of our planar mosaicing system by blending all keyframes

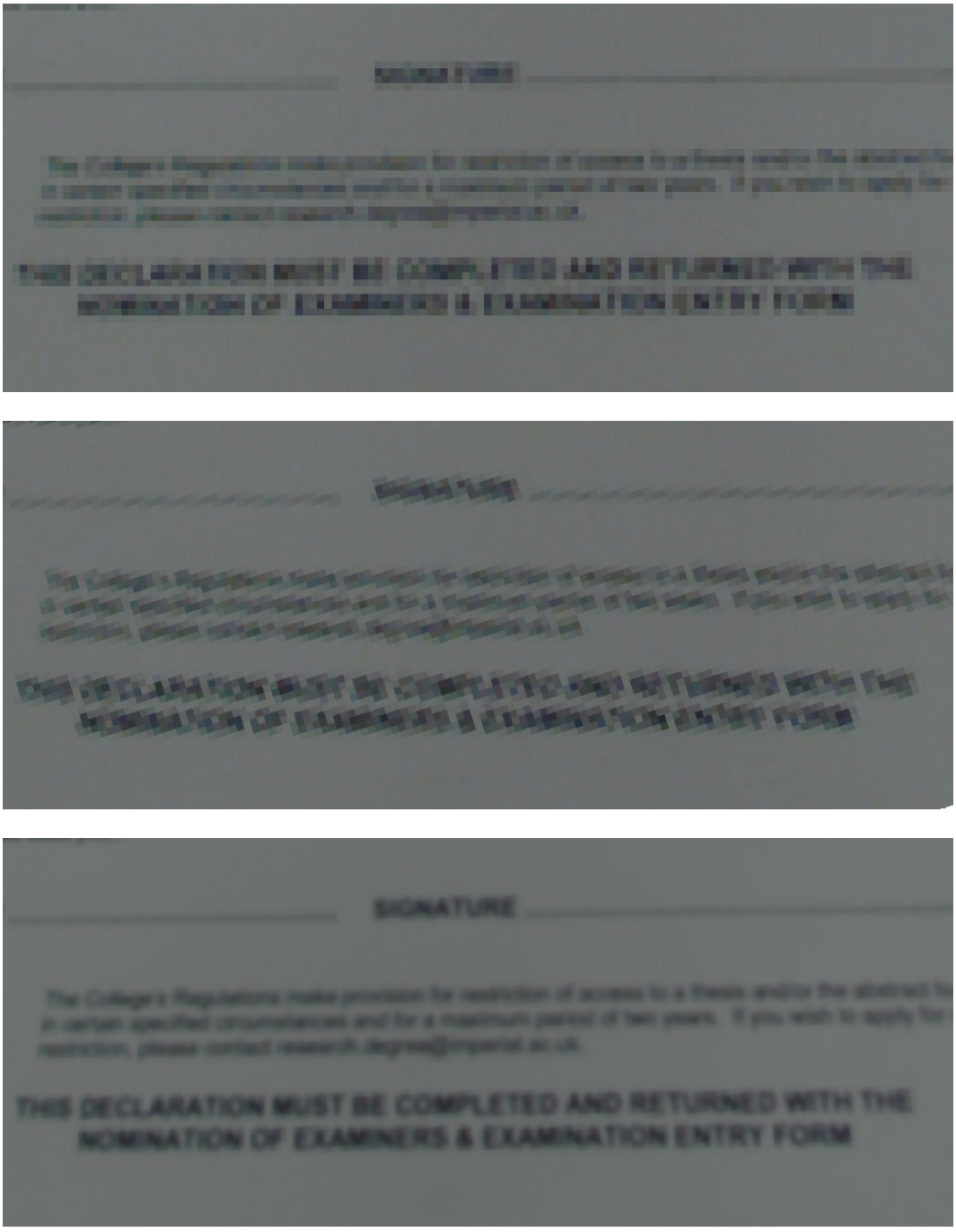


Figure 5.26: Multiple keyframes are composited by simple averaging into a higher resolution image (bottom), producing fine details not visible in constituent images (top, middle).

with equal weight into a fine grid which demonstrates the improvements increased sampling can have on reconstruction quality.

The systems presented in this chapter have focussed on alignment accuracy and live estimation. We have demonstrated largely qualitative results upon which to judge our methods, though a more in depth analysis using synthetic data for example might be possible. To our knowledge, no mosaicing datasets for benchmarking are available. Further, we have neglected how we can increase perceived quality through more sophisticated blending schemes; by taking more robust pixel means for instance, or by intelligently selecting pixels through segmentation when objects appear and disappear. These issues have been well studied already and we refer to Szeliski's thorough review for details [124].

Chapter 6

DTAM: Dense Tracking and Mapping in Real-Time

Work within this chapter describes the system DTAM and was conducted in close collaboration with Richard Newcombe, leading to the publication: 'DTAM: Dense Tracking and Mapping in Real-Time' by Richard Newcombe, Steven Lovegrove and Andrew Davison [95].

6.1 Introduction

Within the last few chapters we have looked at real-time direct parametric tracking and mapping in the context of spherical and planar mosaicing. We have seen how through the very simple application of coarse-to-fine refinement and iteratively reweighted least squares, we have been able to track through highly dynamic camera motions experiencing significant motion blur and heavy accelerations (Section 4.6), whilst maintaining accurate tracking when image quality is good (Section 5.3.7).

Although state of the art methods for live 3D monocular SLAM have matured, perhaps at last offering a degree of robustness suitable for non-expert use, their utility is at present limited by the quality of their maps. The sparse features underlying typical systems exist as a basic necessity for keeping a drift free reference frame;

features themselves are only landmarks for pose estimation. For many applications such as robotics and augmented reality, the live pose of the camera and the locations of sparse landmarks are not enough — a robot needs object geometry in order to avoid hitting things in the world and perhaps to interact with them; augmented reality also requires surface geometry in order for virtual objects and characters to be fused with the physical.

Within this chapter, we will look at how direct parametric tracking methods can be combined with fast, dense multi-view stereo to produce a fully dense monocular SLAM system for arbitrary 3D environments, whose map consists not of sparse point features, but a dense and accurate surface model. We will continue by demonstrating the benefits of such a representation and of using all pixels for tracking and mapping; namely robust and precise pose estimation and straightforward occlusion and occupancy data for target application areas such as augmented reality and robotics.

6.2 Background: Towards Dense 3D SLAM

6.2.1 Piecewise Planar Models

We might attempt to draw inspiration from the graphics community, where a number of representations of geometry have been proposed, many for real-time and interactive purposes. Rasterisation of polygonal primitives has been popular in gaming and interactive graphics to achieve increasingly detailed live 3D renderings, with current graphics hardware highly accelerated for this purpose. Looking at some of the top performers in the offline multi-view stereo (MVS) literature [112], we notice that a number compute surfaces by infinitesimal piecewise planar estimation (e.g. Habbecke and Kobbelt [46], Furukawa and Ponce [41]). Earlier work by Habbecke and Kobbelt [45] demonstrated much coarser reconstructions based on larger planar regions but using very similar machinery.

An avenue of work that is enticing is to construct a coarse-to-fine piece-wise planar world, starting perhaps from where our last chapter finished, refining a piece-wise planar model which, given time, could take us right up to state of the art MVS reconstructions of the world. This idea of coarse-to-fine geometry sounds appealing

— make coarse strokes with little computation to enable robust tracking, refining areas suffering from the poorest predictive quality, staying within real-time bounds. In this chapter we instead take an approach which builds reconstructions from depth map units, but it would be interesting to return to multi-planar modelling in the future; or perhaps to even more restrictive models like a 'Manhattan World' where all planes are orthogonal.

6.2.2 Depth Maps and Multi-View Stereo

From two views of a textured scene where the relationship between the cameras is known, dense structure can be estimated in a process known as *dense stereo*. Each pixel in an image corresponds to a ray in space, along which the scene it has imaged with the given colour must lie — the projection of this ray into the image of a second known camera is a line which is known as the epipolar line. If the relative pose of the camera has been estimated accurately, then the corresponding projection of this scene point into the second camera must lie along this epipolar line [48].

Epipolar geometry gives rise to strong matching constraints that make it feasible to approximate *dense* surface geometry for every pixel in a stereo pair; though, for areas of low texture where matching is ill posed or where there is little baseline between the images, we are likely to see very noisy measurements. From two views, only the geometry of structure observed in *both* views can be reconstructed. For this reason, geometry is typically represented as a per-pixel depth in one of the views to form an image which we refer to as a depth map, as demonstrated in Figure 6.1. For calibrated cameras, it is possible to pre-warp the images to form a rectified stereo pair such that the epipolar line of each pixel in one image corresponds to a pixel aligned row in the other [30]. In this case, the pixel displacement between image pairs is termed *disparity*. Disparity is proportional to inverse depth and can equivalently be stored per pixel as a disparity image.

The difficulty in dense stereo is in attempting to accurately find pixel correspondences. A single pixel from one image lacks discrimination when matching along epipolar geometry in another, so using this method to establish depth can be very noisy. It is common to use a small patch around a pixel as support when determining match score along the epipolar line to increase discrimination and thus reduce noise. The original application of the Lucas-Kanade method that we have used through-

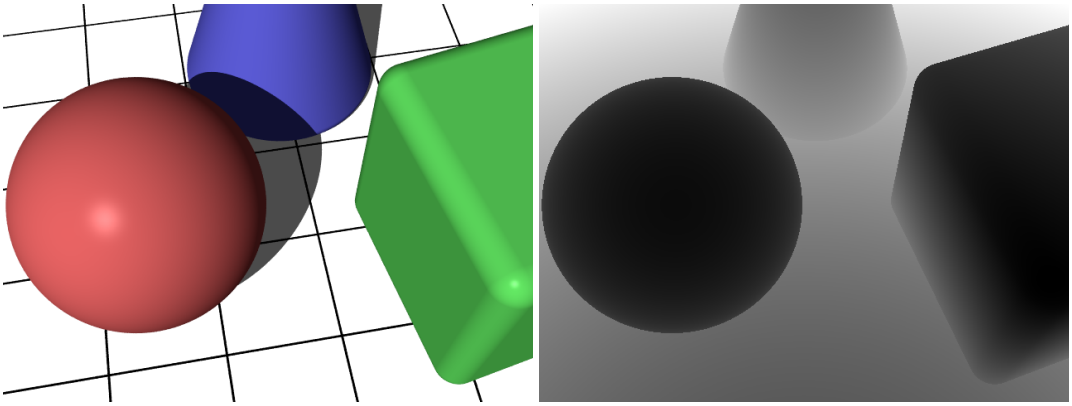


Figure 6.1: Illustration of a simple 3D scene with projective camera image (left), and associated depth map (right). In the depth map, each pixel has a depth value which corresponds to the scene z ordinate captured at that point in the projective colour image. Depth values are show here represented by intensity; black pixels represents those that are nearest the camera and white pixels represent those furthest away.

out this thesis was in this domain, matching small patches in one image along the one-dimensional epipolar line in another [77]. Patch matching can however lead to over smoothing where we miss fine structure, since the patch must match well in its entirety. It is also quite expensive, especially for large patches. The size of the patch will affect the scale at which it can reduce noise but also the size of structure which it can no longer resolve.

The very simplest dense stereo methods take patches around each pixel in one image and search along the epipolar line at whole pixel intervals in the second, recording the whole pixel disparity which best minimises a sum of square difference patch score. Over the years, many methods have been proposed for increasing the quality of dense stereo results. Some authors have suggested different match scores or different methods for sampling along the epipolar line. Another option is to regularise the depth-map by incorporating smoothness priors; that neighbouring pixels are likely to share similar depth values for example. Scharstein and Szeliski offer a thorough review of classic literature and place them in a taxonomy of algorithms [111].

Yoon and Kweon demonstrate that through an adaptive filtering of depth maps using associated colour image gradient data, depth map quality can be improved [133]. They note that discontinuities in depth images frequently occur where the corre-

sponding image data has high image gradients — filtering is therefore weighted by gradient intensity to smooth depth data mostly in areas of homogeneous colour, preserving occlusion boundaries lying along image edges.

Building a 3D model from more than two images is termed multiple-view stereo (MVS). Pollefeys *et al.* [103] demonstrated that by 'chaining' several dense stereo depth maps into the frame of reference of one, increased quality can be obtained. They also show how multiple depth maps can be fused within a volumetric approach to obtain models of arbitrary topology. A triangular surface mesh can be obtained from implicit surface representations using marching cubes [70].

High quality multiple-view stereo can be computationally expensive. Methods that consider global regularisation are generally more costly than those which rely on purely local methods. In recent years however, high quality multiple view stereo methods have started to become real-time capable [42, 134]. This is largely thanks to highly parralelisable algorithms which permit implementation on increasingly performant programmable graphics hardware.

Beginning with the work of Newcombe and Davison [93], however, it was shown that dense, fine and accurate structure could be extracted interactively within a monocular SLAM system using commodity hardware. Their system comprised of a point-based map of features used for camera tracking from which a smooth base mesh was formed. The dense mesh was refined by warping to accurately reflect the true structure by considering epipolar constrained optic flow between a bundle of camera frames and the view prediction from the current geometry. As the quality of the model improves, so too do the view predictions into the live camera, where consistency induces no surface warping. Another quite similar system was developed in parallel by Stuehmer *et al.* [122].

From the perspective of dense tracking, the most similar work to our own is that of Comport *et al.* who demonstrate that every pixel from a stereo sequence can be used to estimate accurate drifting visual odometry in a sequential but non real-time process [22]. From their stereo video sequence, dense stereo is used to estimate a depth map in the frame of reference of one of the views. An ESM (Section 4.3.6) based minimisation is formulated to minimise a photometric cost function with respect to the 6 DOF quadrifocal arrangement of stereo pairs through time. Meilland *et al.* presented work conducted in parallel but independently from

our own based on localisation from spherical depth maps which also bears many similarities with the work in this chapter [85]

6.3 Method

6.3.1 Overview

The structure of DTAM draws on the separation of tracking and mapping as decoupled but dependent processes, inspired by the state of the art in monocular feature-based SLAM, PTAM [62]. Like PTAM, DTAM also stores a collection of historic images sampled from the video stream as keyframes with known pose. However, rather than using a sparse feature-based representation of the world from which to track, each keyframe within DTAM also has associated with it a dense depth map, and so records not only the projected colour of the scene but also its dense surface structure for every pixel in the sampled image. Each of these depth maps is efficiently generated using a dense stereo technique from potentially hundreds of video images from nearby positions of the live tracked camera (Section 6.3.2). This part of the system is an improvement on the dense depth map creation element of [93] which does not require warping of an approximate base surface but solves directly for a depth at each pixel (in this sense being more similar to [122], though with further improvements in the details).

The union of the overlapping textured depth maps forms a model of the surface of the world (Figure 6.2). Capturing fine details and arbitrary topology, this model can be rendered from a new view for a realistic, photometric prediction of what the real camera will observe from that location.

DTAM departs most from previous dense visual SLAM systems in the way that the live pose of the camera is estimated. Rather than relying on a background structure of point features for tracking, a pose estimate is established by whole image alignment of the current image against a projection of the entire dense model (Section 6.3.3). Through an efficient direct method with coarse-to-fine warping, taking full advantage of the restricted motion between video frames, camera tracking can proceed at frame-rate over varied and highly dynamic motions.

The tight coupling of timely multi-view stereo and whole-image tracking from

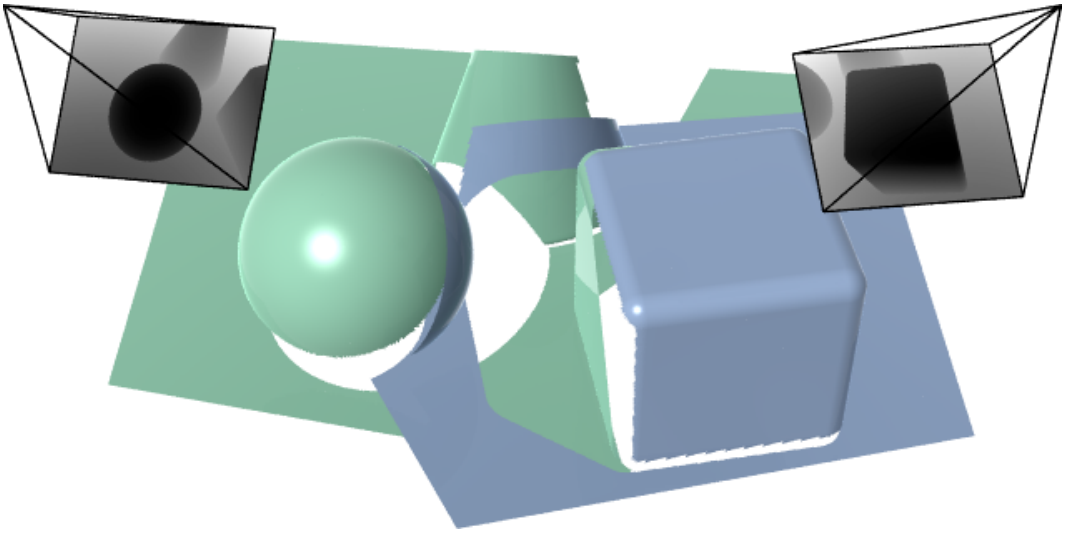


Figure 6.2: Overlapping depth maps recreate the geometry of our familiar block scene. Surface connectivity within a depth map must be established based on some smoothness prior. Here we 'cut' the surface at depth discontinuities which will lead to holes in occluded regions. Multiple depth maps can fill in missing areas, as seen above.

a known model leads to a real-time algorithm capable of incrementally building accurate dense models of a desktop scale scene. By using each narrow baseline video frame within an incremental reconstruction procedure, tracking can proceed in new areas quickly from the best data that is currently available.

A large advantage of the methods used in DTAM for tracking and mapping is that they are inherently and trivially parallelisable. This is a major enabling factor for their real-time use, implemented as they are on commodity graphics hardware, scaling naturally in processing cost with different image resolutions.

6.3.2 Dense Depth Maps from Video

In order to generate a new depth map, an image from the video stream is first chosen as the depth map's reference frame and saved. The pose of the camera when taking this image is also stored, and these together form an uninitialised keyframe. For each new keyframe we create, we also initialise a three-dimensional projective voxel data structure in which we will incrementally accumulate data, called a *projective cost volume*. For every pixel in the reference image, and for every depth in a discrete

range, the cost volume records the *cost* associated with assigning to this pixel a particular depth value.

Depth map construction proceeds by computing these per-pixel, per-depth costs from consideration of many comparison images. At any time, we can take the minimum cost depth per pixel to form our depth image. This measurement however is noisy, and we choose instead to regularise this depth map within a variational procedure before it is used for tracking. Regularisation can be performed incrementally, allowing us to include the depth map in our model for tracking whilst also integrating new data to make the depth map better. Once the depth map has fully converged and we are happy not to include more data, we can destroy the associated cost volume, allowing its memory to be used for a different keyframe.

Projective Cost Volume

Figure 6.3 illustrates the items associated with a dense keyframe r, including the image $\mathtt{I}^r$, pose $\mathtt{T}^{rw}$ and cost volume $\mathbf{C}_r$. For video images of size $M \times N$, the associated cost volume is a three-dimensional array of size $M \times N \times S$, where S is the number of inverse depth bins sampling the range $\xi_{\min}$ to $\xi_{\max}$. A row $\mathbf{C}_r(\mathbf{u})$ in the projective cost volume (called a disparity space image in stereo matching [125], and generalised more recently in [106] for any discrete per-pixel labelling) stores the accumulated average photometric error as a function of inverse depth d.

For every pixel $\mathbf{u}$, each inverse depth sample d within the cost volume represents a hypothetical surface point $\mathbf{X} = \pi^{-1}(\mathbf{u}, d)$. The average photometric error $\mathbf{C}_r(\mathbf{u}, d)$ at this point is computed by projecting it into each of the overlapping comparison images $\mathtt{I}^m \in \mathcal{I}(r)$, taking the average of the absolute photometric errors for all projections that fall in valid image bounds. Let $\mathcal{I}(r, \mathbf{u}, d) \subseteq \mathcal{I}(r)$ be the set of images that $\pi^{-1}(\mathbf{u}, d)$ projects into, then:

$$\mathbf{C}_r(\mathbf{u}, d) = \frac{1}{|\mathcal{I}(r, \mathbf{u}, d)|} \sum_{\mathtt{I}^m \in \mathcal{I}(r, \mathbf{u}, d)} \| \rho_r(\mathtt{I}^m, \mathbf{u}, d) \|_1 \,, \tag{6.1}$$

where the photometric error for each overlapping image is:

$$\rho_r(\mathtt{I}^m, \mathbf{u}, d) = \mathtt{I}^r(\mathbf{u}) - \mathtt{I}^m\left(\pi\left(\mathtt{KT}^{mr}\pi^{-1}(\mathbf{u}, d)\right)\right). \tag{6.2}$$

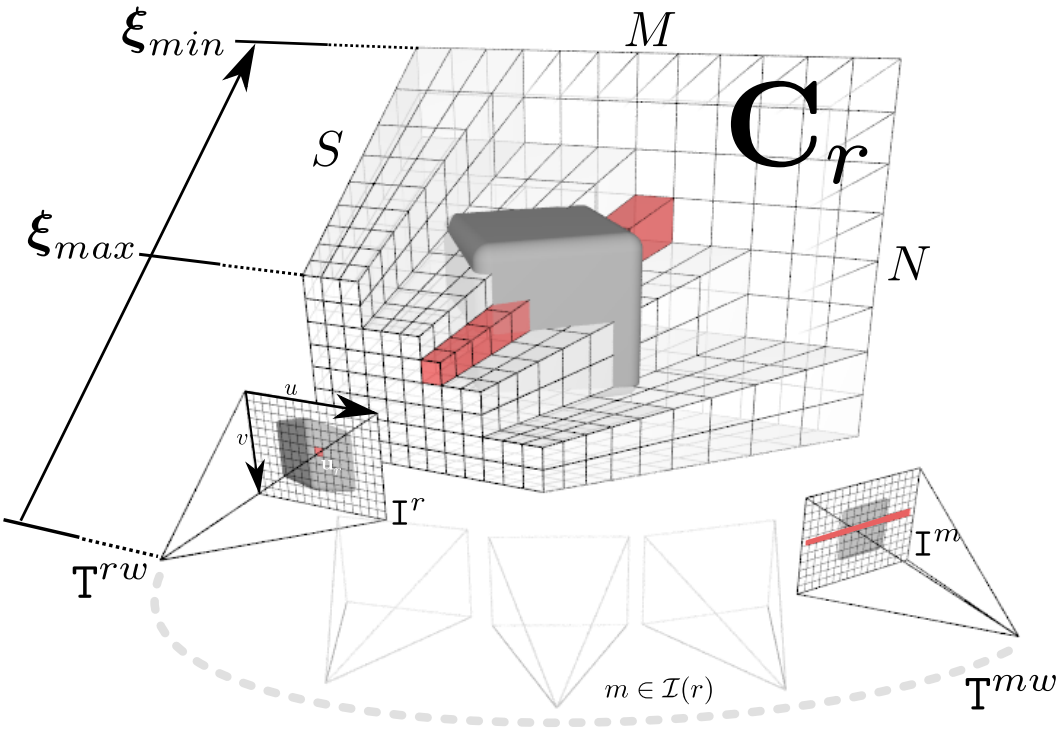


Figure 6.3: A keyframe r consists of a reference image $\mathbf{I}^r$ with pose $\mathbf{T}^{rw}$ and cost volume $\mathbf{C}_r$ (visualised here as a cut away 3D grid). Each pixel of the reference frame $\mathbf{u}_r$ has an associated row of entries $\mathbf{C}_r(\mathbf{u}_r)$ (shown in red) that store the average photometric error computed for each inverse depth $d \in \mathcal{D}$ in the inverse depth range $\mathcal{D} = [\xi_{min}, \xi_{max}]$. We use tens to hundreds of video frames indexed as $m \in \mathcal{I}(r)$, where $\mathcal{I}(r)$ is the set of frames nearby and overlapping r, to compute the values stored in the cost volume.

Naturally, this simple sum can be computed incrementally as new images $\mathbf{I}^m \in \mathcal{I}(r)$ are observed by the live camera — we can simply record a running average and counter per element within the volume. Critically, the set of comparison images $\mathcal{I}(r)$ need not be stored after integration, and so the memory requirement for incorporating any number of images is constant. Integration of a new comparison image is simple and highly parallelisable, allowing us to incorporate each and every video frame into a single cost volume live alongside camera pose estimation.

Having integrated a number of comparison images into a reference cost volume, we can extract the most photo-consistent depth map from the volume by finding per pixel the depth with lowest cost. Despite such a simple pixel-wise photometric cost metric, the average of many individually indiscriminate costs will frequently

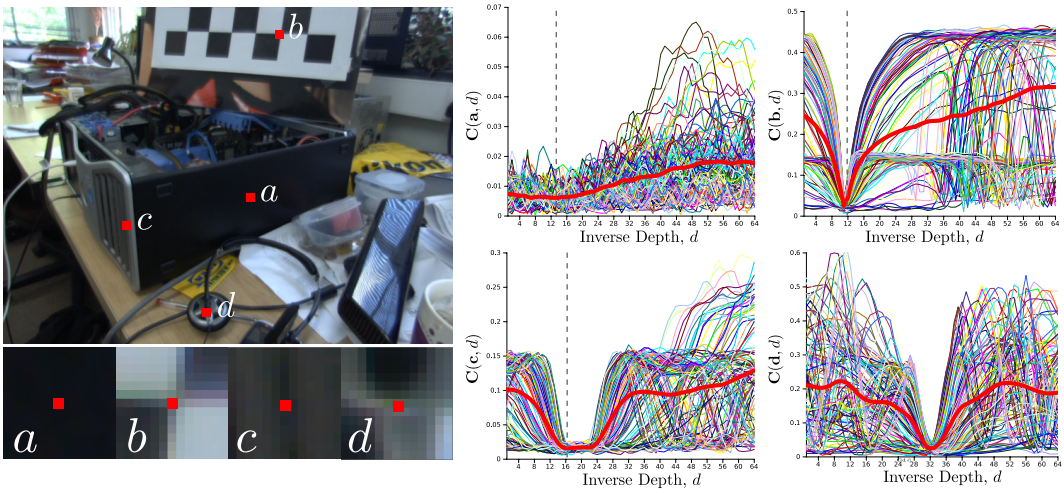


Figure 6.4: Plots for the single pixel photometric functions $\rho(\mathbf{u})$ and the resulting total data cost row $\mathbf{C}(\mathbf{u})$ are shown for three example pixels in the reference frame, chosen in regions of differing discernibility. Pixel (a) is in a textureless region and not well localisable; (b,d) are within strongly textured regions where a point feature might be detected; and (c) is in a region of linear repeating texture. While the individual costs exhibit many local minima, the total cost shows a clear minimum in all except nearly homogeneous regions.

tend toward the correct surface depth. Figure 6.4 demonstrates the sum of absolute costs for illustrative pixels over the cost volumes depth range — although individual comparison image pixel error plots are highly multi-modal, their sum is much smoother, removing ambiguities that it might not be possible to resolve in a single stereo pair.

Cost Volume Regularisation

Through the construction of the photometric cost volume from many comparison images, the per-pixel minimum cost depths can be taken to construct the most photo-consistent depth map; but sometimes the per-pixel cost over depth does not contain a clear minimum or it is multi-modal. There is nothing that stops neighbouring pixels from this *raw* depth map from taking on wildly different depth values.

In order to improve the reconstruction beyond a purely photometric cost function, we must include prior information. For our problem of depth map construction, we assume that the inverse depth solution consists of regions that vary smoothly across

the image and discontinuities that occur due to occluding boundaries. Observing that occluding boundaries frequently project onto strong edges within the image, we use an edge weighted regulariser that penalises deviations from a smooth surface based on the norm over the gradient of the inverse depth map. For reference image $\mathtt{I}^r$ and corresponding depth image ξ, the regulariser term for pixel $\mathbf{u}$ is $g(\mathbf{u})\|\boldsymbol{\nabla}\xi\left(\mathbf{u}\right)\|_\epsilon$, where:

$$g(\mathbf{u}) = e^{-\alpha\|\boldsymbol{\nabla}\mathbf{I}_r(\mathbf{u})\|_2^\beta}\ , \tag{6.3}$$

forms the edge weighting based on image gradient magnitude and

$$\|x\|_\epsilon = \begin{cases} \frac{\|x\|_2^2}{2\epsilon} & \text{if } \|x\|_2 \leq \epsilon \\ \|x\|_1 - \frac{\epsilon}{2} & \text{otherwise} \end{cases} \tag{6.4}$$

defines the Huber norm, which is an L_2^2 norm within $\|\boldsymbol{\nabla}\xi_r\|_2 \leq \epsilon$, and L_1 otherwise. The resulting energy functional therefore contains a non-convex photometric error data term and a convex regulariser whose strength is controlled by the constant λ:

$$E_\xi = \int_\Omega \Big\{ g(\mathbf{u})\|\boldsymbol{\nabla}\xi(\mathbf{u})\|_\epsilon + \lambda\mathbf{C}\left(\mathbf{u}, \xi(\mathbf{u})\right) \Big\} \mathrm{d}\mathbf{u}. \tag{6.5}$$

This energy is minimised to yield the regularised depth map through an efficient primal-dual variational approach. We refer to the original paper [95] for details of this energy including how to efficiently solve it, as this work represents the sole effort of Richard Newcombe.

Although the photometric costs associated with depth reside within the discretised cost volume, the regularised depth solution per pixel can take any continuous values. To achieve high quality reconstructions, sub-sample minimum are sought within the iterative minimisation of the local energy. Figure 6.5 illustrates the incremental construction of the cost volume and regularised solution. Figure 6.6 illustrates the quality of map which can be produced live, versus the sparse feature map maintained by PTAM.

6.3.3 Tracking From a Dense Model

Given a dense model consisting of one or more keyframes with depth maps, we can synthesise realistic novel views over wide baselines by projecting the entire model

Figure 6.5: Incremental cost volume construction; we show the current inverse depth map extracted as the current minimum cost for each pixel row $d_{\mathbf{u}}^{min} = \arg\min_d \mathbf{C}(\mathbf{u}, d)$ as 2, 10 and 30 overlapping images are used in the data term (left). Also shown is the regularised solution that we solve to provide each keyframe inverse depth map (bottom-right).

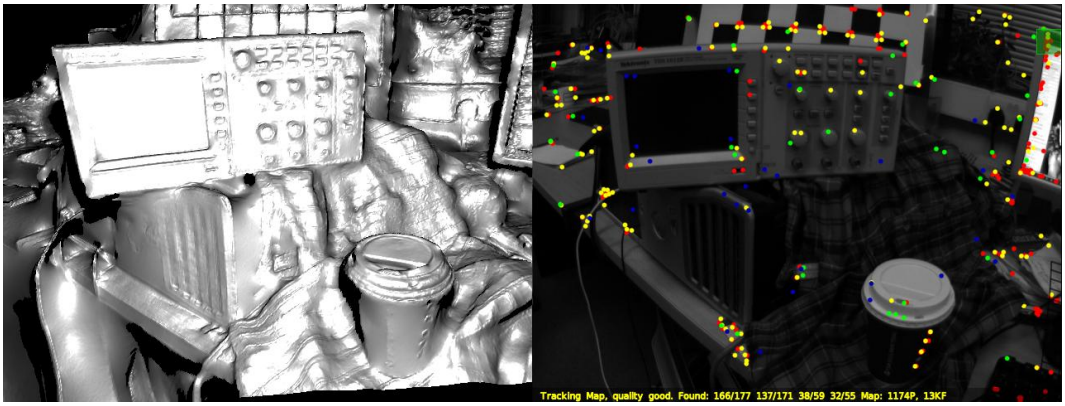


Figure 6.6: DTAM's dense surface map, Phong shaded (left), compared to a snapshot of the PTAM system with its sparse feature map visible as coloured points (right).

into a virtual camera. Since such a model is maintained live, we benefit from a fully predictive surface representation, handling occluded regions and back faces naturally. We estimate the pose of a live camera by finding the parameters of motion which generate a synthetic view which best matches the live video image.

We refine the live camera pose in two stages; first with a constrained inter-frame rotation estimation, and second with an accurate 6 DOF full pose refinement against the model. Both are formulated as iterative Lucas-Kanade style non-linear least-squares problems as we have seen before, iteratively minimising an every-pixel photometric cost function.

To ensure that we converge to the global minimum whilst registering each new video frame to our model, we must initialise the system within the convex basin of the true solution. We use a coarse-to-fine strategy over a Gaussian power of two image pyramid (Section 4.4) for efficiency and to increase our range of convergence.

Inter-frame Rotation Estimation

Noticing that gross image motion for a fast hand-held moving camera is typically dominated by rotation, the pose of the live frame is first refined by considering only rotational motion that may have occurred since the previous video image. Making the simplifying assumption that no translation has taken place, we are able to disregard the scene and align only the previous and live images, $\mathtt{I}^p$ and $\mathtt{I}^l$, by a homography parameterised by the rotation $\mathtt{R}^{pl}$. This optimisation is more stable than 6 DOF estimation when the number of pixels considered is low, helping to converge for large pixel motions at lower resolutions in the pyramid, even when the true rotation is not strictly rotational. Figure 6.7 illustrates the photometric cost function being minimised for different pyramid strategies where inter-frame rotation estimation and full pose refinement occur at different levels within the image pyramid. We can see that inter-frame estimation can help avoid local minima.

By estimating inter-frame rotation first we also gain increased robustness to motion blur since consecutive images are taken at similar velocities and are hence similarly blurred. PTAM's tracker performs a similar but less direct step to inform feature matching, computing first the 2D rigid body pixel transform between consecutive images and then computing the 3D rotational motion which best fits

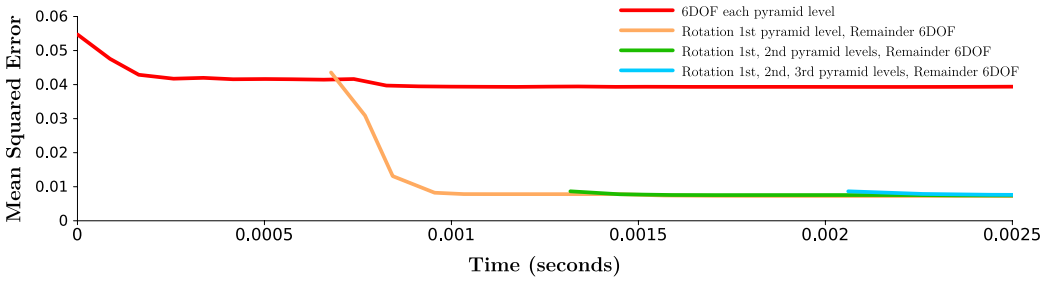


Figure 6.7: MSE convergence plots over time for an illustrative tracking step estimating rapid motion using different combinations of rotation and full pose iterations. Estimating rotation first can help to avoid local minima.

this image transform [63]. This coarse motion estimation can lead to more accurate feature location predictions when attempting to establish data association.

In order to achieve robustness to motion blur relative to the fixed, unblurred model, we would need either to attempt to deblur the live video image before matching, or incorporate a model of exposure within the cost function, effectively blurring the model within each iteration, as described by Mei and Reid [83]. These methods significantly increase the processing cost of alignment, making prior inter-frame estimation appealing to reduce total iterations for model-based blurring methods.

We proceed using the machinery from Section 4.6 to estimate $\mathtt{R}^{pl}$. Our current *best* live camera pose, $\hat{\mathtt{T}}^{wl}$ can be computed via composition and refined by a full pose minimisation against the model.

View Prediction

Given our best estimate of the live camera pose, $\hat{\mathtt{T}}^{wl}$, we project the dense model into a *virtual* camera v at location $\mathtt{T}^{wv} = \hat{\mathtt{T}}^{wl}$, with colour image $\mathtt{I}^{v}$ and inverse depth image ξ_v. These can be rendered efficiently on the graphics card by setting up an OpenGL camera that matches the intrinsics of the real camera, situated in the desired location. Using an off-screen framebuffer, we render each keyframe's depth map with texture using Vertex Buffer Objects to store the depth data, and Pixel Buffer Objects to store the texture (Section 2.5.3). Vertex and Fragment Shaders (Section 2.5.4) allow for the colour image $\mathtt{I}^{v}$, and depth image, ξ_v to be generated quickly within the same rendering process.

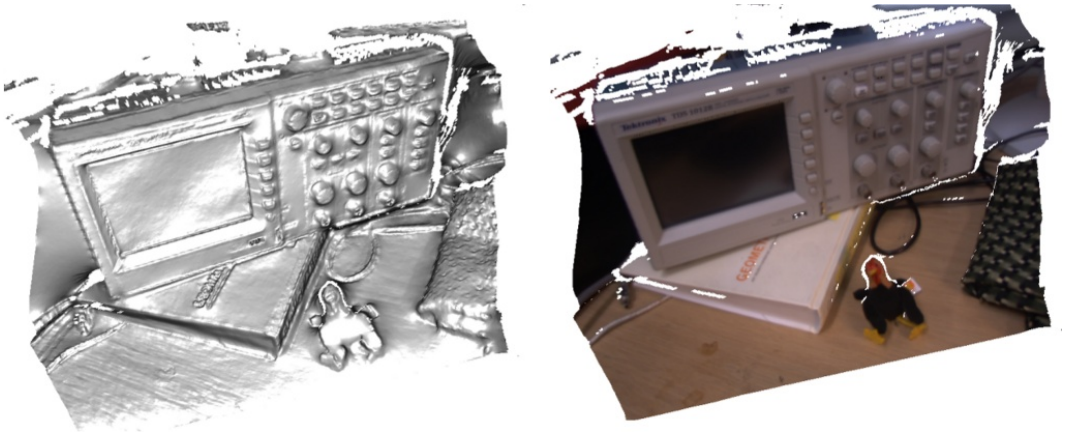


Figure 6.8: The dense textured surface model enables view prediction over wide baselines.

By rendering each keyframe into the virtual camera in this way to create a new predicted coloured depth map, occluded surfaces are naturally hidden provided the occluding surface has been modelled. Since we have a continuous surface representation, we can additionally invalidate back faces — keeping them as occluding surfaces, but excluding per-pixel costs that include them for tracking. One disadvantage of this scheme is that overlapping surface data is not merged into a consistent surface. As such, noisy depth data cannot be naturally improved. This is an area of future work.

During view prediction, we also record the minimum and maximum observed depths within the scene which helps to set up a more appropriate interval ξ_{min} to ξ_{max} for future cost volumes.

Full Pose Refinement

Performing a full view prediction from the complete model into a virtual camera v with pose $\mathtt{T}^{wv} = \hat{\mathtt{T}}^{wl}$, we assume that v is close to the true pose of the live camera, and perform a 2.5D alignment between $\mathtt{I}^v$ and the live image $\mathtt{I}^l$ to estimate $\mathtt{T}^{lv}$, and hence the true pose $\mathtt{T}^{wl} = \mathtt{T}^{wv}\mathtt{T}^{vl}$. We parameterise an update to our current estimate $\hat{\mathtt{T}}^{lv}$ by $\boldsymbol{\psi} \in \mathbb{R}^6$ belonging to the Lie algebra $\mathfrak{se}_3$, and define a forward-

compositional cost function relating photometric error to changing parameters:

$$F(\boldsymbol{\psi}) = \frac{1}{2}\sum_{\mathbf{u}\in\Omega}\Big(f_{\mathbf{u}}\left(\boldsymbol{\psi}\right)\Big)^2 = \frac{1}{2}\|f(\boldsymbol{\psi})\|_2^2, \tag{6.6}$$

$$f_{\mathbf{u}}(\boldsymbol{\psi}) = \mathtt{I}^l\left(\pi\left(\mathtt{K}\hat{\mathtt{T}}^{lv}\mathtt{T}^{lv}(\boldsymbol{\psi})\pi^{-1}\left(\mathbf{u},\xi_v\left(\mathbf{u}\right)\right)\right)\right) - \mathtt{I}^v\left(\mathbf{u}\right) \tag{6.7}$$

$$\mathtt{T}^{lv}(\boldsymbol{\psi}) = \exp\left(\sum_{i=1}^{6}\boldsymbol{\psi}_i \underset{\mathbb{SE}(3)}{\mathrm{gen}_i}\right). \tag{6.8}$$

This cost function does not take into account occluded surfaces directly; it will allow the live image to be warped such that pixels overlap with no consideration in the cost function for if they are visible. Instead, we assume that the optimisation operates over only a narrow baseline from the original model prediction. We could perform a full prediction setting $\mathtt{T}^{wv} = \hat{\mathtt{T}}^{wl}$ at every iteration but find it is not required.

We attempted to apply an ESM formulation for the 6 DOF pose refinement, but found experimentally that the simple forward-compositional approach was more stable with increase convergence. Baker *et al.* discuss 2.5D Image alignment in the context of medical datasets where they state that inverse compositional approaches are not appropriate in this setting [8]. Since ESM can be seen as both forward and inverse compositional, we believe their argument applies to ESM too.

We iteratively find the minimiser $\boldsymbol{\psi}^\circ = \arg\min_{\boldsymbol{\psi}} F(\boldsymbol{\psi})$ and apply the update $\hat{\mathtt{T}}^{lv} \leftarrow \hat{\mathtt{T}}^{lv}\mathtt{T}(\hat{\boldsymbol{\psi}})$ until $\hat{\boldsymbol{\psi}} \approx \mathbf{0}$ marking convergence, or until we run out of time. For details of the method, refer to Section 4.3.4.

Robustified Tracking

Feature-based trackers typically perform individual hard data-association when matching stored map elements in the live image. Mismatches are dealt with after-the-fact by establishing a *consistent set* of features where outliers have been discarded based on methods such as RANSAC. RANSAC relies on generating hypotheses by sampling from random minimal sets of features and testing their support. At any time, the set with the greatest support can be taken as a basis upon which to establish

inliers and outliers. Subsequently, the inlier set can be used to obtain a least squares fit which we now assume has not been corrupted by non-Gaussian outliers.

Under the sum of squared error cost function previously described for tracking, any observed pixels which do not belong to our static scene model or whose colour has dramatically changed due to lighting, may have a large negative influence within the solution. Minimisation of this cost function amounts to estimating the most likely pose given that all observed differences in live versus projected model pixels are Normally distributed about zero, which is not the case. Not only do unmodelled objects induce gross pixel-changes, but the brightness-constancy assumption — that observed scene colour does not change between frames — is violated by even simple Lambertian surfaces, where brightness is view-point dependent.

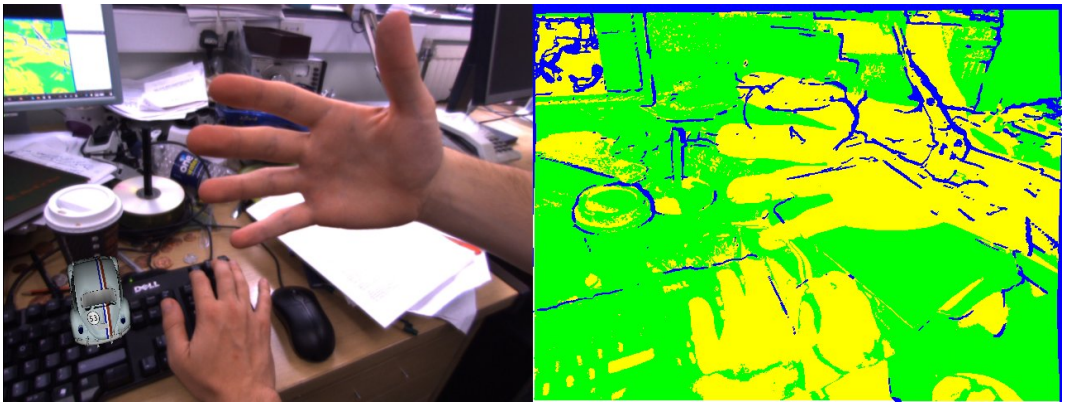


Figure 6.9: Augmented reality car appears fixed rigidly to the world as an unmodelled hand is waved in front of the camera. Pixels in **green** are used for tracking whilst **blue** do not exist in the original prediction and **yellow** are rejected (hand / monitor / shadow).

Analogously to RANSAC for feature-based pose estimation, we might think about a minimal set of pixels and testing support from the remainder of the image, but this represents too great of a cost for real-time application, with very many pixels each having little discrimination.

As discussed earlier in Section 4.5, non-linear least-squares problems such as ours can be robustified by changing the weight of pixels based on their error in a process called iteratively reweighted non-linear least-squares. This weight allows us to effectively change the error norm, minimising a different cost function to yield a more robust solution. In this way, all data is considered whilst the global model is

explicitly enforced, preventing the one-time hard data association of feature-based methods.

We choose the Tukey influence function which allows us to find the most-likely solution given assuming inlier pixel differences are Normally distributed and outliers follow a uniform distribution [48]. By changing a tuning parameter, the variance of the inlier distribution can be modified. The net result is that pixels whose photometric error falls above some threshold are disregarded completely.

One problem with this scheme is that the cost landscape can be made more complicated, particularly for non-convex penalisation terms such as that associated with the Tukey M-estimator. This can adversely affect performance where minimisation 'locks into' local modes, and where convergence is slowed by disregarding outliers which only appear that way when we are far from the solution.

Within DTAM, we dynamically change the Tukey tuning parameter within each least squares iteration. It is ramped down within the coarse-to-fine scheme as we converge to consider fewer outliers whilst maintaining a fast and wide basin of convergence. This makes it practical to track densely whilst observing unmodelled objects (Figure 6.9).

6.3.4 Initialisation

Since pose and dense structure are estimated in alternation and require one another to progress, we cannot use either of these when the system is first started. Instead, we bootstrap DTAM with a sparse feature-based tracking method to estimate camera pose before we have generated a model. Sparse feature-based pose estimation continues until the first keyframe with depth map has been constructed, when dense tracking can commence.

Finding a method for real-time dense initialisation of pose and structure remains an open topic. One approach may be to jointly estimate pose and coarse dense structure from some parametric representation, such as a regular grid of independent planes.

6.3.5 Keyframe Management

Refining a depth map requires that we have kept its associated cost volume — we typically keep only a few at a time on the GPU and must swap them in and out from main memory if we wish to work with more. A static, finalised depth map does not require a cost volume and uses little memory. We can keep many on the GPU for tracking purposes, taking just fractions of a second to render each during view predictions. For the small desktop environment that we target, we do not need to consider removing keyframes or selecting which of those that are visible we should use from tracking; we can efficiently use upwards of 20 at any time.

Given a current model and the live tracked pose of the camera, the biggest questions in maintaining the map are when should new keyframes be added, and which keyframes should comparison video images be integrated into. Since processing time is a premium for every video frame, the number of keyframes a comparison image can be incorporated into at one time is limited. Additionally, the memory required to store a cost volume is large relative to that which is available on current graphics hardware.

DTAM can operate in a number of modes which suit differing applications. The simplest policy is to add a keyframe when the user requests it, and to keep a single keyframe 'active' receiving comparison images at any one time. In this setting, each live video image is integrated into the cost volume of the active keyframe to continually improve its quality. When the user selects to add a new keyframe, the cost volume is detached from the current keyframe, cleared, and reattached to a new one whose reference image and pose becomes the current live frame and pose. This keyframe is included in the map and used for tracking once it has integrated enough data, typically just ten frames or so, and has sufficiently converged. Incremental regularisation of this depth map can be performed at frame-rate meaning that a keyframe can be used for tracking and also refined simultaneously. The previously active keyframe is now fixed and cannot be refined.

By integrating hundreds of narrow baseline video images into new depth maps, keyframe quality converges quickly. Handily, the required quality of a depth map for accurate tracking increases with baseline just as the accuracy of the depth map itself does. Since the view predictive quality of a single dense keyframe is so high, tracking can be sustained quite adequately from just one keyframe over a range of

poses and across large baselines, both from near and far away (see Figure 6.13 for example).

Operating in a fully automatic mode, DTAM is also able to determine when new keyframes are required. Since the system operates from a dense predictive model, each time a view prediction is made we can count the number of pixels in the virtual image that observe a portion of the model. When this threshold falls below a user-set threshold, a new keyframe can be added and we can progress as per the manual interactive mode. One problem with this method is that fast exploration can lead to depth maps of deteriorating quality.

The final keyframe policy that we have looked at is to switch the keyframe that is considered as *active* and will receive comparison images for refinement. All cost volumes are kept until we are sure that their depth maps are stable and they have received enough data. We select which keyframe is active by finding the keyframe whose average scene point is closest to the forward ray of the camera. If this keyframes cost volume is not in graphics memory, it is switched in from main memory; space is made by swapping out the cost volume currently on the GPU seen least recently.

The current nature of our system is that the pose of each keyframe remains fixed after it has been added. Although this representation is rather inflexible since structural adjustments cannot be made when closing large loops for example, in practice, it works very well in the small desktop scenarios for which it is designed. With dense structure estimated so accurately, and tracking occurring against the entire visible model, drift is very low. Extending DTAM to scale beyond the desktop remains future work.

6.4 Evaluation and Results

We have evaluated DTAM in the same desktop setting where PTAM has been successful. In all experiments, we have used a Point Grey Flea2 camera, operating at 30Hz with 640×480 resolution and 24bit RGB colour. The camera has pre-calibrated intrinsics. We run on a commodity system consisting of an NVIDIA GTX 480 GPU hosted by an i7 quad-core CPU. We present a qualitative comparison of the live running system including extensive tracking comparisons with PTAM and augmented

reality demonstrations in an accompanying video (see Appendix B).

6.4.1 Tracking Robustness

We have evaluated the tracking performance of our system against the openly available PTAM system, which includes many state of the art point feature tracking methods (Figure 6.10). Our results highlight both DTAM's local accuracy and extreme resilience to degraded images and rapid motion.

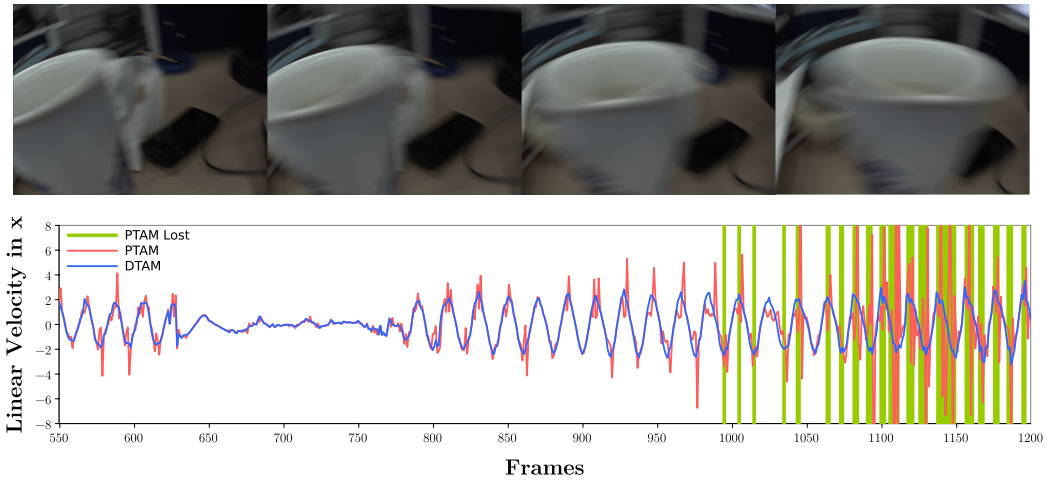


Figure 6.10: Linear velocities for DTAM (blue) and PTAM (red) over a challenging high acceleration back-and-forth trajectory close to a cup. Areas where PTAM lost tracking and resorted to relocalisation are shown in green. In comparison, DTAM's relocaliser was disabled. Notice that DTAM's linear velocity plot reflects smoother motion estimation.

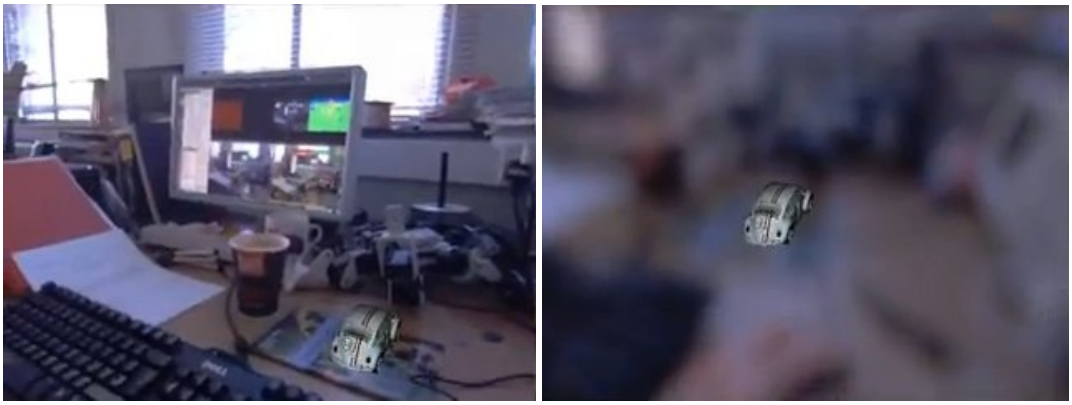


Figure 6.11: DTAM tracking stably throughout camera defocus.

6.4.2 Augmented Reality

In augmented reality (AR), we are interested in seeing virtual entities interacting live with real ones. Real-time localisation systems based on fiducial markers and real-time SLAM systems that operate in unconstrained environments have enabled virtual coordinate frames to be aligned to fixed real frames of reference. These can allow virtual objects to appear fixed rigidly to the real world by compositing live and real video.

For true immersion however, establishing a common frame-of-reference is not usually enough. In order for a virtual character to feel as though they inhabit a space, they must interact with it — this requires knowledge of the scenes structure. If scene geometry is known a-priori, model-based tracking methods such as those based on edges have enabled virtual characters to be appropriately occluded by real-world surfaces [60].

In the absence of known geometry, previous systems have shown that extracting even simple structure in constrained environments can offer interesting AR. For sparse feature-based methods, dominant planes can be extracted by best fit, enabling virtual objects to walk along flat surfaces [62]. Coarse multi-planar fits to sparse data have also seen Ninjas jump between simple planar objects [17]. In contrast with sparse feature-based SLAM systems, DTAM's dense model additionally enables high quality occlusion and occupancy reasoning for truly interactive AR.

Figure 6.12: The dense surface model can be used for augmented reality, offering a depth map for the live camera view and a rigid mesh with which physical models can interact.

With very accurate and smooth tracking, DTAM offers greater immersion for augmented reality applications, where it becomes easier to believe that virtual characters truly exist in the same coordinate system as the real world, and are not just mimicking it. Accurate occlusion geometry and free-space reasoning come for free from the map representation, enabling interactive physical characters, such as the toy car demonstrated in Figure 6.12. Making the most of the available pixels for both tracking and mapping enable high quality maps to be reconstructed, which allow for wide-baseline view prediction and stable tracking through varied motion and close to far camera pose (Figure 6.13).

Figure 6.13: The dense model enables view predictions to be made over extreme viewpoint changes, allowing tracking to remain accurate even over obscure angles.

6.4.3 Failure Modes and Possible Solutions

There are several limitations within the current system leading to many ways in which it can fail. These issues can be broadly classified into map corruption, limited rate of exploration and limited illumination robustness. These are not inherent shortcomings of a dense approach, but rather outstanding areas for research and engineering.

Starting on the subject of map corruption, it is worth emphasising that keyframes within our model are not adjusted in pose with respect to one another, in spite of the potentially large overlap between keyframes. This means that DTAM cannot make corrections after having explored a large loop for instance. Since DTAM tracks from the complete collection of all keyframes simultaneously, using all pixels, drift from

composing keyframes is low. Nevertheless, DTAM is unable to correct drift within its model when closing a loop around a room for instance.

DTAM's lack of global adjustment also means that poor quality keyframes can have a lasting negative effect on the map. Work within Chapter 5 is relevant here, where a pose graph structure is employed to achieve global consistency. Whereas in previous chapters we have looked at planar alignment, for DTAM a 6 DOF coloured depth-map alignment would form edges within the graph. For this, the photometric alignment already described is valid but does not make use of all the data — it ignores the fact that *both* keyframes have associated depth maps. Instead, a cost metric based on both forward and backward warping might be appropriate, or perhaps integration of a structure alignment metric, such as iterative closest point in addition to colour.

The uncertainty that arises from overlapping keyframes having slightly differing surfaces is not rigorously handled within tracking either. For efficiency, we use the surface that is closest (which occludes other surfaces) and make no effort to fuse the alternate hypotheses or weight their contribution. This can lead to poor model quality in regions of low texture and in turn compounding drift. One of the surprising features of DTAM is just how well individual depth maps line up without explicit fusion, and this helps to illustrate the accuracy of the dense approach.

Merrell *et al.* detail a method for fusing depth images by reasoning about visibility and occlusion, presenting results for building reconstructions from ground vehicles [86]. A different route forward is to take the approach of Newcombe *et al.*'s KinectFusion system, where depth maps are fused into a volumetric signed distance function which can adequately represent surface uncertainty [94]. This would enable the model to improve over time — the result becoming better than any of the individual depth maps. The difficulty here would be in allowing such a volumetric method to scale.

The second broad class of failure is in the limited rate of exploration supported by the system. This is due in part to the mechanisms by which we handle keyframes, usually choosing to finish a depth map before moving on to another. Since we do not adjust keyframes, moving on too quickly can lead to depth maps of poor quality. Although DTAM is competitive with other systems in terms of supported motions and explorations, it performs best when the user is in the loop and performs

motions that maximise information gain. The so called 'SLAM wiggle' is visible in our supporting video. Once an area has been mapped however, tracking is incredibly robust, so the user can then proceed within this space without too much care.

The third broad cause of failure for DTAM is in one of our primary assumptions: brightness constancy. We assume brightness constancy in all stages of reconstruction and tracking. Changing camera parameters can easily be handled by considering the camera-response function as a function of these parameters. In this way, pixel values can be normalised with respect to changing exposure. Although Section 6.3.3 describes how we can handle local illumination changes whilst tracking, we are not robust to real-world global illumination changes that can occur.

Irani and Anandan [52] showed how a normalised cross correlation measure can be integrated into the objective function for more robustness to local and global lighting changes. Instead of treating reflective changes in illumination as outliers, in future work, we are interested in joint modelling of the dense lighting and reflectance properties of the scene to enable more accurate photometric cost functions to be used. We see this as a route forward in attempting to recover a more complete physically predictive description of a scene.

Chapter 7

Conclusions

7.1 Contributions

We have presented a number of novel contributions to the problem of monocular SLAM centred around the belief that a dense, every-pixel approach at all stages has advantages over sparse feature-based methods both in the final quality of the map or model generated, and in actually improving the quality of SLAM operation itself, via for instance more accurate or more robust tracking.

The preliminary work in Chapter 3 showed a tetrahedralisation and visibility reasoning method for dense surface reconstruction from a feature-based map, but this path was quickly abandoned for the use of parametric direct image alignment methods. In Chapter 4 we showed that with modern adaptations and parallel GPGPU implementation we could use variants of the Lucas-Kanade method to create a high performance visual gyroscope. In addition, we demonstrated accurate on-board visual odometry for a normal road-going vehicle travelling at standard city speeds using a camera only observing road texture. This signal is suitable for real-time fusion with consumer GPS for practical low-cost motion estimation.

Chapter 5 presented full SLAM methods building on this alignment approach able to build consistent scene maps in real-time for either purely rotating cameras or those moving generally in 3D above a plane — both cases with practical applications. We also looked in depth at real-time visualisation of the reconstructions achieved.

Our systems included interleaved optimisation components to ensure global map consistency during live operation, and in the case of pure rotation we were also able to estimate camera intrinsic parameters during SLAM. The planar mapping was demonstrated both indoors and outdoors, and also in a document scanning application able to produce distortion free, super-resolved reconstructions live.

Chapter 6 presented the last main contribution, the DTAM system capable of fully dense real-time monocular SLAM from a hand-held single camera and a potential fully dense replacement for PTAM [62]. Our methods for alignment-based tracking are used here for live camera tracking by 6DOF model alignment against the dense 3D model. This approach improves significantly on point feature tracking due to factors such as true occlusion handling. Our tracking framework has robustness to extreme image blur and can handle rapid motion or even camera defocus.

7.2 Discussion and Future Research

The work presented within this thesis has built on literature from several domains, including SLAM, structure from motion, image registration and multiple view stereo. We have benefited from the massive increase in computational resources made available in commodity programmable graphics hardware to sit at the intersection of these research fields by considering parallelisable algorithms. More work is still required before we can state that we have solved any of the application areas listed in Section 1.2, but the research community as a whole has made significant progress over the past few years.

One of the clearest and most important areas for future work is in enabling a dense surface representation that can support uncertainty in order to improve the map as it is reobserved. We discussed this briefly in the previous chapter and pointed toward work such as that of Newcombe *et al.* who fuse depth map measurements from a Kinect sensor into a volumetric signed distance function [94].

With the right representation we might ask, can we make an even better model? The likely answer to this question is yes — photometric stereo is a field unto itself which has demonstrated that controlled light can allow us to reconstruct fine surface details by directly calculating surface normals. By forming an approximate dense model, we can start to look at how the appearance of the surface changes with

viewpoint. From here, we are in a good position to reason about environment lighting and perhaps other objects that have not been directly observed. With approximately known lighting, the photometric stereo literature becomes accessible and we might imagine putting this in the loop to really extract the most from the video images, helping to improve tracking quality further through more accurate view prediction.

Another important area for future work is to explore how we can make dense approaches scale to larger areas. Although storing dense models of the world sounds significantly more memory hungry than a sparse point-based map of the world, there is only a factor between them — both representations are sampling from the two-dimensional world surface. The current state of the art in scalable constant-time visual SLAM is well represented by the work of Strasdat *et al.* where the map is only locally metric and overall topological, following a recent trend of relative bundle adjustment approaches [119]. It is easy to imagine how dense systems like DTAM might be extended to make use of these approaches to scalable SLAM, relating local metric areas to one another via a 6 or 7 DOF transform.

Throughout this thesis, we have considered only rigid, immovable worlds. As a small concession to reality, we have attempted to develop robust systems that will continue to function when our core premise of rigidity is partly broken. To really find exciting and general use in robotics and augmented reality, visual SLAM systems must overcome this limitation. The first step might be simply to realise when the world has changed — from one static world to another, without worrying so much about the thing that moved. Forming these *life-long maps* are important for systems that will operate in uncontrolled spaces, such as road networks, over long periods.

A simple progression from the rigid world might be a world formed from multiple rigid bodies. In the tracking approaches presented in previous chapters, we use robust estimation to eliminate contributions from outlying pixels when determining camera pose. These outlying pixels can be segmented, and it might be interesting to recursively initialise a new tracker limited to this region in order to estimate an independent rigid body transform for each successive clutter of rejected pixels.

Humans of course are capable of perceiving and reasoning not only about rigid bodies, but deformable soft bodies, liquids, vapours, transparency and more. With future work in visual SLAM, perhaps one day, robots will too.

Appendix A

Lie Group Generators

Special Euclidean, $\mathbb{SE}(2)$

$$\underset{\mathbb{SE}(2)}{\text{gen}_0} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SE}(2)}{\text{gen}_1} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SE}(2)}{\text{gen}_2} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Special Linear, $\mathbb{SL}(3)$

$$\underset{\mathbb{SL}(3)}{\text{gen}_0} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SL}(3)}{\text{gen}_1} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SL}(3)}{\text{gen}_2} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

$$\underset{\mathbb{SL}(3)}{\text{gen}_3} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SL}(3)}{\text{gen}_4} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SL}(3)}{\text{gen}_5} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$\underset{\mathbb{SL}(3)}{\text{gen}_6} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SL}(3)}{\text{gen}_7} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

Special Orthogonal, $\mathbb{SO}(3)$

$$\underset{\mathbb{SO}(3)}{\text{gen}_0} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}, \underset{\mathbb{SO}(3)}{\text{gen}_1} = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SO}(3)}{\text{gen}_2} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Special Euclidean, $\mathbb{SE}(3)$

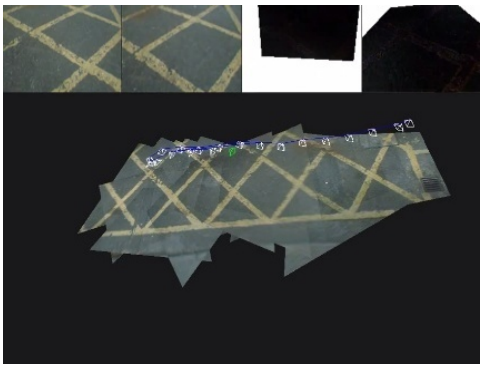
$$\underset{\mathbb{SE}(3)}{\text{gen}_0} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SE}(3)}{\text{gen}_1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SE}(3)}{\text{gen}_2} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\underset{\mathbb{SE}(3)}{\text{gen}_3} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SE}(3)}{\text{gen}_4} = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \underset{\mathbb{SE}(3)}{\text{gen}_5} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

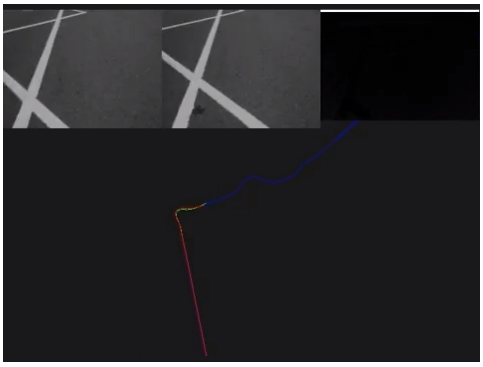
Appendix B

Video Material

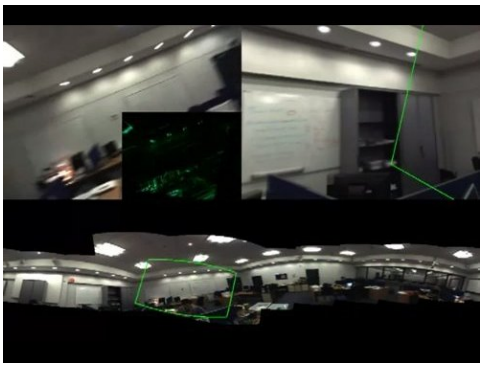
DTAM: Dense Tracking and Mapping in Real-time, *ICCV 2011*
R. A. Newcombe, S. J. Lovegrove and A. J. Davison.
http://youtu.be/Df9WhgibCQA



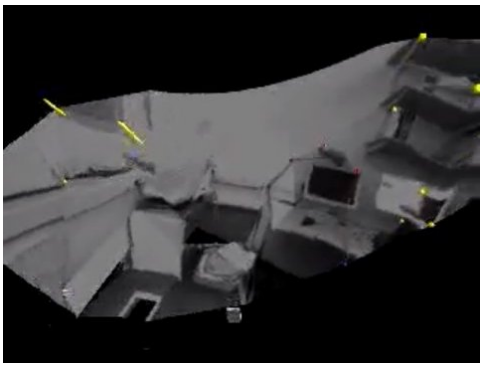
Real-time 6 DOF Planar Mosaicing
http://youtu.be/J-MCURRdJEs



Accurate Visual Odometry from a Rear Parking Camera, *IV 2011*
S. J. Lovegrove, A. J. Davison and J. Ibanez-Guzmán
http://youtu.be/l4P61eD0Fsc



Real-time Spherical Mosaicing using Whole Image Alignment, *ECCV 2010*
S. J. Lovegrove and A. J. Davison
http://youtu.be/9cY7ahgtZdI



Real-time Modelling from Sparse Point Features
http://youtu.be/olyHVM9zvjM

List of Figures

Bibliography

[1] L. Agapito, E. Hayman, and I. Reid. Self-calibration of rotating and zooming cameras. *International Journal of Computer Vision (IJCV)*, 45(2):107–127, 2001. 108

[2] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen. Interactive digital photomontage. In *ACM Transactions on Graphics (SIGGRAPH)*, 2004. 145

[3] S. H. Ahn. Song Ho Ahn's information and tutorials on OpenGL. URL http://www.songho.ca/opengl. 40, 45

[4] A. Angeli, D. Filliat, S. Doncieux, and J.-A. Meyer. Fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics (T-RO)*, 24(5):1027–1037, 2008. 117

[5] T. Azuma, S. Sugimoto, and M. Okutomi. Egomotion estimation using planar and non-planar constraints. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2010. 92

[6] A. Bak, S. Bouchafa, and D. Aubert. Detection of independently moving objects through stereo vision and ego-motion estimation. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2010. 91

[7] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework: Part 1. *International Journal of Computer Vision (IJCV)*, 56(3):221–255, 2004. 11, 68, 76

[8] S. Baker, R. Patil, K. M. Cheung, and I. Matthews. Lucas-Kanade 20 years on: Part 5. Technical report, Robotics Institute, Carnegie Mellon University, 2004. Technical Report CMU-RI-TR-04-64. 168

[9] P.A. Beardsley, A. Zisserman, and D.W. Murray. Sequential updating of projective and affine structure from motion. *International Journal of Computer Vision (IJCV)*, 23(3):235–259, 1997. 53

[10] S. Benhimane and E. Malis. Integration of euclidean constraints in template based visual tracking of piecewise-planar scenes. *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 133

[11] S. Benhimane and E. Malis. 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)*, 2004. 69, 86, 107

[12] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 1992. 81

[13] M. Bosse, P. Newman, J. J. Leonard, M. Soika, W. Feiten, and S. Teller. An atlas framework for scalable mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003. 17

[14] R. A. Brooks and T. Arbel. Generalizing inverse compositional and esm image alignment. *International Journal of Computer Vision (IJCV)*, 87(3):191–212, 2010. 79

[15] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision (IJCV)*, 74:59–73, 2007. 108

[16] J. A. Castellanos. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. PhD thesis, Universidad de Zaragoza, Spain, 1998. 16

[17] D. Chekhlov, A.P. Gee, A. Calway, and W. Mayol-Cuevas. Ninja on a plane: Automatic discovery of physical planes for augmented reality using visual SLAM. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007. 174

[18] A. Chiuso, P. Favaro, H. Jin, and S. Soatto. Structure from motion causally integrated over time. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(4):523–535, 2002. 16

[19] J. Civera, D. R. Bueno, A. J. Davison, and J. M. M. Montiel. Camera self-calibration for sequential bayesian structure from motion. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009. 108

[20] J. Civera, A. J. Davison, J. A. Magallón, and J. M. M. Montiel. Drift-free real-time sequential mosaicing. *International Journal of Computer Vision (IJCV)*, 81(2):128–137, 2009. 86, 109

[21] L. A. Clemente, A. J. Davison, I. Reid, J. Neira, and J. D. Tardós. Mapping large loops with a single hand-held camera. In *Proceedings of Robotics: Science and Systems (RSS)*, 2007. 17

[22] A. I. Comport, E. Malis, and P. Rives. Accurate quadri-focal tracking for robust 3D visual odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007. 69, 92, 157

[23] M. Cummins and P. Newman. FAB-MAP: Probabilistic localization and mapping in the space of appearance. *International Journal of Robotics Research (IJRR)*, 27(6):647–665, 2008. 117

[24] J. Davis. Mosaics of scenes with moving objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1998. 145

[25] A. J. Davison. *Mobile Robot Navigation Using Active Vision*. PhD thesis, University of Oxford, 1998. 16

[26] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2003. 16, 52, 68

[27] A. J. Davison, N. D. Molton, I. Reid, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(6):1052–1067, 2007. 11, 63

[28] F. Dellaert and M. Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research (IJRR)*, 25:1181–1203, 2006. 18

[29] F. Devernay and O. Faugeras. Straight lines have to be straight. *Machine Vision and Applications*, 13:14–24, 2001. 29, 30

[30] U.R. Dhond and J.K Aggarwal. Structure from stereo - a review. *Proceedings of the International Conference on Systems, Man and Cybernetics, (SMC)*, 19(6):1489 –1510, 1989. 155

[31] T. Drummond and R. Cipolla. Visual tracking and control using Lie algebras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999. 35, 75

[32] D.-Z. Du and F. Hwang. *Computing in Euclidean geometry*. World Scientific Publishing Co., Inc., 1992. 62

[33] E. Eade and T. Drummond. Edge landmarks in monocular SLAM. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2006. 19

[34] E. Eade and T. Drummond. Scalable monocular SLAM. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006. 18, 68

[35] E. Eade and T. Drummond. Monocular SLAM as a graph of coalesced observations. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2007. 18

[36] C. Everitt. Projective texture mapping. Technical report, NVIDIA, 2001. White Paper. 43

[37] O. Faugeras, QT Luong, and S. Maybank. Camera self-calibration: Theory and experiments. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 1992. 108

[38] O. D. Faugeras and F. Lustman. Motion and structure from motion in a piecewise planar environment. *International Journal of Pattern Recognition in Artificial Intelligence*, 2(3):485–508, 1988. 134

[39] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 108

[40] A. W. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 311–326. Springer-Verlag, June 1998. 16

[41] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007. 11, 52, 154

[42] David Gallup, Marc Pollefeys, and J. M. Frahm. 3D reconstruction using an n-layer heightmap. In *Proceedings of the DAGM Symposium on Pattern Recognition*, 2010. 157

[43] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Proceedings of Robotics: Science and Systems (RSS)*, 2007. 18

[44] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, 1999. 17

[45] M. Habbecke and L. Kobbelt. Iterative multi-view plane fitting. In *Proceedings of the International Workshop on Vision, Modelling and Visualization (VMV)*, pages 73–80, 2006. 134, 154

[46] M. Habbecke and L. Kobbelt. A surface-growing approach to multi-view stereo reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007. 11, 134, 154

[47] C. G. Harris and J. M. Pike. 3D positional integration from image sequences. In *Proceedings of the Alvey Vision Conference*, pages 233–236, 1987. 16

[48] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004. 83, 155, 170

[49] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981. 67

[50] P. J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):pp. 73–101, 1964. 85

[51] A. Iketani, T. Sato, S. Ikeda, M. Kanbara, N. Nakajima, and N. Yokoya. Video mosaicing based on structure from motion for distortion-free document digitization. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, 2007. 145

[52] M. Irani and P. Anandan. Robust multi-sensor image alignment. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 1998. 68, 177

[53] M. Irani and P. Anandan. All about direct methods. In *Proceedings of the International Workshop on Vision Algorithms, in association with ICCV*, 1999. 68

[54] M. Irani, P. Anandan, J. Bergen, R. Kumar, and S. Hsu. Efficient representations of video sequences and their applications. In *Proceedings of Signal Processing: Image Communication (SPIC)*, pages 327–351, 1996. 68, 85, 107, 131

[55] M. Irani, P. Anandan, and M. Cohen. Direct recovery of planar-parallax from multiple frames. In *Proceedings of the International Workshop on Vision Algorithms, in association with ICCV*, pages 1528–1534, 1999. 68

[56] IXSEA, Landins: Georeferencing and positioning system. URL http://www.ixsea.com/en/products/10/landins.html, 2010. 97

[57] I.K. Jung and S. Lacroix. High resolution terrain mapping using low altitude aerial stereo imagery. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2003. 53

[58] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of the Eurographics Symposium on Geometry Processing*, 2006. 53

[59] B. Kitt, A. Geiger, and H. Lategahn. Visual odometry based on stereo image sequences with RANSAC-based outlier rejection scheme. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2010. 91

[60] G. Klein. *Visual Tracking for Augmented Reality*. PhD thesis, University of Cambridge, 2006. 96, 174

[61] G. Klein and T. Drummond. A single-frame visual gyroscope. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2005. 86

[62] G. Klein and D. W. Murray. Parallel tracking and mapping for small AR workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007. 11, 18, 49, 109, 110, 117, 158, 174, 180

[63] G. Klein and D. W. Murray. Improving the agility of keyframe-based SLAM. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2008. 19, 166

[64] G. Klein and D. W. Murray. Parallel tracking and mapping on a camera phone. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2009. 68, 86

[65] K. Konolige. Sparse sparse bundle adjustment. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2010. 106

[66] K. Konolige, M. Agrawal, and J. Solà. Large-scale visual odometry for rough terrain. In *Robotics Research*, volume 66 of *Springer Tracts in Advanced Robotics*, pages 201–212. Springer Berlin / Heidelberg, 2011. 69

[67] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g^2o: A general framework for graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011. 18, 99

[68] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:1465—-1479, 2006. 116

[69] S. Lieberknecht, S. Benhimane, P. Meier, and N. Navab. A dataset and evaluation methodology for template-based tracking algorithms. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2009. 69

[70] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Transactions on Graphics (SIGGRAPH)*, 1987. 157

[71] M.I.A. Lourakis and A.A. Argyros. The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm. Technical report, Institute of Computer Science (ICS), Foundation for Research and Technology — Hellas (FORTH), 2004. 106

[72] S. J. Lovegrove and A. J. Davison. Real-time spherical mosaicing using whole image alignment. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2010. 21

[73] S. J. Lovegrove, A. J. Davison, and J. Ibanez-Guzmán. Accurate visual odometry from a rear parking camera. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2011. 21, 90

[74] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 1999. 108

[75] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4(4):333–349, 1997. 17, 131

[76] B. D. Lucas. *Generalized Image Matching by the Method of Differences*. PhD thesis, Robotics Institute, Carnegie Mellon University, 1984. 67

[77] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1981. 11, 20, 67, 156

[78] E. Malis. Improving vision-based control using efficient second-order minimization techniques. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004. 11, 20, 69, 75, 81

[79] E. Malis and M. Vargas. Deeper understanding of the homography decomposition for vision-based control. Research Report RR-6303, INRIA, 2007. 134

[80] C. Martin and Hans Moravec. Robot evidence grids. Technical Report CMU-RI-TR-96-06, Robotics Institute, Pittsburgh, PA, March 1996. 57

[81] P. F. McLauchlan and A. Jaenicke. Image mosaicing using sequential bundle adjustment. *Image and Vision Computing (IVC)*, 20(9–10):751–759, 2002. 108

[82] C. Mei, S. Benhimane, E. Malis, and P. Rives. Efficient homography-based tracking and 3-D reconstruction for single-viewpoint sensors. *IEEE Transactions on Robotics (T-RO)*, 24(6):1352–1364, 2008. 75

[83] C. Mei and I. Reid. Modeling and generating complex motion blur for real-time tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008. 166

[84] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid. A constant time efficient stereo SLAM system. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2009. 92

[85] M. Meilland, A. I. Comport, and P. Rives. Dense visual mapping of large scale environments for real-time localisation. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2011. 158

[86] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J.-M. Frahm, R. Yang, D. Nistér, and M. Pollefeys. Real-time visibility-based fusion of depth maps. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2007. 176

[87] M. Miksch, B. Yang, and K. Zimmerman. Automatic extrinsic camera self-calibration based on homography and epipolar geometry. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2010. 97

[88] N. D. Molton, A. J. Davison, and I. Reid. Locally planar patch features for real-time structure from motion. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2004. 19

[89] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2002. 17

[90] C. Morimoto and R. Chellappa. Fast 3D stabilization and mosaic construction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1997. 107

[91] P. Moutarlier and R. Chatila. Stochastic multisensory data fusion for mobile robot location and environement modelling. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, 1989. 16

[92] A. Napier, G. Sibley, and P. Newman. Real-time bounded-error pose estimation for road vehicles using vision. In *Proceedings of the International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2010. 92

[93] R. A. Newcombe and A. J. Davison. Live dense reconstruction with a single moving camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 11, 157, 158

[94] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011. 176, 180

[95] R. A. Newcombe, S. Lovegrove, and A. J. Davison. DTAM: Dense tracking and mapping in real-time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011. 21, 153, 163

[96] P. Newman. *On the Structure and Solution of the Simultaneous Localization and Map Building Problem*. PhD thesis, University of Sydney, 1999. 16

[97] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004. 11, 91

[98] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23(1):–, 2006. 69

[99] E. Olson, J. J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006. 17

[100] Q. Pan, G. Reitmayr, and T. Drummond. ProFORMA: Probabilistic feature-based on-line rapid model acquisition. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2009. 51, 59

[101] Y. Park, V. Lepetit, and W. Woo. ESM-Blur: Handling & rendering blur in 3D tracking and augmentation. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2009. 90

[102] M. A. Paskin. Thin junction tree filters for simultaneous localization and mapping. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003. 18

[103] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. Visual modeling with a hand-held camera. *International Journal of Computer Vision (IJCV)*, 59:207–232, 2004. 57, 157

[104] M. Pollefeys, R. Koch, and L. Van Gool. Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 1998. 16

[105] M. Pupilli and A. Calway. Real-time visual SLAM with resilience to erratic motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006. 18

[106] Christoph Rhemann, Asmaa Hosni, Michael Bleyer, Carsten Rother, and Margrit Gelautz. Fast cost-volume filtering for visual correspondence and

beyond. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 160

[107] H. S. Sawhney and S. Ayer. Compact representations of videos through dominant and multiple motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 18:814–830, 1996. 85, 145

[108] H. S. Sawhney, S. Hsu, and R. Kumar. Robust video mosaicing through topology inference and local to global alignment. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 103–119, 1998. 107, 131

[109] H. S. Sawhney and R. Kumar. True multi-image alignment and its application to mosaicing and lens distortion correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 21:235–243, March 1999. 109

[110] D. Scaramuzza, F. Fraundorfer, M. Pollefeys, and R. Siegwart. 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)*, 2009. 92

[111] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision (IJCV)*, 47:7–42, 2001. 156

[112] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006. 154

[113] H.-Y. Shum and R. Szeliski. Construction and refinement of panoramic mosaics with global and local alignment. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 1998. 145

[114] G. Silveira and E. Malis. Real-time visual tracking under arbitrary illumination changes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007. 11, 68

[115] G. Silveira, E. Malis, and P. Rives. An efficient direct approach to visual SLAM. *IEEE Transactions on Robotics (T-RO)*, 24(5):969–979, 2008. 19, 132, 134

[116] P. Smith, I. Reid, and A. J. Davison. Real-time single-camera SLAM with straight lines. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2006. 19

[117] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Uncertainty in Artificial Intelligence*, pages 435–461. Elsevier, 1988. 16

[118] C. V. Stewart. Robust parameter estimation in computer vision. *SIAM Reviews*, 41:513–537, 1999. 84

[119] H. Strasdat, A. J. Davison, J. M. M. Montiel, and K. Konolige. Double window optimisation for constant time visual SLAM. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011. 19, 181

[120] H. Strasdat, J. M. M. Montiel, and A. J. Davison. Real-time monocular SLAM: Why filter? In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010. 19, 68, 109

[121] H. Strasdat, J. M. M. Montiel, and A. J. Davison. Scale drift-aware large scale monocular SLAM. In *Proceedings of Robotics: Science and Systems (RSS)*, 2010. 92

[122] J. Stuehmer, S. Gumhold, and D. Cremers. Real-time dense geometry from a handheld camera. In *Proceedings of the DAGM Symposium on Pattern Recognition*, 2010. 11, 157, 158

[123] R. Szeliski. Image mosaicing for tele-reality applications. In *Proceedings of the IEEE Workshop on Applications of Computer Vision (WACV)*, pages 44–53, 1994. 107

[124] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):1–104, 2006. 85, 107, 152

[125] R. Szeliski and D. Scharstein. Sampling the disparity space image. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26:419–425, 2004. 160

[126] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A. Y. Ng. Simultaneous mapping and localization with sparse extended information filters. In

Proceedings of the Fifth International Workshop on Algorithmic Foundations of Robotics, 2002. 18

[127] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical report, Technical Report CMU-CS-91-132, Carnegie Mellon University, 1991. 67, 108

[128] P. H. S. Torr and A. Zisserman. Feature based methods for structure and motion estimation. In *Proceedings of the International Workshop on Vision Algorithms, in association with ICCV*, 1999. 68

[129] M. Unger, T. Pock, M. Werlberger, and H. Bischof. A convex approach for variational super-resolution. In *Proceedings of the DAGM Symposium on Pattern Recognition*, 2010. 150

[130] R. Unnikrishnan. *Globally Consistent Mosaicking for Autonomous Visual Navigation*. PhD thesis, Carnegie Mellon University, 2002. 131

[131] B. Williams, G. Klein, and I. Reid. Real-time SLAM relocalisation. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2007. 116

[132] L. Williams. Pyramidal parametrics. In *ACM Transactions on Graphics (SIGGRAPH)*, 1983. 34

[133] K.-J. Yoon and I.S. Kweon. Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2006. 156

[134] C. Zach. Fast and high quality fusion of depth maps. In *Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, 2008. 157

[135] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime TV-L1 optical flow. In *Proceedings of the DAGM Symposium on Pattern Recognition*, 2007. 11

[136] Z. Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and Vision Computing (IVC)*, 15:59–76, 1997. 83, 84, 85

[137] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22:1330–1334, 2000. 29

[138] Z. Zhang and A. R. Hanson. 3D reconstruction based on homography mapping. In *Proceedings of the ARPA Image Understanding Workshop*, 1996. 134