Imperial College London
Department of Computing

# Dense Semantic SLAM

Renato F. Salas-Moreno

October 2014

Supervised by Prof. 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**

Simultaneous Localisation and Mapping (SLAM) began as a technique to enable real-time robotic navigation on previously unexplored environments. The created maps however were designed for the sole purpose of localising the robot (*i.e.* what is the position and orientation of the robot in relation to the map) and several systems demonstrated the increasing descriptive power of map representations, which on vision-only SLAM solutions consisted of simple sparse corner-like features as well as edges, planes and most recently fully dense surfaces that abandon the notion of sparse structures altogether.

Early sparse representations enjoyed the benefit of being simple to maintain as features could be added, optimised and removed independently while being memory and compute efficient, making them suitable for robust real-time camera tracking that relies on a consistent map. However, sparse representations are limiting when it comes to interaction, as for example, a robot aiming to safely navigate in an environment would need to sense complete surfaces in addition to empty space. Furthermore, sparse features can only be detected on highly-textured areas and during slow motion.

Recent dense methods overcome the limitations of sparse methods as they can work in situations where corner features would fail to be detected due to blurry images created during rapid camera motion and also enable to correctly reason about occlusions and complete 3D surfaces, thus raising the interaction capabilities to new levels. This is only possible thanks to the advent of commodity parallel processing power and large amount of memory on Graphic Processing Units (GPUs) that needs careful consideration during algorithm design.

However, increasing the map density makes creating consistent structures more challenging due to the vast amount of parameters to optimise and the interdependencies amongst them. More importantly, our interest is in making interaction even more sophisticated by abandoning the idea that an environment is a dense monolithic structure in favour of one composed of discrete detachable objects and bounded regions having physical properties and metadata.

This work explores the development of a new type of visual SLAM system representing the map with semantically meaningful objects and planar regions which we call Dense Semantic SLAM, enabling new types of interaction where applications that can go beyond asking the question of "where am I" towards "what is around me and what can I do with it". In a way it can be seen as a return to lightweight sparse-based representations while keeping the predictive power of dense methods with added scene understanding at the object and region levels.

**Acknowledgements**

This work could not be possible without the help and encouragement of my lab team and family.

My supervisors Prof. Andrew Davison and Prof. Paul Kelly have been a formidable source of guidance and generosity. They made it possible to receive the scholarship required to begin and complete my studies as there are very limited sources of funding for top-class research education available to non-European students coming from developing countries. I also thank Prof. Davison for giving me the space and faith to try new concepts as it motivated my creativity while keeping the independence that I always strive for.

I thank Ankur Handa for his friendship and wisdom that enabled me to understand intricate mathematical problems and making weekend work fun. Thanks to Richard Newcombe for keeping the bar high when it comes to producing solid work, Steven Lovegrove for writing beautiful programming tools like Pangolin and Hauke Strasdat for his approachability and patience. A very smart set of lab members that joined after me like Jan Jachnik, Jacek Zienkiewicz, Robert Lukierski, Lukas Platinsky and Hanme Kim have given me the confidence that great new work will be developed at the Robot Vision group that hopefully will leverage the output and ideas developed during the past 3.5 years of research. It has been a great pleasure to share lunch with all of them. Big thanks to Becky for her company and joyfulness.

Finally I express my gratitude to all my family: Corina, Fernando, Bruno and Maria del Pilar for their unconditional support on these many years of education.

# Contents

Chapter 1

# Introduction

Simultaneous Localisation and Mapping (SLAM) is the process of self-localising a moving entity in a previously unknown and uncontrolled environment. Examples of those entities include: automatic vacuum cleaners, domestic robots, quad-copters and spatially-aware smart phones or eyewear. An uncontrolled environment is one that has not been previously instrumented to aid navigation for instance by means of beacons, markers or motion-capture cameras. This problem is normally simplified by the general assumption that the environment is rigid and static and the only moving entity (*e.g.* the robot) has no previous knowledge of the environment structure.

In order to self-localise, such an entity will need to incrementally create a map of the environment as it moves. This map consists of a unified and persistent representation of the surroundings, and is commonly structured to be immediately queried for keeping track of the entity's motion.

The fundamental utility of the map is at the very minimum to serve localisation. Several SLAM systems have been developed over the years to make this process more robust, extensible and in general more efficient. Our work focuses around a narrower yet practical type of SLAM system using a single hand-held camera as the sole sensory input (commonly referred to as *Visual SLAM*) which can perform consistent, drift-free localisation and mapping in real-time at room-size scales.[1]

As we will see in more detail in Section 1.2, SLAM has evolved as a means to unify

[1] This is in contrast to methods like visual odometry that can estimate incremental motion accurately but suffer long-term drift as they do not attempt to build a consistent map.

the previously disjoint processes of mapping and tracking. Sensor technology, computing resources, but more importantly application domains themselves have shaped the approaches over the years. Solutions to address a particular domain are not necessarily readily applicable to another. For example, initial exploration of SLAM was rooted in the necessity of robots to instantly sense obstacles in a previously unknown environment and move freely in two dimensional space. These methods were developed before the availability of digital cameras and relied mostly on sonar readings and wheel odometry. Meanwhile, approaches to faithfully reconstruct 3D spaces out of photographs for visualisation purposes had no need to achieve immediate results to be valuable. These two parallel developments however have found common ground and are now seen as equivalent formulations of the same underlying problem of building a static map and estimating precise sensor position within it.

What we propose in this thesis is to shape a SLAM approach for the purpose of interaction via semantic interpretation of environments in real-time that we call *Dense Semantic SLAM*: a system capable of identifying meaningful discrete elements in a map using all available sensory information within the loop of SLAM itself, for the purpose of self-localisation and complex interaction. Not only are we interested in knowing the exact position and orientation of an entity in its environment, but equally important to us is to reason about the nature of the map structure that is incrementally being built, as we expect that any form of emergent intelligence between entities depends on their capability to anticipate the behaviour of the sensed world.

In doing so, several assumptions that have previously constrained earlier methods can be re-examined and improved. For instance, while most SLAM approaches assume a static environment and treat moving parts as outliers, we can anticipate that certain objects such as people and cars are likely to move and therefore consciously rely on background areas that are truly static for the purpose of determining reliable ego-motion. Or in order to achieve scalability and memory efficiency, knowledge of similar object shapes such as repeated chairs or even planar structures can be modelled more efficiently by sharing common geometric properties. More profound is the impact such anticipation could have if it were to enable robots to attempt to move obstructing objects that are in fact movable or to carefully operate in the presence of people.

## 1.1 Dense Semantic SLAM enabled applications

Moving beyond the goal of localising an entity would unlock the potential of more complex interactions between many cooperating agents in 3D space, rather than just using the resulting map for the sole purpose of tracking. Here we highlight a few applications enabled by the use of the *Dense Semantic SLAM* framework we propose:

1. **Object-aware personal robotics**. We envision future service robots that can safely move in uncontrolled environments like offices and houses, are able to interpret the nature of objects around and use this to achieve more sophisticated activities like moving suitable objects to make way (*e.g.* cleaning below a table initially obstructed by chairs), grasping them from the correct region or anticipating the likely location of people ahead of time for enhanced safety. Raw shape and colour information obtained from cameras are not enough to carry out these tasks. We believe that a database of objects annotated by humans with semantic properties like typical use, weight, grasping regions or strength will enable robots to leapfrog the limitations of pure vision systems.

2. **Object-aware augmented reality (AR)**. AR has been widely used to correctly position virtual elements on top of real video and several commercial applications are now available ranging from games to walking directions around the city. But particularly in video games where the focus is on interactivity rather than passive visualisation, very little progress has been made to enable virtual entities to interact with the real world in believable ways. The use of objects in the loop of SLAM would enable next-generation AR games where virtual elements can change their behaviour depending on what objects they are interacting with (*e.g.* bouncing differently on hitting a wall or a pillow). A complete realisation of this approach would be to create a *scene graph* representation of an environment from visual data only, which is incrementally built and queried in real-time.[2]

3. **Realistic virtual teleconferencing**. We envision a situation where participants in a video conference wearing smart eyewear are able to perceive

[2] A *scene graph* is a tree-like data structure commonly used to represent the internal state of a video game, composed of passive and active objects (or actors), shared geometric data, lighting sources, etc. An example scene graph subsystem commonly used in real-time graphics is OpenSG: http://www.opensg.org/.

remote users (rendered with avatars) as being physically present in the host environment which is mapped at the level of objects. For the host participants the experience resembles that of *augmented reality*, whereas for the guest participants (who are not actually physically present) the experience is that of *virtual reality*. Another benefit of the approach is the increased amount of compression that can be achieved by modelling the underlying image formation process and transmitting only state transitions. A full realisation of this usage scenario would require overcoming the uncanny valley with realistic modelling and animation of human characters.

4. **Mixed-reality shared spaces**. A semantic SLAM system used in conjunction with smart eyewear would be able to interpret the shape of objects and the extent of surfaces and project information onto the environment while using tracking information to render perspectively-correct views to all the users sharing the object-level map in real-time. This form of immersive computing contrasts with current ways to display information with devices like Google Glass where widgets float in front of the wearer, with the possibility of dangerously obstructing their field of view while walking through space.

5. **Virtual replacement of environment structures**. Understanding environment structures like walls, floor, ceiling and furniture would enable a system to generate novel renderings of the real world, for instance by virtually changing the type of carpet or colour of walls, or by eliminating structures altogether (*e.g.* unwanted furniture) and seeing through walls.

These exciting novel potential applications would only begin to emerge once dense semantic SLAM becomes widely available to users and robotics scenarios, having been engineered for mobile use and scalability while keeping real-time performance.

In the following section we will review the evolution of traditional SLAM systems that constituted the main building blocks to elaborate complex yet real-time systems and have already captured the imagination of users with devices like self-driving cars, spatially-aware phones and autonomous vacuum cleaners that are beginning to be introduced into the consumer world.

## 1.2 The evolution of Visual SLAM

Before the 1990s, robot localisation relied on *a priori known* maps of the environments in which the entity was deployed. For example the work of Leonard *et al.* performed robot localisation using a predefined map of geometric beacons such as planes, corners and cylinders [86][87] sensed by sonar arrays. The task of performing both localisation and mapping as part of the same simultaneous loop remained challenging due to the correlated nature of both problems, since inaccuracies in one lead to incorrect results in the other, necessitating a sound probabilistic model of their joint behaviour.

It was not until the pioneering work of Smith and Cheeseman in 1986 [148] and Moutarlier and Chatila [111] in 1989 that the fundamental theoretical components commonly present in modern SLAM systems were clearly defined. They proposed to maintain joint probabilistic estimates for *both* map landmarks and sensor poses that are refined over time as new noisy measurements arrive. Their proposed use of the Extended Kalman Filter (EKF) [75] made it the mainstream framework upon which many SLAM systems relied for the next 20 years [15][22][36][43][108][114]. For example, in perhaps the first implementation of a SLAM system, Leonard *et al.* [88] revisited their previous work and applied the EKF framework to successfully localise a robot and *simultaneously* build a map of the scene using servo mounted sonar sensors. The first systems that implemented fully joint EKF filters for SLAM on real robots came towards the late 1990s such as [22][36][114].

The use of vision for mobile robot navigation can be traced back to the work of Moravec in 1977 [109][110] for obstacle avoidance using image region matching. Probably the first system which implemented full joint EKF SLAM approach using vision was by Davison *et al.* in 1998 [36] which performed active vision to focus the camera head on prominent features for building a 3D map and localising a wheeled robot platform moving in 2D space (see Figure 1.1). Thanks to these developments, future SLAM systems had the building blocks necessary to begin to move away from wheeled robots with strong 2D motion assumptions.

The DROID system developed by Harris and Pike [64] successfully builds 3D maps from video sequences in real-time though it neglects correlations due to the common camera motion and is therefore incapable of closing loops and correcting drift.

In 2003, Davison *et al.* presented a remarkable Visual SLAM system called Mono-

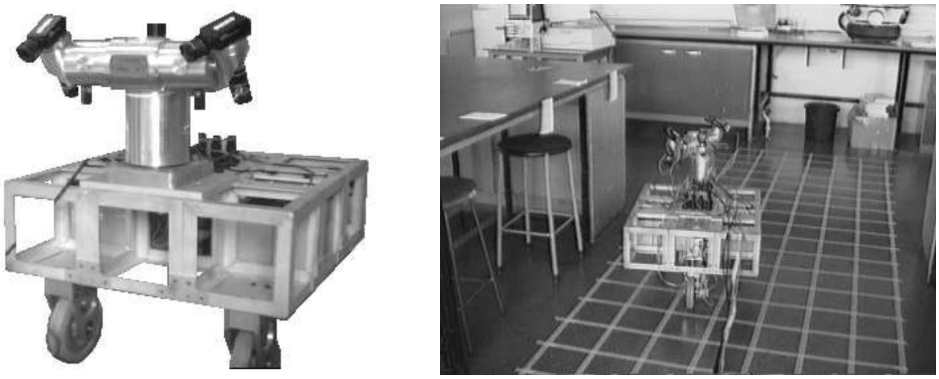


Figure 1.1: Robotic platform used in the active vision SLAM work of Davison *et al.* [36] in 1998. © Andrew Davison.

SLAM [37] (see Figure 1.2), being the first to demonstrate the use of a hand-held monocular camera to perform long-term incremental real-time tracking and mapping solution following the EKF approach and adopting image-based feature matching and pose estimation techniques studied in the closely related field of Structure from Motion (SfM). Due to the joint state estimation of both landmarks and camera poses the system had limited scalability to no more than 100 features in order to stay within the budget for real-time operation (the worst case complexity of filtering methods like the EKF is proportional to the cube of the size of the state vector). An earlier approach by Chiuso *et al.* [27] described a monocular visual SLAM system based on the EKF assuming however that the features were always visible, while Davison's was able to initialise new features, re-detect them after periods of occlusion and even handle small loop closures.

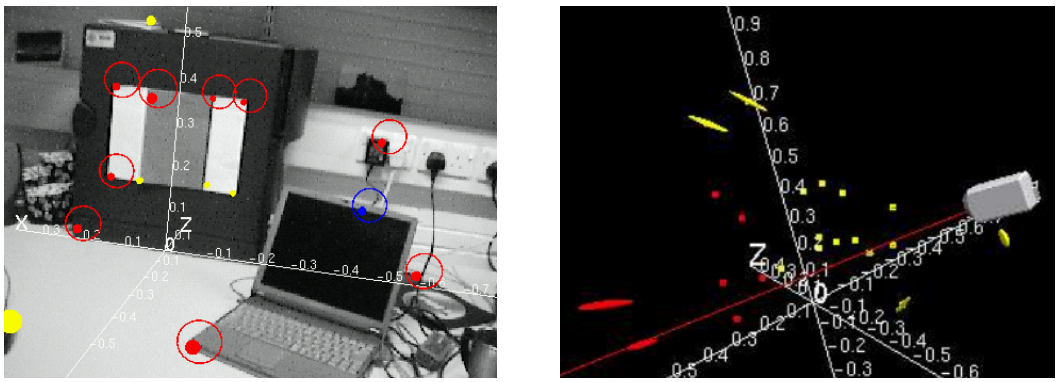


Figure 1.2: MonoSLAM [37][38] introduced in 2003, used a sparse set of corner features measured from a monocular camera, mapped in 3D and used to keep track of the 6 DoF sensor pose. © Andrew Davison.

The sequential nature of the EKF that linearises the non-linear sensor and motion models also produces errors, adding another factor to limit the scalability of filtering approaches like MonoSLAM for creating larger maps. To address this, other work focused on creating more limited sub-maps to compose larger ones (see for instance Bosse *et al.* [16] and Clemente *et al.* [28]).

Rather than keeping an explicit map of landmarks, a different approach known as *consistent poses* was developed by Lu and Milios in 1997 [99]. This approach consisted of keeping only pose-to-pose constraints generated after successful alignment of consecutive range scan measurements. Similarly Agrawal *et al.* [3] and Grisetti *et al.* [62] adopted a pose-graph optimisation approach that is able to distribute errors across a graph of constraints once loop closures are detected. Closely related is the work by Milford *et al.* on RatSLAM between 2003 and 2008 [105][106][107] where a *semi-metric topological map* of the environment is kept, inspired by computational models of rodents' hippocampus.

Other prominent approaches that move away from a metric map representation include that of Cummins *et al.* who developed a pure topological system called FAB-MAP between 2007-2009 [32][33][34]. Here the captured images are transformed into the space of appearance to build the map via a *bags of words* representation [147] and used this to query the likelihood that new measurements came from known places of the map or a new place, while taking into account the effect of perceptual aliasing arising from common structures such as repeated wall patterns. This method is useful during large scale exploration followed by loop closing as it is unlikely that two identical places visited after a period of neglect will be associated based solely on metric alignment as drifting errors are large.

As previously stated, *structure from motion* (SfM) is a closely related discipline aiming to extract high fidelity 3D representations from a set of 2D images by means of an offline joint optimisation process known as *bundle adjustment*. This enables it to find optimal 3D point positions and camera poses that minimise reprojection errors across the complete image set. It was formalised by Brown [21] in the area of photogrammetry back in 1958 and was revisited by Triggs *et al.* in 1999 [158] for the computer vision community. Bundle adjustment was applied in the work of Fitzgibbon and Zisserman to recover camera poses from image sequences [52] and Pollefeys *et al.* [124] for 3D reconstruction and self-calibration of cameras in 1998. Notable recent work in this field includes that of Agarwal *et al.* in 2009 [2] entitled

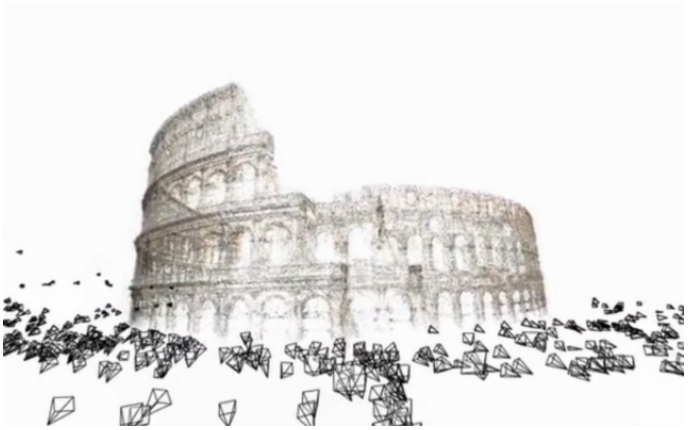


Figure 1.3: Building Rome in a day. In the work of Agarwal *et al.* [2] from 2009, thousand of internet images were collected to jointly optimise the 3D point position and camera poses with bundle adjustment to faithfully reconstruct structures like the Colosseum. © Sameer Agarwal.

'Building Rome in a day' that uses a collection of web images to generate convincing reconstructions of cities (see Figure 1.3).

What makes SfM different to Visual SLAM is its emphasis on accuracy at the expense of computation time, assuming all the data to process is already available and therefore has no need to deal with the uncertainty in sequential estimation. In addition, SfM assumes unordered image sets having feature correspondences at long baselines, therefore temporal constraints are of little importance. Ideas from SfM were nevertheless successfully applied to the real-time Visual SLAM domain by McLauchlan and Murray [103] using a variable state-dimension filter able to add or remove features as new image data arrives, and sliding window approaches like Nister *et al.* [115] to locally apply bundle adjustment and achieve good quality visual odometry in real-time without keeping a globally consistent map (which would drift in the long run).

A cornerstone new system that fully embraced SfM's bundle adjustment but showed how it could be use in real-time for globally consistent maps is PTAM, presented by Klein and Murray in 2007 [79]. As in MonoSLAM, PTAM represents the map via sparse features detected and tracked using a single monocular camera only, but it is able to substantially increase the features' density thanks to the realisation that mapping doesn't need to occur at the same rate as localisation (see Figure 1.4). This allows more processing time to be used for keeping a consistent

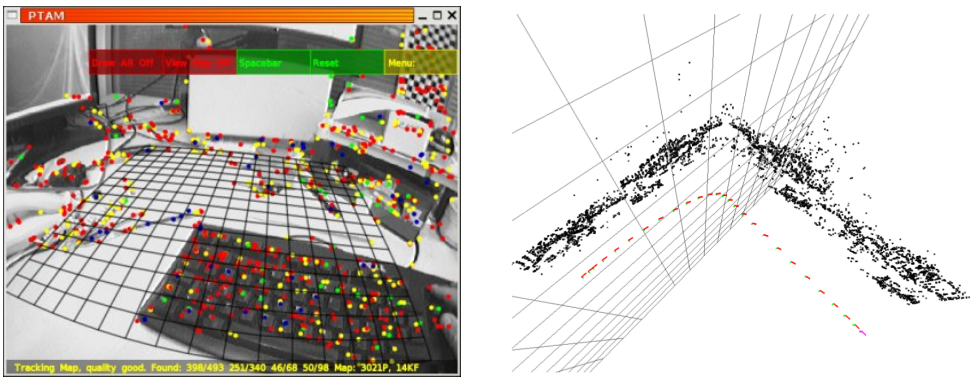



Figure 1.4: In PTAM presented by Klein and Murray in 2007 [79], mapping and tracking are performed in parallel threads with the mapping back-end able to spend more time to optimise many more points while the tracking front-end maintains framerate performance. © Georg Klein.

map by running bundle adjustment in a background thread, while a much lighter tracking thread maintains frame-rate operation. A globally consistent map is kept using keyframes that are sampled spatially rather than temporally and therefore the system is reliable and fast under long operational periods within a limited space.

Improved EKF filtering methods like Eade and Drummond [43] as well as enhanced keyframe-based bundle adjustment methods like Konolige and Agrawal [80] attacked the scalability, robustness and real-time performance of localisation via sparsification of the maps. The method of Eade and Drummond coalesced observations into independent local nodes that are connected into a common graph for global optimisation, while Konolige and Agrawal only kept a reduced set of relative pose information via skeletons to approximate the larger system.

The choice between filtering-based methods and keyframe-based bundle adjustment for achieving accurate mapping and localisation is therefore non-trivial. Strasdat *et al.* [149] offered some insights into the nature of both approaches and conducted experiments leading to the conclusion that accurate localisation performance is tightly coupled to the number of mapped landmarks and therefore any attempts to improve the descriptive quality of maps will win. This is the reason why keyframe approaches with their capability to optimise many more points in a map are a better choice at modern processing levels.

Corner-like feature detection is a popular abstraction in computer vision to simplify further image processing stages and has been extensively used in the previously

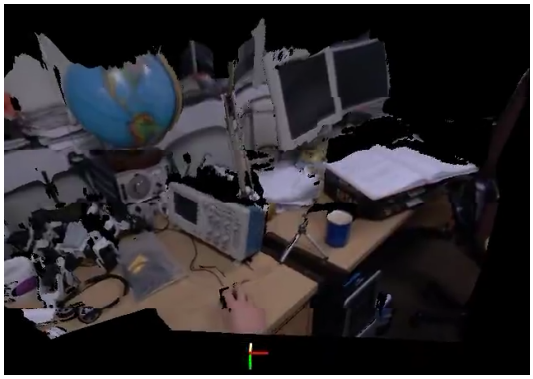


Figure 1.5: DTAM [113] **(left)** and KinectFusion [112] **(right)** developed by Newcombe *et al.* in 2011, redefined tracking and mapping as they were performed densely without feature based abstractions allowing reasoning about smooth surfaces while robustly tracking a camera even in the presence of motion blur. © Richard Newcombe.

reviewed Visual SLAM methods. However corners can be limiting for understanding smooth surface shapes and therefore precise occlusion handling. Furthermore, detecting them is difficult in regions of low texture or due to sensory artefacts like blur during fast motion.

Newer systems such as DTAM [113] and KinectFusion [112] presented in 2011 by Newcombe *et al.* abandoned feature detection in favour of dense surface representations, aided by the vast and commodity GPU processing power (see Figure 1.5). DTAM requires only a moving monocular camera and is able to generate depth maps by combining several measurements following a non-convex optimisation process that adds photometric error data terms and spatial regularisation to generate smooth depth estimates in areas lacking texture. In KinectFusion, the sensor used is able to directly extract depth samples in hardware even in areas of low texture by projecting an infrared speckle pattern onto the environment. In both systems, the extracted dense depth maps are further processed and merged into a unified 3D model, followed by dense tracking to estimate motion using all the available pixel information, thus making the system robust against fast motion and being naturally occlusion-aware.[3]

It is precisely the sophistication achieved by the two previously described dense SLAM systems and related platform technologies that inspired most of the work

[3]Motion estimation performed via *efficient second-order minimisation* (ESM) [100] in DTAM and *iterative closest point* (ICP) in KinectFusion [25].

described in this thesis. The arrival of commodity depth sensing devices like Kinect, access to the massive parallelism of GPU's and high quality mapping and tracking results (as well as their limitations) obtained by algorithms like KinectFusion showed us the path to devise the contributions described next.

## 1.3 Contributions

With the navigation utility of mapping already in place it is time to move beyond localisation towards interaction. Dense maps already provide substantial geometric quality for smooth and robust navigation but the entities we are interested in are certainly not just moving without purpose. An automatic vacuum cleaner's task is to clean the floor and it uses its map to track its motion while avoiding obstacles, whereas a smart phone uses the sensory tracking information to present well registered augmented imagery on top of captured video to enhance the user's perception.

Beyond obstacle avoidance and augmented representations, the entity's interaction with the environment could be further enhanced by making sense of the map structure itself: what objects are present in the scene, how extensive is the floor, how many steps are in a stairway, or can an object be moved and if so what is the best grasping region? This level of awareness is the driving factor behind our novel Dense Semantic SLAM approach that we contribute to the field.

Our journey towards semantic SLAM began by the realisation that many man-made environments consist of repeated elements and it is wasteful to re-map them from scratch. A background in video games development gave insights for representing complex virtual environments efficiently via instantiation, where a single geometric model of an object is shared across lightweight copies. Furthermore, to accelerate production times it is common to reuse pre-designed objects from a database rather than modelling or animating them from the beginning. At this stage, an initial attempt was to use the already mapped features from systems like MonoSLAM or DTAM to query a database of objects and superimpose the matches over the already built map.

SLAM is by definition a mobile system and tightly coupled to the available sensory hardware and processing units. Therefore any attempt to improve performance at the system level needs to consider the properties of measurements, algorithm

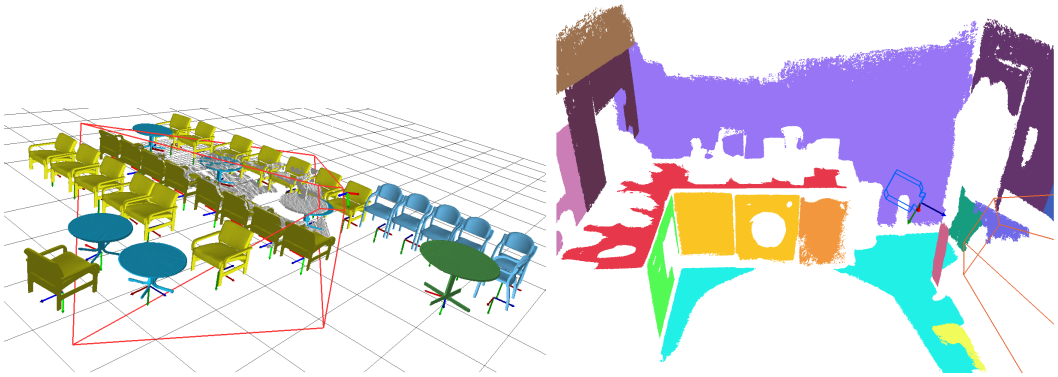


Figure 1.6: **(left)** SLAM++ [135] presented in 2013, maps scenes at the level of objects that are used immediately as landmarks to keep track of the sensor pose. **(right)** Dense Planar SLAM [134] presented in 2014, is able to detect planar regions in real-time and extend them as a camera browses the scene while maintaining high quality surface density.

characteristics, processing throughput and power consumption. With this view, we also explored implementing useful techniques for map optimisation such as bundle adjustment on newer hybrid hardware architectures consisting of integrated CPUs and GPUs.

With improved insight on the existing computing architecture landscape and the limitations of optimising large-scaled maps, we returned to examine object-level representations. This time however realising the advantages of using the objects directly as map features and being part of the SLAM loop itself, bringing semantic information and even higher levels of compression. This work led to the development of 'SLAM++: Simultaneous Localisation and Mapping at the Level of Objects' (see Figure 1.6 left).

SLAM++ needed careful pre-processing for segmentation and the detection algorithm limited its scalability to a handful of objects. Another recognition approach based on *random forests* [5][18] was later explored to better take advantage of the parallel processing power available and multi-class support. However interactive segmentation was still an obstacle and techniques to tackle this were evaluated such as GrabCut [132] and plane detection, using them inside the loop of SLAM itself leading to our newest system called 'Dense Planar SLAM' which is able to incrementally expand planar representations of environments starting from limited measurements (see Figure 1.6 right).

## 1.4 Publications

The following publications are the results of research carried during this PhD:

**SLAM++: Simultaneous Localisation and Mapping at the Level of Objects** [135]. Renato F. Salas-Moreno, Richard A. Newcombe, Hauke Strasdat, Paul H. J. Kelly and Andrew J. Davison. *Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013*

**Dense Planar SLAM** [134]. Renato F. Salas-Moreno, Ben Glocker, Paul H. J. Kelly and Andrew J. Davison. *Proceeding of the International Symposium on Mixed and Augmented Reality (ISMAR), 2014*

## 1.5 Thesis Structure

The rest of the thesis is structured as follows. In Chapter 2 we provide an extensive background review of key developments for 3D object recognition and semantic labelling. This material influenced the development of real-time object and plane recognition algorithms suitable for SLAM that were later applied in our SLAM++ and Dense Planar SLAM systems. Chapter 3 provides a description of mathematical notation, terminology and equations that serves as foundation for the rest of the thesis. In this chapter we also review novel computer architectures and programming models on which we base our work as the real-time constraints of SLAM make algorithmic development tightly coupled to these technologies. Chapter 4 describes our attempt to accelerate bundle adjustment on the now prevalent hybrid GPU/CPU architectures for parallel computing. Chapters 5 and 6 describe the core contributions of this thesis with SLAM++ and Dense Planar SLAM respectively. Finally in Chapter 7 we hypothesize future work required in the short and long term to make Dense Semantic SLAM even more interesting and applicable to real-world scenarios and offer closing remarks of the outcome of this research.

Chapter 2

# Towards Dense Semantic SLAM

Having introduced the aims of the thesis in Chapter 1 and reviewed the evolution of Visual SLAM systems as they stood at the beginning of this research, we now shift our attention to the recognition technology necessary to take us towards the semantic capabilities we are seeking.

During initial stages of our research we began approaching semantic SLAM by assuming that scenes are entirely made of objects and therefore reviewed several techniques to enable detection and pose estimation of objects from a database, potentially in real-time. We describe the reviewed methods in Section 2.1.

Later on we arrived at the conclusion that having object-level representations is not enough to semantically describe man-made environments as most scenes consist of large flat areas that can be more succinctly described parametrically without the need of objects. Under this new approach, we reviewed methods for scene labelling which we describe in Section 2.2.

## 2.1 Approaches for 3D Object Recognition

Recognising objects in images is a fundamental problem in computer vision and different approaches have been widely documented in the literature. Our exploration of this wide field has always been driven by certain ideal requirements on methods

which could be usable in a SLAM setting: real-time learning and prediction, precise 6 degrees of freedom (DoF) pose estimation, large-scale multiple class output and robustness against clutter and partial occlusions.

Object recognition is required when no prior information exists regarding the presence (found or not), location (or pose) or identity of an object. It is not necessary for instance when temporal information is available that could allow us to predict the location of an object in a new image based on evidence from a previous frame (a process known as *tracking*). However this prior information is not always reliable, particularly when tracking is performed over long periods of time which could lead to *drift* due to the accumulation of noise or total failure due to complete occlusion. Therefore even when an object could be initially detected manually (from a canonical pose or human annotation) it is always desirable to automate this process and integrate it into the system loop to correct long-term errors.

Since the sought object doesn't necessarily occupy a complete image, a common approach consists of polling candidate locations across the full image and accumulating evidence until the best candidates are extracted. Candidate poses are often discretised to accelerate the recognition process and this could be good enough in certain applications. Since we also require an accurate 6 DoF pose estimate of objects, an optimisation procedure is needed to minimise misalignment errors of the candidate poses (provided they lie within the basin of convergence for the optimisation procedure to find a solution). Under this view we can then describe object detection as a process to *bring likely objects towards the basin of convergence of a pose optimisation algorithm.*

Two main families of approaches for 3D object recognition exist in the literature as categorised by Lepetit *et al.* [90]. The first one consists of sliding an exemplar bounding box (sliding window) of a possible object view over an image until a suitable location is achieved by comparing the exemplar and target image statistics (we will refer to this as a *global approach*). The second one consists of testing individual pixels or discrete features and their close neighbours followed by grouping individual contributions to find peaks in a low-dimensional parameter space (we will refer to this as a *local approach*). This categorisation is not exclusive as for example newer *part-based approaches* first perform feature sampling and then extract area statistics in small regions around the feature positions.

### 2.1.1 Global Approaches

Global approaches rely on statistics from all the pixels in a bounding box to determine the likelihood of an object match. As such they can be susceptible to background clutter or partial occlusions if the window does not enclose the object precisely. Local approaches on the other hand rely on discrete pixels or features and as such can be made more robust by extracting feature descriptors that are robust to viewpoint or illumination changes or by grouping only likely members (with spurious matches removed via RANSAC [50] or geometric constraints).

Real-time global approaches are a popular choice for class specific detectors of objects like faces. The remarkable algorithm by Viola and Jones [161] uses an Adaboost cascade classifier based on simple Haar-like features to quickly reject unlikely regions and direct more complex processing to good candidates. This is also a learning-based method where a set of positive and negative samples are used to train the classifier offline (usually a few hours) while prediction runs in real-time thanks to the pre-computation of summed area tables of an image (also known as an *integral image*) for fast computation of statistics via subtraction of image regions.

The Histogram of Gradients (HoG) method proposed by Dalal and Triggs [35] describes objects by counting the gradient orientations of pixels within cells of an image sample and normalising their response across group of cells (called blocks), making the descriptor robust against changing illumination conditions and local deformations. This method has been successfully applied to problems like pedestrian detection.

Other interesting real-time detection algorithms using templates include the work of Hinterstoisser *et al.* that is targeted at texture-less objects. Beginning with Dominant Orientation Templates (DOT) [72] it encodes at each pixel of the template and image the most dominant orientations inside a neighbourhood (discretised orientations are used due to their robustness to illumination changes). This makes it robust against small deformations and shifting, and substantially reduces the number of test locations at which the template needs to be tested. The authors pay special attention to modern computer architecture features to efficiently compute error functions using *single instruction multiple data* (SIMD) bitwise operations, achieving about 12 fps on a dual-core laptop for a few hundred templates representative of an object. An extension known as LINE-MOD [71] uses multimodal templates to complement the

gradient information extracted from intensity images (mainly found on the object contour) with surface normals extracted via depth sensors like the Kinect (mainly found on the object interior). It further avoids cache misses by the use of linearised response maps as detailed in [70]. Later work [73] simplified the creation of templates from known 3D models and obtains refined 6 DoF pose estimates. This method automatically samples templates from a virtual camera positioned around a hemisphere that encloses the object at multiple scales (each template is also annotated with the known camera pose). Once the template is matched at runtime, the annotated pose is retrieved and refined via volumetric *iterative closest point* (ICP) [51].

In a similar spirit of improving the efficiency of sliding window approaches, Lampert *et al.* [83] developed a branch-and-bound algorithm to partition the search space of candidate windows hierarchically and reject those subsets whose upper-bounds are inferior than a globally determined score.

Another way to reject unlikely windows is through the concept of *Objectness*, first introduced by Alexe *et al.* [4], which is a measure to signal the presence of an object of any class. They define an object as having certain cues such as well-defined closed boundaries, appearance different from the background and saliency, which are combined in a Bayesian framework to measure the objectness of a window (sampled from the distribution of possible windows). A remarkable new procedure by Cheng *et al.* [26] is well suited for mobile real-time applications as it is able to estimate candidate windows at only 3ms on a laptop CPU while still achieving subsequent detection rates above 96.2%. The authors noted that under the concept of objects having a well defined closed-boundary, their normalised gradients are strongly correlated and therefore likely windows can be obtained by filtering the quantised search space with an $8 \times 8$ normalised gradient learned from annotated object data, which can be applied quickly with few bitwise operations.

### 2.1.2 Local Approaches

As previously described, local approaches rely on sampling individual pixel contributions. Some algorithms first extract salient features that are informative while others consider dense sampling of every pixel's contribution. An initial feature extraction phase reduces the complexity during object recognition as only the most informative pixels are passed through the pipeline. However, feature extraction is

not always reliable, especially during camera motion due to blurring effects and usually only works well on highly textured objects. For that reason, more recent work skips feature extraction altogether and consider every pixel information densely as we will see later.

An approach that we initially consider for detecting objects in our SLAM++ system (see Chapter 5) was based on the method developed by Drost *et al.* [41] and substantially accelerated using *GPU-Compute*. Drost's method consists of first describing an object shape globally with a data structure encoding the space of all possible *Point-Pair Feature* (PPF) values of the object. PPF's are extracted by pairing every two vertices of the object mesh and computing for each pair a four-dimensional descriptor of the relative position and normals of the oriented vertices. The PPF vectors are grouped in a hash-table, where each entry holds a group of similar descriptors hashed by the discretised PPF value, allowing constant-time access to any group. Once the global shape description is created, detecting an object at runtime consists of first identifying reference points on the scene depth image and then pairing them with every other point of the image (usually randomly sampled) to extract PPF and use their values to query similar ones in the model via the hash table. The similar PPF are used to cast votes in an accumulator space in a manner similar to the *Generalised Hough Transform* [8], indexed by the model reference point and the angle that would put matching vectors into alignment after pivoting around the reference point normal. Peaks in this accumulator correspond to likely scene reference points where most model PPF vectors commonly vote on a pivot. Once peaks are identified, the 3D object pose can be extracted via simple model to scene transformations.

Local methods like the one described above rely on having a fixed 3D model of the object to detect and the availability of depth information in the test images; as such they are normally known as *Shape Matching* in the literature. The benefits of these approaches are that they do not involve a lengthy training procedure and are simple to implement. When such geometric information is not available, or when the object intra-class variability is high (*e.g.* shapes of people), learning-based methods have been shown to exhibit better performance and are described next.

Among the best examples for performing object recognition via an initial feature extraction phase is the seminal work of Lepetit *et al.* [91]. He proposed a method to reformulate feature matching as a classification process allowing substantial runtime

speed-up as most of the complexity of the matching methods is moved into a training phase and only compact class representations are needed at test time. It also allows relaxation of the planarity assumption considered in previous approaches. To cast this problem as a classification method, each class is considered to be different views of a keypoint (extracted via the Harris corner detector). Interestingly the use of synthetically generated viewsets proved to work remarkably well and could be cheaply created from 3D texture mapping techniques.

To keep the misclassification rate low only the most characteristic keypoints are kept which are those that can be detected across many views. Illumination invariance is achieved by normalising the view intensities such that all views have the same minimum and maximum intensity. The generated viewsets are processed via PCA followed by K-means clustering to compactly represent them. To find the corresponding point, a nearest neighbour search is performed on the set of means. The whole object detection procedure takes 200ms on a 2GHz CPU compared to 1s using a SIFT based approach [97].

The combined PCA, K-means and nearest neighbour search classification procedure was later replaced by random forests [5][18] and described in [90] (we will describe in more detail the theory behind random forests in Section 2.3). This new approach is more robust and faster allowing real-time object tracking by detection. Increased robustness is achieved by only keeping stable keypoints across different scales. Runtime speeds are improved to 40 ms however training required about 15 minutes when growing 20 trees with the classic information gain approach for up to depth 10 and 200 keypoints with 100 views each.

In their follow-up work [89] Lepetit *et al.* picked only random tests instead of the entropy minimisation approach described previously, which slightly reduces detection performance but significantly decreases training time to a few seconds rather than minutes. Similarly Ozuysal *et al.* [123] followed the same random tests approach to pre-generate a forest structure enabling feature harvesting for online training using point features extracted as a camera orbits an object.

Ozuysal *et al.* then realised that the power of simply using random tests to train a forest originates not from the tree structure itself but from the combination of groups of binary tests. This led to the development of a new method referred to as *ferns* [122] which unlike trees has no hierarchy and uses a semi-naive Bayesian

approach to combine feature responses. This in turn allows handling of many more classes at higher classification rates than trees. The approach was also demonstrated to improve the tracking robustness in a SLAM system [121]. Matching 300 keypoints against 200 classes takes about 20ms on VGA images, while training takes 5mins.

A simpler and almost immediate training of ferns was successfully applied by Glocker *et al.* [61] to re-localise a camera in the event of tracking failure on a SLAM system like KinectFusion. Here sufficiently distinct keyframes annotated with camera poses are downscaled and binary encoded via simple responses to random tests. In this way any query image can be quickly compared with a block-wise Hamming distance to retrieve the most likely candidate keyframes and corresponding poses.

### 2.1.3 Part-based approaches

To improve recognition performance in challenging situations like varying illumination conditions and with object deformations, global approaches moved towards part-based models which can be seen as combining a local approach to initially identify parts with a global approach to group them and predict the location of entire objects. Some parts detection methods rely on *generative approaches*, requiring to model rather complex joint distributions to achieve good discrimination while others rely on *discriminative approaches* to directly model the conditional probability given some feature input.

In [84][85] Leibe *et al.* described an approach to establish a generative codebook of interest points paired with offsets to the object centroid. To detect an object instance in an image, interest points are detected and matched to the codebook, followed by the accumulation of votes in a Hough space to find the centroid and generate the best object location estimate from peaks. Interestingly, identifying the supporting parts of the peak (back-projection) allows segmentation of the object from its background; the complete method therefore can be seen as an intertwined bottom-up (recognition) and top-down (segmentation) approach.

Rather than matching interest points to codebook entries (which can be inefficient when large collections are involved), Gall *et al.* [54][55] formulated a discriminative approach using random forests [5][18] that directly maps parts to their corresponding Hough vote (thus calling this method *Hough Forests*), in a manner similar to Lepetit's approach [91] previously described, enabling improved performance by

sidestepping the time-consuming codebook matching of Leibe *et al.* [84]. Training is achieved by interleaving classification and regression information gain measures to reduce class and offset uncertainty during tree growing, leading to leaves having samples with similar labels (foreground or background) and similar locations (offsets with respect to object centroid). At runtime, patches are extracted from the target image and classified using the trained forest; the associated offsets of the patches classified as foreground are used to cast votes for the object centroid, followed by peak finding to localise one or more instances.

Extensions of the Hough Forests method were made to achieve robustness and real-time performance for head pose estimation and facial features detection by Fanelli *et al.* [45][46][44]. Other follow-up work included allowing multiple-class output in a single unified model by Razavi *et al.* [128] that scales sub-linearly with the number of classes (enabled by having a shared codebook of parts) while retaining similar detection accuracy compared to the linear scalability of applying class-specific Hough Forests in sequence.

Another discriminative approach is that of Felzenszwalb *et al.* [48] where object class recognition is achieved via a multiscale deformable parts model where each part captures local appearance and they are interconnected with springs (a representation known as Pictorial Structures [49]). This method can handle a much richer variability in class appearance (compared to global approaches like HoG) by using a mixture of models. For example one model can be trained to represent frontal views of an object while another to represent side views.

Other remarkable part-based approaches include Shotton *et al.*'s work for human pose recognition in real-time [141]. In this method, skeleton joint proposals are generated by first classifying individual pixels in a Kinect depth map and labelling them with different body parts using the random forest framework, requiring only simple depth comparison features that can be evaluated very quickly (and in parallel using the GPU as described in [140]). Once pixels are labelled, 3D joint positions for each part are determined by polling individual pixel contributions with mode-finding using mean shift [29]. As in Lepetit's method [90], impressive performance is achieved by using synthetic data for training.

Rather than classifying body parts, Taylor *et al.* [153] demonstrated how to directly infer the dense model coordinates of a canonical human pose by using a random

forest with a regression metric; correspondences are then used to obtain the pose by minimizing an energy function directly, avoiding the alternation between correspondence finding and pose optimisation as done for instance in ICP [14][25]. Similar dense correspondence achieved by a regression forest was successfully applied by Shotton *et al.* [142] to quickly re-localise a camera given a previously reconstructed 3D map of a scene.

Recently, Aubry *et al.* [7] presented a part-based method to detect objects in photographs from a large repository of 3D models (such as Google or Trimble 3D Warehouse). The method consists in matching individual image patches to an 800K collection of mid-level visual elements represented with HoG descriptors [35]. These visual elements are created offline from rendered 3D views of the models having their score responses globally calibrated. Individual detections are enforced to follow spatial layout and viewpoint constraints following the star model of Felzenszwalb *et al.* [48].

### 2.1.4 Information Retrieval Approaches

When the number of objects to recognise is substantially large (more than 1K), and particularly when we are interested in discriminating between individual instances within a class (*e.g.* tell the difference between a Toyota Prius and a Tesla Model-S rather than recognising them as cars) information retrieval approaches provide the scalability required.

Prominent work in this field includes that of Schmid and Mohr [138] where point features engineered to be invariant to viewpoint changes, illumination, scale, and partial occlusion form the basis of a voting-based image retrieval system.

Newer methods adopted a technique commonly known as the *Bags of Visual Words* (BoW) model, which is a visual analogue of text retrieval and consists of indexing and retrieval phases. Indexing begins by extracting salient keypoints in an exemplar image, computing a descriptor of the regions around them (such as histogram of edges if using SIFT [97] or responses to box filters if using SURF [11]), followed by vector quantisation of their responses (by K-means clustering for instance) and assigning each cluster to a word of a predefined dictionary (*codebook* generation). Finally a bit-vector (the *document*) is created to represent the presence or absence of words. At retrieval time, the same keypoint detection, description, and

codebook generation is applied to target images followed by a matching process to obtain a distance metric, resulting in likely candidates that maximise the number of matches.

Sivic and Zisserman utilised the BoW model on their Video Google system [147], bringing ideas from large-scale text retrieval systems into the computer vision community. They described a system capable of identifying a query object in many video frames, taking advantage of the temporal continuity of the data stream to reject false positives.

Nister and Stewenius [116] improved the recognition quality and scalability of the BoW model by creating a vocabulary tree. The branch-and-bound algorithm of Lampert *et al.* [83] described earlier was also used in order to efficiently localise objects within an image following the BoW model.

Cummins *et al.* [32] also leveraged the BoW model for place recognition to detect loop closures on a large-scale SLAM system.

The BoW model was also used to quickly obtain the pose of a camera in a given scene assuming a previously created 3D map of the environment (created for instance with SfM). This works by matching corresponding points in 2D images and 3D maps followed by the n-point-pose algorithm and filtering outliers with RANSAC. As matching is a time consuming process, Sattler *et al.* [136] observed that it would be necessary to have knowledge of the likely search cost that each feature would incur *before* matching them. After feature descriptors are assigned to words (using a vocabulary of about 100K words), the number of members represented by each cluster is a good indication of the search cost as a feature would only have to be linearly searched amongst other members of the same word. Therefore accelerated feature matching is achieved by sorting words in order of increasing expected cost for fast prioritised search. To reduce the impact of quantisation errors, Sattler *et al.* [137] also explored the co-occurrence of matches to perform a backwards 3D-to-2D search to enable features surrounding correctly matches on the 3D model to look back for those in the image via a smaller vocabulary on which the initial 2D features were assigned during the first pass.

A recent approach to accelerate feature matching was developed by Hartmann *et al.* [68]. They predict the *matchability* likelihood of features using a random forest trained on image sequences using as positive samples the features that can be

matched at least once. This allows to discard about 70% of features while retaining about 60% of the matches.

## 2.2 Approaches for 3D Scene Labelling

Semantically labelling images into objects classes (*e.g.* floor, wall, window, road, etc.) is a very challenging problem in computer vision. The task is not only difficult to achieve given the variety of changing conditions that objects undergo such as pose, illumination or non-rigid deformations but also the fact many ambiguities are difficult to resolve when considering local information alone. As an example, a leg can be part of a variety of animals, but only when the context is taken into account, such as animals with feathers, ambiguities can be resolved.

Early approaches tackle segmentation as an independent data-driven process without recourse to recognition. The benefits of their joint formulation began to be appreciated with the need to extract consistent segmentations and semantic labelling of regions. The seminal work of Tu *et al.* [159] combines the two approaches for the first time, demonstrating the ability to parse an image into generic regions (background) and meaningful objects (text and faces) by creating generative models of regions and objects activated by bottom-up image proposals. Similarly Winn and Jojic [163] described a method for unsupervised learning of a single object class via a hierarchical generative model that is directly usable for segmentation, combining bottom-up color and edge features with a top-down shape and pose model. Another early approach is that of Leibe and Schiele [85] who used parts-based recognition and Hough voting to recognise an object class and segment it with back-propagation of winning votes.

To achieve higher quality pixel-level segmentations newer approaches rely on *Conditional Random Fields* (CRF) [82], a discriminative graphical model initially formulated for labelling 1D text sequences. It proceeds by minimising an energy function consisting of unary and pairwise terms such that per-pixel labelling is both individually accurate and locally consistent between neighbours. Compared to related approaches like *Markov Random Fields* (MRF) [58], CRF avoids the construction of a complex generative model, making it much easier to train while incorporating long-range dependencies.

He *et al.* call this problem *image labelling* [69]. Their work uses features at the

local and global scales to take advantage of context and puts them into a CRF formulation generalised for 2D labelling.

The prominent work of Shotton *et al.* called *TextonBoost* [144] resolves recognition ambiguities by modelling shape, appearance and context information using textons as features for a discriminative boosted classifier that, when combined with edge information, can obtain spatially coherent segmentations for more than 20 object classes. This can be seen as a unification of recognition and segmentation in a similar spirit as the previously described approach of Leibe and Schiele [85] but applied for many more classes. An improved method known as *Semantic Texton Forest* [143] achieves even higher recognition performance and real-time speeds using a random forest.

Brostow *et al.* [20] follow the previously described texton forest framework in order to segment outdoor scenes captured from a moving vehicle. However, rather than using appearance features directly, they incorporate 3D structure and motion cues (such as the feature's height and closest distance to the camera path) and project them to the 2D image plane. These projected features can then produce accurate segmentations after dense pixel classifications using the texton forest.

Silberman and Fergus [145] contributed a dataset (NYU dataset) of scenes measured with a RGBD camera labelling the depth and colour frames with 13 classes such as bed, window, ceiling, floor, etc. This serves as an input to a CRF framework with unary potentials consisting of intensity and depth descriptors (such as SIFT [98] and Spin images [76]) in addition to location priors. 2D location priors provide the contextual information of likely places of objects after being projected onto the image plane (*e.g.* beds are usually on the lower part of images while ceilings are on the top) while 3D location priors offer likely depths of objects (*e.g.* walls are sensed further than furniture). Their follow-up work [146] was able to infer support relations between interacting objects leading to enhanced segmentation performance.

Floros and Leibe [53] generate semantic segmentation of street scenes via a CRF, taking advantage of the underlying 3D structure to generate valid 2D segmentations from projections in addition to temporal consistency between frames.

The recent approach by Farabet *et al.* [47] uses a multiscale convolutional network trained with features learned directly from images (rather than being hand-crafted), allowing it to capture shape, appearance and context (with the multiscale property

allowing to exploit long-range information). Couprie *et al.* [30] improved the previous approach with the addition of depth information, making it also amenable to real-time prediction with FPGAs.

## 2.3 Random Forests

We have previously described a few methods such as [54][90][141] that rely on random forests for discriminative class or regression prediction that are shown to outperform matching-based object recognition approaches. In this section we provide a more detailed look at the theory behind random forests as it will serve as a foundation for our real-time recognition work described in Section 5.2.4. A more extensive review of this technique and its application to computer vision problems can be found in the recent book by Criminisi and Shotton [31], from which we also the borrow some of the notation that follows.

Random forests can be used in tasks such as classification, regression, density estimation, semi-supervised learning, amongst others. A random forest consists of a set of $T$ trees such that a battery of tests are applied independently on each of them and results are later combined across the set. Each tree also contains internal nodes (splits) and terminal nodes (leaves) arranged in a hierarchy. Starting from the root split node at the top, each node branches into two child nodes across $D$ levels until leaf nodes $L$ are reached (see Figure 2.1). Each split node applies a test on incoming data and steers it to the right or left child according to the test result. Once the test data lands at a leaf node, the stored statistics are retrieved providing a class or regression prediction. The split node tests as well as leaf node statistics are established during a training phase from a dataset.

As an example, for the task of classification, suppose that we have a dataset containing a set of training points $\mathbf{v}$ alongside their class labels $c \in C$, with $C = \{c_k\}_{k=1}^{|C|}$. Each point is a vector of feature responses $\mathbf{v} = (x_1, ..., x_d) \in \mathbb{R}^d$. The classification forest's goal is to estimate the class probability $p(c|\mathbf{v})$ for a given test point. Each split node is a weak learner trained by optimising the parameter vector:

$$\boldsymbol{\theta}_j = \underset{\boldsymbol{\theta} \in \mathcal{T}_j}{\operatorname{argmax}}\, G(S_j, \boldsymbol{\theta}) \ , \tag{2.1}$$

where $\mathcal{T}_j$ is the space of all possible split parameters for node $j$ and $S_j$ is the set of training points arriving at the node. Usually the information gain metric is used as

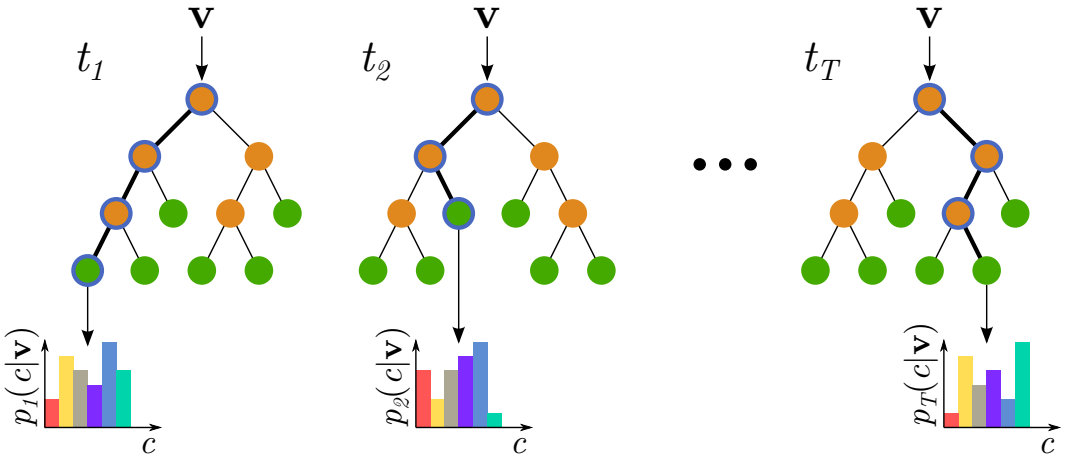



Figure 2.1: Random forest structure. A set of $T$ trees consisting of split nodes (orange) and leaf nodes (green) steers the incoming data $\mathbf{v}$ left or right according to the binary test results from the split nodes. The paths taken through each tree are highlighted and the final leaf node reached contains an empirical distribution of classes obtained from trained data. The trees shown contain up to $D = 3$ levels.

the objective function $G$ (see Equation 2.2). This metric tries to divide the incoming data points $S_j$ into left $S_j^{\mathrm{L}}$ and right $S_j^{\mathrm{R}}$ subsets, such that their class label impurity (or entropy) $H_c(S_j^i)$ is reduced. Therefore leaf nodes have much less class uncertainty compared to nodes further up the tree.

$$G(S_j, \boldsymbol{\theta}) = H_c(S_j) - \sum_{i \in \{\mathrm{L},\mathrm{R}\}} \frac{|S_j^i|}{|S_j|} H_c(S_j^i) \ , \tag{2.2}$$

$$H_c(S) = -\sum_{c \in C} p(c) \log \ p(c) \ . \tag{2.3}$$

The class histograms of data points arriving at leaf nodes are used as an empirical posterior $p_t(c|\mathbf{v})$. The final class posterior is usually computed as the average prediction amongst the set of trees:

$$p(c|\mathbf{v}) = \frac{1}{T} \sum_{t=1}^{T} p_t(c|\mathbf{v}) \ . \tag{2.4}$$

The use of trees for classification and regression tasks can be traced back to the initial work described by Breiman *et al.* [19], but was limited at that time to lower dimensional data. Later Amit and Geman [5] introduced the concept of using ensembles of trees with random node tests to improve accuracy and generalisation. Breiman further developed the method using *bagging* to randomly sample the training data and called the new technique *random forests* [18] as it is now known.

Random forests provide several desirable qualities that we are interested to have in a recognition module. These include the ability to handle multiple classes, generalisation to new data, suitability for parallel GPU architectures for real-time prediction [140] and a probabilistic output allowing control of which predictions to keep.

Application of the standard random forest method however is not sufficient for our task of object recognition. While a classification forest would allow us to predict the type of object at each individual pixel (see for example the dense classification results of Shotton *et al.* [141]) we are also interested in estimating the accurate 6 DoF object pose.

### 2.3.1 Hough Forests

Gall and Lempitsky [54] introduced a new method called *Hough Forests*, allowing them to combine both a classification and regression metric followed by a Hough-style voting mechanism [8] to jointly predict class and object locations in 2D images. The Hough Forest method is both fast and robust and was successfully applied for tasks such as pedestrian detection and head pose estimation [45][46]. We adapt this method to detect object instances in depth images while estimating their 6 DoF pose.

In the original formulation of Gall *et al.* the training points consisted of a set of patches $\{P_i = (I_i, c_i, \boldsymbol{\delta}_i)\}$ of $16 \times 16$ pixels each extracted from training images, with $I_i$ the appearance of the patch in different channels (raw intensities, derivative filter responses, etc.), $c_i$ class labels and $\boldsymbol{\delta}_i$ 2D offsets (see Figure 2.2). The method considered two class labels: foreground ($c_i = 1$) and background ($c_i = 0$). Foreground patches are those extracted from inside a bounding box of the target object while background patches lie outside the box. The 2D offset vector $\boldsymbol{\delta}_i$ is obtained by connecting the center of the patch to the centroid of the bounding box; in practice this value is only required for foreground patches and therefore a background patch's offset remains undefined. After training, the leaves will contain a class distribution of patches, and for those containing foreground labels the associated offsets vectors are stored and used at test time to cast votes for the object center in a Hough space.

The feature vector $\mathbf{v}_i = [h_1, h_2, ..., h_k]^\top$ for a patch $P_i$ consists of all the binary responses parametrised by $\boldsymbol{\theta} = (a, \mathbf{u}_1, \mathbf{u}_2, \tau)$, where $a$ is an appearance channel, $\mathbf{u}_1$

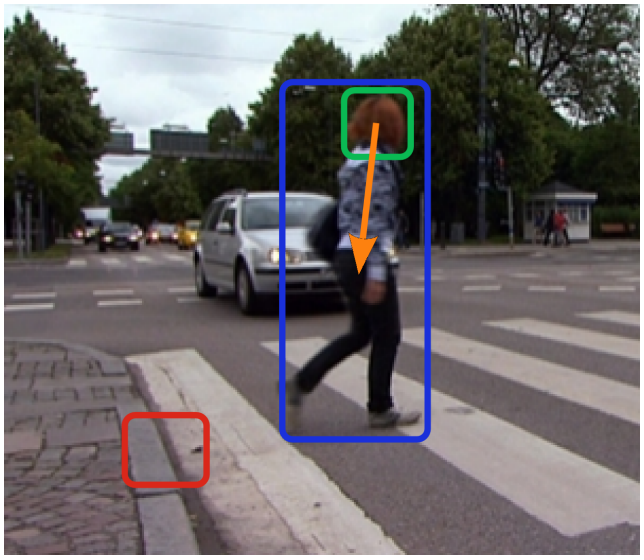


Figure 2.2: Hough Forest patches for 2D object localisation. Foreground patches (green) are extracted from inside the bounding box of the target object (blue), each having an offset vector pointing to the centroid of the box (orange). Background patches (red) are extracted from outside the bounding box.

and $\mathbf{u}_2$ two probe positions, $\tau$ is a threshold value, and the binary test is defined as:

$$h(\boldsymbol{\theta}, I) = \begin{cases} 0 & \text{if } I^a(\mathbf{u}_1) < I^a(\mathbf{u}_2) + \tau \\ 1 & \text{otherwise .} \end{cases} \tag{2.5}$$

Differently from standard classification or regression forests, a Hough forest tries to jointly decrease both the class label *and* offset vector uncertainty towards the leaves by interleaving the information gain measure (see Equation 2.2) with a regression metric defined as:

$$H_r(S) = \sum_{i:c_i=1} (\boldsymbol{\delta}_i - \boldsymbol{\delta}_S)^2 \ . \tag{2.6}$$

Therefore, likely patches with high foreground class probability are able to cast votes with low uncertainty for the centroid of the object.

## 2.4 Summary

In this chapter we reviewed the available methods found in the literature that will lead us to design novel real-time object recognition and scene labelling algorithms which will become important components in our semantic SLAM pipeline.

We found that most techniques have very little interest in real-time execution speeds and are mostly dedicated to dealing with single colour images. Only recent methods address execution speed as an algorithmic design consideration and fully embrace modern computer architecture for speed-ups.

Recent trends prompt for the re-evaluation of classical approaches for object recognition and SLAM, such as commodity parallel processing and depth sensing video devices. Therefore in the next chapter, in addition to introducing some useful mathematical definitions, we will review the latest computing architecture landscape that was available during this research.

Chapter 3

# Preliminaries

In this chapter we begin by establishing the notation used to describe equations consistently throughout the thesis and review some mathematical concepts that are frequently used in our area of research, such as rigid body transformations, Lie groups and their algebra and non-linear least squares optimisation.

In addition, we will describe a modern GPU architecture offering commodity parallel processing. This key processing technology enabled the development of dense SLAM systems like ours that can operate in real-time.

It would be challenging to code algorithms without a programming model that is both expressive and of high-performance. Therefore at the end of this chapter we will also review modern programming models that we used extensively to accelerate algorithms for tasks such as bundle adjustment, object recognition and plane detection.

## 3.1 Notation

We represent an $m$ dimensional vector in lower case **boldface** and columnar form:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix}, \tag{3.1}$$

or in row form $\mathbf{v}^\top = [v_1, v_2, ..., v_m]$. Matrices are denoted with upper case boldface such as $\mathbf{A}$.

Points in space are frequently represented using *homogeneous coordinates* rather than *Cartesian coordinates* as they allows us to represent a variety of transformations (like perspective transformation, translation, etc.) via matrix-vector multiplication. To homogenize Cartesian coordinates we add an extra $w$ term. For example the homogeneous coordinates of a 3D point $\mathbf{p} = [x, y, z]^\top$ are represented as $\dot{\mathbf{p}} = [x, y, z, w]^\top$, whereas a 2D point $\mathbf{r} = [x, y]^\top$ is represented as $\dot{\mathbf{r}} = [x, y, w]^\top$:

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ w \end{bmatrix} . \tag{3.2}$$

To convert back to Cartesian coordinates we divide by the added $w$ term (this is called *homogeneous projection* and is denoted by $\pi$). For example $\dot{\mathbf{p}} = [x, y, z, w]^\top$ becomes $\mathbf{p} = [x/w, y/w, z/w]^\top$. Setting $w = 1$ expresses the original Cartesian point and setting $w = 0$ expresses points at infinity (direction vectors):

$$\pi(\dot{\mathbf{x}}) = \begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix} . \tag{3.3}$$

## 3.2 Rigid Body Transformations

We represent rotation in 3D with a $3 \times 3$ matrix $\mathbf{R}$ and translation with a $3 \times 1$ vector $\mathbf{t}$. A point in space $\mathbf{p}$ can be rigidly transformed (rotated and translated) to a different place with respect to its reference frame or transformed to another reference frame via a matrix-vector multiplication:

$$\mathbf{p}_\mathrm{a} = \mathbf{R}_\mathrm{ab}\mathbf{p}_\mathrm{b} + \mathbf{t}_\mathrm{ab} = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} \mathbf{p}_\mathrm{b} + \begin{bmatrix} t_0 \\ t_1 \\ t_2 \end{bmatrix} , \tag{3.4}$$

where the subscripts on the points indicate the coordinate frames where their values are represented (*e.g.* $\mathbf{p}_\mathrm{b}$ is a point in coordinate frame b) and the subscripts on the transformation indicate the source and target coordinate frames, which must be read from right to left (*e.g.* $\mathbf{R}_\mathrm{ab}$ is a rotation bringing a point from coordinate frame b into coordinate frame a).

The expression above can be made more compact by using homogeneous coordinates for the point and storing the rotation and translation components in a single $4 \times 4$ matrix:

$$\dot{\mathbf{p}}_{\mathrm{a}} = \mathbf{T}_{\mathrm{ab}}\dot{\mathbf{p}}_{\mathrm{b}} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dot{\mathbf{p}}_{\mathrm{b}} = \left[\begin{array}{c|c} \mathbf{R}_{\mathrm{ab}} & \mathbf{t}_{\mathrm{ab}} \\ \hline \mathbf{0}^{\top} & 1 \end{array}\right] \dot{\mathbf{p}}_{\mathrm{b}} \ . \tag{3.5}$$

## 3.3 Lie Groups and Lie Algebra

Some of the problems that we solve require an optimisation framework allowing us to find model parameters that best explain a measured phenomena, for example to obtain the transformation parameters describing the motion of a camera in space by comparing a predicted view of a 3D map and a measured view of the scene from a camera. Such problems require solving non-linear systems iteratively, where transformations are updated in a number of small steps. The transformation matrices that we saw in the previous section however are complicated to use iteratively as only a small subset of the possible matrices are valid transformations, requiring extra verification steps to preserve, for instance, orthonormal columns in the case of rotation matrices.

The above mentioned complications arise from the *over-parametrisation* of transformation matrices, where 9 values are used to describe rotation or 12 for a the rigid body transformation matrix. In reality rotations only have 3 degrees of freedom (DoF), and therefore it is enough to express them with 3 parameters. Likewise a full rigid body transformation has 6 degrees of freedom. We note that there are several widely used minimal parametrisations of 3D transformations (*e.g.* Euler angles). However here we will show that understanding these transformations as members of a *Lie Group* leads to a straightforward way to use minimal parametrisations for optimisation. The following is an overview of the fundamental concepts. More detailed descriptions can be found on textbooks such as [56] and [160].

A Lie Group is a smooth differentiable manifold with derivatives of all orders. As such, moving between their immediate neighbours can be achieved with a minimal parametrisation.

As a group, they must satisfy the following properties when performing an oper-

ation $\otimes$ between members:

**Closure**: if $\mathbf{A}, \mathbf{B} \in G$ then $\mathbf{A} \otimes \mathbf{B} \in G$.
**Associativity**: if $\mathbf{A}, \mathbf{B}, \mathbf{C} \in G$ then $(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$.
**Identity**: an element $\mathbf{1}_G \in G$ satisfies $\mathbf{1}_G \otimes \mathbf{A} = \mathbf{A} \otimes \mathbf{1}_G = \mathbf{A}$ where $\mathbf{A} \in G$.
**Inverse**: there is an element $\mathbf{B} \in G$ such that $\mathbf{A} \otimes \mathbf{B} = \mathbf{B} \otimes \mathbf{A} = \mathbf{1}_G$ where $\mathbf{A} \in G$.

A 3 DoF rotation is member of the special orthogonal group $\mathbb{SO}(3)$ while a 6 DoF rigid body transformation is a member of the special Euclidean group $\mathbb{SE}(3)$.

### 3.3.1 Rotation Group: $\mathbb{SO}(3)$

Considering the Euler-angle representation of a rotation matrix as an example, with $\phi, \theta, \psi$ the rotation angles about the $x, y, z$ axis respectively, we know that:

$$\mathbf{R} = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\theta\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} .$$

The above can be further simplified using small-angle approximations (*i.e.* $\sin(\alpha) \approx \alpha, \cos(\alpha) \approx 1$):

$$\mathbf{R} \approx \begin{bmatrix} 1 & -\psi + \phi\theta & \phi\psi + \theta \\ \psi & 1 + \phi\theta\psi & -\phi + \theta\psi \\ -\theta & \phi & 1 \end{bmatrix} ,$$

and removing second order terms leads to:

$$\mathbf{R} \approx \begin{bmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\phi \\ -\theta & \phi & 1 \end{bmatrix} = \mathbf{I} + \phi \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} + \theta \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} + \psi \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} ,$$

or:

$$\mathbf{R} \approx \mathbf{I} + \phi\mathbf{G}_1 + \theta\mathbf{G}_2 + \psi\mathbf{G}_3 . \tag{3.6}$$

As we can see, any rotation matrix near the identity differs from it by a linear combination of the *Generator* matrices $\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$ with coefficients given by $\boldsymbol{\omega} = [\phi, \theta, \psi]^\top \in \mathfrak{so}(3)$. $\mathfrak{so}(3)$ is called the Lie algebra of the $\mathbb{SO}(3)$ group and spans the tangent space near the group's identity, with the generators being the basis for this

space with:

$$\left.\frac{\partial \mathbf{R}}{\partial \omega_i}\right|_{\boldsymbol{\omega}=\mathbf{0}} = \mathbf{G}_i \ . \tag{3.7}$$

To generalise this to other elements of the group we consider a point $\mathbf{p}$ rotating about an axis $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^\top$ with unit angular velocity ($\|\boldsymbol{\omega}\| = 1$) and an instantaneous velocity given by:

$$\frac{\partial \mathbf{p}}{\partial t} = \boldsymbol{\omega} \times \mathbf{p} \ . \tag{3.8}$$

We can express the above cross product as a matrix-vector multiplication by representing the vector $\boldsymbol{\omega}$ via the skew-symmetric matrix $[\boldsymbol{\omega}]_\times$:

$$[\boldsymbol{\omega}]_\times = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} = \omega_1 \mathbf{G}_1 + \omega_1 \mathbf{G}_2 + \omega_3 \mathbf{G}_3 = \sum_{i=1}^{3} \omega_i \mathbf{G}_i \ , \tag{3.9}$$

leading to the following differential equation:

$$\frac{\partial \mathbf{p}}{\partial t} = [\boldsymbol{\omega}]_\times \mathbf{p} \ , \tag{3.10}$$

with solution:

$$\Rightarrow \mathbf{p} = e^{[\boldsymbol{\omega}]_\times t} \mathbf{p}_{t=0} \ . \tag{3.11}$$

In $\theta$ units of time the point experiences a rotation of:

$$\mathbf{R}(\boldsymbol{\omega}, \theta) = e^{[\boldsymbol{\omega}]_\times \theta} \ . \tag{3.12}$$

If we let the axis of rotation have a norm of $\|\boldsymbol{\omega}\| = \theta$ we can express rotation about the origin as a vector $\boldsymbol{\omega}$ whose direction is the axis of rotation and whose length is the angle. Therefore the above simplifies to:

$$\mathbf{R}(\boldsymbol{\omega}) = e^{[\boldsymbol{\omega}]_\times} \ . \tag{3.13}$$

The previous *exponential map* can be expressed with a Taylor series as:

$$\exp([\boldsymbol{\omega}]_\times) = e^{[\boldsymbol{\omega}]_\times} = \mathbf{I} + [\boldsymbol{\omega}]_\times + \frac{1}{2!}[\boldsymbol{\omega}]_\times^2 + \frac{1}{3!}[\boldsymbol{\omega}]_\times^3 + ...\infty \ , \tag{3.14}$$

yielding the *Rodriguez formula* [56] given by:

$$e^{[\boldsymbol{\omega}]_\times} = \mathbf{I} + [\boldsymbol{\omega}]_\times \frac{\sin(\theta)}{\theta} + [\boldsymbol{\omega}]_\times^2 \frac{1 - \cos(\theta)}{\theta^2} \ . \tag{3.15}$$

The exponential map therefore takes a skew symmetric matrix of the reduced 3 parameters needed for rotation of $\boldsymbol{\omega} \in \mathfrak{so}(3)$ and turns it into a matrix $\in \mathbb{R}^{3\times3} \subset \mathbb{SO}(3)$:

$$\mathbf{R}(\boldsymbol{\omega}) = \exp\left(\sum_{i=1}^{3} \omega_i \mathbf{G}_i\right) : \mathfrak{so}(3) \to \mathbb{SO}(3) \ . \tag{3.16}$$

Going back from $\mathbb{SO}(3)$ to $\mathfrak{so}(3)$ is achieved with the logarithm:

$$\ln(\mathbf{R}) = \frac{\alpha}{2\sin(\alpha)}(\mathbf{R} - \mathbf{R}^\top) \ , \tag{3.17}$$

$$\alpha = \arccos\left(\frac{\mathrm{tr}(\mathbf{R}) - 1}{2}\right) \ , \tag{3.18}$$

$$\boldsymbol{\omega} = \lfloor \ln(\mathbf{R}) \rfloor \ , \tag{3.19}$$

where $\mathrm{tr}(\mathbf{R})$ gives the trace of $\mathbf{R}$ and the operator $\lfloor \bullet \rfloor$ returns the unique off-diagonal elements of a matrix.

### 3.3.2 Rotation and Translation Group: $\mathbb{SE}(3)$

The general rigid body transformation consisting of a translation vector $\mathbf{t} \in \mathbb{R}^3$ and rotation about the origin $\boldsymbol{\omega} \in \mathfrak{so}(3)$ can be minimally parametrised by:

$$\boldsymbol{\delta} = [\mathbf{u}, \boldsymbol{\omega}]^\top \in \mathfrak{se}(3) \ . \tag{3.20}$$

The exponential map to go from $\mathfrak{se}(3)$ to $\mathbb{SE}(3)$ is given by:

$$\mathbf{T}(\boldsymbol{\delta}) = \exp\left(\sum_{i=1}^{6} \delta_i \mathbf{G}_i\right) : \mathfrak{se}(3) \to \mathbb{SE}(3) \ , \tag{3.21}$$

with generator matrices to express differential translations and rotations given by:

$$\left.\frac{\partial \mathbf{T}}{\partial \delta_i}\right|_{\boldsymbol{\delta}=\mathbf{0}} = \mathbf{G}_i \ , \tag{3.22}$$

$$\mathbf{G}_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{G}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{G}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{3.23}$$

$$\mathbf{G}_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{G}_5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{G}_6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} . \tag{3.24}$$

The linear combination of generators can be expressed compactly in block-matrix form as:

$$\sum_{i=1}^{6} \delta_i \mathbf{G}_i = \left[\begin{array}{c|c} [\boldsymbol{\omega}]_\times & \mathbf{u} \\ \hline \mathbf{0}^\top & 0 \end{array}\right] . \tag{3.25}$$

The exponential map can then be expanded with a Taylor series as:

$$\exp\left(\begin{array}{c|c} [\boldsymbol{\omega}]_\times & \mathbf{u} \\ \hline \mathbf{0}^\top & 0 \end{array}\right) = \mathbf{I} + \left(\begin{array}{c|c} [\boldsymbol{\omega}]_\times & \mathbf{u} \\ \hline \mathbf{0}^\top & 0 \end{array}\right) + \frac{1}{2!}\left(\begin{array}{c|c} [\boldsymbol{\omega}]_\times^2 & [\boldsymbol{\omega}]_\times \mathbf{u} \\ \hline \mathbf{0}^\top & 0 \end{array}\right) + ...\infty , \tag{3.26}$$

$$\Rightarrow \exp\left(\begin{array}{c|c} [\boldsymbol{\omega}]_\times & \mathbf{u} \\ \hline \mathbf{0}^\top & 0 \end{array}\right) = \left(\begin{array}{c|c} \exp([\boldsymbol{\omega}]_\times) & \mathbf{V}\mathbf{u} \\ \hline \mathbf{0}^\top & 0 \end{array}\right) , \tag{3.27}$$

with:

$$\mathbf{V} = \mathbf{I} + \left(\frac{1-\cos(\theta)}{\theta^2}\right)[\boldsymbol{\omega}]_\times + \left(\frac{\theta - \sin(\theta)}{\theta^3}\right)[\boldsymbol{\omega}]_\times^2 , \tag{3.28}$$

$$\theta = \|\boldsymbol{\omega}\| . \tag{3.29}$$

Going back from $\mathbb{SE}(3)$ to $\mathfrak{se}(3)$ is achieved with the logarithm:

$$\boldsymbol{\omega} = \lfloor \ln(\mathbf{R}) \rfloor , \tag{3.30}$$

$$\mathbf{u} = \mathbf{V}^{-1}\mathbf{t} , \tag{3.31}$$

with

$$\mathbf{V}^{-1} = \mathbf{I} - \frac{1}{2}[\boldsymbol{\omega}]_\times + \frac{1}{\theta^2}\left(1 - \frac{\theta \sin(\theta)}{2(1-\cos(\theta))}\right)[\boldsymbol{\omega}]_\times^2 . \tag{3.32}$$

## 3.4 Pinhole Camera Model

We follow the pinhole camera model [67] as depicted in Figure 3.1. The *image plane* has origin at the top-left corner and a central point called the *principal point* $\mathbf{p} = [u_0, v_0]^\top$ defined as the intersection between the image plane and the *optical axis*. The pixels are assumed to be rectangular with size $p_w \times p_h$. The *focal length* $f$ in the pinhole camera model is the distance between the image plane and the *optic centre*; the shorter the focal length, the wider the angle of view and consequently the coverage of the scene captured. Together the principal point, focal length and pixel size define the *camera intrinsic parameters*.[1]

[1] We ignore distortions coefficients for the purpose of this research as we mainly used the Kinect depth and colour sensors, which are already of good quality.

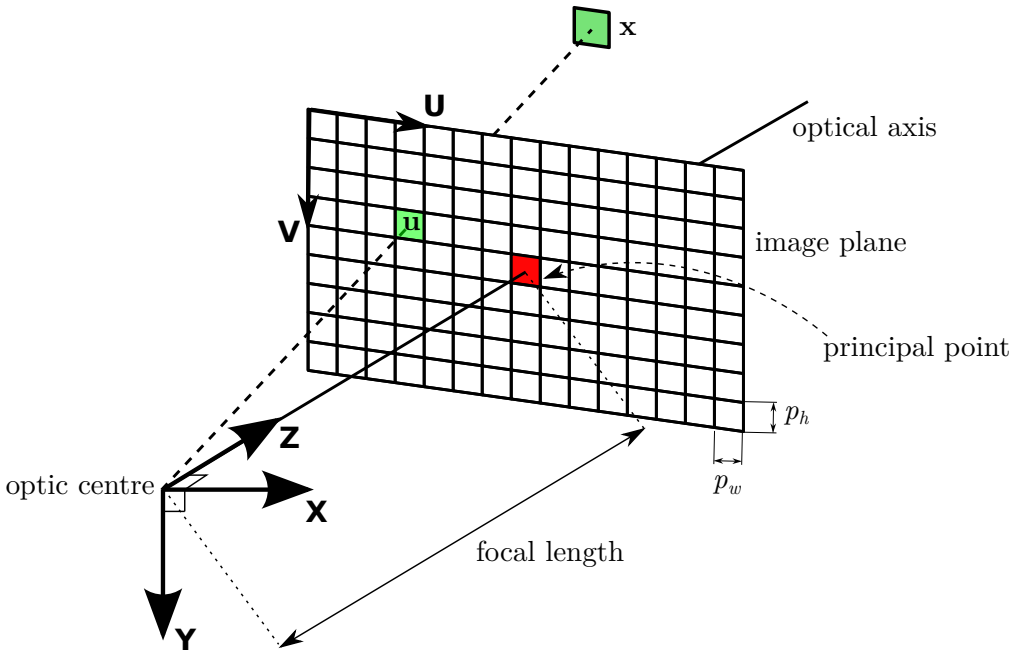



Figure 3.1: The pinhole camera model defines the relationship between points $\mathbf{x}$ in 3D space and their projections $\mathbf{u}$ into the image plane.

A 3D point $\mathbf{x} = [x, y, z]^\top$ in the scene is projected as a 2D point $\mathbf{u} = [u, v]^\top$ on the image plane as function of its depth and the camera intrinsic parameters:

$$u = f_u \left(\frac{x}{z}\right) + u_0 \ , \qquad v = f_v \left(\frac{y}{z}\right) + v_0 \ , \tag{3.33}$$

$$f_u = \frac{f}{p_w} \ , \qquad f_v = \frac{f}{p_h} \ . \tag{3.34}$$

It is convenient to write the camera intrinsic parameters in matrix form as:

$$\mathbf{K} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \ . \tag{3.35}$$

This allows us to transform a point $\mathbf{x}$ into homogeneous image space coordinates $\dot{\mathbf{u}}$ via a matrix-vector multiplication:

$$\dot{\mathbf{u}} = \mathbf{K}\mathbf{x} \ , \tag{3.36}$$

$$\dot{\mathbf{u}} = \begin{bmatrix} f_u x + u_0 z \\ f_v y + v_0 z \\ z \end{bmatrix} \ . \tag{3.37}$$

The inverse calibration matrix $\mathbf{K}^{-1}$ is given by:

$$\mathbf{K}^{-1} = \begin{bmatrix} \frac{1}{f_u} & 0 & -\frac{u_0}{f_u} \\ 0 & \frac{1}{f_v} & -\frac{v_0}{f_v} \\ 0 & 0 & 1 \end{bmatrix} . \tag{3.38}$$

This will be useful when determining direction vectors $\mathbf{l}$ from homogeneous pixel coordinates $\dot{\mathbf{u}} = [u, v, 1]^\top$:

$$\mathbf{l} = \mathbf{K}^{-1}\dot{\mathbf{u}} . \tag{3.39}$$

## 3.5 Non-linear least squares optimisation

A number of SLAM problems that will be described in the following chapters require an optimisation framework to find optimal model parameters from measured data corrupted by noise. One such model to optimise is for instance the transformation that best aligns a consistent 3D map to a noisy depth image acquired with an RGB-D camera; this process is called *iterative closest point* (ICP) and will be used for things such as live camera pose estimation and object alignment; it will be described in detail in Section 3.5.1. Another model to optimise consists of finding the set of camera poses and 3D point positions that best explain 2D measurements in a set of images; this process is called *bundle adjustment* (BA) and will be described in detail in Chapter 4.

Models like these have cost functions which are non-linear in the parameters and solving them require linear approximations to obtain partial solutions iteratively.

More formally, we aim to find the optimal parameters $\hat{\mathbf{a}} \in \mathbb{R}^p$ of the model $f(\mathbf{a})$ that best explain a set of $N$ observations:

$$\hat{\mathbf{a}} = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{i=1}^{N} E_i(\mathbf{a})^2 . \tag{3.40}$$

We define the *error function* $E(\mathbf{a}) \in \mathbb{R}$ as:

$$E(\mathbf{a}) = \sum_{i=1}^{N} E_i(\mathbf{a})^2 . \tag{3.41}$$

There are a variety of methods available to solve 3.40 [117][125]. The most straightforward is called *gradient descent* that proceed by stepping towards the local

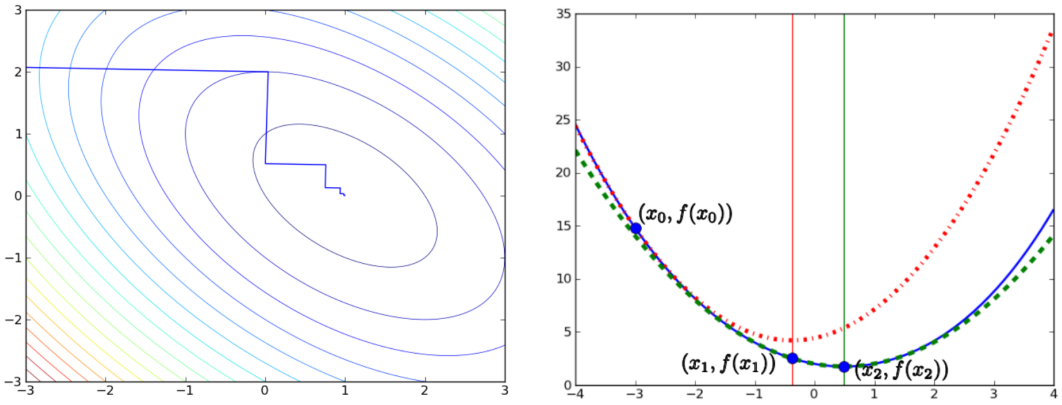



Figure 3.2: **(left)** The gradient descent method for the quadratic form $\frac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x} - \mathbf{b}^\top\mathbf{x} + c$ shows slow convergence rate due to the "zig-zag" path towards the local minimum. **(right)** The Newton method to find the minimum of a higher-order polynomial (blue curve) approximates the function with a one-dimensional quadratic form (red and green curves) on each iteration step $x_i$. Image courtesy of Hauke Strasdat.

minimum of $E$ along the direction of the negative gradient $-\nabla_E$ until the function stops decreasing. Gradient descent exhibits a "zig-zag" path leading to slow convergence rate particularly close to the minimum (see Figure 3.2 left). The *Newton* method can be more efficient provided the function is twice differentiable (see Figure 3.2 right) but requires calculating the Hessian of $E(\mathbf{a})$ which can be intractable for high-dimensional problems. The *Gauss-Newton* method, described in detail in the rest of this Section, uses an approximation to the Hessian and thus can be efficiently used for the type of problems we are interested in SLAM. The Newton-type methods work well close to the local minimum, but since they cannot distinguish between minima, maxima or saddle points they can fail to converge. The method known as Levenberg-Marquardt (LM) [92][101] alternates between Gauss-Newton and gradient descent to achieve fast and guaranteed convergence. The LM method will be explored in detail in Chapter 4 for solving the *bundle adjustment* problem.

The Gauss-Newton method starts by approximating the error function via a first order Taylor expansion around $\mathbf{a}$:

$$E(\mathbf{a}+\mathbf{x}) \approx E(\mathbf{a}) + \nabla_E^\top\mathbf{x} + \frac{1}{2}\mathbf{x}^\top\mathbf{H}_E\mathbf{x}\ , \tag{3.42}$$

where $\nabla_E$ is the gradient of $E(\mathbf{a})$:

$$\nabla_E = \frac{\partial E(\mathbf{a})}{\partial \mathbf{a}} = \left[\frac{\partial E(\mathbf{a})}{\partial a_1}, ..., \frac{\partial E(\mathbf{a})}{\partial a_p}\right]^\top , \tag{3.43}$$

and $\mathbf{H}_E$ is the Hessian of $E(\mathbf{a})$:

$$\mathbf{H}_E = \frac{\partial^2 E(\mathbf{a})}{\partial a_i \partial a_j} = \begin{bmatrix} \frac{\partial^2 E(\mathbf{a})}{\partial a_1^2} & \cdots & \frac{\partial^2 E(\mathbf{a})}{\partial a_1 \partial a_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\mathbf{a})}{\partial a_1 \partial a_p} & \cdots & \frac{\partial^2 E(\mathbf{a})}{\partial a_p^2} \end{bmatrix} . \tag{3.44}$$

Expressing the *residuals* as an error vector: $\mathbf{e}(\mathbf{a}) = \{E_i(\mathbf{a})\}_{i=1}^N$, the error function becomes:

$$E(\mathbf{a}) = \mathbf{e}^\top \mathbf{e} , \tag{3.45}$$

and therefore:

$$\nabla_E = 2\mathbf{J}^\top \mathbf{e} , \tag{3.46}$$

where $\mathbf{J}$ is the $N \times p$ Jacobian matrix of the error vector:

$$\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{a}} = \begin{bmatrix} \frac{\partial E_1(\mathbf{a})}{\partial a_1} & \cdots & \frac{\partial E_1(\mathbf{a})}{\partial a_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial E_N(\mathbf{a})}{\partial a_1} & \cdots & \frac{\partial E_N(\mathbf{a})}{\partial a_p} \end{bmatrix} . \tag{3.47}$$

Likewise the Hessian becomes:

$$\mathbf{H}_E = 2\mathbf{J}^\top \mathbf{J} + 2\frac{\partial \mathbf{J}^\top}{\partial \mathbf{a}} \mathbf{e} . \tag{3.48}$$

The second term in the Hessian is multiplied by residuals that are generally small and distributed with 0 mean and can be ignored, leading to the Gauss-Newton approximation to the Hessian given by:

$$\mathbf{H}_E \approx 2\mathbf{J}^\top \mathbf{J} . \tag{3.49}$$

At each iteration step we will determine a small update $\mathbf{x}$ to the model parameters $\mathbf{a}$ that minimises the error function. This should happen when:

$$\nabla_{E(\mathbf{a}+\mathbf{x})} = \mathbf{0} . \tag{3.50}$$

Differentiating 3.42 and ignoring higher order terms we obtain:

$$\nabla_{E(\mathbf{a}+\mathbf{x})} = \nabla_E + \mathbf{H}_E\mathbf{x} = \mathbf{0} \ . \tag{3.51}$$

Replacing 3.46 and 3.49 in 3.51 we obtain:

$$\nabla_{E(\mathbf{a}+\mathbf{x})} = 2\mathbf{J}^\top\mathbf{e} + 2\mathbf{J}^\top\mathbf{J}\mathbf{x} = \mathbf{0} \ . \tag{3.52}$$

Reordering the above leads to the so called *normal equation*:

$$\mathbf{J}^\top\mathbf{J}\mathbf{x} = -\mathbf{J}^\top\mathbf{e} \ . \tag{3.53}$$

Finally the parameter update at each iteration $k$ is given by:

$$\mathbf{x} = -(\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top\mathbf{e} \ , \tag{3.54}$$

$$\mathbf{a}_{k+1} = \mathbf{a}_k \oplus \mathbf{x} \ . \tag{3.55}$$

The sequence of iterations is stopped if the error norm stops decreasing from one iteration to the next or a maximum number of iterations is reached.

### 3.5.1 Iterative Closest Point (ICP)

ICP [14] is a non-linear least squares optimisation process for registering two sets of points representing overlapping 3D surfaces (referred to as the *model* and *data* surfaces), thus placing both sets of data into a common reference frame.

It is assumed that exact correspondences are not known beforehand and therefore the procedure will make an initial 'guess' on each iteration trying to find likely correspondences that will be refined as the method converges to the solution. The initial guess can be based on finding the *closest* model point to a particular data point. However when the data to be aligned contains noise and outliers the closest-point approach can generate a large number of mismatches, slowing the convergence rate. A more robust metric of correspondence that we will be using is the *point-to-plane* distance as described by Rusinkiewicz and Levoy [133], which uses projections that are less sensitive to noise.

The model and data surfaces to align each consist of a vertex and normal map ($\mathcal{V}_\mathrm{m}, \mathcal{N}_\mathrm{m}$ and $\mathcal{V}_\mathrm{d}, \mathcal{N}_\mathrm{d}$ respectively). Figure 3.3 shows an example scene where the

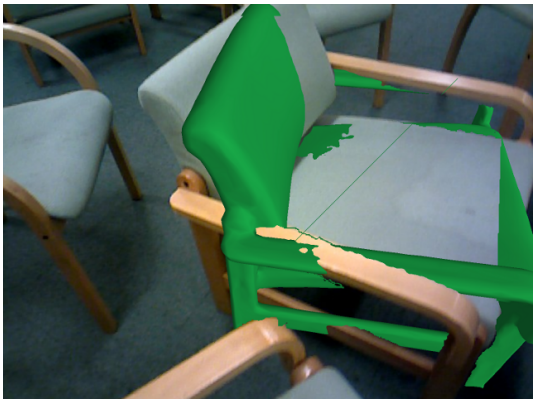


Figure 3.3: A scene with a nearby object before ICP alignment. The colour scene represents the *data* surface, while the green object is the *model*.

model of a chair is close to its final position, perhaps after detection, and requires further ICP iterations to complete alignment.

A vertex map $\mathcal{V}$ is computed from a depth map $\mathcal{D}$ by back-projecting depth values along direction vectors $\mathbf{r}$ connecting pixels to the optic centre obtained with the inverse calibration matrix:[2]

$$\mathbf{r}(\mathbf{u}) = \mathbf{K}^{-1}\dot{\mathbf{u}} \ , \tag{3.56}$$

$$\mathcal{V}(\mathbf{u}) = \mathbf{r}(u)\mathcal{D}(\mathbf{u}) \ . \tag{3.57}$$

When the 3D surface is densely sampled on a regular grid (as in the Kinect camera), an approximation of the normal map $\mathcal{N}$ can be computed with the cross product of neighbouring vertices as follows:

$$\mathcal{N}(\mathbf{u}) = s\left[(\mathcal{V}(u+1,v) - \mathcal{V}(u,v)) \times (\mathcal{V}(u,v+1) - \mathcal{V}(u,v))\right] \ , \tag{3.58}$$

where $s[\mathbf{x}] = \mathbf{x}/\|\mathbf{x}\|_2$.

The model alignment transformation $\mathbf{T}_{\text{md}} \in \mathbb{SE}(3)$ will take points from the data reference frame and transform them to the model reference frame.

We will refine the model parameters by estimating a sequence of $n$ incremental updates $\{\tilde{\mathbf{T}}^n_{\text{md}}\}^n_{k=1}$ parametrised with a vector $\mathbf{x} \in \mathfrak{se}(3)$ with $\tilde{\mathbf{T}}^{k=0}_{\text{md}}$ set as the identity. Taking the solution vector $\mathbf{x}$ to an element in $\mathbb{SE}(3)$ via the exponential map,

[2] A depth map stores for each pixel a distance to a scene point along the optical axis

we compose the computed incremental transform at iteration $k+1$ onto the previous estimated transform $\tilde{\mathbf{T}}^i_{\mathrm{md}}$:

$$\tilde{\mathbf{T}}^{k+1}_{\mathrm{md}} \leftarrow \exp(\mathbf{x})\tilde{\mathbf{T}}^k_{\mathrm{md}} \ . \tag{3.59}$$

The error function we will use for ICP with a point-to-plane distance is expressed as:

$$E(\mathbf{a}) = \sum_{\mathbf{u}\in\Omega} E^2_{\mathbf{u}}(\mathbf{a}) \ , \tag{3.60}$$

$$E_{\mathbf{u}}(\mathbf{a}) = \mathcal{N}_{\mathrm{m}}(\mathbf{u}')^\top \left(\exp(\mathbf{a})\hat{\mathcal{V}}_{\mathrm{d}}(\mathbf{u}) - \mathcal{V}_{\mathrm{m}}(\mathbf{u}')\right) \ , \tag{3.61}$$

where $\Omega$ is the surface domain. As we are mostly dealing with surfaces extracted from depth images the domain corresponds to pixels in a uniform 2D grid.

Here $\mathcal{V}_{\mathrm{m}}(\mathbf{u}')$ and $\mathcal{N}_{\mathrm{m}}(\mathbf{u}')$ are the projectively data associated model vertex and normal estimated by projecting the data vertex $\mathcal{V}_{\mathrm{d}}(\mathbf{u})$ at pixel $\mathbf{u}$ from the live depth map into the reference frame with camera intrinsic matrix $\mathbf{K}$ and standard pin-hole projection function $\pi$:

$$\mathbf{u}' = \pi(\mathbf{K}\hat{\mathcal{V}}_{\mathrm{d}}(\mathbf{u})) \ . \tag{3.62}$$

A data vertex is transformed into the model frame using the current incremental transform $\tilde{\mathbf{T}}^k_{\mathrm{md}}$:

$$\hat{\mathcal{V}}_{\mathrm{d}}(\mathbf{u}) = \tilde{\mathbf{T}}^k_{\mathrm{md}}\mathcal{V}_d(\mathbf{u}) \ . \tag{3.63}$$

We will first calculate the Jacobian for a single residual $E_u$:

$$\mathbf{J}_{\mathbf{u}} = \frac{\partial E_{\mathbf{u}}(\mathbf{a})}{\partial \mathbf{a}} = \frac{\partial}{\partial \mathbf{a}}\left[\mathcal{N}_{\mathrm{m}}(\mathbf{u}')^\top \left(\exp(\mathbf{a})\hat{\mathcal{V}}_{\mathrm{d}}(\mathbf{u}) - \mathcal{V}_{\mathrm{m}}(\mathbf{u}')\right)\right] \ , \tag{3.64}$$

$$= \frac{\partial \mathcal{N}_{\mathrm{m}}(\mathbf{u}')^\top \left(\exp(\mathbf{a})\hat{\mathcal{V}}_{\mathrm{d}}(\mathbf{u}) - \mathcal{V}_{\mathrm{m}}(\mathbf{u}')\right)}{\partial \exp(\mathbf{a})\hat{\mathcal{V}}_{\mathrm{d}}(\mathbf{u})} \frac{\partial \exp(\mathbf{a})}{\partial \mathbf{a}} \hat{\mathcal{V}}_{\mathrm{d}}(\mathbf{u}) \ . \tag{3.65}$$

After simplification, the previous Jacobian becomes:

$$\mathbf{J}_{\mathbf{u}} = \mathcal{N}_{\mathrm{m}}(\mathbf{u}')^\top \frac{\partial \exp(\mathbf{a})}{\partial \mathbf{a}} \hat{\mathcal{V}}_{\mathrm{d}}(\mathbf{u}) \ , \tag{3.66}$$

$$= \mathcal{N}_{\mathrm{m}}(\mathbf{u}')^\top [\mathbf{G}_1, ..., \mathbf{G}_6] \hat{\mathcal{V}}_{\mathrm{d}}(\mathbf{u}) \ , \tag{3.67}$$

$$\Rightarrow \mathbf{J}_{\mathbf{u}} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}^\top \begin{bmatrix} 1 & 0 & 0 & 0 & v_z & -v_y \\ 0 & 1 & 0 & -v_z & 0 & v_x \\ 0 & 0 & 1 & v_y & -v_x & 0 \end{bmatrix} \ . \tag{3.68}$$

with

$$\mathcal{N}_{\mathrm{m}}(\mathbf{u}') = [n_x, n_y, n_z]^\top \ ,$$
$$\hat{\mathcal{V}}_{\mathrm{d}}(\mathbf{u}) = [v_x, v_y, v_z]^\top \ .$$

The normal equation for this ICP formulation can be written as:

$$\mathbf{J}^\top \mathbf{J}\mathbf{x} = -\mathbf{J}^\top \mathbf{e} \ . \tag{3.69}$$

with:

$$\mathbf{J}^\top \mathbf{J} = \sum_{u \in \Omega} \mathbf{J}_\mathbf{u}^\top \mathbf{J}_\mathbf{u} \ , \tag{3.70}$$
$$\mathbf{J}^\top \mathbf{e} = \sum_{u \in \Omega} \mathbf{J}_\mathbf{u}^\top E_\mathbf{u} \ . \tag{3.71}$$

Solving for the parameter update at each iteration $k$ we finally get:

$$\mathbf{x} = -(\mathbf{J}^\top \mathbf{J})^{-1} \mathbf{J}^\top \mathbf{e} \ . \tag{3.72}$$

We use the previous solution to update the alignment transformation using Equation 3.59 and repeat the iterative process as described in Section 3.5 until the convergence criteria are met.

## 3.6 Principal Components Analysis

It can be useful and more efficient to represent a measured multidimensional dataset $\mathbf{X} = \{\mathbf{x}_i\}$, $i = 1...p, \mathbf{x}_i \in \mathbb{R}^n$ in a lower dimensional space $k < n$. For example, as shown in Figure 3.4, the plotted 2D data can be more succinctly expressed in 1D if we assume it was originally generated along the axis $\mathbf{u_1}$ going through the middle of the points and then corrupted by some noise along the second axis $\mathbf{u_2}$ perpendicular to it. The major axis $\mathbf{u_1}$ should be chosen such that variance is maximised when data is projected onto it. Essentially this method allow us to compress the dataset by considering only the most informative dimensions.

To do this we first need to normalise the data by subtracting its mean:

$$\bar{\mathbf{x}} = \frac{1}{p} \sum_{i=1}^{p} \mathbf{x}_i \ , \tag{3.73}$$
$$\hat{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}} \ . \tag{3.74}$$

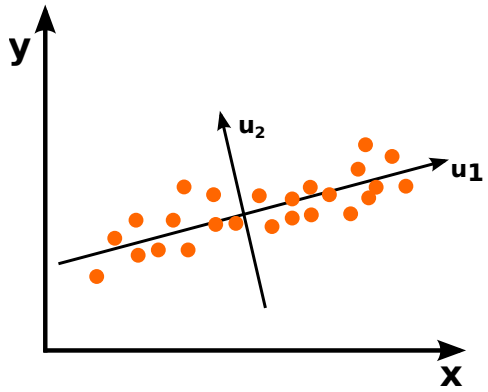



Figure 3.4: Plotted 2D data showing its two eigenvectors $\mathbf{u_1}$ and $\mathbf{u_2}$. The data could be compressed with minimal loss of information by projecting into $\mathbf{u_1}$ as it maximises its underlying variance.

The projection of a point $\hat{\mathbf{x}}_i$ on a unit vector $\mathbf{u}$ is given by $\hat{\mathbf{x}}_i^\top \mathbf{u}$. In order to maximise the variance of all such projected points from the dataset we will choose a vector $\mathbf{u}$ that maximises:

$$e = \frac{1}{p}\sum_{i=1}^{p}(\hat{\mathbf{x}}_i^\top \mathbf{u})^2 \,, \tag{3.75}$$

$$e = \frac{1}{p}\sum_{i=1}^{p}\mathbf{u}^\top \hat{\mathbf{x}}_i \hat{\mathbf{x}}_i^\top \mathbf{u} = \mathbf{u}^\top \left(\frac{1}{p}\sum_{i=1}^{p}\hat{\mathbf{x}}_i \hat{\mathbf{x}}_i^\top\right)\mathbf{u} \,, \tag{3.76}$$

$$e = \mathbf{u}^\top \boldsymbol{\Sigma}(\hat{\mathbf{x}})\mathbf{u} \,, \tag{3.77}$$

subject to $\|\mathbf{u}\| = 1$.

$\boldsymbol{\Sigma}(\hat{\mathbf{x}})$ is the *covariance matrix* and represents the spread of the data between every pair of dimensions:

$$\boldsymbol{\Sigma}(\hat{\mathbf{x}}) = \frac{1}{p}\sum_{i=1}^{p}\hat{\mathbf{x}}_i \hat{\mathbf{x}}_i^\top = \frac{1}{p}\begin{bmatrix} \sum \hat{x}^0\hat{x}^0 & \sum \hat{x}^0\hat{x}^1 & \cdots & \sum \hat{x}^0\hat{x}^n \\ \sum \hat{x}^1\hat{x}^0 & \sum \hat{x}^1\hat{x}^1 & \cdots & \sum \hat{x}^1\hat{x}^n \\ \vdots & \vdots & \ddots & \vdots \\ \sum \hat{x}^n\hat{x}^0 & \sum \hat{x}^n\hat{x}^1 & \cdots & \sum \hat{x}^n\hat{x}^n \end{bmatrix} \,, \tag{3.78}$$

where $\hat{x}^d$ is the $d$th dimension of $\hat{\mathbf{x}}$.

Furthermore, the expression $\boldsymbol{\Sigma}(\hat{\mathbf{x}})\mathbf{u}$ can be thought of as a matrix-vector transformation applied to $\mathbf{u}$. Therefore, in order to maximise $e$ we would like to choose a unit vector $\mathbf{u}$ whose direction is not affected by such a transformation:

$$\boldsymbol{\Sigma}(\hat{\mathbf{x}})\mathbf{u} = \lambda\mathbf{u} \,. \tag{3.79}$$

Such vector $\mathbf{u}$ is the principal *eigenvector* of the covariance matrix and $\lambda$ its corresponding *eigenvalue*. For a given square matrix of size $n$, there exist $n$ such eigenvectors and eigenvalues, and usually these are computed numerically. The eigenvectors are also orthogonal to each other.

To represent the dataset in a lower dimensional space $k < n$ we would then choose the first $k$ eigenvectors sorted by decreasing eigenvalue; the chosen eigenvectors can be seen a new basis $\mathbf{u}_1, ..., \mathbf{u}_k$ for the data. After projecting the data into the new basis it becomes:

$$\hat{\mathbf{y}}_i = \begin{bmatrix} \mathbf{u}_1^\top \hat{\mathbf{x}}_i \\ \mathbf{u}_2^\top \hat{\mathbf{x}}_i \\ \vdots \\ \mathbf{u}_k^\top \hat{\mathbf{x}}_i \end{bmatrix} , \tag{3.80}$$

with $\hat{\mathbf{y}} \in \mathbb{R}^k$.

The final approximated data (*i.e.* compressed) after discarding the less informative dimensions is reconstructed by:

$$\hat{\mathbf{x}}'_i = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_k \end{bmatrix} \hat{\mathbf{y}}_i , \tag{3.81}$$

$$\mathbf{x}'_i = \hat{\mathbf{x}}'_i + \bar{\mathbf{x}} , \tag{3.82}$$

with $\mathbf{x}'_i \in \mathbb{R}^n$.

## 3.7 Modern GPU architecture

Our research has mainly targeted the NVIDIA GPU architecture code-named 'Fermi' released in September 2009 [119]. Fermi is the successor of the 'Tesla' architecture introduced in November 2006 that brought a unified graphics and compute model [118] and it has now been superseded by the newest 'Kepler' architecture revealed in May 2012 featuring more power efficiency [120], making it suitable for battery-powered devices (with the NVIDIA Tegra K1 being an example GPU built on the Kepler architecture).

We will see in Section 3.8.2 how the CUDA programming model enables access to compute capabilities of GPUs via a high-level language similar to C++. In the meantime it is worth mentioning that parallel threads are grouped into blocks to enable cooperative execution via synchronisation primitives and local memory.

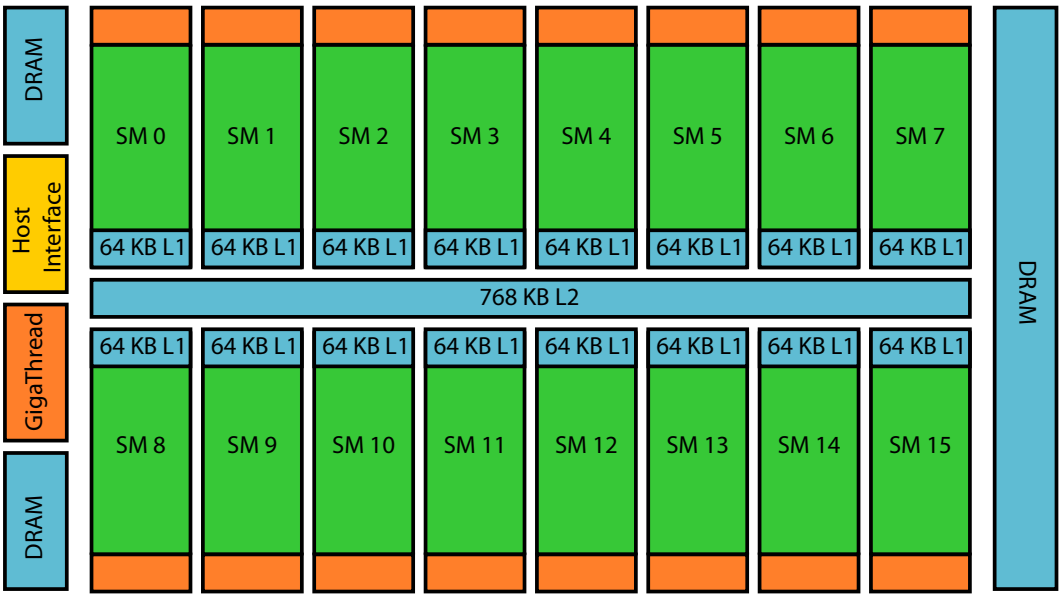



Figure 3.5: The Fermi GPU architecture. A set of 16 Streaming Multiprocessors (SM) execute parallel operations organised by the GigaThread global scheduler. Each SM contains a local configurable L1 memory while fast GPU-wide caching is achieved via a L2 cache. Adapted from [164].

Internally, finer groups of 32 threads called *warps* are executed at the same clock-step. The routines written by a programmer to describe the parallel computations are commonly referred to as *kernels*.

The Fermi architecture consists of a set of Streaming Multiprocessors (SM, typically 16) each containing 32 CUDA cores to perform parallel computations (see Figures 3.5 and 3.6).

Fermi GPUs schedule activities at both the GPU and SM level. A GigaThread global scheduler dynamically organises blocks of threads to be executed in parallel amongst the SMs. In addition, each SM also contains dual Warp Schedulers and Instruction Dispatch units to allow simultaneous execution of two warps on the CUDA cores and other execution units. To maximise GPU utilisation, the Fermi architecture allows concurrent kernel execution where more than one kernel executes simultaneously on the same GPU.

This architecture also offers a memory hierarchy consisting of L1/L2 caches and DRAM memory and is connected to the host CPU via a PCI-Express bus. Inside the SM there is a local memory of 64 KB whose capacity can be configured to act

mostly as L1 cache or shared memory (48 KB of L1 cache and 16KB of shared memory, or 48 KB of shared memory and 16KB of L1 cache). When more L1 cache is desired it improves performance during random memory accesses and also reduces the chance of spilling registers to DRAM to reduce latency when the on-chip register file is saturated. Conversely, when more shared memory is desired, this could enable blocks of threads in a SM to cooperate by sharing intermediate results or to programatically cache data from DRAM. Fast GPU-wide data caching is allowed via the 768 KB L2 cache.

Each CUDA core contains both an integer arithmetic logic unit (ALU) and floating point unit (FPU) (see Figure 3.6) with support for fused multiply-add (FMA) instructions for enhanced precision of intermediate results on common operations such as $D = A \times B + C$. The Load/Store Units calculate source or destination addresses and move data into the caches or DRAM. Transcendental instructions such as sine, cosine and square root are executed by Special Function Units (SFU).

This new architecture also offers up to 4.2x double precision performance enhancement compared to the previous generation as well as faster context-switching between graphics and compute tasks, a desirable property in Dense SLAM systems as some algorithms are more naturally handled via OpenGL rendering techniques (see for example Section 6.2.2).

## 3.8 Parallel Programming Models

An expressive programming model should be able to describe an algorithmic intent without resorting to low-level hardware constructs, and should also enable performance portability by delaying target processor optimisations to the runtime sub-system.

In the context of SLAM, it is beneficial to succinctly instruct a computing device to execute the modules of a SLAM pipeline in order to manage its inherit complexity and real-time requirements. As will be described in Section 5.2.3, we will use parallel primitive operations supported by template libraries like Thrust [12] to compactly accelerate an object recognition algorithm to achieve real-time speeds, and also to perform fast dense data-association for planar regions in Section 6.3.2.

While it is in theory possible to produce a dense SLAM system written entirely

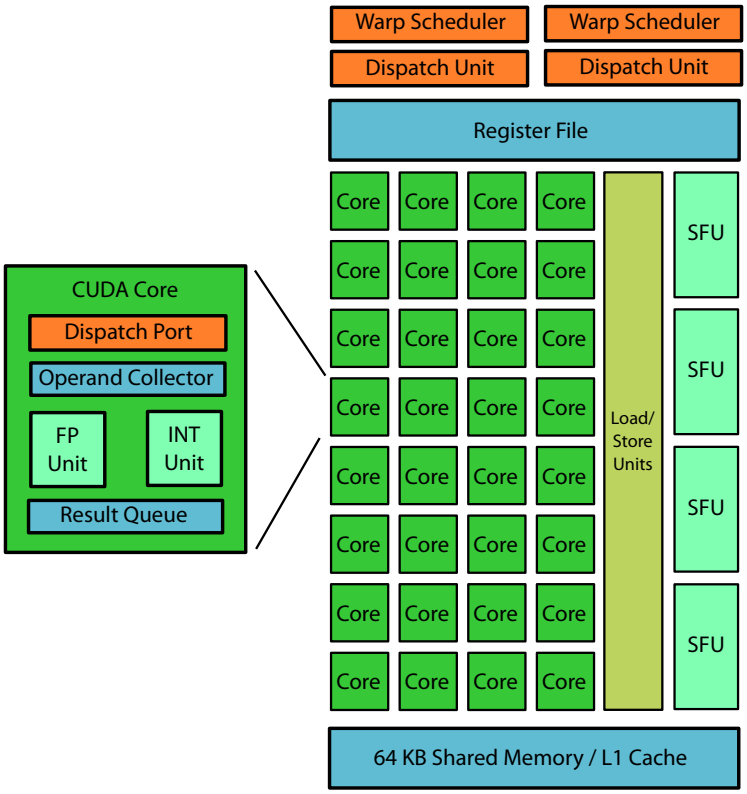



Figure 3.6: Streaming Multiprocessors (SM) architecture. Each SM holds 32 CUDA cores with a configurable shared memory/L1 cache. A dual warp scheduler and dispatch unit allows simultaneous execution of two warps. Special Function Units (SFU) accelerate computation transcendental operations. Adapted from [164].

for the CPU with a language like C++, it would likely be of low performance and of little practical use. Furthermore, the dense SLAM systems of interest to us produce hole-free surface predictions that can only be rendered quickly enough by using hardware accelerated rasterisation units present in GPUs, with many routines such as frustum culling, depth-order testing and triangle filling available 'for free' thanks to the progress in the field of computer graphics. To take advantage of years of real-time graphics research it is worth reviewing the modern GPU pipeline and its programming interface via OpenGL.

### 3.8.1 Modern GPU Pipeline and OpenGL

The current GPU architecture is the result of more than 20 years of development of hardware accelerated 3D graphics and programming models, evolving from being a fixed function triangle rasterisation unit [139], to allowing user-programmable

graphic stages [93], and finally becoming a general-purpose parallel processor [94].

Today we can describe it in abstract terms as a pipeline consisting of both fixed and programmable stages in a chain (see Figure 3.7), each one performing operations in parallel to every input data element.

More concretely, current GPUs hold two logical pipelines: The *graphics pipeline* (see Figure 3.7) and the *compute pipeline* (see Figure 3.8). As their names imply, the graphics pipeline deals with the processing of elements resulting in graphical output while the compute pipeline process data elements in parallel for general purpose tasks. This new logical partition was only recently apparent with the advent of programming languages such as CUDA, OpenCL and Compute Shaders as early attempts to exploit the massive parallelism of GPUs were made within the graphics pipeline itself by tricking the system to perform non-graphical tasks via pixel shaders and storing the input/output data elements in non-visible buffers.

In the graphics pipeline the raw graphics data is first assembled into primitive elements such as points, lines and triangles and passed to the first programmable stage called the *vertex shader* which transforms individual vertices, for example by perturbing them to simulate water effects. The resulting data is passed into the *tessellation shader* to programmatically increase the geometric detail (*e.g.* based on viewing distance).

Following this is the *geometry shader* allowing the creation of new geometric primitives (*e.g.* from point to triangle). For example, a single triangle primitive can be partitioned into 3 independent lines, an oriented point can be transformed into normal whiskers for visualisation or even amplified to a triangle strip to resemble an hexagon for splat rendering with surfels (see Figure 6.3 in Chapter 6).

With the geometric data already processed it is time for the *rasterisation* process to convert them into candidate pixels (commonly known as fragments) which are input to the *fragment shading* stage where final color processing takes place at the pixel scale.

The *compute pipeline* can accept more general buffers than the graphics pipeline and enables more control over the operations performed and work dispatch.

Fortunately during our research the programming models to harness the parallelism of GPU's for graphics and compute tasks were mature enough for desktop class

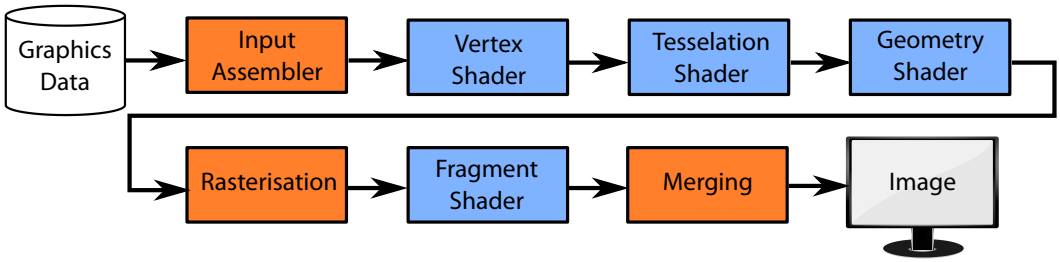



Figure 3.7: The graphics pipeline consists of both fixed (orange) and programmable stages (blue) operating on data elements in parallel, ending in the final rendered image.

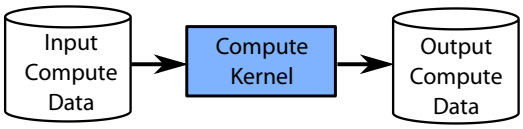



Figure 3.8: The compute pipeline consists of a single programmable stage operating in parallel on every data element.

applications, and therefore we were able to easily write code in OpenGL and CUDA whenever needed. Other languages such as OpenCL or C++AMP are equally capable to expose the GPU processing power but their slower adoption and platform availability limit the choices of supporting libraries.

### 3.8.2 CUDA

To harness the parallel compute power of GPU's, earlier approaches masked general computations inside graphics shading languages such as CGSL (a practice known as *GPGPU* and popularised during 2001-2005). NVIDIA introduced the CUDA platform in 2006 comprising a new hardware architecture and programming model to enable higher-level access to compute capabilities with a syntax more commonly used by non-graphics programmers (this is now known as *GPU-Compute*). Even though CUDA can be seen as a complete hardware and software platform with many language bindings, supporting libraries and tools, for the purpose of this section we will focus on the description of the programming model enabled via the C/C++ language extension.

The CUDA programming model expresses parallel operations via self-contained *kernels* written by a programmer to hold instructions that are executed in parallel by

Listing 3.1: Example CUDA kernel to add two vectors A and B into C

```
1  // Each thread instantiating this kernel will operate
2  // on a different element according to its ID.
3  __global__ void cuVectorAddition(
4    //output
5    float* cArr,
6    //input
7    const float* aArr,
8    const float* bArr,
9    const int elementCount)
10 {
11   // Compute the thread ID
12   int id = threadIdx.x + blockIdx.x*blockDim.x;
13
14   // Bounds check
15   if (id < elementCount)
16     cArr[id] = aArr[id] + bArr[id];
17 }
```

*threads*. Each thread instantiates the same set of kernel instructions and additionally contains a unique thread ID, private local memory, program counter, registers and varying data inputs or outputs. Typically a kernel would read input data from the DRAM *global memory* and output data would be written back to it once execution completes.

An example kernel to compute the addition of two vectors is presented in Listing 3.1. The listing contains the kernel body with a set of instructions to be executed independently for each thread in parallel. It can also be seen as the body inside a traditional *for* loop used in serial CPU code. To identify the data element the kernel is operating on, CUDA provides built-in variables such as threadIdx, blockIdx and blockDim.

Threads are organised in *blocks* and these in turn are organised in *grids* (see Figure 3.9). Thread blocks enable cooperative parallel routines via fast data sharing and barrier synchronisation primitives. The grid is sent to the GPU and the blocks are distributed amongst the SM and corresponding CUDA cores. The grid provides GPU-wide synchronisation by sharing data via global memory between kernel calls.

The programmer or the compiler has control over the parameters to define the thread organisation that is most suitable to the algorithm, but there are limitations on the target hardware that must be taken into account, particularly when shared

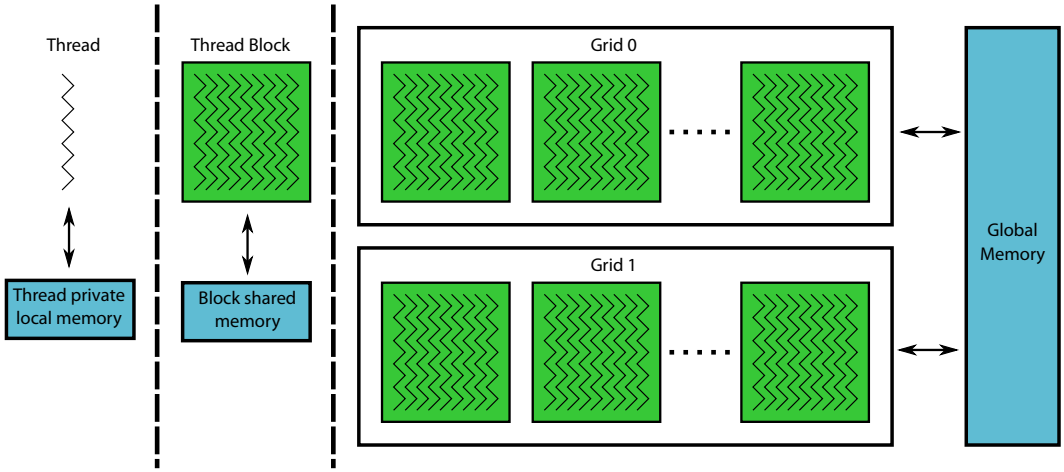



Figure 3.9: CUDA threads organisation. **left:** A single thread holds private local memory. **Middle:** Threads are organised in blocks to cooperate via shared memory. **right:** Thread blocks are further organised in grids and cooperate via global memory. Adapted from [119].

memory is required as this is a limited resource on each SM. An example kernel organisation is given in Listing 3.2. Here the configuration parameters are written by the programmer prior to the kernel call inside '<<<, , >>>'.

### 3.8.3 Thrust

While the CUDA programming model reduced the overhead to implement parallel GPU-Compute algorithms, finding the right thread organisation still posses a challenge to maximise the resource utilisation of GPUs. Furthermore, many common routines such as sort, search and reductions consist of a generic sequence of steps that only need to be specialised to suit a particular algorithm.

To simplify this, some libraries such as CUDA Data Parallel Primitives (CUDPP) were initially developed by Harris *et al.* [65], followed by the introduction of template-based libraries like Thrust by NVIDIA [13] or Bolt by AMD [1], which we used extensively to achieve peak performance of procedures with reduced coding.

To motivate the need of a higher level of abstraction we present an example of a parallel reduction operation consisting of summing up all the elements of a vector as this is a task used at various places in the development of our Semantic SLAM systems. For a vector of size $n$, a multi-threaded parallel reduction has an improved work efficiency of $O(\log n)$ compared to the equivalent single thread version of $O(n)$.

Listing 3.2: Example CUDA kernel organisation and execution

```
int main()
{
  int elementCount = 32768;
  float* aArr = NULL;
  float* bArr = NULL;
  float* cArr = NULL;

  //[...] Setup input data and copy from host (CPU)
  //to device (GPU) memory.

  //Organise threads into blocks and grids
  int blockSize = 1024; //threads in block
  int gridSize = elementCount/blockSize; //blocks in grid

  //Execute the kernel
  cuVectorAddition<<<gridSize, blockSize>>>(
    //output
    cArr,
    //input
    aArr,
    bArr,
    elementCount);

  //[...] Copy output data from device (GPU)
  //to host (CPU) memory.

  return 0;
}
```

As described in Section 3.8.2, threads are organised into blocks that can hold shared memory allowing thread cooperation via synchronisation primitives and this is beneficial for a parallel reduction algorithm. Listing A.2 shows the kernel code required to perform this operation using CUDA alone. The idea is to operate on smaller parts of the data to be reduced at the block-level (divide-and-conquer approach); for this each thread within a block is responsible for loading its data item and placing it in a shared memory space, followed by the reduction operation on a paired element to obtain a final result for the block. To avoid data hazards, the previous operations must be followed by synchronisation to ensure that write instructions are completed by all the threads within a block. Since thread cooperation occurs at the level of blocks, in order to complete a GPU-wide synchronisation an additional reduction operation has to be performed to finalise the block-level partial results.

Listing 3.3: Parallel Sum reduction with Thrust

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/reduce.h>
#include <thrust/functional.h>

int main(void)
{
  int elementCount = 32768;

  //Create input array on host
  thrust::host_vector<float> h_inputArr(elementCount);
  //[...] Fill input array with data

  //Copy input data from host to device
  thrust::device_vector<float> d_inputArr = h_inputArr;

  //Binary sum operation
  thrust::plus<float> sumOp;

  // compute sum on the device
  int sumResult = thrust::reduce(
    d_inputArr.begin(), d_inputArr.end(), 0.0f, sumOp);

  return 0;
}
```

Prior to launching a reduction kernel, device arrays must be created and data copied to them from the equivalent host-side arrays. As described previously, two kernel launches are required to compute the partial block-level summation and a second one for the global sum. This is shown in Listing A.1. Notice that the amount of shared data to be allocated per block is defined as the third argument inside the '<<<,,>>>' kernel launch parameters.

In contrast to the very verbose code of Listing A.2 and A.1 we can more compactly describe the same reduction operation using Thrust. Not only does this reduce the amount of code required, but also the configuration parameters are determined automatically by the compiler with no additional runtime cost, as shown in Listing 3.3. Allocation, deallocation and memory management are also greatly simplified with the use of STL-like semantics and iterators.

## 3.9 Summary

Having reviewed some fundamental mathematical concepts, the target GPU architecture we based our work on and its programming model, we are now ready to describe the work performed during this research and highlight our novel contributions.

In the next chapter we explore the challenges and advantages of a hybrid CPU/GPU implementation of Bundle Adjustment, allowing faster map optimisation for large number of points.

Chapter 4

# Hybrid GPU/CPU Bundle Adjustment

Next we describe a hybrid implementation of Bundle Adjustment (BA), an algorithm used for back-end map optimisation in some SLAM systems like PTAM [79] as well as offline structure from motion systems like Building Rome in a Day [2]. Until recently it was only seen as time-consuming serial optimisation procedure running as a background process on the CPU, but which under closer examination encompasses several sub-steps suitable for parallelisation on the GPU.

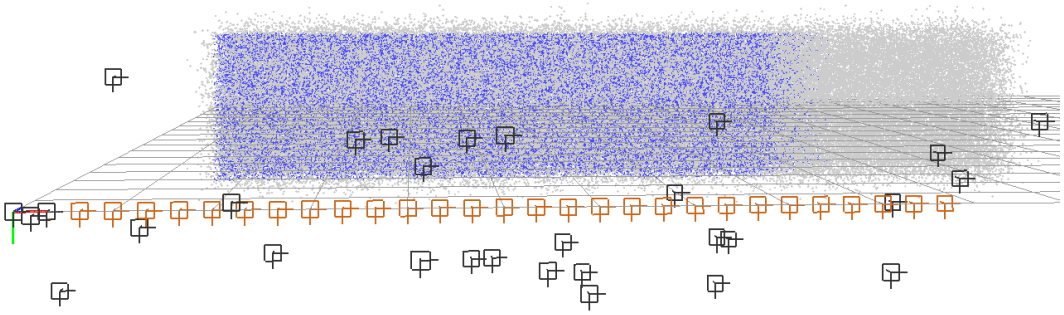


Figure 4.1: A synthetic scene consisting of 100K noisy points (gray) from a vertical plane reconstructed densely and 30 noisy initial camera estimates (black) following a linear trajectory from left to right trajectory. After running bundle adjustment we obtain the optimized points (blue) and cameras (orange) with reduced reprojection errors. This scene was optimised in 4.98 seconds with our hybrid GPU/CPU implementation. A CPU-only version takes about 40 seconds to converge.

BA is required to control the influence of errors and obtain globally consistent maps from measured data, as depicted in Figure 4.1. These errors can come from a variety of sources such as the measurement procedure (*e.g.* sampling artifacts), unmodeled influences (*e.g.* translucent materials), sensor shortcomings (*e.g.* motion blur), numerical round-off, to mention a few.

The motivation behind this work came from the sheer number of data points requiring optimisation on dense maps created by systems like KinectFusion [112] and DTAM [113]. An optimal solution should jointly consider estimating camera and map structure parameters satisfying all the measurement constraints simultaneously. In practice however, systems like KinectFusion interleave those steps: first optimising the map with fixed camera parameters followed by camera optimisation assuming a fixed map. In doing so, the formulation is simplified and the system is able to achieve real-time speeds. In this work we instead investigate practical limits for performing joint optimisation without sacrificing too much speed by taking advantage of the sparsity on the BA formulation and opportunities for parallelisation.

In Sections 4.1 and 4.2 we will introduce the standard BA algorithm on serial processors. From 4.3 we will analyse the structure of the solution and look at accelerating this with parallel computations.

## 4.1 Objective Function

Assuming we are given an initial point-based 3D reconstruction of a scene along with poses describing the trajectory of a moving camera, the goal of Bundle Adjustment is to jointly optimise a set of $n$ points $\{\mathbf{b}_i\}_{i=0}^{n-1}, \mathbf{b}_i \in \mathbb{R}^3$ and $m$ camera parameters $\{\mathbf{a}_j\}_{j=0}^{m-1}, \mathbf{a}_j \in \mathbb{SE}(3)$ in order to minimize the re-projection error across a set of keyframes (either a running window over the last $k$ frames or across the whole video sequence). A simplified example with 4 points and 2 cameras is shown in Figure 4.2. In the following description, it is assumed that exact correspondences are given.

Due to the non-linear nature of the projective transformation, the parameters are optimised following a non-linear least squares formulation [96], expressed as:

$$[\hat{\mathbf{a}}_j, \hat{\mathbf{b}}_i] = \underset{\mathbf{a}_j, \mathbf{b}_i}{\operatorname{argmin}} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (\mathbf{x}_{i,j} - \mathbf{Q}(\mathbf{a}_j, \mathbf{b}_i))^2 \ , \tag{4.1}$$

$$\mathbf{Q}(\mathbf{a}_j, \mathbf{b}_i) = \pi(\mathbf{K}\mathbf{T}_{\mathrm{a}_j w}\mathbf{b}_i) \ , \tag{4.2}$$

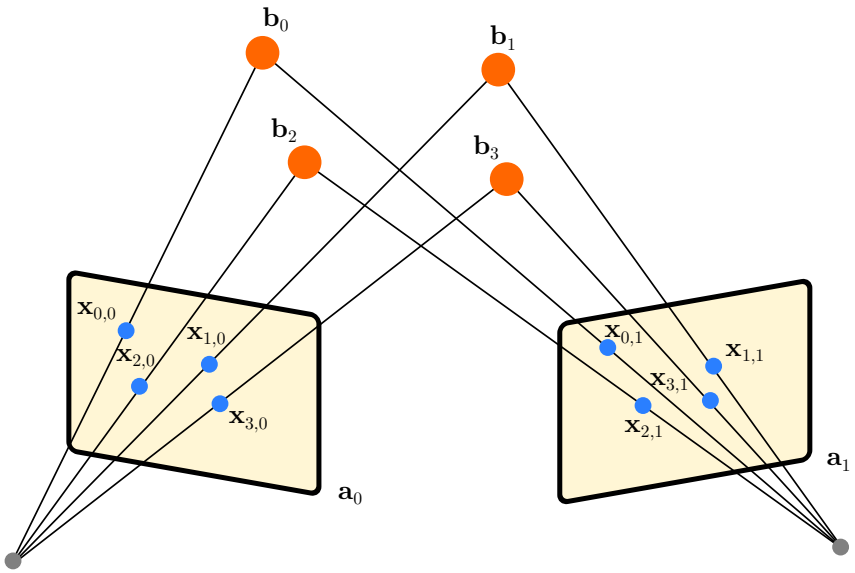



Figure 4.2: Bundle adjustment example consisting of 3D points $\mathbf{b}_i$ (orange) and cameras $\mathbf{a}_j$ (yellow). A 2D measurement point $\mathbf{x}_{i,j}$ (blue) represents the projection of point $\mathbf{b}_i$ on camera $\mathbf{a}_j$.

where $\mathbf{Q}(\mathbf{a}_j, \mathbf{b}_i)$ represents the projection of the reconstructed 3D point $i$ on the camera $j$ and $\mathbf{x}_{i,j} \in \mathbb{R}^2$ is the corresponding 2D image measurement (obtained as the result of running a corner detection algorithm such as FAST [130]).

The error function is therefore defined as:

$$\mathbf{E}(\mathbf{a}_j, \mathbf{b}_i) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (\mathbf{x}_{i,j} - \mathbf{Q}(\mathbf{a}_j, \mathbf{b}_i))^2 \ . \tag{4.3}$$

The most used method to solve (4.1) in the BA literature consists of applying the Levenberg-Marquardt (LM) procedure coupled with the *Schur complement* trick and an exact sparse solver such as sparse-Cholesky (we based our hybrid GPU/CPU implementation on this). This is because LM provides fast and guaranteed convergence properties. In addition, with the Schur complement trick we can first compute an expensive solution for a small subset of parameters (*e.g.* camera parameters) followed by a straightforward back-substitution on a larger subset (*e.g.* point parameters). A solver such as sparse-Cholesky exploits the sparse matrix nature of the normal equations to become more memory and compute efficient than a dense solver. However, as we will see later, this is not an ideal algorithm for mapping computations to a GPU, where inexact methods seem to excel.

## 4.2 Normal Equation

To simplify the BA formulation described next, we will follow closely the notation used by Lourakis and Argyros [96] in their 'Sparse Bundle Adjustment' software package.

Let us define a combined *parameter vector* by grouping the set of $m$ cameras and $n$ points as follows:

$$\mathbf{P} = (\mathbf{a}_0^\top, ..., \mathbf{a}_{m-1}^\top, ..., \mathbf{b}_0^\top, ..., \mathbf{b}_{n-1}^\top)^\top \ , \tag{4.4}$$

where $\mathbf{a}_j \in \mathbb{SE}(3)$ and $\mathbf{b}_i \in \mathbb{R}^3$.

Similarly, we define the *measurement vector* as:

$$\mathbf{X} = (\mathbf{x}_{0,0}^\top, ..., \mathbf{x}_{0,m-1}^\top, \mathbf{x}_{1,0}^\top, ..., \mathbf{x}_{1,m-1}^\top, ..., \mathbf{x}_{n-1,0}^\top, ..., \mathbf{x}_{n-1,m-1}^\top)^\top \ , \tag{4.5}$$

where $\mathbf{x}_{i,j}^\top \in \mathbb{R}^2$ represents the 2D measurement of point $i$ in camera $j$.

Using the combined parameter vector 4.4 and measurement vector 4.5, we define the residual vector as:

$$\mathbf{r} = \mathbf{X} - \mathbf{Q}(\mathbf{P}) \ , \tag{4.6}$$

where $\mathbf{Q}(\mathbf{P})$ applies the projective transformation $\mathbf{Q}$ defined in 4.2 to the parameter vector $\mathbf{P}$. The gradient of $\mathbf{r}$ is given by:

$$\nabla_{\mathbf{r}} = \frac{\partial \mathbf{r}}{\partial \mathbf{P}} = -\mathbf{J} \ , \tag{4.7}$$

where $\mathbf{J}$ is the Jacobian of $\mathbf{Q}$:

$$\mathbf{J} = \frac{\partial \mathbf{Q}}{\partial \mathbf{P}} = \frac{\partial \mathbf{X}}{\partial \mathbf{P}} \ . \tag{4.8}$$

Therefore we can express the error function defined in 4.3 as:

$$\mathbf{E}(\mathbf{P}) = \mathbf{r}^\top \mathbf{r} \ . \tag{4.9}$$

The minimum of $\mathbf{E}$ is achieved when a gradient is zero:

$$\nabla_{\mathbf{E}} = 2(\nabla_{\mathbf{r}})^\top \mathbf{r} = -2\mathbf{J}^\top \mathbf{r} = 0 \ . \tag{4.10}$$

To find a solution we begin by first linearly approximating (**??**) with a first-order Taylor expansion around $\hat{\mathbf{P}}$ for a small step $\boldsymbol{\delta}_\mathbf{P}$:

$$\mathbf{Q}(\mathbf{P} + \boldsymbol{\delta}_\mathbf{P}) \approx \mathbf{Q}(\hat{\mathbf{P}}) + \mathbf{J}\boldsymbol{\delta}_\mathbf{P} \ . \tag{4.11}$$

Then, starting from a close solution $\hat{\mathbf{P}}_0$ (our initial 3D reconstruction and camera poses) and measurements $\mathbf{X}$, we iteratively use this linear approximation to find a sequence $\{\hat{\mathbf{P}}_1, \hat{\mathbf{P}}_2, ...\hat{\mathbf{P}}^k\}$ converging to a local minimum.

Replacing the linearised model from 4.11 in 4.6 we obtain:

$$\mathbf{r} = \mathbf{X} - (\mathbf{Q}(\hat{\mathbf{P}}) + \mathbf{J}\boldsymbol{\delta}_\mathbf{P}) = \mathbf{X} - \mathbf{Q}(\hat{\mathbf{P}}) - \mathbf{J}\boldsymbol{\delta}_\mathbf{P} \ . \tag{4.12}$$

Defining the reprojection error estimate as:

$$\boldsymbol{\epsilon} = \mathbf{X} - \mathbf{Q}(\hat{\mathbf{P}}) \ , \tag{4.13}$$

and replacing it in 4.12 we get:

$$\mathbf{r} = \boldsymbol{\epsilon} - \mathbf{J}\boldsymbol{\delta}_\mathbf{P} \ . \tag{4.14}$$

With this in place we replace 4.14 in 4.10 obtaining:

$$-2\mathbf{J}^\top(\boldsymbol{\epsilon} - \mathbf{J}\boldsymbol{\delta}_\mathbf{P}) = 0 \ , \tag{4.15}$$

$$-2\mathbf{J}^\top\boldsymbol{\epsilon} + 2\mathbf{J}^\top\mathbf{J}\boldsymbol{\delta}_\mathbf{P} = 0 \ . \tag{4.16}$$

This leads to the following normal equation that we use to solve for $\boldsymbol{\delta}_\boldsymbol{p}$:

$$\mathbf{J}^\top\mathbf{J}\boldsymbol{\delta}_\mathbf{P} = \mathbf{J}^\top\boldsymbol{\epsilon} \ . \tag{4.17}$$

Thus, on each iteration, our local minimizer becomes:

$$\mathbf{P}^{new} = \mathbf{P}^{old} + \boldsymbol{\delta}_\mathbf{P} \ . \tag{4.18}$$

The Levenberg-Marquardt algorithm augments Equation (4.17) with a damping term $\mu$ resulting in:

$$(\mathbf{J}^\top\mathbf{J} + \mu\mathbf{I})\boldsymbol{\delta}_\mathbf{P} = \mathbf{J}^\top\boldsymbol{\epsilon} \ . \tag{4.19}$$

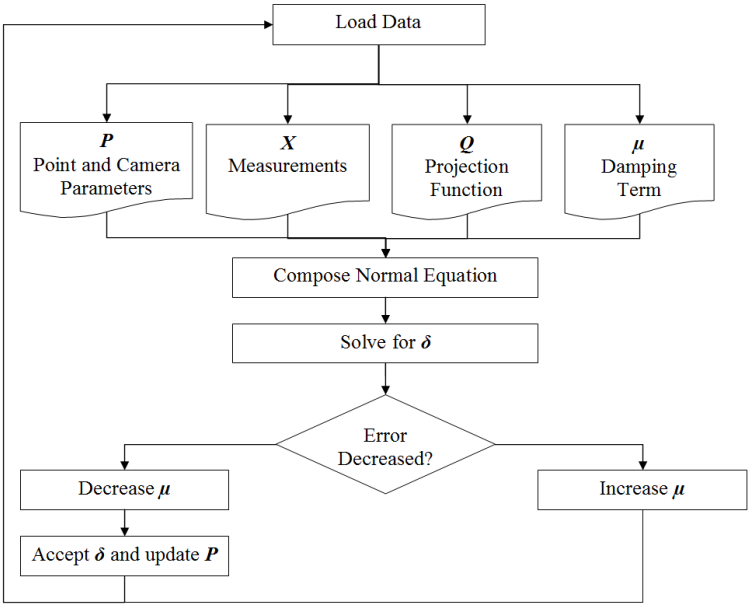



Figure 4.3: The bundle adjustment process solved with an iterative Levenberg-Marquardt algorithm.

The damping term $\mu$ can be thought of as a steering variable that is dynamically updated, allowing us to shift the solution along the direction of steepest-descent: $\mathbf{J}^\top \boldsymbol{\epsilon}$ (slow but with guaranteed convergence) or Gauss-Newton: $\boldsymbol{\delta_p}$ (fast quadratic convergence but with possibilities of singularities).

If the reprojection error is reduced, the damping term $\mu$ is decreased, skewing the solution towards a faster Gauss-Newton direction. If however we fail to reduce the error, the damping term is increased to move along a safer steepest-descent direction (see Figure 4.3).

## 4.3 Jacobian matrix structure

We will now analyse the structure of the BA optimisation problem with the aim of understanding the role that parallel processing could play in a hybrid CPU/GPU implementation.

To facilitate understanding, we will initially concentrate on a problem consisting of 2 cameras and 4 points, and assume that all points are visible across the camera set.

Expanding the Jacobian matrix defined in 4.8 where each row corresponds to a different data point and each column corresponds to a different parameter, we get:

$$\mathbf{J} = \left[\begin{array}{cc|cccc} \frac{\partial \hat{\mathbf{x}}_{0,0}}{\partial \mathbf{a}_0} & \frac{\partial \hat{\mathbf{x}}_{0,0}}{\partial \mathbf{a}_1} & \frac{\partial \hat{\mathbf{x}}_{0,0}}{\partial \mathbf{b}_0} & \frac{\partial \hat{\mathbf{x}}_{0,0}}{\partial \mathbf{b}_1} & \frac{\partial \hat{\mathbf{x}}_{0,0}}{\partial \mathbf{b}_2} & \frac{\partial \hat{\mathbf{x}}_{0,0}}{\partial \mathbf{b}_3} \\ \frac{\partial \hat{\mathbf{x}}_{0,1}}{\partial \mathbf{a}_0} & \frac{\partial \hat{\mathbf{x}}_{0,1}}{\partial \mathbf{a}_1} & \frac{\partial \hat{\mathbf{x}}_{0,1}}{\partial \mathbf{b}_0} & \frac{\partial \hat{\mathbf{x}}_{0,1}}{\partial \mathbf{b}_1} & \frac{\partial \hat{\mathbf{x}}_{0,1}}{\partial \mathbf{b}_2} & \frac{\partial \hat{\mathbf{x}}_{0,1}}{\partial \mathbf{b}_3} \\ \frac{\partial \hat{\mathbf{x}}_{1,0}}{\partial \mathbf{a}_0} & \frac{\partial \hat{\mathbf{x}}_{1,0}}{\partial \mathbf{a}_1} & \frac{\partial \hat{\mathbf{x}}_{1,0}}{\partial \mathbf{b}_0} & \frac{\partial \hat{\mathbf{x}}_{1,0}}{\partial \mathbf{b}_1} & \frac{\partial \hat{\mathbf{x}}_{1,0}}{\partial \mathbf{b}_2} & \frac{\partial \hat{\mathbf{x}}_{1,0}}{\partial \mathbf{b}_3} \\ \frac{\partial \hat{\mathbf{x}}_{1,1}}{\partial \mathbf{a}_0} & \frac{\partial \hat{\mathbf{x}}_{1,1}}{\partial \mathbf{a}_1} & \frac{\partial \hat{\mathbf{x}}_{1,1}}{\partial \mathbf{b}_0} & \frac{\partial \hat{\mathbf{x}}_{1,1}}{\partial \mathbf{b}_1} & \frac{\partial \hat{\mathbf{x}}_{1,1}}{\partial \mathbf{b}_2} & \frac{\partial \hat{\mathbf{x}}_{1,1}}{\partial \mathbf{b}_3} \\ \frac{\partial \hat{\mathbf{x}}_{2,0}}{\partial \mathbf{a}_0} & \frac{\partial \hat{\mathbf{x}}_{2,0}}{\partial \mathbf{a}_1} & \frac{\partial \hat{\mathbf{x}}_{2,0}}{\partial \mathbf{b}_0} & \frac{\partial \hat{\mathbf{x}}_{2,0}}{\partial \mathbf{b}_1} & \frac{\partial \hat{\mathbf{x}}_{2,0}}{\partial \mathbf{b}_2} & \frac{\partial \hat{\mathbf{x}}_{2,0}}{\partial \mathbf{b}_3} \\ \frac{\partial \hat{\mathbf{x}}_{2,1}}{\partial \mathbf{a}_0} & \frac{\partial \hat{\mathbf{x}}_{2,1}}{\partial \mathbf{a}_1} & \frac{\partial \hat{\mathbf{x}}_{2,1}}{\partial \mathbf{b}_0} & \frac{\partial \hat{\mathbf{x}}_{2,1}}{\partial \mathbf{b}_1} & \frac{\partial \hat{\mathbf{x}}_{2,1}}{\partial \mathbf{b}_2} & \frac{\partial \hat{\mathbf{x}}_{2,1}}{\partial \mathbf{b}_3} \\ \frac{\partial \hat{\mathbf{x}}_{3,0}}{\partial \mathbf{a}_0} & \frac{\partial \hat{\mathbf{x}}_{3,0}}{\partial \mathbf{a}_1} & \frac{\partial \hat{\mathbf{x}}_{3,0}}{\partial \mathbf{b}_0} & \frac{\partial \hat{\mathbf{x}}_{3,0}}{\partial \mathbf{b}_1} & \frac{\partial \hat{\mathbf{x}}_{3,0}}{\partial \mathbf{b}_2} & \frac{\partial \hat{\mathbf{x}}_{3,0}}{\partial \mathbf{b}_3} \\ \frac{\partial \hat{\mathbf{x}}_{3,1}}{\partial \mathbf{a}_0} & \frac{\partial \hat{\mathbf{x}}_{3,1}}{\partial \mathbf{a}_1} & \frac{\partial \hat{\mathbf{x}}_{3,1}}{\partial \mathbf{b}_0} & \frac{\partial \hat{\mathbf{x}}_{3,1}}{\partial \mathbf{b}_1} & \frac{\partial \hat{\mathbf{x}}_{3,1}}{\partial \mathbf{b}_2} & \frac{\partial \hat{\mathbf{x}}_{3,1}}{\partial \mathbf{b}_3} \end{array}\right] . \tag{4.20}$$

As we will see in Section 4.4, the elements of the above matrix are also sub-matrices because the data and model parameters are vectors.

Furthermore we will decompose the Jacobian matrix $\mathbf{J}$ based on two components, one for the camera parameters $\mathbf{a}_j$ and another for the point parameters $\mathbf{b}_i$:

$$\mathbf{A}_{ij} = \frac{\partial \hat{\mathbf{x}}_{i,j}}{\partial \mathbf{a}_j} \ , \mathbf{B}_{ij} = \frac{\partial \hat{\mathbf{x}}_{i,j}}{\partial \mathbf{b}_i} \ . \tag{4.21}$$

Here the *Frame Jacobian* $\mathbf{A}_{ij}$ expresses a differential change in the 2D estimate $\hat{\mathbf{x}}_{i,j}$ due to changes in the camera parameters $\mathbf{a}_j$ and likewise the *Point Jacobian* $\mathbf{B}_{ij}$ expresses a differential change of $\hat{\mathbf{x}}_{i,j}$ due to changes in the point parameters $\mathbf{b}_i$ (see Figure 4.4).

Note that $\frac{\partial \hat{\mathbf{x}}_{i,j}}{\partial \mathbf{a}_k} = \mathbf{0}, \forall j \neq k$ because changing the parameters of camera $k$ has no effect on the estimates at camera $j$. Similarly $\frac{\partial \hat{\mathbf{x}}_{i,j}}{\partial \mathbf{b}_k} = \mathbf{0}, \forall i \neq k$ because the points are assumed to be independent and therefore changing the parameters of point $k$ has no effect on the estimates at point $i$ [66]. Therefore 4.20 simplifies to the following

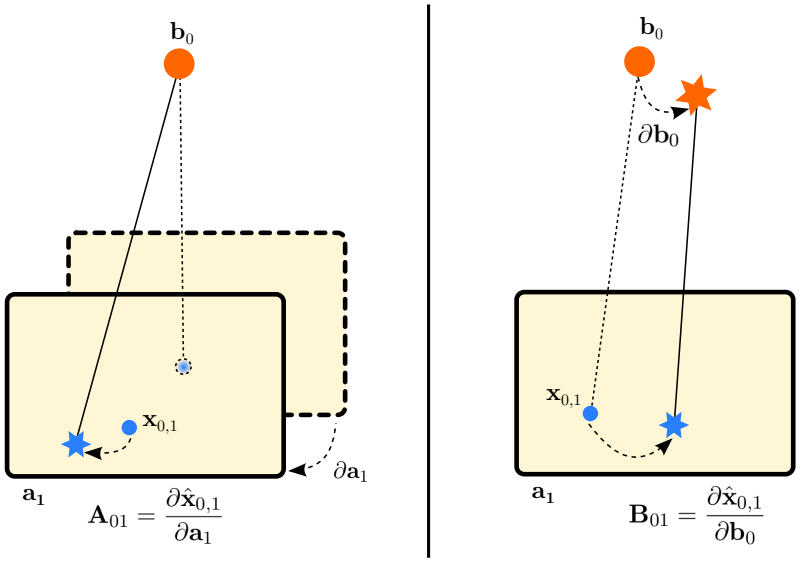



Figure 4.4: **(left)** The frame Jacobian $\mathbf{A}_{01}$ expresses a differential change in the 2D estimate $\hat{\mathbf{x}}_{0,1}$ due to a change $\partial\mathbf{a}_1$ in the camera parameters $\mathbf{a}_1$. **(right)** The point Jacobian $\mathbf{B}_{01}$ expresses a differential change in the 2D estimate $\hat{\mathbf{x}}_{0,1}$ due to a change $\partial\mathbf{b}_0$ in the point parameters $\mathbf{b}_0$.

sparse matrix:

$$\mathbf{J} = \left[\begin{array}{cc|cccc} \frac{\partial\hat{\mathbf{x}}_{0,0}}{\partial\mathbf{a}_0} & \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{0,0}}{\partial\mathbf{b}_0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{0,1}}{\partial\mathbf{a}_1} & \frac{\partial\hat{\mathbf{x}}_{0,1}}{\partial\mathbf{b}_0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \frac{\partial\hat{\mathbf{x}}_{1,0}}{\partial\mathbf{a}_0} & \mathbf{0} & \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{1,0}}{\partial\mathbf{b}_1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{1,1}}{\partial\mathbf{a}_1} & \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{1,1}}{\partial\mathbf{b}_1} & \mathbf{0} & \mathbf{0} \\ \frac{\partial\hat{\mathbf{x}}_{2,0}}{\partial\mathbf{a}_0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{2,0}}{\partial\mathbf{b}_2} & \mathbf{0} \\ \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{2,1}}{\partial\mathbf{a}_1} & \mathbf{0} & \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{2,1}}{\partial\mathbf{b}_2} & \mathbf{0} \\ \frac{\partial\hat{\mathbf{x}}_{3,0}}{\partial\mathbf{a}_0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{3,0}}{\partial\mathbf{b}_3} \\ \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{3,1}}{\partial\mathbf{a}_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial\hat{\mathbf{x}}_{3,1}}{\partial\mathbf{b}_3} \end{array}\right] \tag{4.22}$$

$$= \left[\begin{array}{cc|cccc} \mathbf{A}_{00} & \mathbf{0} & \mathbf{B}_{00} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{01} & \mathbf{B}_{01} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{10} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{10} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{11} & \mathbf{0} & \mathbf{B}_{11} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{20} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{20} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{21} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{21} & \mathbf{0} \\ \mathbf{A}_{30} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{30} \\ \mathbf{0} & \mathbf{A}_{31} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{31} \end{array}\right]. \tag{4.23}$$

To compose the normal equation $\mathbf{J}^\top \mathbf{J} \boldsymbol{\delta}_\mathbf{P} = \mathbf{J}^\top \boldsymbol{\epsilon}$ from 4.17, we define the following:

$$\mathbf{U}_j = \sum_i \mathbf{A}_{ij}^\top \mathbf{A}_{ij} \ , \tag{4.24}$$

$$\mathbf{V}_i = \sum_j \mathbf{B}_{ij}^\top \mathbf{B}_{ij} \ , \tag{4.25}$$

$$\mathbf{W}_{ij} = \mathbf{A}_{ij}^\top \mathbf{B}_{ij} \ . \tag{4.26}$$

Therefore $\mathbf{J}^\top \mathbf{J}$ can be expressed compactly as:

$$\mathbf{J}^\top \mathbf{J} = \begin{bmatrix} \mathbf{U}_0 & \mathbf{0} & \mathbf{W}_{00} & \mathbf{W}_{10} & \mathbf{W}_{20} & \mathbf{W}_{30} \\ \mathbf{0} & \mathbf{U}_1 & \mathbf{W}_{01} & \mathbf{W}_{11} & \mathbf{W}_{21} & \mathbf{W}_{31} \\ \mathbf{W}_{00}^\top & \mathbf{W}_{01}^\top & \mathbf{V}_0 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}_{10}^\top & \mathbf{W}_{11}^\top & \mathbf{0} & \mathbf{V}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{W}_{20}^\top & \mathbf{W}_{21}^\top & \mathbf{0} & \mathbf{0} & \mathbf{V}_2 & \mathbf{0} \\ \mathbf{W}_{30}^\top & \mathbf{W}_{31}^\top & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{V}_3 \end{bmatrix} \ . \tag{4.27}$$

We partition $\boldsymbol{\delta}_\mathbf{P}$ into two sets, one for the camera parameters and another for the point parameters:

$$\boldsymbol{\delta}_\mathbf{P} = [\boldsymbol{\delta}_\mathbf{a}, \boldsymbol{\delta}_\mathbf{b}]^\top \ . \tag{4.28}$$

Similarly, we also partition $\mathbf{J}^\top \boldsymbol{\epsilon}$ into a camera parameters set and point parameters set:

$$\mathbf{J}^\top \boldsymbol{\epsilon} = [\boldsymbol{\epsilon}_{\mathbf{a}_j}, \boldsymbol{\epsilon}_{\mathbf{b}_i}]^\top \ , \tag{4.29}$$

$$\boldsymbol{\epsilon}_{\mathbf{a}_j} = \sum_i \mathbf{A}_{ij}^\top \epsilon_{ij} \ , \tag{4.30}$$

$$\boldsymbol{\epsilon}_{\mathbf{b}_i} = \sum_j \mathbf{B}_{ij}^\top \epsilon_{ij} \ . \tag{4.31}$$

With this, the normal equation can be rewritten in block form as:

$$\left[\begin{array}{cc|cccc} \mathbf{U}_0 & \mathbf{0} & \mathbf{W}_{00} & \mathbf{W}_{10} & \mathbf{W}_{20} & \mathbf{W}_{30} \\ \mathbf{0} & \mathbf{U}_1 & \mathbf{W}_{01} & \mathbf{W}_{11} & \mathbf{W}_{21} & \mathbf{W}_{31} \\ \hline \mathbf{W}_{00}^\top & \mathbf{W}_{01}^\top & \mathbf{V}_0 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}_{10}^\top & \mathbf{W}_{11}^\top & \mathbf{0} & \mathbf{V}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{W}_{20}^\top & \mathbf{W}_{21}^\top & \mathbf{0} & \mathbf{0} & \mathbf{V}_2 & \mathbf{0} \\ \mathbf{W}_{30}^\top & \mathbf{W}_{31}^\top & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{V}_3 \end{array}\right] \begin{bmatrix} \boldsymbol{\delta}_{\mathbf{a}_0} \\ \boldsymbol{\delta}_{\mathbf{a}_1} \\ \boldsymbol{\delta}_{\mathbf{b}_0} \\ \boldsymbol{\delta}_{\mathbf{b}_1} \\ \boldsymbol{\delta}_{\mathbf{b}_2} \\ \boldsymbol{\delta}_{\mathbf{b}_3} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\epsilon}_{\mathbf{a}_0} \\ \boldsymbol{\epsilon}_{\mathbf{a}_1} \\ \boldsymbol{\epsilon}_{\mathbf{b}_0} \\ \boldsymbol{\epsilon}_{\mathbf{b}_1} \\ \boldsymbol{\epsilon}_{\mathbf{b}_2} \\ \boldsymbol{\epsilon}_{\mathbf{b}_3} \end{bmatrix} \ . \tag{4.32}$$

Notice how the top-left and bottom-right blocks form block diagonal matrices and also the bottom-left is the transpose of the top-right block. We can further simplify the above expression as:

$$\begin{bmatrix} \mathbf{U}^* & \mathbf{W} \\ \mathbf{W}^\top & \mathbf{V}^* \end{bmatrix} \begin{bmatrix} \boldsymbol{\delta}_\mathbf{a} \\ \boldsymbol{\delta}_\mathbf{b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\epsilon}_\mathbf{a} \\ \boldsymbol{\epsilon}_\mathbf{b} \end{bmatrix} , \tag{4.33}$$

where $\mathbf{U}^*$ is the block diagonal matrix with elements being the sub-matrices $\mathbf{U}_0$, $\mathbf{U}_1$, ..., $\mathbf{U}_{m-1}$ and similarly for $\mathbf{V}^*$.

In this form, the solution for either $\boldsymbol{\delta}_\mathbf{a}$ or $\boldsymbol{\delta}_\mathbf{b}$ depends on both the camera and point parameters, but by multiplying the above expression with:

$$\begin{bmatrix} \mathbf{I} & -\mathbf{W}\mathbf{V}^{*-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} , \tag{4.34}$$

it can be simplified to:

$$\begin{bmatrix} \mathbf{U}^* - \mathbf{W}\mathbf{V}^{*-1}\mathbf{W}^\top & \mathbf{0} \\ \mathbf{W}^\top & \mathbf{V}^* \end{bmatrix} \begin{bmatrix} \boldsymbol{\delta}_\mathbf{a} \\ \boldsymbol{\delta}_\mathbf{b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\epsilon}_\mathbf{a} - \mathbf{W}\mathbf{V}^{*-1}\boldsymbol{\epsilon}_\mathbf{b} \\ \boldsymbol{\epsilon}_\mathbf{b} \end{bmatrix} . \tag{4.35}$$

This allow us to compute the solution for camera parameters first by solving:

$$\mathbf{S}\boldsymbol{\delta}_\mathbf{a} = \boldsymbol{\epsilon}_\mathbf{a} - \mathbf{W}\mathbf{V}^{*-1}\boldsymbol{\epsilon}_\mathbf{b} , \tag{4.36}$$

$$\mathbf{S} = \mathbf{U}^* - \mathbf{W}\mathbf{V}^{*-1}\mathbf{W}^\top , \tag{4.37}$$

where $\mathbf{S}$ is the *Schur complement* of $\mathbf{V}^*$. This is more efficient as typically the number of camera parameters is much smaller than the number of point parameters. Finally the point parameters are solved with:

$$\mathbf{V}^*\boldsymbol{\delta}_\mathbf{b} = \boldsymbol{\epsilon}_\mathbf{b} - \mathbf{W}^\top\boldsymbol{\delta}_\mathbf{a} . \tag{4.38}$$

## 4.4 Frame and Point Jacobian matrices

Following the insights given in [42], we will present a closed form solution for the frame Jacobian $\mathbf{A}_{ij}$ and point Jacobian $\mathbf{B}_{ij}$ matrices.

The rigid body transformation parametrised by $\mathbf{a}_j = [\mathbf{R}, \mathbf{t}]^\top \in \mathbb{SE}(3)$ rotates and translates a point $\mathbf{b}$ into the camera frame of reference as follows:

$$\mathbf{y} = \mathbf{R}\mathbf{b} + \mathbf{t} , \tag{4.39}$$

$$\mathbf{y} = [x, y, z]^\top . \tag{4.40}$$

From **??**, the projected 2D position is given by:

$$\mathbf{x} = \mathbf{Q}(\mathbf{a}, \mathbf{b}) = \mathbf{Q}(\mathbf{y}) \ . \tag{4.41}$$

Applying the chain rule to the point Jacobian $\mathbf{B}$ we get:

$$\mathbf{B} = \frac{\partial \mathbf{Q}}{\partial \mathbf{b}} = \frac{\partial \mathbf{Q}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{b}} \ . \tag{4.42}$$

The first part $\frac{\partial \mathbf{Q}}{\partial \mathbf{y}}$ is given by:

$$\frac{\partial \mathbf{Q}}{\partial \mathbf{y}} = \frac{1}{z} \begin{bmatrix} 1 & 0 & -x/z \\ 0 & 1 & -y/z \end{bmatrix} \ , \tag{4.43}$$

while the second part becomes:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{b}} = \frac{\partial (\mathbf{R}\mathbf{b} + \mathbf{t})}{\partial \mathbf{b}} = \mathbf{R} \ , \tag{4.44}$$

therefore combining 4.43 and 4.44 into 4.42 we obtain:

$$\mathbf{B} = \frac{1}{z} \begin{bmatrix} 1 & 0 & -x/z \\ 0 & 1 & -y/z \end{bmatrix} \mathbf{R} \ . \tag{4.45}$$

Applying the chain rule to the frame Jacobian $\mathbf{A}$ we get:

$$\mathbf{A} = \frac{\partial \mathbf{Q}}{\partial \mathbf{a}} = \frac{\partial \mathbf{Q}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{a}} \ . \tag{4.46}$$

The first part $\frac{\partial \mathbf{Q}}{\partial \mathbf{y}}$ is already given by 4.43. For the second part $\frac{\partial \mathbf{y}}{\partial \mathbf{a}}$, we will expresses the differential transformation with respect to camera parameters using an element $\varepsilon \in \mathfrak{so}(3)$ of its tangent space:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{a}} \approx \frac{\partial \exp(\varepsilon)\mathbf{y}}{\partial \varepsilon} = [\mathbf{G}_1, ..., \mathbf{G}_6]\mathbf{y} \ , \tag{4.47}$$

$$\Rightarrow \frac{\partial \mathbf{y}}{\partial \mathbf{a}} = \begin{bmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{bmatrix} \ . \tag{4.48}$$

Finally combining 4.43 and 4.48 into 4.46 we obtain:

$$\mathbf{A} = \begin{bmatrix} \frac{1}{z} & 0 & -\frac{x}{z^2} & -\frac{xy}{z^2} & 1 + \frac{x^2}{z^2} & -\frac{y}{z} \\ 0 & \frac{1}{z} & -\frac{y}{z^2} & -\left(1 + \frac{y^2}{z^2}\right) & \frac{xy}{z^2} & \frac{x}{z} \end{bmatrix} \ . \tag{4.49}$$

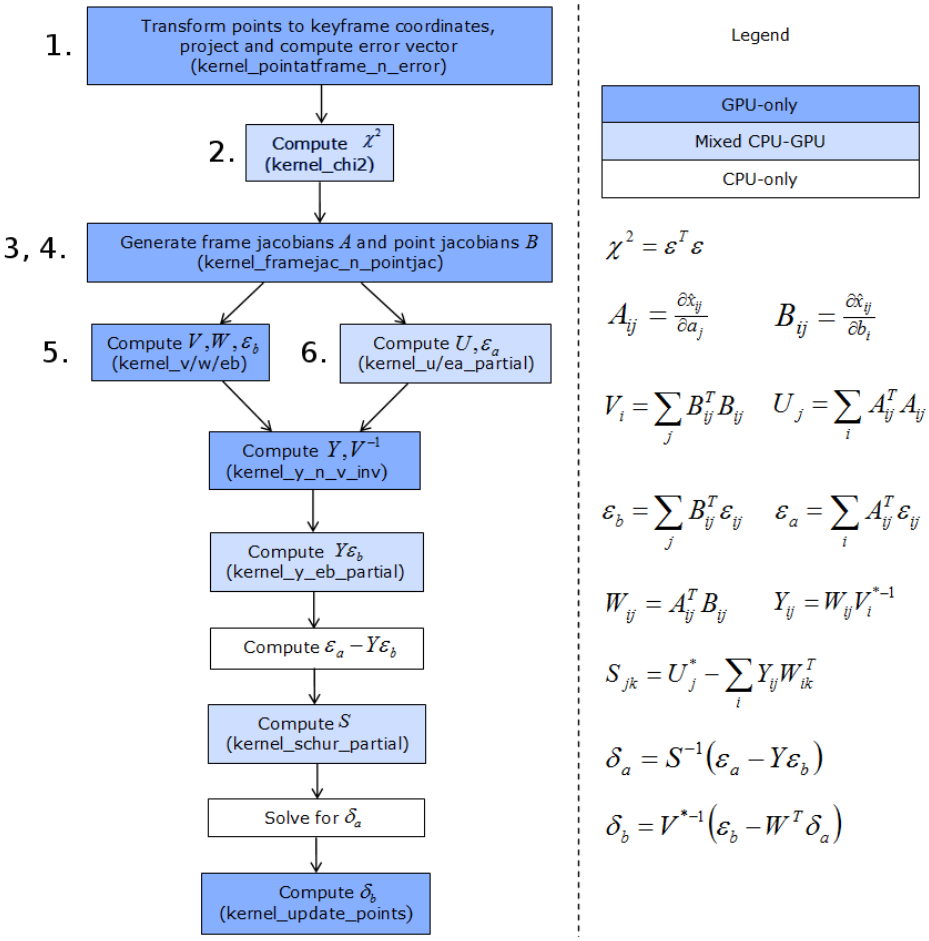



Figure 4.5: Mapping BA sub-steps to the GPU and CPU.

## 4.5 Implementation details

We are now ready to consider how an efficient hybrid CPU/GPU solution should be constructed.

Equation (4.19) may appear simple initially, but composing its terms is a compute and memory intensive process. We have identified a series of sub-steps and distinguished them according to how well they map to a GPU-only, CPU-only or a hybrid implementation (see Figure 4.5).

We will elaborate on the implementation of a few key sub-steps to highlight the implementation ideas.

1. **Error Vector (GPU-only)**: We load the point parameter vector $\mathbf{P}_b$ =

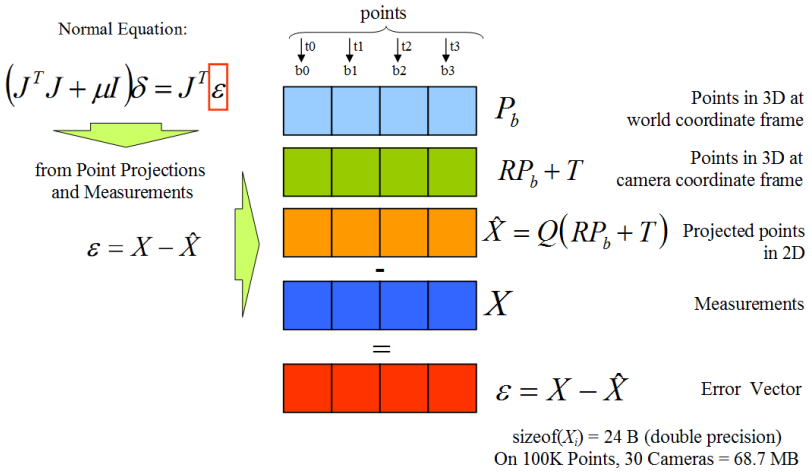



Figure 4.6: Error vector computation, an example of a GPU-only step.

$\{\mathbf{b}_i\}_{i=0}^{n-1}$ into the GPU global memory and process each point by a different thread by transforming them into the camera coordinate frame, projecting them into 2D and subtracting them from the measurement vector $\mathbf{X}$. The resulting error vector $\boldsymbol{\epsilon}$ is placed back into global memory. Note that this procedure is repeated for each camera (Figure 4.6).

2. **Error Vector Norm (Hybrid GPU/CPU)**: We load the previously computed error vector $\boldsymbol{\epsilon}$ and pairwise multiply it with itself, using one thread per point. The vector norm $\chi^2$ is calculated by summing the product results with a *parallel reduction* algorithm (See Section 3.8.3). Since we cannot synchronise threads across blocks, the final norm is generated on the CPU (Figure 4.7).

3. **Frame Jacobian (GPU-only)**: We need to generate a $2 \times 6$ frame Jacobian matrix $\mathbf{A}_{ij}$ per measurement (for each point on each camera where it is visible). This represents the differential change in our 2D projection estimates due to a change in camera parameters (Figure 4.8).

4. **Point Jacobian (GPU-only)**: Similar to the previous step, we need to generate a $2 \times 3$ point Jacobian matrix $\mathbf{B}_{ij}$ per measurement. This represents the differential change in our 2D projection estimates due to a change in point parameters (Figure 4.9).

5. **V block matrix (GPU-only)**: As described before, $\mathbf{J}^\top\mathbf{J}$ has a sparse-block structure and $\mathbf{V}$ is a block-matrix of its lower diagonal elements, with $\mathbf{V}_i =$

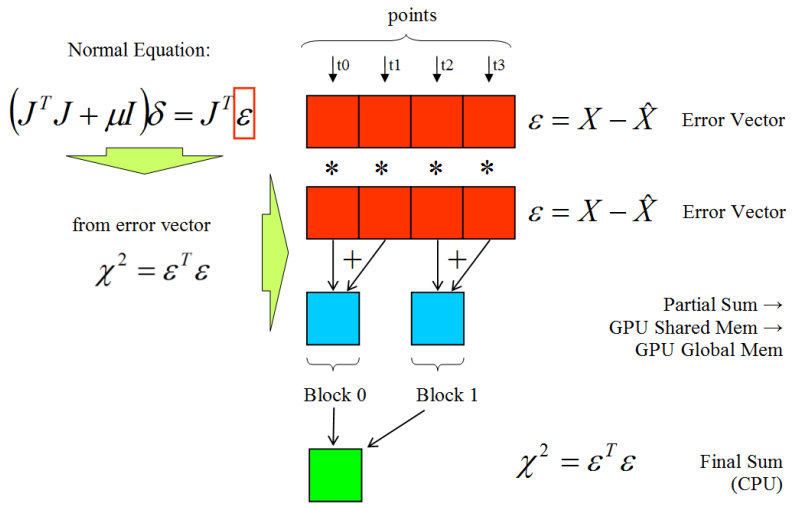



Figure 4.7: Error vector norm computation, an example of a hybrid GPU/CPU step.

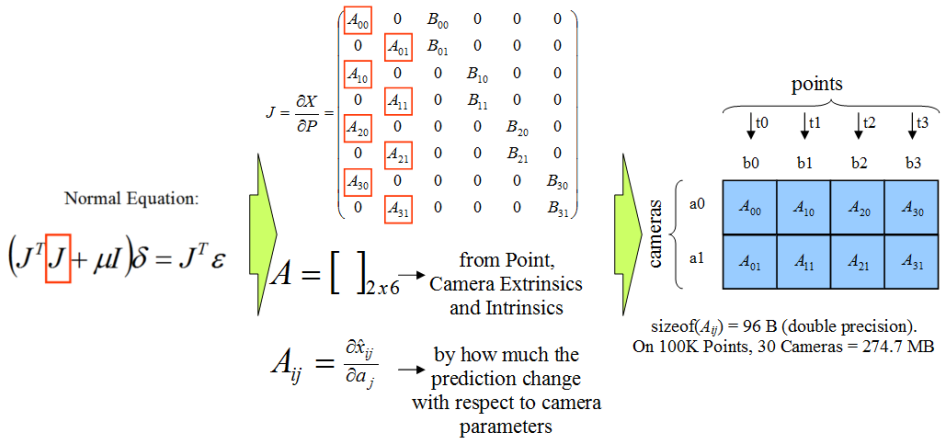



Figure 4.8: Frame Jacobian computation, an example of a GPU-only step.

$\sum_j(\mathbf{B}_{ij}^\top\mathbf{B}_{ij})$ being a $3 \times 3$ matrix (Figure 4.10). In this case a thread is launched per point, multiplying the point Jacobians and adding the results across cameras. Therefore we obtain a different $\mathbf{V}_i$ matrix per point.

6. **U block matrix (Hybrid GPU/CPU)**: $\mathbf{U}$ is a block-matrix of the upper diagonal elements of $\mathbf{J}^\top\mathbf{J}$, with $\mathbf{U}_j = \sum_i(\mathbf{A}_{ij}^\top\mathbf{A}_{ij})$ being a $6\times6$ matrix (Figure 4.11). A thread is launched per point and we need to synchronise them to perform multiplication and addition across points for a particular camera. The final result is generated on the CPU.

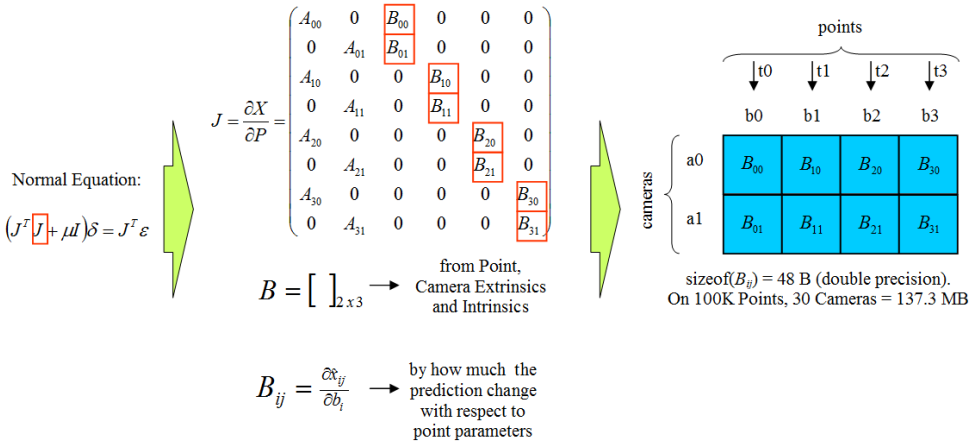



Figure 4.9: Point Jacobian computation, another GPU-only step.

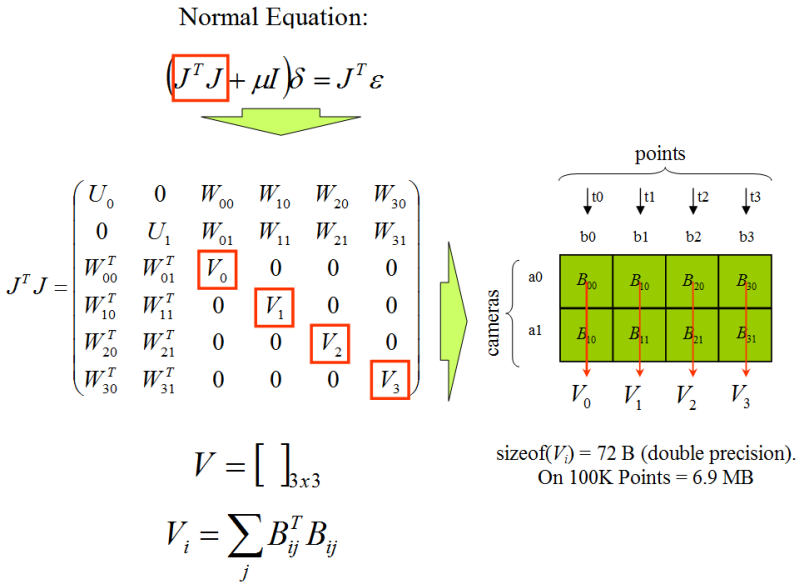



Figure 4.10: **V** block matrix computation, a GPU-only step.

Other sub-steps follow the same pattern as the examples above: independent computations are suitable for straightforward parallelism on the GPU, however, reduction operations typically need the cooperation of the CPU since there is no thread synchronisation across GPU blocks. Once the normal equation is composed, the actual solution is performed on the CPU using Cholesky factorisation.

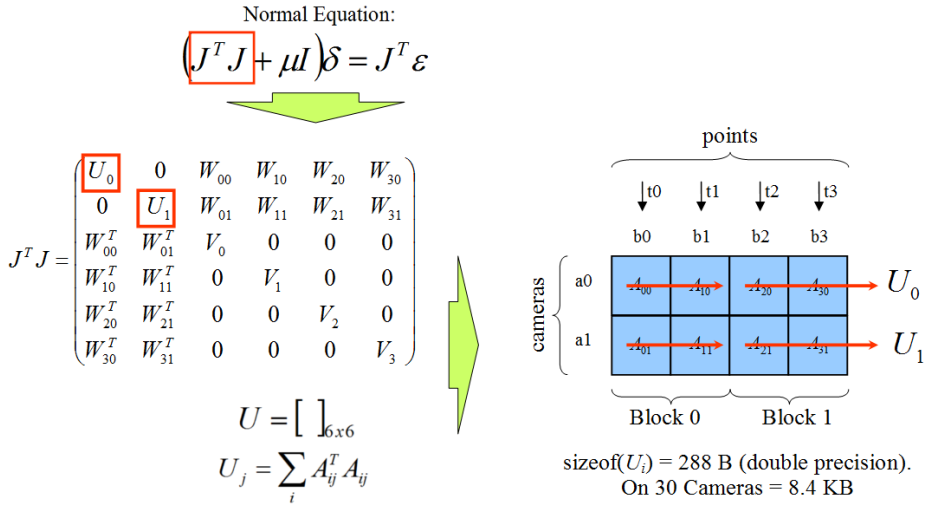



Figure 4.11: **U** block matrix computation, a hybrid GPU/CPU step.

## 4.6 Results

To evaluate the performance of our hybrid GPU/CPU bundle adjustment implementation, we generate synthetic point-cloud scenes and camera poses corrupted by Gaussian noise, as seen in Figure 4.1. The scenes are created with increasing number of points (from 100 to 250K) while keeping a fixed number of cameras (30). We assume that all points remain visible across the camera set. For each scene, we run our optimisation method 5 times and average the time for convergence. We do the same for a recent CPU-only version [149]. We use desktop PC equipped with an Intel i7-960 CPU running at 3.2GHz and an NIVIDIA GTX 480 GPU.

We achieved up to 10.5x speedup with our hybrid implementation compared to the CPU-only version (see Figure 4.12 and 4.13). We can observe consistent improvements for problems sizes of hundred of thousands of points, up to about 250K points and 30 cameras due to memory limitations. It is worth noting that under closer examination, our improvements are only valid when our problem size surpasses around 360 points. This is because the benefits of parallel computations on the GPU for a small problem are diminished due to transfer costs of data from main memory to GPU memory plus the overhead of thread scheduling.

Figure 4.14 displays the overall timing of GPU kernels. We can see that the most expensive computation is concentrated on computing the Schur complement of the

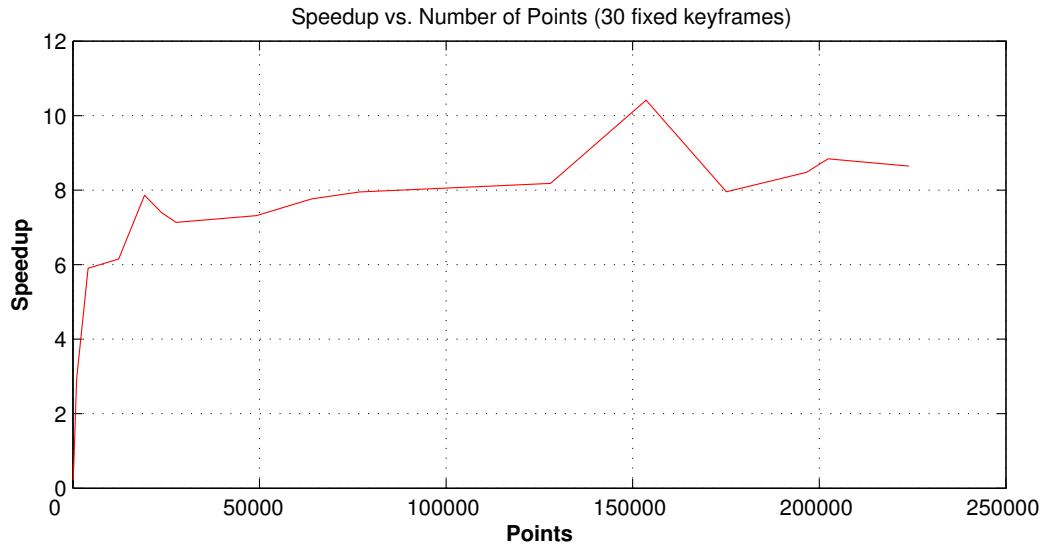



Figure 4.12: Speedup of BA running on the GPU/CPU vs. CPU-only, as a function of problem size (number of points). The sudden peaks of GPU performance at about 20K and 150K points are due to improved memory coalescing for those particular problem sizes.

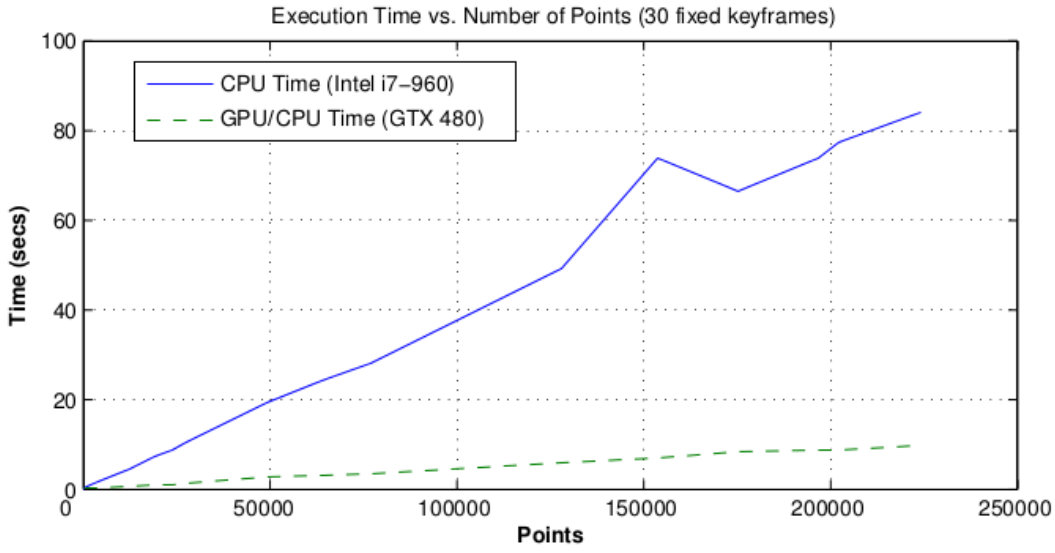



Figure 4.13: Runtime for BA on the GPU/CPU vs. CPU-only, as a function of problem size (number of points).

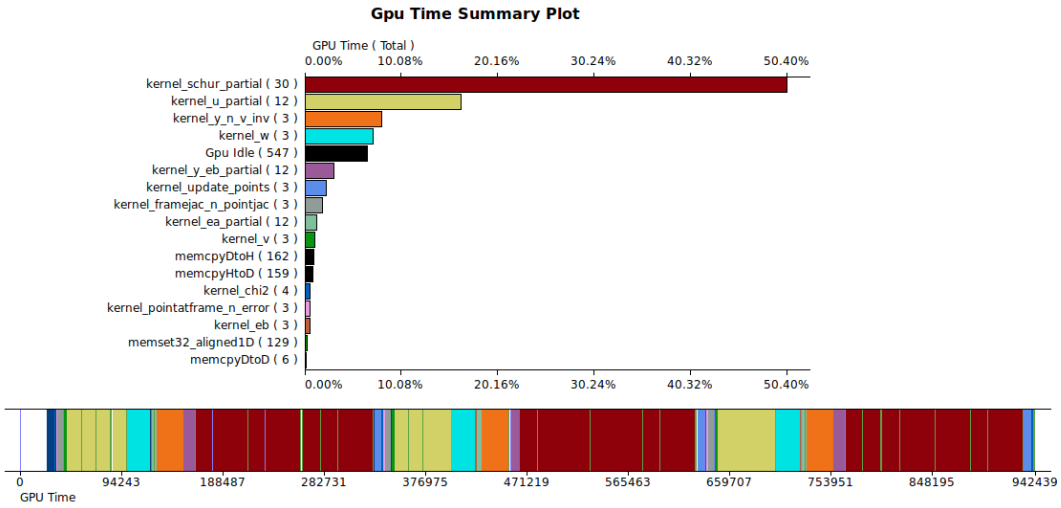



Figure 4.14: Overall time spend per GPU kernel (top) and order of calls for 3 iterations (bottom).

normal equation (about 50% of GPU time), followed by the partial computation of the **U** block matrix. Unfortunately, the Schur complement is difficult to parallelise as a whole and we resort to sequentially scheduling per-camera sub-tasks.

## 4.7 Difficulties

One of the reasons for the low performance of computing the **U** block matrix comes from the parallel reduction of its large $6 \times 6$ sub-matrix elements across all points. Each $6 \times 6$ sub-matrix occupies 288 bytes of memory, which saturates shared memory and available registers, resulting in low occupancy (the ratio between the actual number of scheduled threads and maximum possible number of working threads). Table 4.1 details the occupancy calculation considering the hardware limitations found on the Streaming Multiprocessors (SM) of NVIDIA GPUs with Fermi architecture:

Table 4.1: Occupancy calculation for the $\mathbf{U_j}$ matrix

| | |
|---|---|
| Size of $\mathbf{U_j}$ | $6 \times 6 \times 8 = 288$ bytes |
| Shared memory per Streaming Multiprocessor(SM) | 49152 bytes |
| Max thread count per SM | $49152/288 = 170.6$ |
| Max thread count per SM as power of 2 | 128 |
| Max allowed thread count per SM | 1536 |
| Achieved occupancy | $128/1536 = 0.083$ |

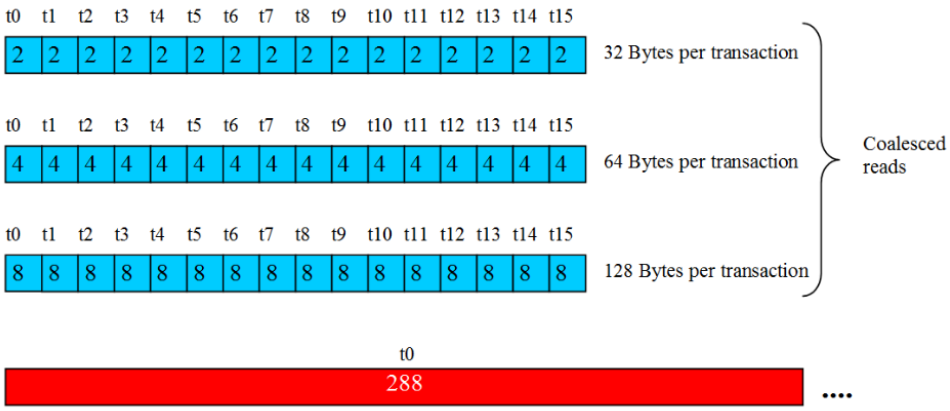



Figure 4.15: Inadequate memory coalescing case. The blue vectors represents coalesced memory access where each thread reads consecutive addresses of either 2, 4 or 8 bytes. In red we see a case where a single thread access a large chunk of 288 bytes corresponding to loading/storing a single $\mathbf{U}_j$ matrix.

Related to the previous issue is the fact that dealing with large matrices per thread complicates opportunities for memory coalescing as depicted in Figure 4.15. A GPU can group together memory access from continuous threads provided each one request chunks of 2, 4 or 8 bytes. For the case of computing and storing a $\mathbf{U}_j$ matrix per thread we can see that we largely move beyond the limit, requesting as much as 288 bytes per thread.

## 4.8 Conclusion and Future Work

We have taken a relatively popular serial implementation of Bundle Adjustment and decomposed it into a series of sub-steps, many of them being suitable for full parallelisation on the GPU, while others are hybridly computed between the GPU and CPU or on the CPU alone. This solution give us up to 10.5x speedup compared to a CPU-only version.

However we can further improve this work in the following ways:

1. Using different data structures, task decomposition or memory access patterns:

   One way to eliminate the need to handle large matrices per thread is to express the actual matrix as a Structure of Arrays (SoA) instead of treating the full set of matrices as an Array of matrix Structures (AoS) [104]. Also, our kernels uses relatively large number of temporary variables resulting in poor

register utilisation; this could be alleviated having slimmer kernels with finer task decomposition. Finally, the fragmented memory access patterns can be improved by having threads to cooperatively load large matrices into shared memory.

2. Using alternative optimisation algorithms that are better suited for GPUs:

   For example, Wu *et al.* [165] demonstrated the superior performance of an Inexact Newton method on the GPU, specifically the Linear Conjugate Gradient (LCG) instead of the Exact Dense or Sparse Cholesky decomposition such as the one we explored. One problem with the Cholesky decomposition is that the initial unfactored sparse matrix can become dense after factorisation. With the LCG the problem can be expressed without requiring to explicitly represent the Jacobian or the Schur complement matrices. This greatly reduces the memory allocation needs, enabling them to solve larger problems. In addition they identified floating point normalisation techniques to eliminate the need to use double precision arithmetic in favour of single precision which can have twice the speed on current generation of GPUs.

3. Abstracting algorithmic representations from the underlying platform implementation:

   The mentioned improvements, being valid and necessary, impose a great deal of programming effort for finding the right balance of computation and memory management across heterogeneous computing devices. Furthermore, the programmer's assumptions can break in newer hardware architectures such as the family of integrated GPU and CPU (AMD Fusion or Intel SandyBridge), with unified memory systems and zero data transfer costs. It is therefore desirable to abstract the algorithmic representation away from the implementation details if we aim to hit peak performance on a variety of platforms and problem sizes. This could lead to the development of a new Domain-Specific Language (DSL) for SLAM, similar in spirit to Halide [126], a language that simplify the development of high-performance image processing code targeting a variety of modern architectures.

A pure data-driven approach for performing joint map and camera optimisation is at the core of traditional SLAM systems. However, the arrival of dense solutions like KinectFusion [112] brought a new level of map resolution allowing to reason about

watertight surfaces and detached objects. Rather than ignoring the benefits of this novel representation, in our following work we decided to attack the underlying object-level structure observed in many man-made scenes, which as we will see in the next chapter, substantially simplifies the optimisation problem while bringing useful semantic information for interaction.

Chapter 5

# SLAM++: Simultaneous Localisation and Mapping at the Level of Objects

Most current real-time SLAM systems operate at the level of low-level primitives (points, lines, patches or non-parametric surface representations such as depth maps), which must be robustly matched, geometrically transformed and optimised over in order that maps represent the intricacies of the real world. Modern processing hardware permits ever-improving levels of detail and scale, and much interest is now turning to semantic labelling of this geometry in terms of the objects and regions that are known to exist in the scene. However, some thought about this process reveals a huge amount of wasted computational effort; and the potential for a much better way of taking account of domain knowledge *in the loop of SLAM operation itself.*

In this chapter, we propose a paradigm for real-time localisation and mapping which harnesses 3D object recognition to jump over low level geometry processing and produce incrementally-built maps directly at the 'object oriented' level. As a hand-held depth camera browses a cluttered scene, prior knowledge of the objects likely to be repetitively present enables real-time 3D recognition and the creation of a simple pose graph map of relative object locations. This graph is continuously optimised as new measurements arrive, and enables always up-to-date, dense and

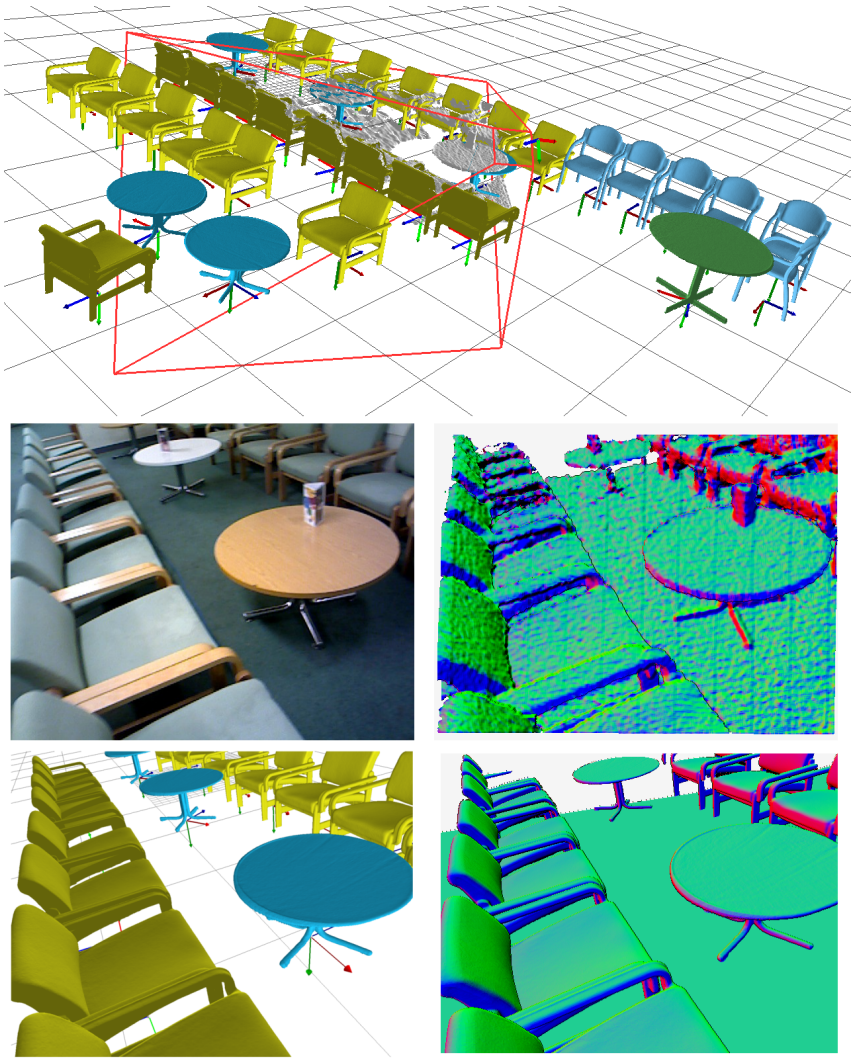


Figure 5.1: **(top)** In SLAM++, a cluttered 3D scene is efficiently tracked and mapped in real-time directly at the object level. **(left)** A live view at the current camera pose and the synthetic rendered objects. **(right)** We contrast a raw depth camera normal map with the corresponding high quality prediction from our object graph, used both for camera tracking and for masking object search.

precise prediction of the next camera measurement. These predictions are used for robust camera tracking and the generation of active search regions for further object detection.

Our approach is enabled by efficient GPU-Compute parallel implementation of recent advances in real-time 3D object detection and 6 DoF ICP-based pose refinement. We show that alongside the obvious benefit of an object-level scene descrip-

tion, this paradigm enables a vast compression of map storage compared to a dense reconstruction system with similar predictive power; and that it easily enables large scale loop closure, relocalisation and great potential for the use of domain-specific priors.

## 5.1 Real-Time SLAM with Hand-Held Sensors

In SLAM, building an internally consistent map in real-time from a moving sensor enables drift-free localisation during arbitrarily long periods of motion. We have still not seen truly 'pick up and play' SLAM systems which can be embedded in low-cost devices and used without concern or understanding by non-expert users, but there has been much recent progress. Until recently, the best systems used either monocular or stereo passive cameras. Sparse feature filtering methods like [37] were improved on by 'keyframe SLAM' systems like PTAM [79] which used bundle adjustment in parallel with tracking to enable high feature counts and more accurate tracking.

Most recently, a breakthrough has been provided by 'dense SLAM' systems, which take advantage of GPU-Compute processing hardware to reconstruct and track full surface models, represented non-parametrically as meshes or implicit surfaces. While this approach is possible with an RGB camera [113], commodity depth cameras have now come to the fore in high performance, robust indoor 3D mapping, in particular via the KinectFusion algorithm [112]. New developments such as [162] have tackled scaling the method via a sliding volume, sub-blocking or octrees; but a a truly scalable, multi-resolution, loop closure capable dense non-parametric surface representation remains elusive, and will always be wasteful in environments with symmetry.

From sparse feature-based SLAM, where the world is modelled as an unconnected point cloud, to dense SLAM which assumes that scenes contain continuous surfaces, we have seen an increase in the prior scene knowledge brought to bear. In SLAM++ we step up to the even stronger assumption that the world has intrinsic organisation in the form of repeated objects. While we currently pre-define the objects expected in a scene, we intend that the paradigm permits the objects in a scene to be identified and segmented automatically as salient, repeated elements.

Object SLAM has many characteristics of a return to feature-based SLAM meth-

ods. Unlike dense non-parametric approaches, the relatively few discrete entities in the map makes it highly feasible to jointly optimise over all object positions to make globally consistent maps. The fact that the map entities are now objects rather than point features, however, puts us in a stronger position than ever before; tracking one object in 6 DoF is enough to localise a camera, and reliable relocalisation of a lost camera or loop closure detection can be performed on the basis of just a small number of object measurements due to their high saliency. Further, and crucially, instant recognition of objects provides great efficiency and robustness benefits via the active approaches it permits to tracking and object detection, guided entirely by the dense predictions we can make of the positions of known objects.

SLAM++ relates strongly to the growing interest in semantically labelling scene reconstructions and maps, in both the computer vision and robotics communities, though we stress the big difference between post-hoc labelling of geometry and the closed loop, real-time algorithm we present. Some of the most sophisticated recent work was by Kim *et al.* [78]. A depth camera is first used to scan a scene, similar in scale and object content to the results we demonstrate later, and all data is fused into a single large point cloud. Off-line, learned object models, with a degree of variation to cope with a range of real object types, are then matched into the joint scan, optimising both similarity and object configuration constraints. The results are impressive, though the emphasis is on labelling rather than aiding mapping and we can see problems with missing data which cannot be fixed with non-interactive capture.

Other good work on labelling using RDB-D data was by Silberman [145] as well as Ren *et al.* [129] who used kernel descriptors for appearance and shape to label single depth camera images with object and region identity.

Several published approaches use object detection, like we do, with the aim of not just labelling but actually improving reconstruction and tracking; but none of these have taken the idea nearly as far as we have with the combination of real-time processing, full 3D operation, dense prediction and a modern graph optimisation back-end. To give a flavour of this work, Castle *et al.* [23] incorporated planar object detection into sparse feature-based monocular SLAM [37]. These objects, once recognized via SIFT descriptors, improved the quality of SLAM due to their known size and shape, though the objects were simple highly textured posters and the scene scale small.

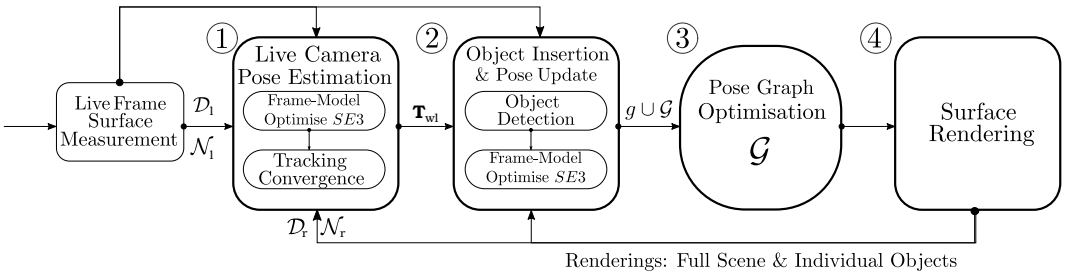



Figure 5.2: Outline of the SLAM++ pipeline. Given a live depth map $\mathcal{D}_\text{l}$, we first compute a surface measurement in the form of a vertex and normal map $\mathcal{N}_\text{l}$ providing input to the sequentially computed camera tracking and object detection pipelines. **(1)** We track the live camera pose $\mathbf{T}_\text{wl}$ with an iterative closest point approach using the dense multi-object scene prediction captured in the current SLAM graph $\mathcal{G}$. **(2)** Next we attempt to detect objects, previously stored in a database, that are present in the live frame, generating detection candidates with an estimated pose in the scene. Candidates are first refined or rejected using a second ICP estimation initialised with the detected pose. **(3)** We add successfully detected objects $g$ into the SLAM graph in the form of a object-pose vertex connected to the live estimated camera-pose vertex via a measurement edge. **(4)** Rendering objects from the SLAM graph produce a predicted depth $\mathcal{D}_\text{r}$ and normal map $\mathcal{N}_\text{r}$ into the live estimated frame, enabling us to actively search only those pixels not described by current objects in the graph. We run an individual ICP between each object and the live image resulting in the addition of a new camera-object constraint into the SLAM graph.

Also in an EKF SLAM paradigm, Ramos *et al.* [127] demonstrated a 2D laser and camera system which used object recognition to generate discrete entities to map (tree trunks) rather than using raw measurements. Finally, the same idea that object recognition aids reconstruction has been used in off-line structure from motion. Bao *et al.* [9] represented a scene as a set of points, objects and regions in two-view SfM, jointly solving the labelling and reconstruction problem via graph optimisation taking account the interactions between all scene entities.

## 5.2 Method

SLAM++ is overviewed in Figure 5.2, and detailed below.

### 5.2.1 Creating an Object Database

Before live operation in a certain scene, we rapidly make high quality 3D models of repeatedly occurring objects via interactive scanning using KinectFusion [112] in a

controlled setting where the object can easily be circled without occlusion. A mesh for the object is extracted from the truncated signed distance volume obtained from KinectFusion using marching cubes [95]. A small amount of manual editing in a mesh tool is performed to separate the object from the ground plane, and mark it up with a coordinate frame such that domain-specific object constraints can be applied. The reconstructed objects are then stored in a simple database.

### 5.2.2 SLAM Map Representation

Our representation of the world is a graph, where each node stores either the estimated $\mathbb{SE}(3)$ pose (rotation and translation relative to a fixed world frame) $\mathbf{T}_{\mathrm{wo}_j}$ of object $j$, or $\mathbf{T}_{\mathrm{w}i}$ of the historical pose of the camera at timestep $i$ (see Figure 5.16). Each object node is annotated with a type from the object database. Each $\mathbb{SE}(3)$ measurement of the pose of an object $Z_{i,o_j}$ from the camera is stored in the graph as a factor (constraint) which links one camera pose and one object pose. Additional factors can optionally be added to the graph; between camera poses to represent camera-camera motion estimates (*e.g.* from ICP), or domain-specific *structural* priors, *e.g.* that certain types of objects must be grounded on the same plane. Details on graph optimisation are given in Section 5.2.6 and Figure 5.16.

### 5.2.3 Real-Time Object Recognition via Shape Matching

We follow the approach of Drost *et al.* [41] for recognising the 6 DoF pose of 3D objects, represented by meshes, in a depth image. We give details of our parallel implementation, which achieves the real-time detection of multiple instances of multiple objects we need by fully exploiting the fine-grained parallelism of GPUs.

As in the Generalised Hough Transform [8], in Drost *et al.*'s method an object is detected and simultaneously localised via the accumulation of votes in a parameter space. The basis of voting is the correspondence between Point-Pair Features (PPFs): four-dimensional descriptors of the relative position and normals of pairs of oriented points on the surface of an object. Points, with normal estimates, are randomly sampled on a bilateral-filtered image from the depth camera. These samples are paired up in all possible combinations to generate PPFs which vote for 6 DoF model configurations containing a similar PPF.

A global description for each object mesh is quickly generated on the GPU by discretising PPFs with similar values and storing them in search data structures built

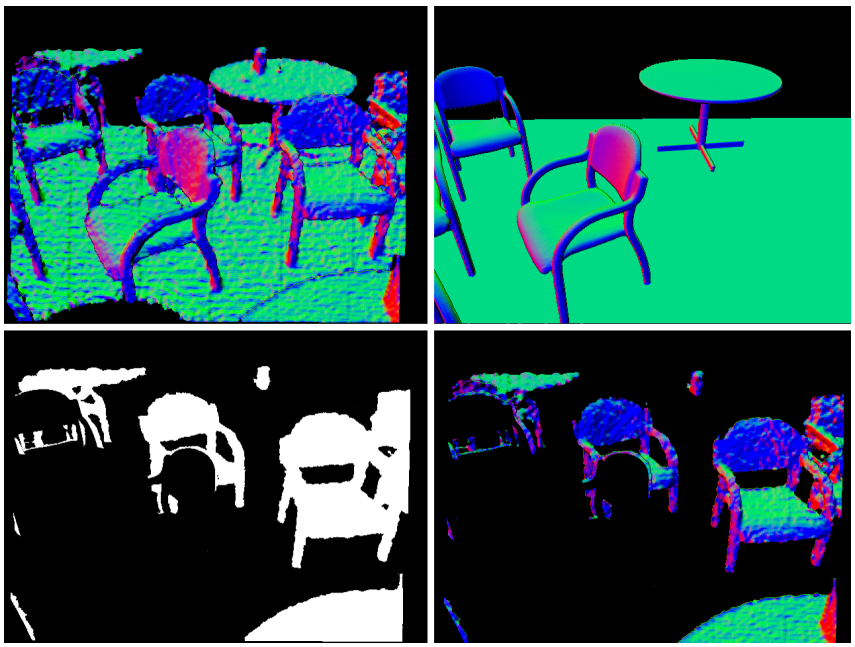


Figure 5.3: Object recognition with active search. **(top-left)** Measured normal map from sensor. **(top-right)** Predicted view from graph. **(bottom-left)** Undescribed regions to be searched are in white. **(bottom-right)** Masked normal map used for recognition.

using parallel primitive operations such as reduction, scan and sort (see Algorithm 1), provided by modern GPU template libraries such as Bolt [1]. Similar structures are also built from each live frame. This process typically takes <5ms for 160K PPFs and could also be used in the future to describe new object classes on the fly as they are automatically segmented.

Matching similar features of the scene against the model can be efficiently performed in parallel via a vectorised binary search, producing a vote for each match. Accumulation of votes into a shared buffer can be prohibitively expensive on the GPU, where many threads would require atomic operations when incrementing a common memory location to avoid race conditions. To overcome this, each vote is represented as a 64-bit integer code (Figure 5.4), which can then be efficiently sorted and reduced in parallel.

Sorting puts corresponding model points, scene reference points and alignment angles contiguously in memory. This is followed by a parallel reduction with a *sum* operation to accumulate equal vote codes (Algorithm 2). After peak finding,

**Algorithm 1:** Build global description from PPFs

**input** : A set of $N$ point-pair features (PPF)
**output**: Set of search data structures

```
// Encode the PPF index and hash key
codes ← new array;
foreach i ← 0 to N - 1 in parallel do
    Compute the hash key hk from PPF i;
    codes[i] = hk ≪ 32 + i;
codes ← Sort(codes);
// Decode PPF index and hash key
key2ppfMap ← new array;
hashKeys ← new array;
foreach i ← 0 to N - 1 in parallel do
    key2ppfMap[i] = ∼(1 ≪ 32) & code[i];
    hashKeys[i] = code[i] ≫ 32;
// Count PPF per hash key and remove duplicate keys
hashKeys, ppfCount ← Reduce(hashKeys, sumOp);
// Find the first PPF index of a block of PPFs in key2ppfMap having
   equal hash key
firstPPFIndex ← ExclScan(ppfCount);
return hashKeys, ppfCount, firstPPFIndex, key2ppfMap;
```

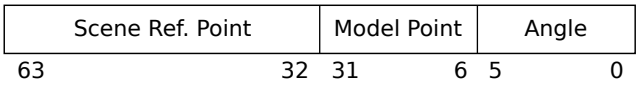



Figure 5.4: Packed 64-bit integer vote code. The first 6 bits encode the alignment angle, followed by 26 bits for the model point and 32 bits for the scene reference point.

pose estimates for each scene reference point are clustered on the CPU according to translation and rotation thresholds as in [41].

**Algorithm 2:** Accumulate Votes

**input** : A set of $M$ vote codes
**output**: Vote count per unique vote code

```
// Sort voteCodes in parallel
voteCodes ← Sort(voteCodes);
// Count votes with equal voteCode
voteCodes, voteCount ← Reduce(voteCodes, sumOp);
return voteCodes, voteCount;
```

#### Active Object Search

Standard application of Drost *et al.*'s recognition algorithm [41] in room scenes is highly successful when objects occupy most of the camera's field of view, but poor for objects which are distant or partly occluded by other objects, due to poor sample point coverage. It is here that we realise one of the major benefits of our in-the-loop SLAM++ approach. The view prediction capabilities of the system mean that we can generate a mask in image space for depth pixels which are not already described by projected known objects. The measured depth images from the camera are cropped using these masks and samples are only spread in the undescribed regions (see Figure 5.3).

The result of object detection is often multiple cluster peaks, and quantised localisation results. These must be passed to ICP refinement as explained in the next section.

### 5.2.4 Real-Time Object Recognition with Hough Forests

The method of Drost *et al.* [41] previously studied was proven to be successful on objects containing little texture. Although we adapted the procedure to execute faster via parallel operations on the GPU there are two main factors preventing it to be more generally useful in an object-oriented SLAM setting, where we expect to recognise hundreds of objects in real-time.

The first limiting factor has to do with the fact that the global description is model-specific, requiring a separate data structure for each different object type and therefore at runtime one has to sequentially test all the global descriptors. Scalability is therefore linear in the number of object classes and as each detection typically takes between 5-8 ms we are limited to describing up to 5 classes to maintain real-time execution speed without dropping camera frames ($< 33$ ms).

Another limitation is the underlying algorithmic complexity of $O(n^2)$ in the number of pixels to evaluate the point-pair features on. Our current strategy is to randomly select a much smaller number of reference points (typically 20% or about 60K pixels on a VGA depth map) in order to maintain real-time speeds. Since the selected reference points become the pivot points at which pose peaks are estimated, incrementing this proportion is beneficial to achieve higher detection rates. At the extreme, evaluating every pixel densely would allow us to handle very small

objects, which is currently not possible with the method of Drost *et al.* as it will be prohibitively too costly to execute.

Motivated by the excellent detection and runtime performance of learning-based methods using *random forests* such the work of Lepetit *et al.* [91][90] on object pose estimation, and Shotton *et al.* [141] for human pose recognition, we decided to experiment with this alternative technique. While the early results achieved here are promising, we expect to develop a more comprehensive study in the future to compare against recent developments with similar goals such as the template-based method of Hinterstoisser *et al.* [73] and the coordinate regression method of Brachmann *et al.* [17].

#### Hough Forests for multi-class 3D object recognition in real-time

Following the training data extraction ideas from Lepetit and Fua [89], Shotton *et al.* [141], Fanelli *et al.* [45] and Girshick *et al.* [60], we obtain training data from synthetic renderings of 3D models as they have shown to be sufficiently descriptive for learning-based methods while being cheap to generate using 3D graphics techniques. We collect a database of 3D objects scanned with KinectFusion [112] (see Figure 5.5) and edit them in a 3D modelling tool such that each object is cleanly segmented from its background, and for each we define a coordinate system such that its contact points to the supporting plane lay in the x-z plane.

In order to generate synthetic training data for each 3D model, we render depth maps from vertices of a hemisphere centred on the object. As we will see in the next Section, it is beneficial to render the models with a supporting plane. The hemisphere is created by refining the faces of an icosahedron twice, each time subdividing the faces into four triangles. This creates 162 vertices but we only keep those above the supporting x-z plane (a strategy similar to the template extraction procedure of Hinterstoisser *et al.* [73]). The rendered depth maps are further corrupted by Gaussian noise to better simulate measurement artefacts.

The virtual camera is assumed to be looking towards the object centre $\mathbf{o}$ from one of the hemisphere vertices $\mathbf{p}$ (see Figure 5.6), having a camera pose matrix (in the

Figure 5.5: 3D object database. We generate a collection of 3D objects modelled with KinectFusion and use them to synthetically generate training data from partial views.

object coordinate frame) defined by:

$$\mathbf{T}_{\text{oc}} = \begin{bmatrix} \hat{x}_x & \hat{y}_x & \hat{z}_x & p_x \\ \hat{x}_y & \hat{y}_y & \hat{z}_y & p_y \\ \hat{x}_z & \hat{y}_z & \hat{z}_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} , \tag{5.1}$$

with orthonormal rotational components given by:

$$\hat{\mathbf{z}} = \frac{\mathbf{o} - \mathbf{p}}{\|\mathbf{o} - \mathbf{p}\|} , \tag{5.2}$$

$$\hat{\mathbf{x}} = \frac{\mathbf{d}_{up} \times \mathbf{z}}{\|\mathbf{d}_{up} \times \mathbf{z}\|} , \tag{5.3}$$

$$\hat{\mathbf{y}} = \hat{\mathbf{z}} \times \hat{\mathbf{x}} . \tag{5.4}$$

Here $\mathbf{d}_{\text{up}}$ is the upward direction of the world coordinate frame (we use $\mathbf{d}_{\text{up}} = [0, 1, 0]^\top$). The inverse of the previous matrix is the object pose matrix (in the

camera coordinate frame), defined by:

$$\mathbf{T}_{\text{co}} = \mathbf{T}_{\text{oc}}^{-1} = \begin{bmatrix} \hat{x}_x & \hat{x}_y & \hat{x}_z & -\hat{\mathbf{x}} \cdot \mathbf{p} \\ \hat{y}_x & \hat{y}_y & \hat{y}_z & -\hat{\mathbf{y}} \cdot \mathbf{p} \\ \hat{z}_x & \hat{z}_y & \hat{z}_z & -\hat{\mathbf{z}} \cdot \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{5.5}$$

Rather than extracting patches like Gall and Lempitsky [54], we randomly extract 500 training points per hemisphere vertex for each object class, using each of the generated synthetic depth maps $\mathcal{D}$. We define the following depth-invariant comparison features following Shotton *et al.* [141]:

$$f(\boldsymbol{\theta}, \mathcal{D}) = \mathcal{D}\left(\mathbf{x} + \frac{\mathbf{u}_1}{\mathcal{D}(\mathbf{x})}\right) - \mathcal{D}\left(\mathbf{x} + \frac{\mathbf{u}_2}{\mathcal{D}(\mathbf{x})}\right) , \tag{5.6}$$

$$h(\boldsymbol{\theta}, \mathcal{D}) = \begin{cases} 0 & \text{if } f(\boldsymbol{\theta}, \mathcal{D}) < \tau \\ 1 & \text{otherwise} , \end{cases} \tag{5.7}$$

parametrised by $\boldsymbol{\theta} = (\mathbf{x}, \mathbf{u}_1, \mathbf{u}_2, \tau)$, where $\mathbf{u}_1$ and $\mathbf{u}_2$ are two randomly chosen offsets from a point $\mathbf{x}$ inside the object contour and $\tau$ is a threshold value (see Figure 5.10). We also extract 500 background training points per hemisphere vertex from real scenes not containing the objects of interest.

Similarly to Fanelli *et al.* [45], but now in 3D, we store offsets $\boldsymbol{\delta} \in \mathbb{R}^3$ from every training point position $\mathbf{x} \in \Omega \subset \mathbb{R}^2$ towards the object centre $\mathbf{o}$:

$$\boldsymbol{\delta} = \mathbf{T}_{\text{co}}\mathbf{o} - \mathcal{V}(\mathbf{x}) , \tag{5.8}$$

with $\mathcal{V}(\mathbf{x}) = \mathbf{K}^{-1}\dot{\mathbf{x}}\mathcal{D}(\mathbf{x})$ and $\mathbf{K}$ the camera intrinsic matrix. Additionally, we store the forward direction vector $\hat{\mathbf{z}} \in \mathbb{R}^3$. The offset and direction vectors will be used to predict at test time the position and orientation of the object respectively. We use a normalised direction vector as this will allow the estimation of mean values across a full unit sphere, compared to other unsuitable representations like euler angles due to their circular domain. Our camera pose representation cannot cope with in-plane rotations (*i.e.* roll), however we expect an upright camera setup with minor deviations that can be corrected by the further ICP optimisation.

The sample labels $\boldsymbol{\phi} = (\boldsymbol{\delta}, \hat{\mathbf{z}})$ are modelled as multivariate Gaussian such that $p(\boldsymbol{\phi}|L) = \mathcal{N}(\boldsymbol{\phi}; \bar{\boldsymbol{\phi}}, \boldsymbol{\Sigma})$ with $\boldsymbol{\Sigma}$ the covariance matrix. Like Fanelli *et al.* [45] we allow only covariances among offset vectors or direction vectors but not between them.

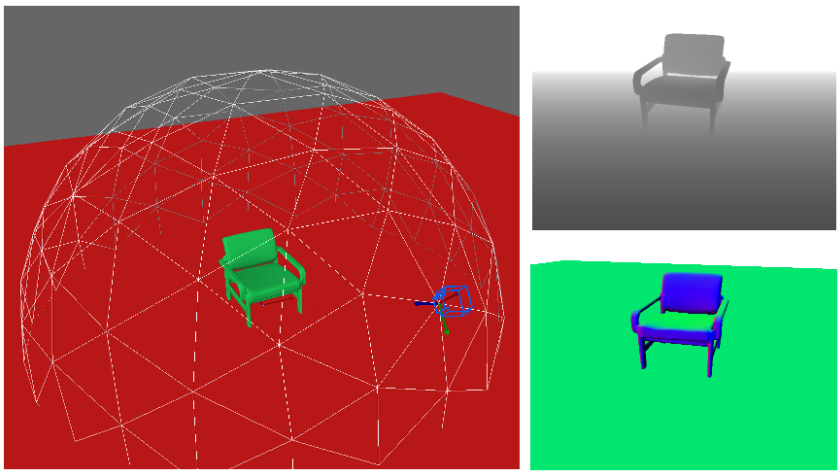


Figure 5.6: Hemisphere used to extract synthetic views of a 3D object **(left)**. A camera positioned at one of the hemisphere vertices looks towards the object centre generating a synthetic depth map **(top-right)** and normal map **(bottom-right)**. Note that we ignore rotations about the view direction (camera is always upright) as the predicted pose is expected to be corrected by ICP to some extent.

At training time, the parameters $\boldsymbol{\theta}$ are randomly sampled and we also randomly choose the test that maximises the classification information gain metric $H_c(S)$ or regression metric $H_r(S)$, except when the frequency of a particular class exceed 95% or when the current tree level $l \geq D - 2$, in which case we always use the regression metric. Training stops when the maximum depth is achieved or the number of samples in a node falls below a threshold (20 samples in our case). Statistics of the samples reaching a leaf are extracted and for each leaf we only store the mean offset vector $\bar{\boldsymbol{\delta}}$, mean direction vector $\bar{\mathbf{z}}$ and trace $\mathrm{tr}(\boldsymbol{\Sigma})$.

At test time we will use $\bar{\boldsymbol{\delta}} + \mathcal{V}(\mathbf{x})$ to predict the object position and the mean forward direction vector $\bar{\mathbf{z}}$ to predict its orientation following Equations 5.1-5.5. We densely sample all pixels from an input depth map and test against each of the forest's trees. This process is easily parallelizable on the GPU and takes approximately 2ms for a VGA image ($640 \times 480$ pixels).

Only those pixels for which the reached leaf satisfies $p(c|L) > 0.75$, and $\mathrm{tr}(\boldsymbol{\Sigma}) < 3$ are used to predict the existence and pose of an object instance. The bottom row of Figure 5.8 provides a visualisation of the dense class prediction. As can be seen, there are sections of class ambiguity, particularly if we observe the backrest of the two chairs on the middle and right side as they have similar geometry. More distinct

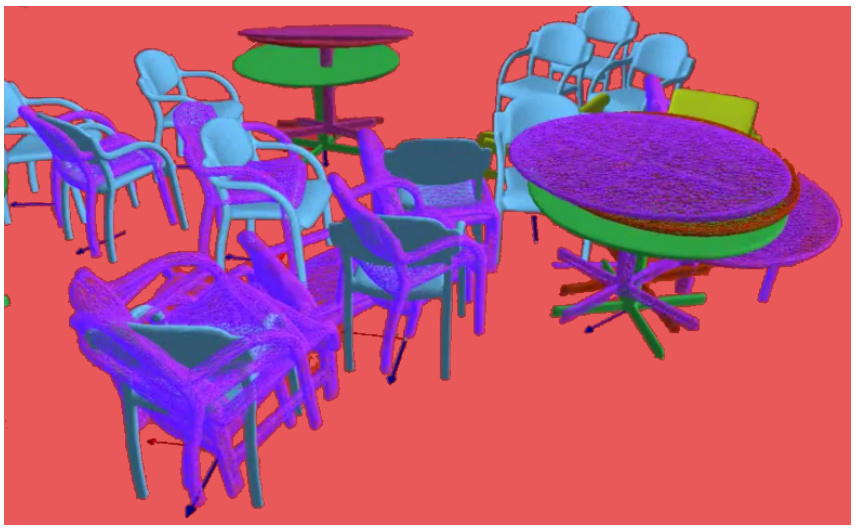


Figure 5.7: Pose clusters for several objects detected in a synthetic scene. Solid objects represent the ground truth.

areas such as the lower support on the yellow chair class help to disambiguate the two objects and this highlights the benefit of using dense predictions.

For the pixels passing the previous probability and variance thresholds we execute clustering using the CPU via *mean shift* [29] in order to obtain object instances. First, we randomly pick 5000 possible centres and assign the remaining pixels to each cluster according to a compatibility criterion: translation $< 10$cm, rotation $< 15°$, and re-estimate cluster centres and memberships for up to 10 iterations or when the centre moves below a threshold. Finally we discard clusters having less than 300 members. Figure 5.7 shows the cluster prediction for several objects in a synthetic scene, while the top-row of Figure 5.8 shows the cluster with the maximum number of members for single objects.

The generated clusters that are close to each other by less than half of the object diameter are sorted by decreasing number of members. This is followed by the execution of dense ICP [133], starting from the first element in the sorted list, in order to complete the alignment of the estimated pose to the measured depth map. We declare an object as detected if the number of correspondences is above 70% of the predicted rendered pixels and stop applying ICP to the remaining clusters. Figure 5.9 shows detected objects after ICP alignment on a real scene, alongside a visualisation for class label prediction.

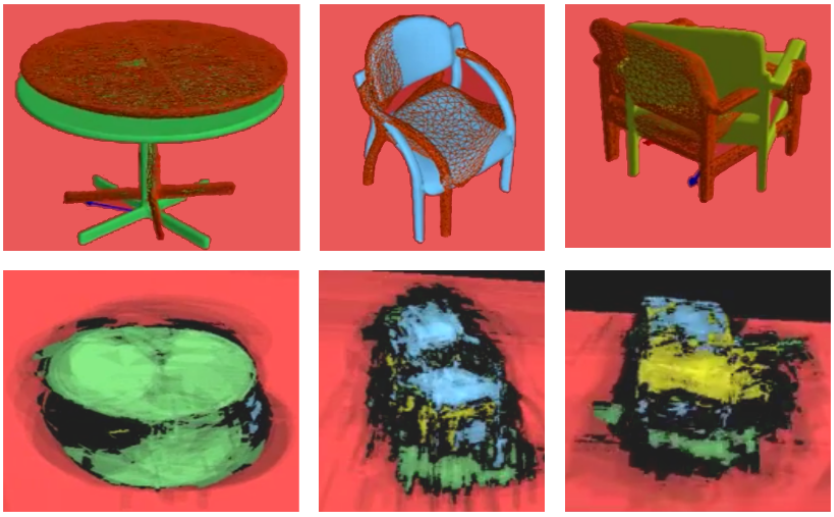


Figure 5.8: **(top-row)** Pose cluster with maximum number of members. Solid objects represent the ground truth. **(bottom-row)** Dense class prediction visualisation. Each pixel is encoded with a colour for the class label with maximum probability (green: table, blue: high chair, yellow: low chair, red: background) and modulated by the probability itself such that brighter colours indicate higher probability.

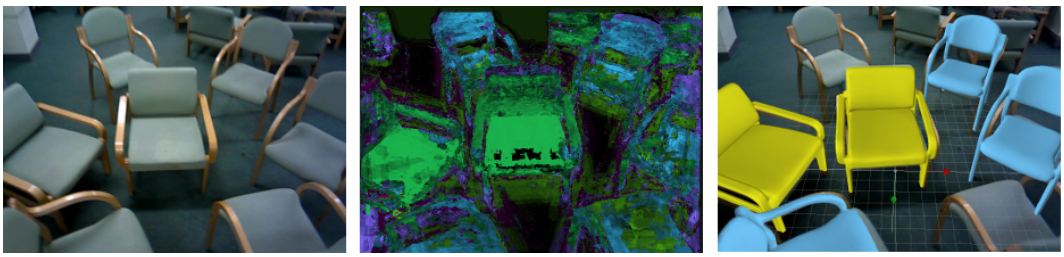


Figure 5.9: Detected objects with multi-class Hough forest after alignment. **(left)** Colour image of the scene, **(middle)** class prediction visualisation, **(right)** detected objects after ICP alignment.

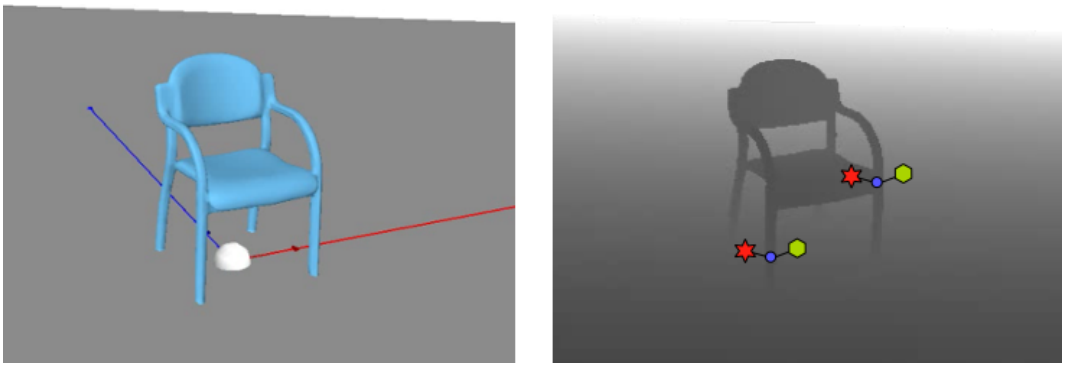


Figure 5.10: **(left)** A chair is rendered on top of a supporting plane to improve recognition performance with the addition of context. **(right)** Rendered depth map with two identical depth comparison feature examples at different places on the object. A probe position (blue) landing on the left leg produces a low depth comparison response based on the offset samples (red and green). A probe position landing on the right side of the chair generates a higher feature response.

### Experiments

In what follows we describe experimenting with different settings to achieve good performance under varying noise levels. To measure the recognition performance we generated 10 synthetic scenes like the one shown in Figure 5.7 and corrupted them with Gaussian noise. All reported results are obtained after aligning objects with ICP and we consider an object to be recognised if its difference in position is less than 1/10 of the object diameter and its orientation differs by less than 15°.

**Using supporting regions:** Contextual information has been shown to improve recognition performance in tasks such as scene labelling [143][145]. Here we evaluate the value of using a supporting plane where an object can rest. The synthetic images generated with a plane now contain pixels with depth information surrounding the target object as can be seen in Figure 5.10. Recognition results when ignoring the supporting plane are shown in Figure 5.11 and when adding it in Figure 5.12. As can be seen, adding the supporting planar region improves performance significantly for certain classes such as big table, short table and bin. This might be explained by the relative lack of features on those objects (large planar sections and vertical symmetry) that are boosted by the addition of contextual information such as the significant depth difference between the top of the table and the floor.

**Distant views:** We experimented with generating views at longer distances by

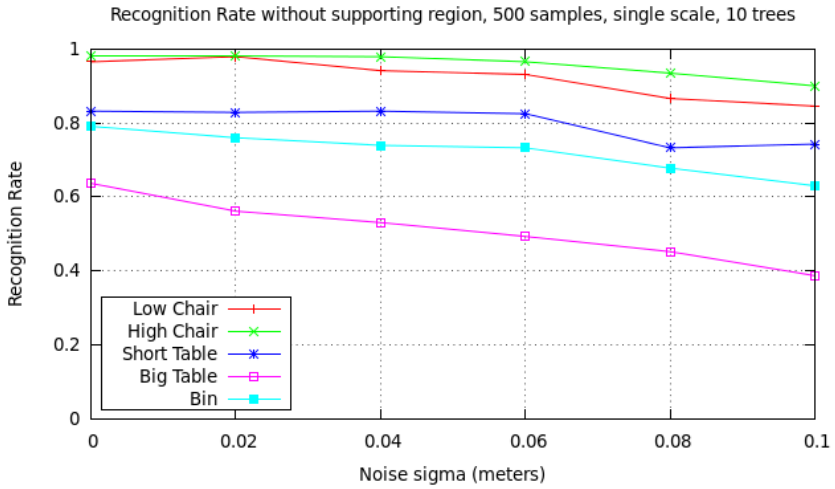



Figure 5.11: Object recognition rate when training without a supporting region, using 500 training samples at a single hemisphere scale and a forest with 10 trees.

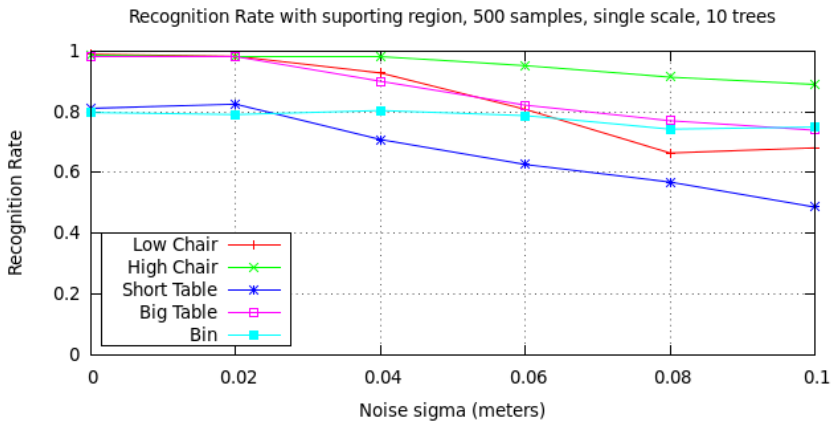



Figure 5.12: Object recognition rate when training with a supporting region, using 500 training samples at a single hemisphere scale and a forest with 10 trees.

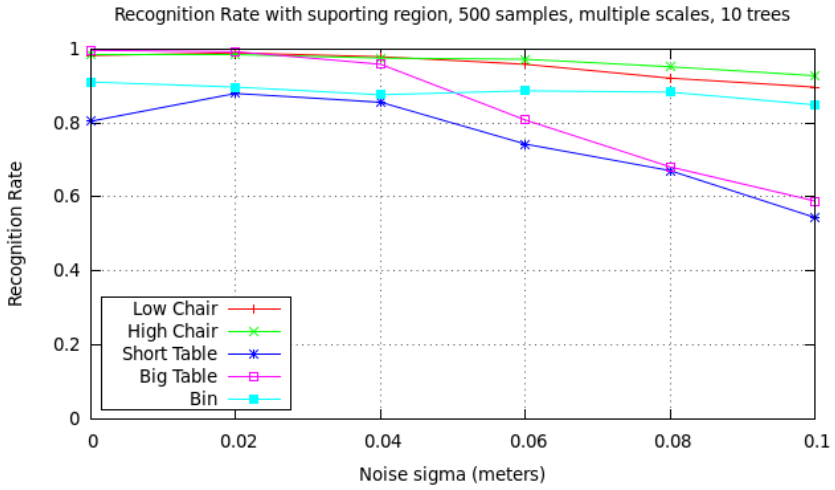



Figure 5.13: Object recognition rate when training with a hemisphere at different scales, using a supporting plane, 500 training samples and a forest with 10 trees.

scaling the icosahedron to 2 and 3 times its original size, similar to the work by Hinterstoisser *et al.* [73]. The results of this experiment are shown in Figure 5.13 and we compare this against Figure 5.12. Adding multiple scales does not improve performance by much, as having depth invariant features seem to negate the need for multiscale sampling. The addition of samples at multiple scales also results in a substantial increase in training time of about 10 hours compared to 2 hours for a single scale.

**Increasing sample size:** Using more samples for training improves recognition performance consistently across all classes. Comparing the previous Figure 5.12 against Figure 5.14 we can see an approximate 10% improvement in recognition rate when increasing the total number of samples for a class from 500 to 2000. However training time also increases in a similar proportion, taking approximately 7.5 hours to complete.

**Reducing the number of trees:** While keeping a large sample number of 2000 we experimented with reducing the number of trees from 10 to 3 in an effort to speed up training time. Surprisingly recognition performance is not affected much, except for the short table object as seen in Figure 5.15 compared to Figure 5.14. A possible explanation for the lower performance of the short table object might be due to the reduced discriminative power of the trees as this class shares many features with the low chair object since their horizontal planes are of the same height.

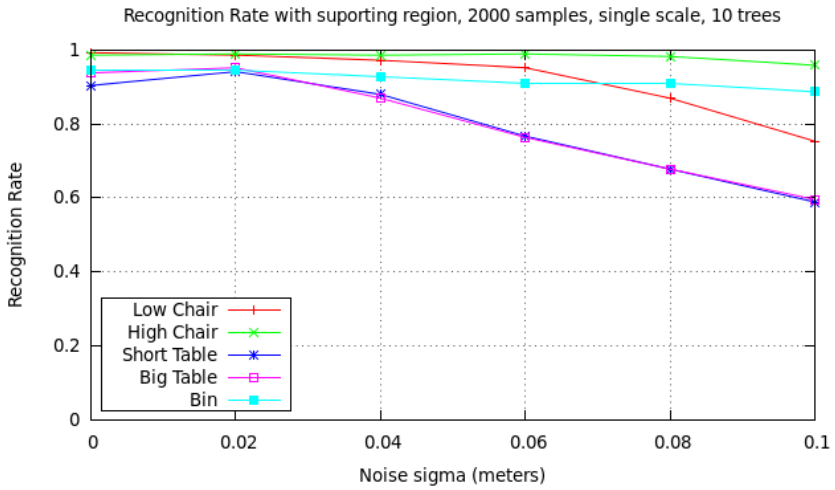



Figure 5.14: Object recognition rate when increasing the number of training samples, using 2000 samples instead of 500, a supporting plane and a forest with 10 trees.

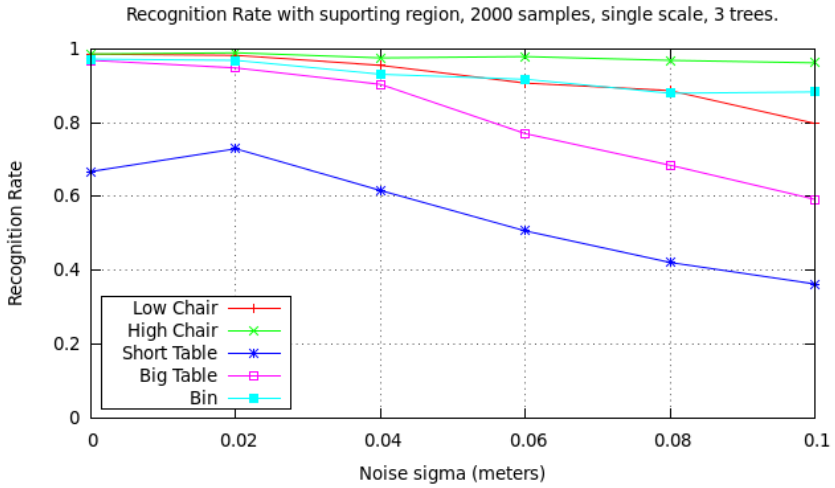



Figure 5.15: Object recognition rate when reducing the number of trees in the random forest from 10 to 3, using a supporting plane, 2000 samples and a single hemisphere scale.

### Hough forests statistics

Table 5.1 reports various performance indicators of the presented multi-class Hough forest method and compares its execution speed to our GPU accelerated method of Drost *et al.* presented in Section 5.2.4. This was executed on a high-performance laptop equipped with an Intel i7 Quad Core CPU at 2.50GHz and NVidia GTX 580M GPU with 2GB of memory. The timing results are for the object recognition without further ICP optimisations, which can take an additional 8 ms.

Training of the forest is done on the CPU with OpenMP, with one thread per tree, while testing is done entirely on the GPU using CUDA, where each individual pixel is passed through the forest in parallel. Clustering is performed on the CPU and accelerated with OpenMP wherever possible.

Table 5.1: Multi-class Hough forests statistics

| | |
|---|---:|
| Class count (excl. background) | 5 |
| Training samples per class | 2000 |
| Number of trees | 10 |
| Maximum tree depth | 15 |
| Training time | 7.5 hours |
| Test time on VGA image | 2.2 ms |
| Clustering time | 6.3 ms |
| Total Hough Forest time | 8.5 ms |
| Drost *et al.* time per object | 5 ms |
| Total Drost *et al.* time | 25 ms |

### Multi-class Hough forests discussion and future work

We have seen in the previous section how the extension of the Hough forest method for multi-class object recognition and pose estimation allowed us to speed up detection performance compared to shape matching methods like Drost *et al.* even after being accelerated on the GPU. As highlighted by Lepetit *et al.* [91], treating matching as a classification problem has the benefit of moving most of the detection complexity to training time, and the use of simple comparison features does not require time consuming filter preprocessing.

The downside of the method is the high training time of several hours to achieve good performance which might preclude the use of the method on exploratory SLAM scenarios where a only new objects are available.

One way to accelerate training time could be the use of completely random tests at the split nodes without resorting to optimising the information gain metric. This method of training is known as Extremely Randomised Trees [59] and it was similarly explored by Lepetit *et al.* [89] who reported only a small loss of reliability but a considerable speed-up in training time.

Similarly, Ozuysal *et al.* [122] noted that when the tests are chosen at random, the classification rates do not depend on the tree structure itself but on the combination of several binary tests and thus introduced simpler structures know as *Ferns* that are much simpler to train and test, at the expense of a slight decreasing in discrimination performance.

In the future we plan to experiment with the previously described methods for real-time object training and recognition. In addition, to improve the experimental setup and provide more standard performance metrics we need to develop a dataset of real-scenes and appropriate ground thruth. Some recent efforts such as Hinterstoisser *et al.* [73] and Brachmann *et al.* [17] provide such datasets and we hope to make use of them to better assess this effort. We will also explore the addition of colour features which can help to increase robustness and to recognise objects in the absence of active depth sensing such as in mobile devices.

Many objects of interest are also non-rigid and this will present a challenge to the previously studied methods. One way to cope with such cases would be to directly regress object coordinates densely as shown by Taylor *et al.* [153] and deform the sought object to fit the measured data.

### 5.2.5 Camera Tracking and Accurate Object Pose Estimation using ICP

**Camera-Model Tracking:** In KinectFusion [112], the up-to-date volumetric map is leveraged to compute a view prediction into the previously estimated camera pose, enabling estimation of the live camera using a fast dense iterative closest point (ICP) algorithm [133]. In SLAM++, in contrast to the relatively incomplete models available at early stages of mapping with KinectFusion, we track against a complete high quality multi-object model prediction. Following [112], we compute a reference view prediction of the current scene geometry consisting of a depth map $\mathcal{D}_\mathrm{r}$ and normal map $\mathcal{N}_\mathrm{r}$ rendered into the previously estimated frames pose $\mathbf{T}_\mathrm{wr}$ representing

the 6 DoF rigid body transform defined as a member of the special Euclidean group $\mathbb{SE}(3)$. We update the live camera to world transform $\mathbf{T}_{\text{wl}}$ by estimating a sequence of $m$ incremental updates $\{\tilde{\mathbf{T}}^n_{rl}\}_{n=1}^m$ parametrised with a vector $\mathbf{x} \in \mathbb{R}^6$ defining a twist in $\mathbb{SE}(3)$, with $\tilde{\mathbf{T}}^{n=0}_{\text{rl}}$ as the identity. We iteratively minimise the whole depth image point-plane metric over all available valid pixels $u \in \Omega$ in the live depth map:

$$E_c(\mathbf{x}) = \sum_{u \in \Omega} \psi\left(e(u, \mathbf{x})\right) \ , \tag{5.9}$$

$$e(u, \mathbf{x}) = \mathcal{N}_{\text{r}}(u')^\top \left(\exp(\mathbf{x})\hat{V}_{\text{l}}(u) - V_{\text{r}}(u')\right) \ . \tag{5.10}$$

Here $V_{\text{r}}(u')$ and $\mathcal{N}_{\text{r}}(u')$ are the projectively data associated predicted vertex and normal estimated at a pixel correspondence $u' = \pi(\mathbf{K}\hat{V}_{\text{l}}(u))$, computed by projecting the vertex $V_{\text{l}}(u)$ at pixel $u$ from the live depth map into the reference frame with camera intrinsic matrix $\mathbf{K}$ and standard pin-hole projection function $\pi$. The current live vertex is transformed into the reference frame using the current incremental transform $\tilde{\mathbf{T}}^n_{\text{rl}}$:

$$\hat{V}_{\text{l}}(u) = \tilde{\mathbf{T}}^n_{\text{rl}} V_{\text{l}}(u) \ , \tag{5.11}$$

$$V_{\text{l}}(u) = \mathbf{K}^{-1} \dot{u} \mathcal{D}_{\text{l}}(u) \ , \tag{5.12}$$

$$V_{\text{r}}(u') = \mathbf{K}^{-1} \dot{u}' \mathcal{D}_{\text{r}}(u') \ . \tag{5.13}$$

We chose $\psi$ as a robust Huber penalty function in place of the explicit point compatibility check used in [112] which enables a soft outlier down weighting. A Gauss-Newton based gradient descent on Equation (5.9) results in solution of the normal equations:

$$\sum_{u \in \Omega} \mathbf{J}(u)^\top \mathbf{J}(u)\mathbf{x} = \sum_{u \in \Omega} \psi'(e)\mathbf{J}(u) \ , \tag{5.14}$$

where $\mathbf{J}(u) = \frac{\partial e(\mathbf{x}, u)}{\partial \mathbf{x}}$ is the Jacobian, and $\psi'$ computes the robust penalty function derivative given the currently estimated error. We solve the $6 \times 6$ linear system using Cholesky decomposition. Taking the solution vector $\mathbf{x}$ to an element in $\mathbb{SE}(3)$ via the exponential map, we compose the computed incremental transform at iteration $n+1$ onto the previous estimated transform $\tilde{\mathbf{T}}^n_{\text{rl}}$:

$$\tilde{\mathbf{T}}^{n+1}_{\text{rl}} \leftarrow \exp(\mathbf{x})\tilde{\mathbf{T}}^n_{\text{rl}} \ . \tag{5.15}$$

The estimated live camera pose $\mathbf{T}_{\mathrm{wl}}$ therefore results by composing the final incremental transform $\tilde{\mathbf{T}}_{\mathrm{rl}}^{m}$ onto the previous frame pose:

$$\mathbf{T}_{\mathrm{wl}} \leftarrow \mathbf{T}_{\mathrm{wr}} \tilde{\mathbf{T}}_{\mathrm{rl}}^{m} \ . \tag{5.16}$$

**Tracking Convergence:** We use a maximum of $m = 10$ iterations and check for poor convergence of the optimisation process using two simple criteria. First, we do not attempt to track against the predicted model if its pixel coverage is less than $\frac{1}{8}$ of a full image. Second, after an optimisation iteration has completed we compute the ratio of pixels in the live image which have been correctly matched with the predicted model ascertained by discounting pixels which induce a point-plane error greater than a specified magnitude $\epsilon_{\mathrm{pp}}$.

**Tracking for Model Initialisation:** We utilise the dense ICP pose estimation and convergence check for two further key components in SLAM++. First, we note that the real-world objects we use as features are often ambiguous, and not well discriminated given a single view. Therefore, given a candidate object and detected pose, we run camera-model ICP estimation on the *detected* object pose, and check for convergence using the previously described criteria. We find that for correctly detected objects, the pose estimates from the detector are erroneous within $\pm 30^{\circ}$ rotation, and $\pm$50cm translation. This allows a more conservatively set threshold $\epsilon_{\mathrm{oi}}$ and early rejection of incorrect objects.

**Camera-Object Pose Constraints:** Given the active set of objects that have been detected in SLAM++, we further estimate relative camera-object pose parameters which are used to induce constraints in the scene pose graph. To that end, we run the dense ICP estimate between the live frame and *each* model object currently visible in the frame. The ability to compute an individual relative pose estimate introduces the possibility to prune poorly initialised or incorrectly tracked objects from the pose graph at a later date. By analysing the statistics of the camera-object pose estimate's convergence we can keep an inlier-outlier style count on the inserted objects, and cull poorly performing ones.

### 5.2.6 Graph Optimisation

We formulate the problem of estimating the historical poses of the depth camera $\mathbf{T}_{\mathrm{w}i}$ at time $i$ and the poses of the static objects $\mathbf{T}_{\mathrm{wo}_j}$ as graph optimisation. $Z_{i,\mathrm{o}_j}$ denotes the 6 DoF measurement of object $j$ in frame $i$ and $\mathbf{\Sigma}_{i,\mathrm{o}_j}^{-1}$ its inverse measurement

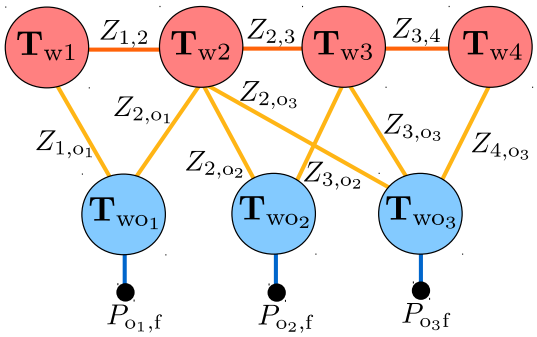



Figure 5.16: Example graph illustrating the pose of the moving camera over four time steps $\mathbf{T}_{\mathrm{w}i}$ (red) as well as the poses of three static objects in the world $\mathbf{T}_{\mathrm{wo}_j}$ (blue). Observations of the object $\mathrm{o}_j$ at time $i$ are shown as binary camera-object constraints $Z_{i,\mathrm{o}_j}$ (yellow) while the relative ICP constraints between two cameras are shown as $Z_{i,i+1}$ (orange). We also apply unary structural constraints $P_{\mathrm{o}_1,f}$, $P_{\mathrm{o}_2,f}$ and $P_{\mathrm{o}_3,f}$ encoding prior information.

covariance which can be estimated using the approximated Hessian $\mathbf{\Sigma}^{-1}_{i,\mathrm{o}_j} = \mathbf{J}^\top\mathbf{J}$ (with $\mathbf{J}$ being the Jacobian from the final iteration of the object ICP). $Z_{i,i+1}$ is the relative ICP constraint between camera $i$ and $i+1$, with $\mathbf{\Sigma}^{-1}_{i,i+1}$ the corresponding inverse covariance. The absolute poses $\mathbf{T}_{\mathrm{w}i}$ and $\mathbf{T}_{\mathrm{wo}_j}$ are *variables* which are modified during the optimisation, while $Z_{i,\mathrm{o}_j}$ and $Z_{i,i+1}$ are measurements and therefore *constants*. All variables and measurements have 6 DoF and are represented as members of $\mathbb{SE}(3)$. An example graph is shown in Figure 5.16.

We minimize the sum over all measurement constraints:

$$\begin{aligned} E_{\mathrm{m}} \quad = \quad & \sum_{Z_{i,\mathrm{o}_j}} ||\log(Z^{-1}_{i,\mathrm{o}_j} \cdot \mathbf{T}^{-1}_{\mathrm{w}i} \cdot \mathbf{T}_{\mathrm{wo}_j})||_{\Sigma_{i,\mathrm{o}_j}} \\ & + \sum_{Z_{i,i+1}} ||\log(Z^{-1}_{i,i+1} \cdot \mathbf{T}^{-1}_{\mathrm{w}i} \cdot \mathbf{T}_{\mathrm{w}i+1})||_{\Sigma_{i,i+1}} \ , \end{aligned} \tag{5.17}$$

with $||\mathbf{x}||_\Sigma := \mathbf{x}^\top\mathbf{\Sigma}^{-1}\mathbf{x}$ the Mahalanobis distance and $\log(\cdot)$ the logarithmic map of $\mathbb{SE}(3)$. This generalised least squares problem is solved using Levenberg-Marquardt; the underlying normal equation's sparsity is exploited using a sparse Cholesky solver [81]. The pose Jacobians are of the form $\pm\frac{\partial}{\partial\epsilon_i}\log(A\cdot\exp(\epsilon)\cdot B)|_{\epsilon=0}$, approximated using higher order Baker-Campell-Haussdorf expansions [150].

### Including Structural Priors

Additional information can be incorporated in the graph in order to improve the robustness and accuracy of the optimisation problem. In our current implementation

we apply a structural planar prior that the objects are located on a common ground plane. The world reference frame w is defined such that the $x$ and $z$-axes lie within the ground plane with the $y$-axis perpendicular into it. The ground plane is implicitly detected from the initial observation $Z_{1,\mathrm{o}_1}$ of the first object; its pose $\mathbf{T}_{\mathrm{wf}}$ remains fixed during optimisation. To penalize divergence of the objects from the ground plane, we augment the energy,

$$E_{\mathrm{m\&p}} = E_{\mathrm{m}} + \sum_{P_{\mathrm{o}_j,\mathrm{f}}} || \log(P_{\mathrm{o}_j,f}^{-1} \cdot \mathbf{T}_{\mathrm{wo}_j}^{-1} \cdot \mathbf{T}_{\mathrm{wf}}) ||_{\Sigma_{o_j,f}} \ , \tag{5.18}$$

using a set of unary constraints $P_{\mathrm{o}_j,\mathrm{f}} = \exp((\boldsymbol{\upsilon}^\top, \boldsymbol{\theta}^\top)^\top)$ with $\boldsymbol{\upsilon}, \boldsymbol{\theta} \in \mathbb{R}^3$ (see Figure 5.16). In particular, we set the translation along the y-axis, $\upsilon_2 = 0$, as well as the rotational components about the $x$ and $z$-axis to zero, $\theta_1 = 0$ and $\theta_3 = 0$. We use the following inverse prior covariances:

$$\boldsymbol{\Sigma}_{\mathrm{o}_j,\mathrm{f}}^{-1} = \mathrm{diag}(0, w_{\mathrm{trans}}, 0, w_{\mathrm{rot}}, 0, w_{\mathrm{rot}}) \ , \tag{5.19}$$

with positive weights $w_{\mathrm{trans}}$ and $w_{\mathrm{rot}}$. Since the objects can be located anywhere within the $x-z$ plane and are not constrained in their rotation about the $y$-axis, the corresponding weights are set to zero; the prior components $\upsilon_1, \upsilon_3, \theta_2$ do not affect the energy and can be set to arbitrary values.

### 5.2.7 Other Priors

The ground plane constraint can have value beyond the pose graph. We could guide point-pair feature sampling and make votes for poses only in the unconstrained degrees of freedom of objects. In the ICP method, again a prior could be used as part of the energy minimisation procedure to pull it always towards scene-consistent poses. While this is not yet implemented we at least cull hypotheses of object positions far from the ground plane. Additional prior constraints such as object intersection and affordances would be interesting to explore.

### 5.2.8 Relocalisation

When camera tracking is lost the system enters a relocalisation mode. Here a new local graph is created and tracked from, and when it contains at least 3 objects it is matched against the previously tracked long-term graph (see Figure 5.17). Graph matching is achieved by considering both the local and long-term graphs as sets of oriented points in a mesh that are fed into the same recognition procedure described

in Section 5.2.3. We use the position of the objects as vertices and their $x$-axes as normals. The matched vertex with highest vote in the long-term graph is used instead of the currently observed vertex in the local graph and camera tracking is resumed from it, discarding the local map.

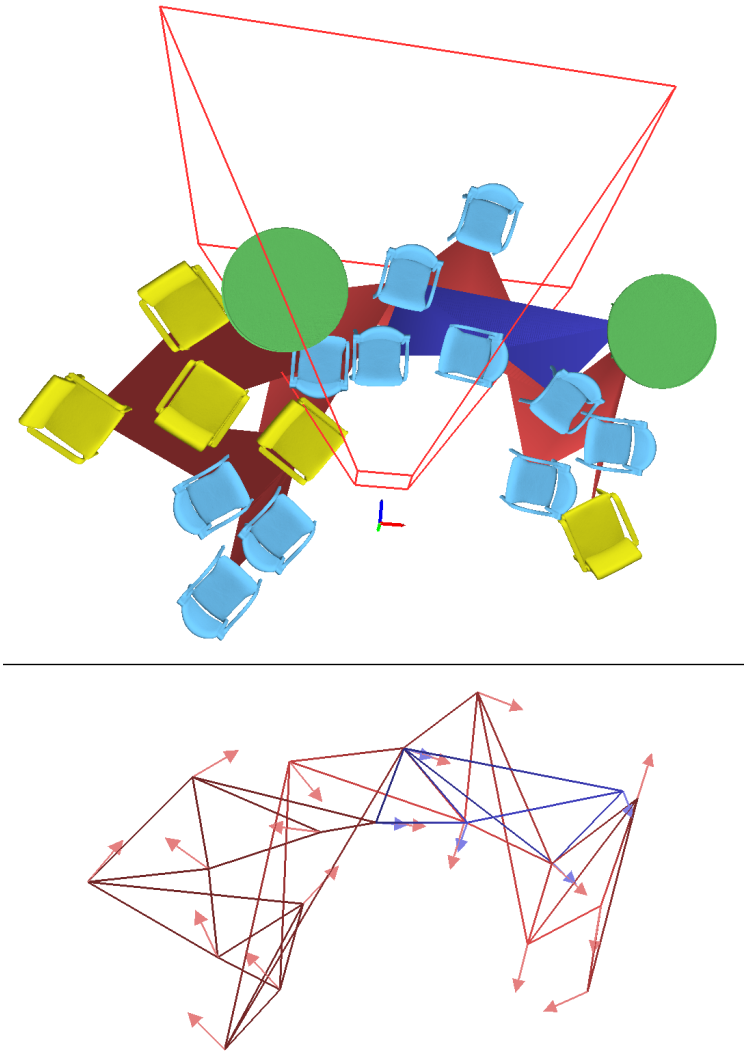


Figure 5.17: Relocalisation procedure. When tracking is lost a local graph (blue) is created and matched against a long-term graph (red). **(top)** Scene with objects and camera frustum when tracking is resumed a few frames after relocalisation. **(bottom)** Oriented points extracted for matching. The connectivity depicts the detection sequence and is not used by the recognition procedure.

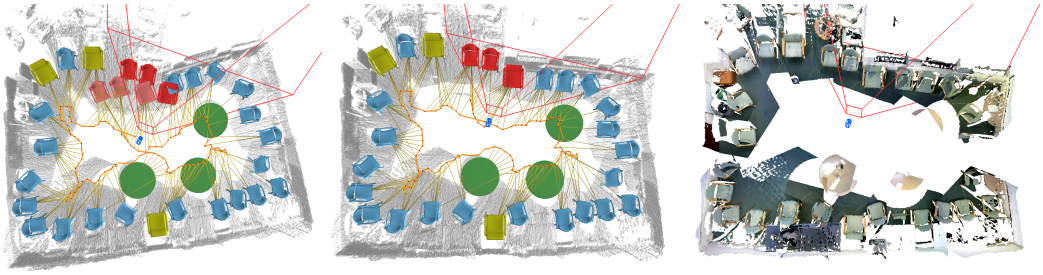


Figure 5.18: Loop closure. **(left)** Open loop drift during exploration of a room; the two sets of objects shown in red were identified as corresponding. The full SLAM++ graph is displayed with yellow lines for camera-object constraints and orange lines for camera-camera constraints. **(middle)** Imposing the new correspondences and re-optimising the graph closes the loop and yields a more metric map. **(right)** For visualisation purposes only (since raw scans are not normally saved in SLAM++), we show a coloured point cloud after loop closure.

## 5.3 Results

The in-the-loop operation of our system is more effectively demonstrated by watching the SLAM++ videos in Appendix B, where the advantages of our method over off-line scene labelling will be more obvious. We present demonstrations of a new level of real-time localisation and mapping performance, surpassing previous SLAM systems in the quality and density of geometry description obtained given the very small footprint of our representation; and rivalling off-line multi-view scene labelling systems in terms of object identification and configuration description.

### 5.3.1 Loop Closure

Loop closure in SLAM occurs when a location is revisited after a period of neglect, and the arising drift corrected. In SLAM++, small loops are regularly closed using the standard ICP tracking mechanism. Larger loop closures (see Figure 5.18), where the drift is too much to enable matching via predictive ICP, are detected in real-time using a module based on matching fragments within the main long-term graph in the same manner as in relocalisation (Section 5.2.8).

### 5.3.2 Large Scale Mapping

We have run SLAM++ in environments including the large common room shown in Figure 5.1 (size 15×10×3m), with two types of chair and two types of round table, all constrained to a common ground plane. The real-time process lasted around

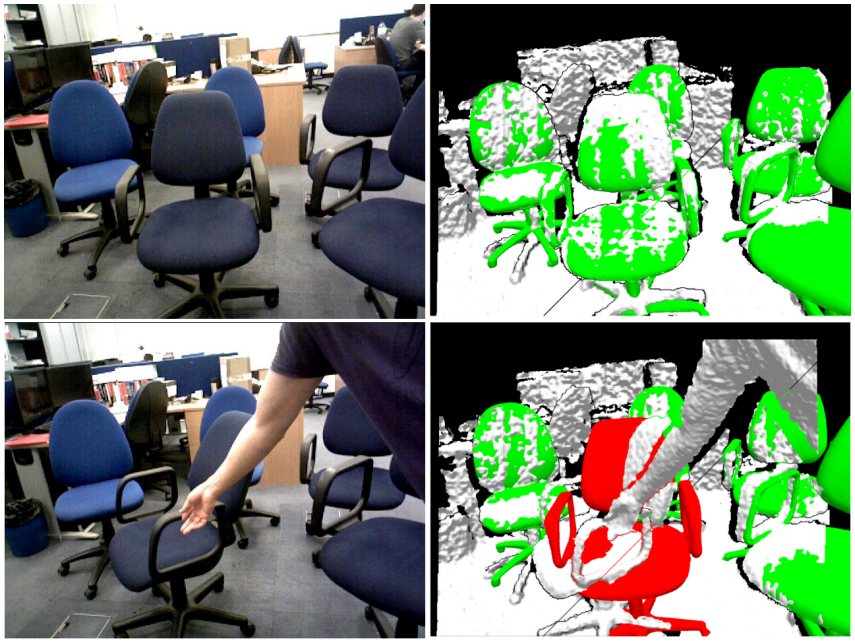


Figure 5.19: When a mapped object is largely inconsistent with good measurements (green) it is marked as invalid (red) and observations from it are stopped to avoid corrupting the rest of the graph.

10 minutes, including various loop closures and relocalisations due to lost tracking. 34 objects were mapped across the room. Figure 5.1 gives a good idea of mapping performance. Note that there are no priors in the system concerning the regular positioning of tables and chairs, only that they sit on the ground plane.

### 5.3.3 Moved Object Detection

We demonstrate the ability to detect the movement of objects, which fail ICP gating due to inconsistency (Figure 5.19).

### 5.3.4 Augmented Reality with Objects

Finally, the ability to semantically predict complete surface geometry from partial views allows novel context-aware AR capabilities such as path finding, to gracefully avoid obstacles while reaching target objects. We apply this to command virtual characters to navigate the scene and find places to sit as soon as the system is started (without the need to scan a whole room). (Figure 5.20).

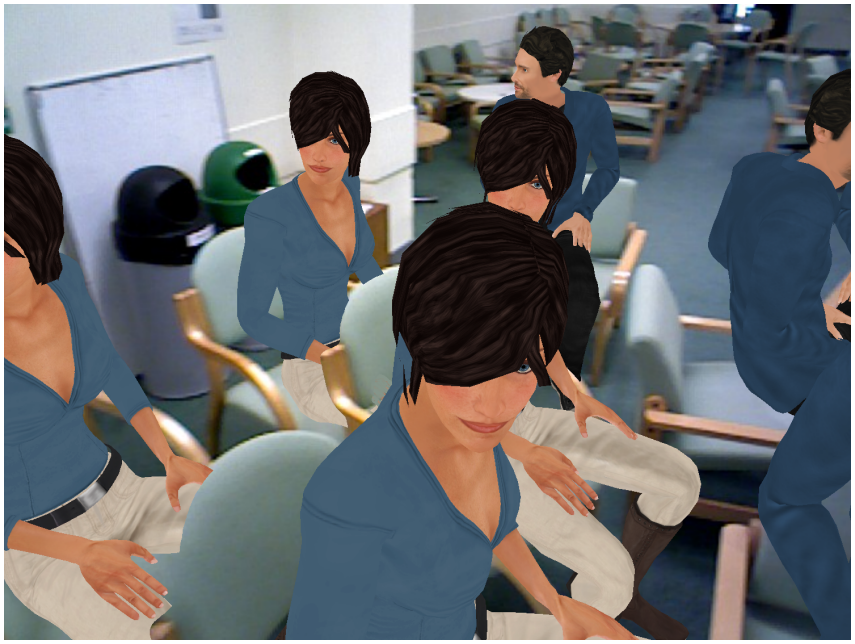


Figure 5.20: Context-aware augmented reality: virtual characters navigate the mapped scene and automatically find sitting places.

### 5.3.5 System statistics

Table 5.2 summarises the results when mapping the room shown in Figure 5.18 (10×6×3m) using a gaming laptop. We compare the memory footprint of SLAM++ with KinectFusion [112] for the same volume (assuming 4 Bytes/voxel, 128 voxels/m).

Table 5.2: System statistics for mapping a large room

| | |
|---|---|
| Framerate | 20 fps |
| Camera Count | 132 |
| Object Count | 35 |
| Object Class Count | 5 |
| Edge Count | 338 |
| Graph Memory | 350 KB |
| Database Memory | 20 MB |
| KinectFusion Memory | 1.4 GB |
| Approx. Compression Ratio | 1/70 |

## 5.4 Conclusions and Future Work

We have shown that using high performance 3D object recognition in the loop permits a new approach to real-time SLAM with large advantages in terms of efficient and semantic scene description. In particular we demonstrate how the tight interaction of recognition, mapping and tracking elements is mutually beneficial to all. Currently our approach is well suited to locations like the interiors of public buildings with many repeated, identical elements, but we believe is the first step on a path to more generic SLAM methods which take advantage of objects with low-dimensional shape variability or in the long term which can segment and define their own object classes. Some non-object areas such as the floor are implicitly inferred (*e.g.* from the objects it supports) however their precise extent is unknown.

In the next chapter we address some of these limitations as we present an efficient real-time approach to densely maps an environment using bounded planes and surfels. This can take advantage of the planarity of many parts of real-world scenes for data regularisation and compression, which in contrast to SLAM++, is purely data-driven as it does not need a pre-made database and still allows the capture of useful semantic information.

Chapter 6

# Dense Planar SLAM

In augmented reality applications, the goals of SLAM systems which can operate in unknown environments with hand-held or wearable cameras have now clearly progressed beyond pure localisation towards capturing significant information about the scene to enable meaningful automatic annotation. We define a 'dense SLAM' system as one which produces not a point cloud but a closed surface geometry scene model, enabling every-pixel depth prediction and occlusion reasoning. A breakthrough in real-time dense SLAM was provided by the KinectFusion algorithm [112], and it was shown that in AR a dense geometric reconstruction can be annotated in any number of interactive or automatic ways [74]. However, KinectFusion and related methods [77][131][162] use non-parametric representations which are both heavyweight in terms of computing resources and lack semantic description.

In the previous chapter, we attacked both of these weaknesses by detecting instances of known 3D objects in the live image stream from a hand-held depth camera and creating a highly efficient object-level map consisting solely of the objects' configuration. The estimated object configuration is used to generate a dense surface prediction for accurate and robust camera pose tracking using ICP as in KinectFusion; and the known objects can easily be used as the basis for content-aware AR effects. However, SLAM++ relies on a database of specific known objects only, and apart from the ability to define and use a ground plane under objects such as chairs and tables, cannot cope with non-object *regions* such as walls, which often have a large extent well beyond the field of view of a camera.

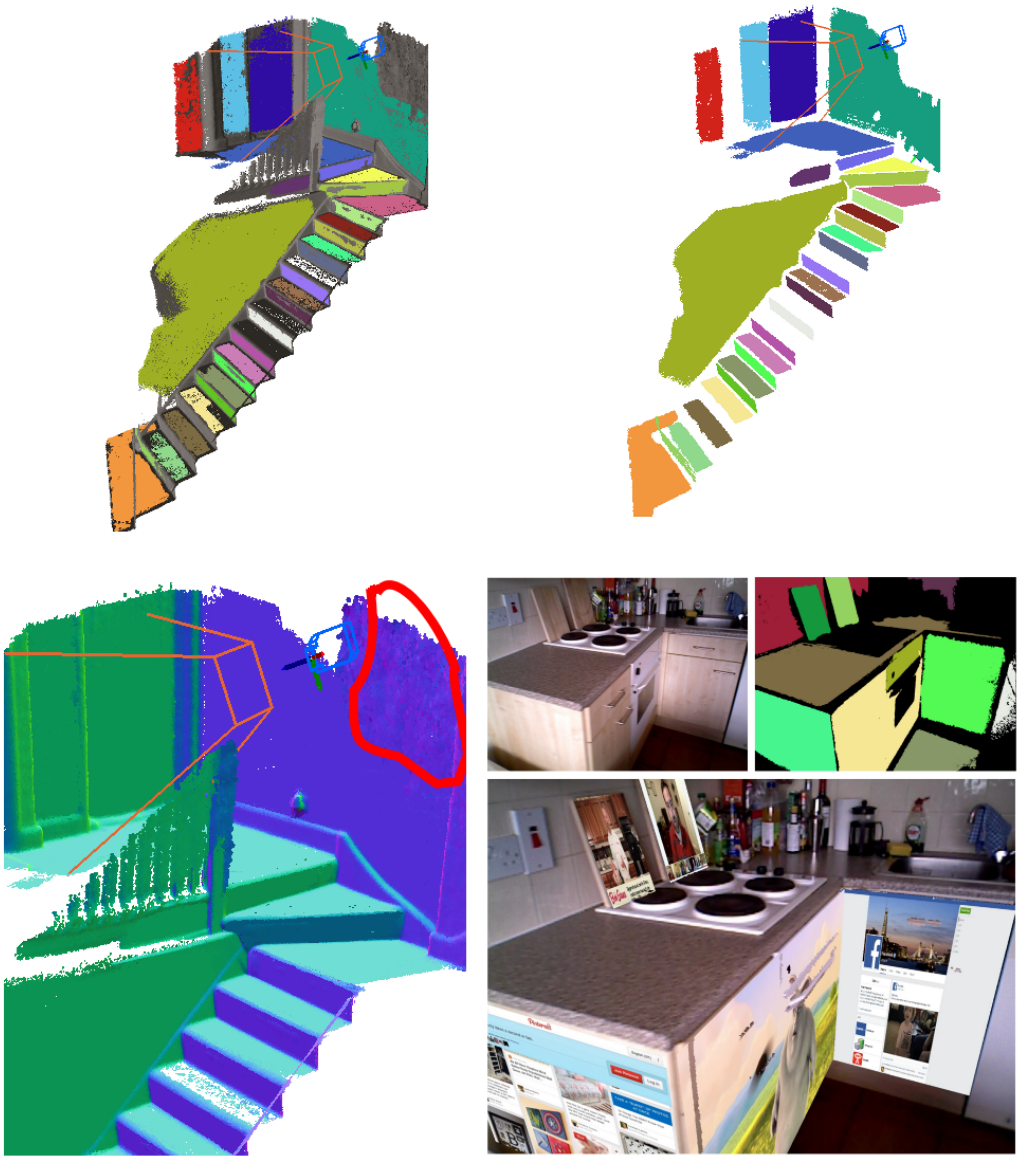


Figure 6.1: Dense Planar SLAM in action. **(top-left)** The stairs of a house have been mapped with both planar and non-planar region surfels. **(top-right)** Planar-only regions. **(bottom-left)** Normal map shows high-quality reconstruction. Observe the lower quality normals on the highlighted red area lacking planar measurements. **(bottom-right)** Some planar regions detected on a kitchen are used to display user's content.

We believe that the crucial measure of the performance of a dense SLAM system is what fraction of the pixels in each new image it is able to explain with its scene model, and that this must be driven up closer to the near 100% that KinectFusion achieves for the efficient and semantic object-based SLAM paradigm to be fully competitive. This requires both the ability to model and use the non-object-like regions of a scene; and to find and add new types of objects which are not present in the prior database. In man-made scenes, both of these requirements strongly motivate the capability to discover and model significant planar scene structures. Planes are extremely common and often occupy large fractions of the field of view of typical images from indoor scenes. Mapping them explains these pixels, potentially with great efficiency; and crucially if all planes in a scene can be mapped then the regions which are *not* planar are relatively few and are often clearly segmented against the planar surfaces which surround them.

In this chapter we focus on the detection and modelling of accurate, bounded planar regions with arbitrary boundary shape. These are extracted and refined over time to form a real-time dense planar SLAM system using depth images such as those produced by RGB-D sensors or via dense multi-view stereo reconstruction methods [113]. While there are many planes in most man-made scenes, mapping them is not as simple as identifying these and instantiating infinite planes in a map. Rather, accurately representing the shape and extent of planes is crucial, and for us this is what defines 'Dense Planar SLAM'. Planar regions will often have irregular boundaries, or holes (when an object is on a table top, or hangs on a wall, for instance). Starting from a surfel map generated in real-time using the point fusion method of [77], we show how planar regions at arbitrary orientations can be segmented and incrementally grown over time. We introduce an efficient representation of the accurate extent of 3D planar regions using a 2D occupancy map approach. This representation is incrementally extensible so that large planar regions can be grown, refined and joined over long observation periods. Our representation also allows on the fly compression and efficient use of memory and processing resources, and we particularly show how it is amenable to parallel GPU-Compute implementation.

While this work was first motivated by high level goals in object aware SLAM, we show that dense planar SLAM alone is very interesting and practical for a number of novel AR applications; in particular the use of walls or other real-world surfaces for the display of information, which will be particularly attractive when used with

see-through AR headsets in the near future.

## 6.1 Related Work

Our work relates to previous real-time and off-line SLAM methods which have attempted to efficiently map scenes using planar assumptions. In the broader context, it relates to the literature on augmenting 3D reconstructions with semantic meaning. Our approach, benefits from the simplification, efficiency and predictive power of semantic model-fitting *in the loop* of real-time operation, a principle that also underpins our previous SLAM++ work [135] (see Chapter 5), which recognises instances of objects from a pre-scanned library and directly builds a map at the object level. This is in contrast to numerous approaches which consider reconstruction and semantic labelling as processes to be applied one after the other (*e.g.* the sophisticated work of Kim *et al.* [78]). Other work which does jointly perform reconstruction and object fitting, such as that by Bao *et al.* [9], is far from being feasible in real-time operation, unlike [135].

Earlier approaches for real-time SLAM using planes include work from Gee *et al.* [57] and Chekhlov *et al.* [24]. Their systems used planes to replace point features and reduce the state space of estimation, because having large number of points becomes unbearable in Kalman filter-based systems. This was improved by Carranza and Calway [102] who directly mapped using planes and points without initialisation delay.

Dou *et al.* [40] improved indoor 3D reconstruction quality via bundle adjustment method that incorporated planar surface alignment errors in addition to 3D point reprojection errors. Their system however was not aimed at real-time scenarios, taking 3 seconds for plane extraction and a few minutes for global optimisation.

Trevor *et al.* [157] combined a Kinect sensor with a 2D laser range scanner to map both close and distant line and plane features. Taguchi *et al.* [152] performed camera tracking by detecting point and plane features and matching them in the complete global map. However this approach resulted in perceptual aliasing and slow tracking. Most recently Ataer-Cansizoglu *et al.* [6] improves on Taguchi's method by predicting the camera motion via a constant velocity model using optical flow, a condition which is difficult to satisfy with handheld camera scenarios. Our method imposes no such assumption and instead directly tracks via ICP alignment from a

dense model, giving a fine pose update that eases the data-association task.

## 6.2 System Overview

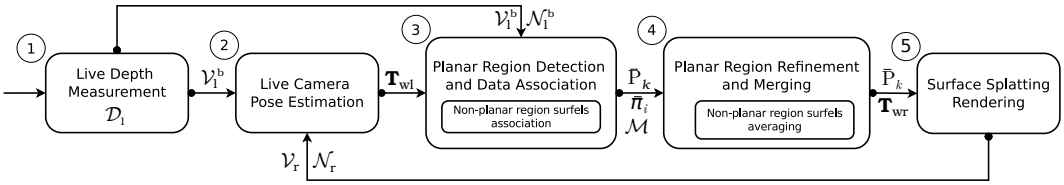



Figure 6.2: Outline of the Dense Planar SLAM pipeline. **(1)** A bilateral filtered depth measurement $\mathcal{D}_l$ is transformed into a metric vertex $\mathcal{V}_l^b$ and normal $\mathcal{N}_l^b$ map and used for both camera pose estimation and plane detection. **(2)** We update the live camera pose $\mathbf{T}_{wl}$ by densely aligning with ICP the measured vertex map $\mathcal{V}_l^b$ against the predicted vertex $\mathcal{V}_r$ and normal map $\mathcal{N}_r$. **(3)** Planes are detected via connected component labelling and incrementally extended with projective data-association. **(4)** Modelled planes $\bar{\pi}_i$ are merged and refined with a running average. **(5)** View prediction is generated by rendering surfels $\bar{\mathbf{P}}_k$ via surface-splatting using the reference pose $\mathbf{T}_{wr}$.

A schematic overview of our system is shown in Figure 6.2. Our starting point is the Point-based Fusion method of Keller *et al.* [77] to densely map the environment with surfels: small disk-shaped entities to describe locally planar regions without connectivity information. Mapping from noisy depth sensors using surfels provides easier management of data-association, insertion, averaging and removal of map entities compared to structured meshes like triangles as well as memory savings compared to voxel-based methods like KinectFusion [112].

In our approach, we aim to label each surfel in the 3D map either with one of a number of discrete plane labels, or to leave it with no label if it is not part of any major plane in the scene. *Planar region surfels* describe large areas with little or no curvature and therefore share common properties (normals and closest distance to the common plane) and are managed together to enforce this. *Non-planar region surfels* on the other hand are located in areas of high curvature and are managed as in the original method of Keller *et al.* [77].

The dense and incremental mapping nature of our method enables easy data-association of modelled and measured planes across consecutive frames: After the current pose of the camera is estimated, plane association is simply done by counting pixel-level matches of projected modelled planes into the currently measured depth

image and handling mismatches in a logical manner.

Data-associated planes are then converted into the same world reference frame and refined with a running average. All modelled surfels belonging to the same plane are enforced to share the same refined normals and closest distance, unlike non-planar region surfels which average in isolation.

Finally, two or more overlapping modelled planes with similar properties are merged together to incrementally extend areas that initially fail to connect due to noise or occlusion.

While a particular planar region is still densely represented by many surfels, it is worth mentioning that surfels' positions and orientations are controlled by the same set of plane parameters. This may seem redundant and memory inefficient at first but allows us to densely represent complex planes with holes in the middle that would otherwise be challenging to model and render incrementally using closed polygons with hulls (*e.g.* as done in [157][152][6]).[1] Section 6.4 demonstrates that we can compress planes by a 9-to-1 ratio to achieve lightweight maps, particularly of indoor environments composed for several planes.

### 6.2.1 Preliminaries

As in [77], we represent the SLAM map with a set of $k$ unstructured surfels $\bar{\mathbf{P}}_k$ with properties such as position $\bar{\mathbf{v}}_k \in \mathbb{R}^3$, normal $\bar{\mathbf{n}}_k \in \mathbb{R}^3$, radius $\bar{r}_k \in \mathbb{R}$, confidence $\bar{c}_k \in \mathbb{R}$, and timestamp $\bar{t}_k \in \mathbb{N}$.[2] Additionally we include a plane ID $\bar{o}_k = i; i = 1, ..., p \in \mathbb{N}$ with $\bar{o}_k = 0$ signalling non-planar region surfels.

Overlapping surfels are representative of the local surface area with a radius size chosen to minimise holes between neighbours; we compute them as: $\bar{r}_k = \sqrt{2}\bar{\mathbf{v}}_{k(z)}/f$, with $f$ the camera focal length.

A live depth measurement $\mathcal{D}_\mathrm{l}$ is transformed into a metric vertex map $\mathcal{V}_\mathrm{l}(\mathbf{u}) = \mathbf{K}^{-1}\dot{\mathbf{u}}\mathcal{D}_\mathrm{l}(\mathbf{u})$, with $\mathbf{K}$ the camera intrinsic matrix, $\mathbf{u} = (x, y)^\top$ a pixel position in the image domain $\mathbf{u} \in \Omega \subset \mathbb{R}^2$ and $\dot{\mathbf{u}}$ its homogeneous representation. The live normal map $\mathcal{N}_\mathrm{l}$ is simply generated from the vertex map by central differences.

Additionally, we apply a bilateral filter [155] to $\mathcal{D}_\mathrm{l}$ generating discontinuity pre-

[1] See for example the plane hole on the washing machine door in Figure 6.9.
[2] The timestamp is the moment at which a measurement is taken such as a simple frame counter.

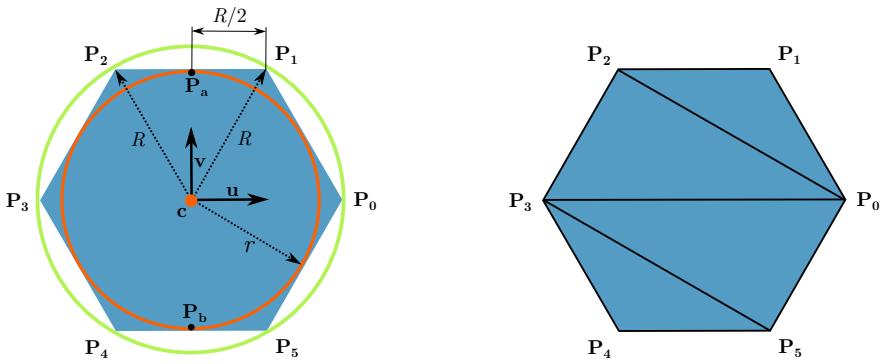



Figure 6.3: **(left)** A surfel approximated by a hexagon. The orange circle with radius $r$ is the surfel disk inscribed in the hexagon while the green circle is circumscribed, having a radius $R = \frac{2}{\sqrt{3}}r$. **(right)** A hexagon partitioned as a triangle strip for efficient drawing with OpenGL using the following vertex ordering: $P_1, P_2, P_0, P_3, P_5, P_4$.

served vertex $\mathcal{V}_{\mathrm{l}}^{\mathrm{b}}$ and normal $\mathcal{N}_{\mathrm{l}}^{\mathrm{b}}$ map with reduced noise.

To update the live camera to world transform $\mathbf{T}_{\mathrm{wl}} = [\mathbf{R}, \mathbf{t}] \in \mathbb{SE}(3)$, $\mathbf{R} \in \mathbb{SO}(3)$, $\mathbf{t} \in \mathbb{R}^3$; a pair of vertex map $\mathcal{V}_{\mathrm{r}}$ and normal map $\mathcal{N}_{\mathrm{r}}$ is rendered via surface-splatting (see Section 6.2.2) using the previous reference pose $\mathbf{T}_{\mathrm{wr}} \in \mathbb{SE}(3)$ that is incrementally aligned to $\mathcal{V}_{\mathrm{l}}^{\mathrm{b}}$ using dense ICP [112] with a point-plane error metric [133]. This produces a series of incremental updates $\{\tilde{\mathbf{T}}_{\mathrm{rl}}^{n}\}_{n=1}^{m}$ composed together to generate $\mathbf{T}_{\mathrm{wl}} \leftarrow \mathbf{T}_{\mathrm{wr}}\tilde{\mathbf{T}}_{\mathrm{rl}}^{m}$.

### 6.2.2 Surface-splatting

We approximate the disk shape of surfels with hexagons (see Figure 6.3). For this, two arbitrary perpendicular vectors $\mathbf{u}$, $\mathbf{v}$ to the normal $\mathbf{n} = [a, b, c]^{\top}$ are extracted such that $\mathbf{u} \cdot \mathbf{n} = 0$ and $\mathbf{v} \cdot \mathbf{n} = 0$:

$$\mathbf{n} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k} \,, \tag{6.1}$$

$$\mathbf{u} = (b - c)\mathbf{i} - a\mathbf{j} + a\mathbf{k} \,, \tag{6.2}$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u} \,. \tag{6.3}$$

The inscribed circle to the hexagon is the surfel being approximated allowing us to draw it without any holes. A circumscribed circle to the hexagon with radius

$R = \frac{2}{\sqrt{3}}r$ helps us to obtain the end points of the hexagon as follows:

$$\begin{aligned}
\mathbf{P_a} &= \mathbf{c} + r\mathbf{v} \ , \\
\mathbf{P_b} &= \mathbf{c} - r\mathbf{v} \ , \\
\mathbf{P_0} &= \mathbf{c} + R\mathbf{u} \ , \\
\mathbf{P_1} &= \mathbf{P_a} + (R/2)\mathbf{u} \ , \\
\mathbf{P_2} &= \mathbf{P_a} - (R/2)\mathbf{u} \ , \\
\mathbf{P_3} &= \mathbf{c} - R\mathbf{u} \ , \\
\mathbf{P_4} &= \mathbf{P_b} - (R/2)\mathbf{u} \ , \\
\mathbf{P_5} &= \mathbf{P_b} + (R/2)\mathbf{u} \ .
\end{aligned}$$

Rendering begins by filling OpenGL buffers containing the position, normal, radius and additional color properties and executing traditional point-based rendering but instructing the system to use the geometry shader stage to amplify geometry (see Listing A.3).

Thanks to this process, there is no need to explicitly store the end points of the hexagon to represent the surfel as they can be procedurally generated in code from the passed properties, thus saving memory space and bandwidth. Once the endpoints are identified, the hexagon is partitioned into a single triangle strip allowing efficient OpenGL rendering (see Figure 6.3 right).

### 6.2.3 Relocalisation

In line with the dense nature of our system, we avoid extracting features to match (such as SIFT or SURF) to retrieve known places and continue tracking when this is lost. Instead we perform whole image encoding of keyframes using Ferns, following the method of Glocker *et al.* [61]. This enables the fast retrieval of near poses when tracking is lost that are later refined with ICP on the dense model.

## 6.3 Mapping with Planes

Assuming an updated live camera pose $\mathbf{T}_{\text{wl}}$, mapping a scene consists of integrating new measurements into the global model. To do so, a $4\times$ super-resolution index map $\mathcal{I}^{\text{s}}$ is created by recording the point index $k$ projected into the live frame at pixel $\mathbf{u}^{\text{s}} = \pi(\mathbf{K}^{\text{s}}\mathbf{T}_{\text{wl}}^{-1}\bar{\mathbf{v}}_k)$ via standard pin-hole projection function $\pi$. A super-resolution

approach is beneficial to avoid wrong correspondences when points project onto the same pixel due to limited precision.

Modelled surfels can now be associated with measurements according to sensor uncertainty, normals agreement, confidence value and distance to viewing ray [77], producing data-associated pairs $a_{\text{surfels}} = \{(i,j)\}; i = 1,...,k; j = 1,...,w \times h$.

### 6.3.1 Planar Region Detection

Similar to Trevor *et al.* [156] we detect planes using connected component labelling [39]. This produces a label map $\mathcal{L}(\mathbf{u}) = i; i = 1,...,q \in \mathbb{N}$ identifying to which of the $q$ measured planes each pixel belongs to. $\mathcal{L}(\mathbf{u}_i) = 0$ is reserved for regions with few connecting components or high curvature (see Figure 6.4 **left**).

Planes are parametrised by $\pi = (n_x, n_y, n_z, d)^\top$ with $\mathbf{n}_\pi = (n_x, n_y, n_z)^\top$ the plane normal and $d$ the closest distance to the common plane.

Detection proceeds by first computing a similarity map efficiently using the GPU. This generates for every pixel a bitmask indicating if the pixel above and to the left of the current one have similar plane distance and normal:

$$\text{Mask}(\mathbf{u}) = (S(\mathbf{u}, \mathbf{up}) \ll 1) \mid S(\mathbf{u}, \mathbf{left}) \ , \tag{6.4}$$

$$S(\mathbf{x},\mathbf{y}) = \begin{cases} 1 & \text{if } \|\mathcal{V}_1^{\text{b}}(\mathbf{x}) \cdot \mathcal{N}_1^{\text{b}}(\mathbf{x}) - \mathcal{V}_1^{\text{b}}(\mathbf{y}) \cdot \mathcal{N}_1^{\text{b}}(\mathbf{y})\| \\ & < \delta_1(\mathcal{V}_1^{\text{b}}(\mathbf{x})_{(z)})^2 \text{ and } \mathcal{N}_1^{\text{b}}(\mathbf{x}) \cdot \mathcal{N}_1^{\text{b}}(\mathbf{y}) > \cos(\Theta_1) \\ 0 & \text{otherwise} \ , \end{cases}$$

$$\mathbf{up} = \mathbf{u} - (0,1)^\top, \ \mathbf{left} = \mathbf{u} - (1,0)^\top \ .$$

The process continues on the CPU by assigning unique labels to similar and contiguous pixels followed by a union-find algorithm to merge equivalent labels. To prevent merging regions with low-quality normals, the process is avoided at depth-discontinuity boundaries (see Figure 6.4 **right**), *i.e.* zeros in the map:

$$\text{Disc}(\mathbf{u}) = \prod_{\mathbf{u}_i \in \omega} S(\mathbf{u}, \mathbf{u}_i) \ . \tag{6.5}$$

with $\omega$ a window with radius 3 centered in $\mathbf{u}$.

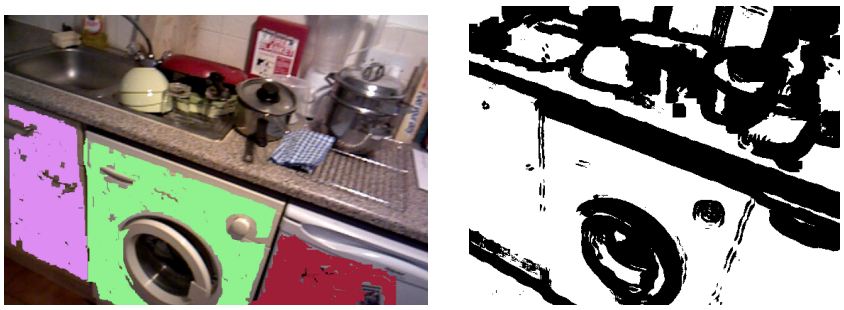


Figure 6.4: Planar region detection. **(left)** Three planes have been detected with connected component labelling using the measured vertex and normal maps. **(right)** A depth discontinuity map is used to avoid labelling regions with low quality normals (black pixels).

Following [156], we discard regions with few connected pixels ($<$ min-inliers). From the rest we fit planes by performing Principal Component Analysis (PCA): first the vertices are normalised by subtracting its mean $\hat{\mathbf{v}}$, followed by the computation of a covariance matrix $\mathbf{\Sigma}$ and its corresponding eigendecomposition. The eigenvector with minimum eigenvalue $\lambda_{\text{min}}$ becomes the plane normal $\mathbf{n}_\pi$. The plane distance is computed as: $d = -\mathbf{n}_\pi \cdot \hat{\mathbf{v}}$ while the plane curvature is: $\kappa = \frac{\lambda_{\text{min}}}{\sum_{i=1}^{3} \lambda_i}$. We discard planes having curvature $\kappa >$ max-curvature.

### 6.3.2 Data-Association with Planes

A SLAM system should be able to incrementally expand its map during exploration. To enable this in our system we need to expand modelled planes as the camera browses a new scene.

Once the current sensor pose is estimated, modelled planar and non-planar region surfels are projected into the current live frame. For each pixel $\mathbf{u}$, one of several data-association cases may occur (see Figure 6.5):

**(A)** A modelled plane and a measured plane intersect. This situation indicates a data-association between the modelled and measured planes, producing pairs $a_{\text{planes}} = \{(i, j)\}; i = 1, ..., p; j = 1, ..., q$.

**(B)** Modelled surfels lack a planar measurement. This indicates a non-planar region due to high-curvature or noisy depth measurements preventing planes to be fitted.

**(C, D)** Modelled planar region surfels lack planar measurement. Due to noise, not every plane is expected to be detected on measured depth maps.

**(E, F)** Unmodelled (measured) planar regions. This happens when a new plane is detected on the live depth map and is yet to be incorporated into the SLAM map.

**(G)** Invalid data. Occurs at pixels where the live depth data from the sensor is invalid (has holes).

Cases C and D as well as E and F have to be disambiguated. To do this we first identify the intersecting pixels (Case A) giving them a unique ID that is flooded into the regions C and E, thus expanding the modelled plane of regions C and A towards region E. At this point we are left with only cases D and F which are purely modelled or unmodelled (measured) respectively.

This disambiguation is efficiently performed in practice via parallel operations evaluated on the GPU [1][12]. First the set of pixels associations $a_{\text{planes}}$ corresponding to Case A are transformed into a hash value $h = j\rho + i$ (with $\rho$ a constant to ensure uniqueness) and sorted in parallel to group pixels belonging to the same modelled plane first and same measured plane second. This is followed by a parallel reduction operation using the hash values as keys and a constant 1 (one) as value, giving a list of possible associations and number of pixels supporting this: $\{(j, i, count)\}$.

A measured plane $\bar{\pi}_j$ could be associated to more than one modelled plane $\bar{\pi}_i$ due to noise or occlusion. To prevent wrong associations to the measured plane we traverse the previous list looking for a modelled plane with similar coefficients: $\|d'_j - d_i\| < \delta_2$, $\mathbf{n}'_j \cdot \mathbf{n}_i > \cos(\Theta_2)$. Similar modelled planes IDs are added into a merge map $\mathcal{M}$ and the one with maximum count $count^\star >=$ min-assoc-count is chosen as the final associated modelled plane and used as key for $\mathcal{M}$.

### 6.3.3 Planar Region Refinement and Merging

A modelled plane $\bar{\pi}_i$ is refined with the associated measured plane $\pi_j$ using a simple running average. First the measured plane coefficients are transformed into the world reference frame:

$$\mathbf{n}'_j = \mathbf{R}\mathbf{n}_j \ , \tag{6.6}$$

$$d'_j = -\mathbf{n}'_j \cdot \mathbf{t} + d_j \ . \tag{6.7}$$

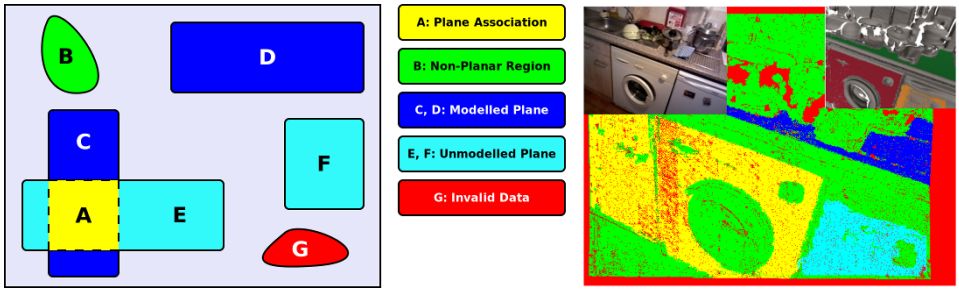



Figure 6.5: Data-association cases. **(left)** The diagram shows all the possible pixel-wise association cases when projecting the SLAM map into the measured live frame. **(right)** Colour-coded visualisation of pixel-wise association cases. The top-left inset shows the rgb data while the top-right shows the densely reconstructed planar and non-planar region surfels.

The modelled plane coefficients are then refined with:

$$\mathbf{n}_i \leftarrow \frac{w\mathbf{n}_i + \mathbf{n}'_j}{w+1} \ , \ d_i \leftarrow \frac{wd_i + d'_j}{w+1} \ , \ w \leftarrow w+1 \ . \tag{6.8}$$

We traverse the merge map $\mathcal{M}$ and for each entry we rename the contained plane IDs with that of the corresponding key.

Finally all surfels belonging to the measured plane $\pi_j$ are projected onto the refined modelled plane $\bar{\pi}_i$.

### 6.3.4 Non-Planar region surfels mapping

Surfels not having planar region measurements have their associated properties updated with a running average $\alpha$ weighted for radial noise as in [77]. This is reproduced here for easier reading:

$$\bar{\mathbf{v}}_k \leftarrow \frac{\bar{c}_k\bar{\mathbf{v}}_k + \alpha\mathbf{T}_{\text{wl}}\mathbf{v}_\text{l}}{\bar{c}_k + \alpha} \ , \ \bar{\mathbf{n}}_k \leftarrow \frac{\bar{c}_k\bar{\mathbf{n}}_k + \alpha\mathbf{R}\mathbf{n}_\text{l}}{\bar{c}_k + \alpha} \ , \tag{6.9}$$

$$\bar{r}_k \leftarrow \frac{\bar{c}_k\bar{r}_k + \alpha r_\text{l}}{\bar{c}_k + \alpha} \ , \ \bar{c}_k \leftarrow \bar{c}_k + \alpha \ , \ \bar{t}_k \leftarrow t \ . \tag{6.10}$$

## 6.4 Map Compression

Planar region surfels need to densely populate their area of coverage, however many of their properties are shared between them (normals, radius size, confidence, timestamp and plane ID). Furthermore their planar position requires a two-dimensional representation only.

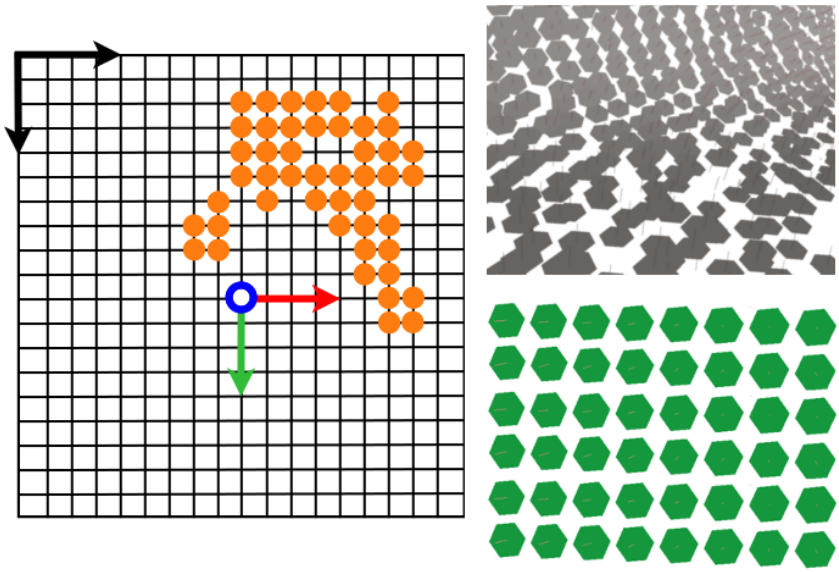


Figure 6.6: (**left**)Virtual Image with on-pixels surfels (orange) representing planar region coverage. The coloured coordinate frame are the eigenvectors centred at the plane centroid, while the black coordinate frame is the virtual image origin. (**top-right**) Non-planar region surfels around a region of high curvature need to explicitly represent individual position and orientation information. (**bottom-right**) Planar region surfels are evenly organised and share common properties. ***Note***: Surfel radius size reduced for easier visualisation.

We compress planar regions whenever they become non-visible (*i.e.* outside the view frustum). First we execute frustum culling by intersecting the plane bounding box with each of the 6 planes enclosing the frustum. Only if the planar region does not intersect all of them we can safely compress the plane, move its data down the memory hierarchy (*i.e.* from GPU memory to RAM or disk) and reclaim the working GPU memory. This form of occlusion culling also helps to maintain an almost consistent frame rate independent of map size, while reducing the need for additional space partitioning techniques like octrees.

Compression begins by performing an additional PCA step in order to estimate the major x-y axis of the extended plane. As in the plane fitting procedure, we first normalise the vertices by subtracting its mean $\hat{\mathbf{v}}$, followed by calculation of the covariance matrix $\boldsymbol{\Sigma}$ and corresponding eigendecomposition to obtain the x-y axis.

We can think of the plane compression mechanism as a way of representing the plane as a binary image: with on-pixels signalling the areas of coverage and off-

pixels for holes (see Figure 6.6). To do so we will convert vertex coordinates into pixel locations in a *Virtual Image* of dimensions: $w_{vi} \times h_{vi}$. In practice both dimensions are set to 65536, allowing us to represent planes with dimensions in the range $(-32.768m, -32.768m)$ to $(32.767m, 32.767m)$ at millimetre accuracy. The fixed virtual image dimensions allow us to further linearise the coordinates in 1D. Computing the compressed index of a surfel is detailed below:

$$\mathbf{v}_{\mathrm{c}} = \mathbf{v} - \hat{\mathbf{v}} \ , \tag{6.11}$$

$$\mathbf{v}_{\mathrm{p}} = (\mathbf{x}_{\mathrm{axis}} \cdot \mathbf{v}_{\mathrm{c}}, \mathbf{y}_{\mathrm{axis}} \cdot \mathbf{v}_{\mathrm{c}})^{\top} \ , \tag{6.12}$$

$$\mathbf{v}_{\mathrm{vi}} = \mathrm{round}(\mathbf{v}_{\mathrm{p}} \times 1000) \ , \tag{6.13}$$

$$\mathbf{v}_{\mathrm{o}} = \mathbf{v}_{\mathrm{v}i} + (w_{\mathrm{v}i}, h_{\mathrm{vi}})^{\top}/2 \ , \tag{6.14}$$

$$\mathrm{index} = \mathbf{v}_{\mathrm{o}(y)} \times w_{\mathrm{v}i} + \mathbf{v}_{\mathrm{o}(x)} \ . \tag{6.15}$$

Here $\mathbf{v}_{\mathrm{c}}$ is the normalised (centred) vertex, $\mathbf{v}_{\mathrm{p}}$ is the vertex on the plane after projecting the $\mathbf{v}_{\mathrm{c}}$ coordinates with the plane axis, $\mathbf{v}_{\mathrm{v}i}$ is the vertex on the virtual image with integer units pre-scaled to preserve millimetre accuracy, $\mathbf{v}_{\mathrm{o}}$ is the vertex on the top-left reference frame, finally *index* is the linearised $\mathbf{v}_{\mathrm{o}}$ coordinates in 1D.

In this way planar region surfels are compressed with an approximate ratio of 9-to-1 as we only need to store their indices with 4 bytes per surfel.[3] In contrast, non-planar region surfels require 36 bytes each: vertices (3 floats), normals (2 floats), radius (1 float), confidence (1 float), timestamp (1 uint), plane ID (1 uint).

Scenes composed of planar and non-planar region surfels produces combined compression ratios of about 2.27. Some compression results on real and synthetic data are shown on the chart in Figure 6.7.

Further compression ratios could be achieved for example by performing run-length encoding of on-pixels but this is not explored yet.

Decompression is trivially achieved by performing the inverse compression steps (6.15 - 6.11) and in practice both tasks take less than 2ms, allowing online operation.

## 6.5 Results

Quantitative experiments using synthetic scenes were performed to measure the quality of tracking with and without planar mapping. Qualitative results of the

[3] Planar regions' surfels share the same normal, radius, confidence, timestamp and plane ID.

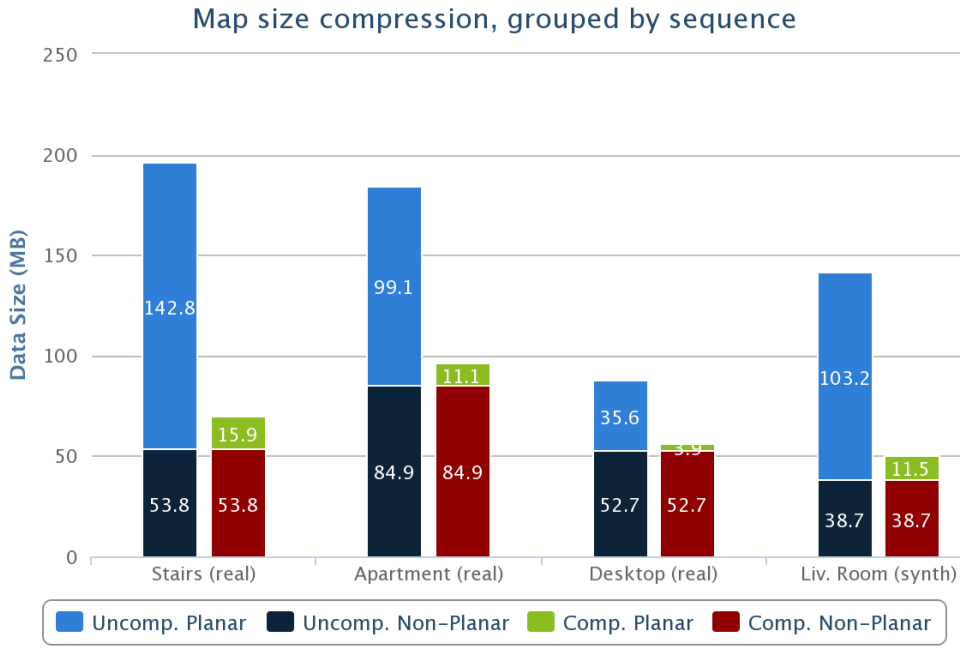



Figure 6.7: Chart showing the map data size of planar and non-planar region surfels with and without compression.

enhanced planar representation are shown for real scenes in Figures 6.1, 6.9 and 6.10 obtained with the Asus Xtion Pro RGB-D sensor.

The plane detection parameters were set to min-inliers = 1000, $\delta_1 = 0.01$m, $\delta_2 = 0.2$m, $\Theta_1 = \Theta_2 = 20°$, max-curvature = 0.00015, and min-assoc-count = 100.

The noise characteristics of the depth sensor make plane detection only usable in the close range ($< 4$m). Our thresholds discourage false-positives by discarding planes with large-curvature (as described in Section 6.3.1). Nevertheless, the fact that mapping and tracking are still possible in the absence of plane detection reinforces the integrated planar/non-planar approach presented.

### 6.5.1 Synthetic scenes

We evaluate our system on synthetic scenes with ground truth camera poses for two trajectories produced by Handa *et al.* [63]. Depth maps are further corrupted by the noise model proposed by Barron and Malik [10] to create data closer to the Kinect sensor. Reconstructions results of the 'living room' sequence are shown on Figure 6.8.

Table 6.1 summarises the Absolute Trajectory Error (ATE) as proposed by Sturm *et al.* [151]. ATE computes the absolute difference between the ground truth and estimated poses after alignment.

Although tracking from a globally consistent dense model is shown to be already of high quality [112] we can see from the RMSE values that using planar regions surfels decreases the trajectory error slightly, this is because the scene contains a large number of planar regions affected by noise and alleviated earlier by our method (compared to KinectFusion [112] or Point-based Fusion [77] that require a few frames to denoise).

While the RMSE of *trajectory-0* is large compared to *trajectory-1* it is worth highlighting that *trajectory-0* is challenging when tracking with ICP since the camera cannot be locked based on 2 planes only (wall and ceiling) and therefore large drift occurs.[4] This artefact was also reported in [63] section VI-D-1.

Table 6.1: Absolute Trajectory Error (ATE) in synthetic scene

| Error | trajectory-0 | | trajectory-1 | |
|---|---|---|---|---|
| | non-planar | planar | non-planar | planar |
| RMSE | 0.254134 | **0.246437** | 0.018997 | **0.016940** |
| Mean | 0.222794 | 0.218559 | 0.016906 | 0.015043 |
| Median | 0.179679 | 0.182547 | 0.014714 | 0.016449 |
| Std | 0.122258 | 0.113857 | 0.008666 | 0.007789 |
| Min | 0.055287 | 0.070420 | 0.003252 | 0.002228 |
| Max | 0.728229 | 0.645284 | 0.032742 | 0.028430 |

### 6.5.2 Real-world scenes

Examples of real-world scenes are shown in Figure 6.1, 6.9 and 6.10. Here the stairs of a house and an apartment have been reconstructed and major planes parametrised incrementally and in real-time. The first case could be particularly useful for stair-climbing robots navigating new environments.

### 6.5.3 Augmented Reality with dense planar maps

We can take advantage of the dense and real-time nature of our system to perform novel Augmented Reality (AR) interactions with fine occlusion handling, requiring very little user input.

[4] This can be seen on minute 0:32 at: http://youtu.be/4O-OaV0h4AQ.

As a first example, we let the user choose a set of planes to augment the original input with an application display using the Oculus Rift paired with an Xtion sensor (see Figure 6.11 and the Dense Planar SLAM videos in Appendix B), essentially converting the real-world into a window manager. To enable this, we first extract the bounding-box of the selected plane(s) and convert it into a quad polygon for efficient texture mapping. This feature could also be very useful in see-through AR headsets as it can replace small floating widgets with large projections on planes without interfering with the wearer's field of view (*e.g.* a limiting factor in the current Google Glass).

Another useful example is virtually replacing the floor style of a house many times until the user is satisfied with the result (see Figure 6.12).

The idea of overlaying information on real planes instead of floating windows allows the user to safely navigate environments without fear of collisions and make tasks like zooming in/out as natural as walking closer/further from surfaces.

### 6.5.4 System Statistics

Table 6.2 summarises the results when mapping the stairs sequence shown in Figure 6.1. This was executed on a high-performance laptop equipped with an Intel i7 Quad Core CPU at 2.50GHz and NVidia GTX 580M GPU with 2GB of memory.

While the overhead of plane detection, data association and compression are not present in systems like [77], we note that further optimisation can be easily engineered to increase frame rate such as moving mapping to a lower priority thread as in Klein *et al.* [79].

## 6.6 Conclusions and Future Work

We have presented a Dense Planar SLAM algorithm which identifies, merges and compresses the arbitrary planar regions which are present in many man-made scenes, and leads to an efficient, robust and real-time plane-aware SLAM system.

In addition, we have shown the highly practical AR applications this permits, in particular the use of planar regions for the display of information in a very natural manner which will fit well with see-through head mounted displays.

Table 6.2: System statistics for the stairs sequence

| **Memory usage** | |
|---|---|
| Non-Planar region surfels count | 1,566,063 |
| Planar region surfels count | 4,159,902 |
| Plane Count | 30 |
| Point-Based Fusion Memory [77] | 196.58 MB |
| Dense Planar SLAM Memory (this work) | 69.64 MB |
| Compression Ratio | 2.82 |
| **Timings** | |
| Frame Prediction | 11.8 ms |
| ICP | 6.08ms |
| Plane Detection | 10.1 ms |
| Data Association | 24.02 ms |
| Averaging | 2.58 ms |
| Surfel Addition/Removal | 9.4 ms |
| Compression and Decompression | 1.8 ms |
| Total time | 65.98 ms |
| Frame Rate | 15.16 fps |

We remain interested in our long-term goals of a fully object-based SLAM system within which this planar mapping will form a crucial component.

Adding a graph-based loop closure optimisation for consistency over long loopy trajectories is a clear short-term goal, and we hope that standard methods as used in [135] will be suitable here. Though careful thought has to be given to how non-planar regions of surfels should be treated when the locations of planes is optimised due to loop closure constraints; presumably each small blob of non-planar surfels should be attached and adjusted rigidly with the same transformation of one or more of its neighbouring planar regions rather than being broken up or sheared.

Also, we will continue to work on using these non-planar regions to extend a library of object types; the main challenge here is rapid learning of efficient detectors for these new object types during real-time operation when the resources for extravagant training are not available.

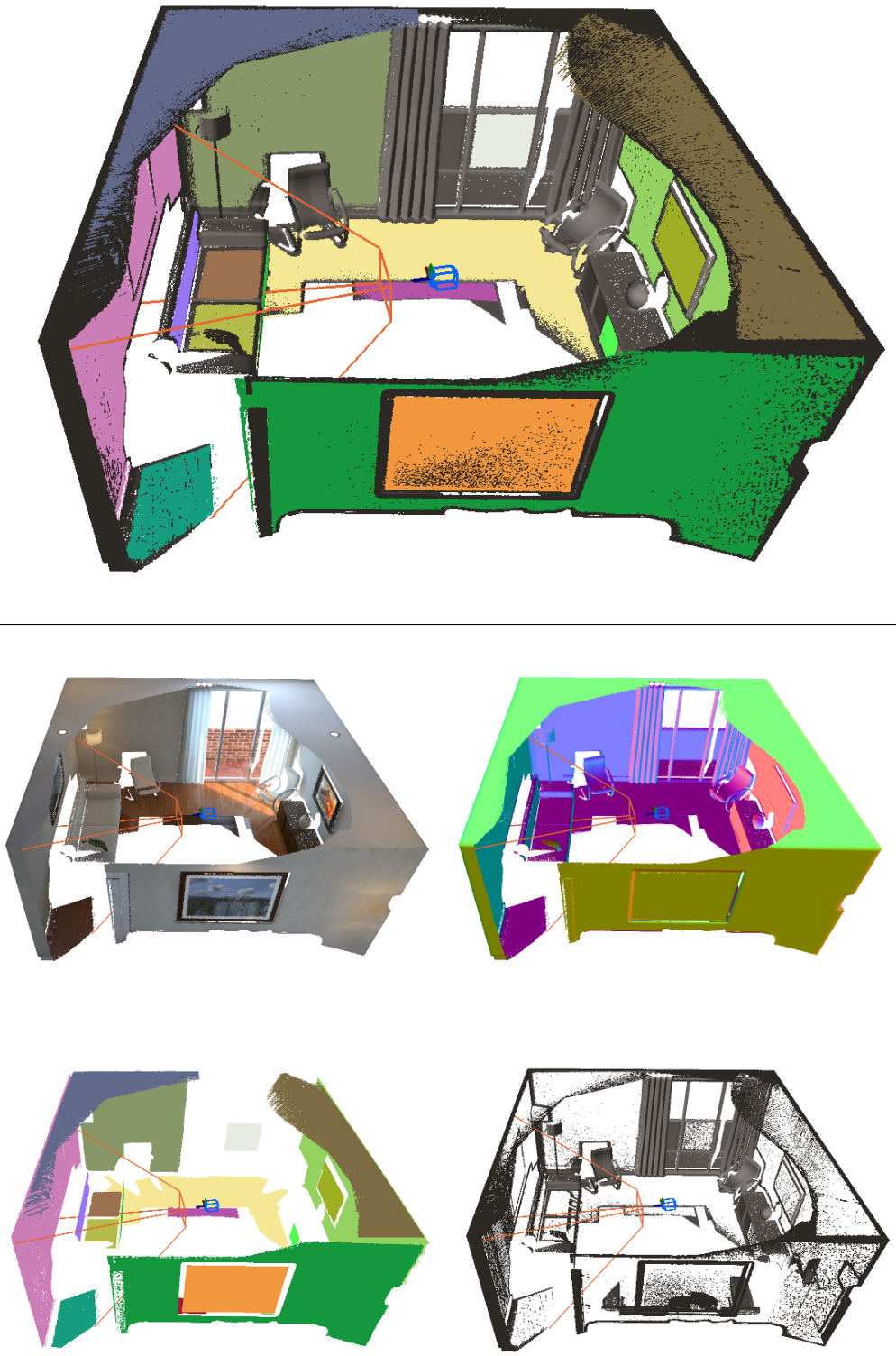


Figure 6.8: Synthetic scene reconstruction of a living room. **(top)** Displaying both planar and non-planar regions surfels. **(bottom)** In clockwise order: Colour output, normal map, non-planar region surfels only, planar region surfels only.

Figure 6.9: Real scene reconstruction of an apartment. **(top)** Displaying both planar and non-planar regions surfels. **(bottom)** In clockwise order: Colour output, Normal Map, Non-Planar region surfels only, Planar region surfels only.

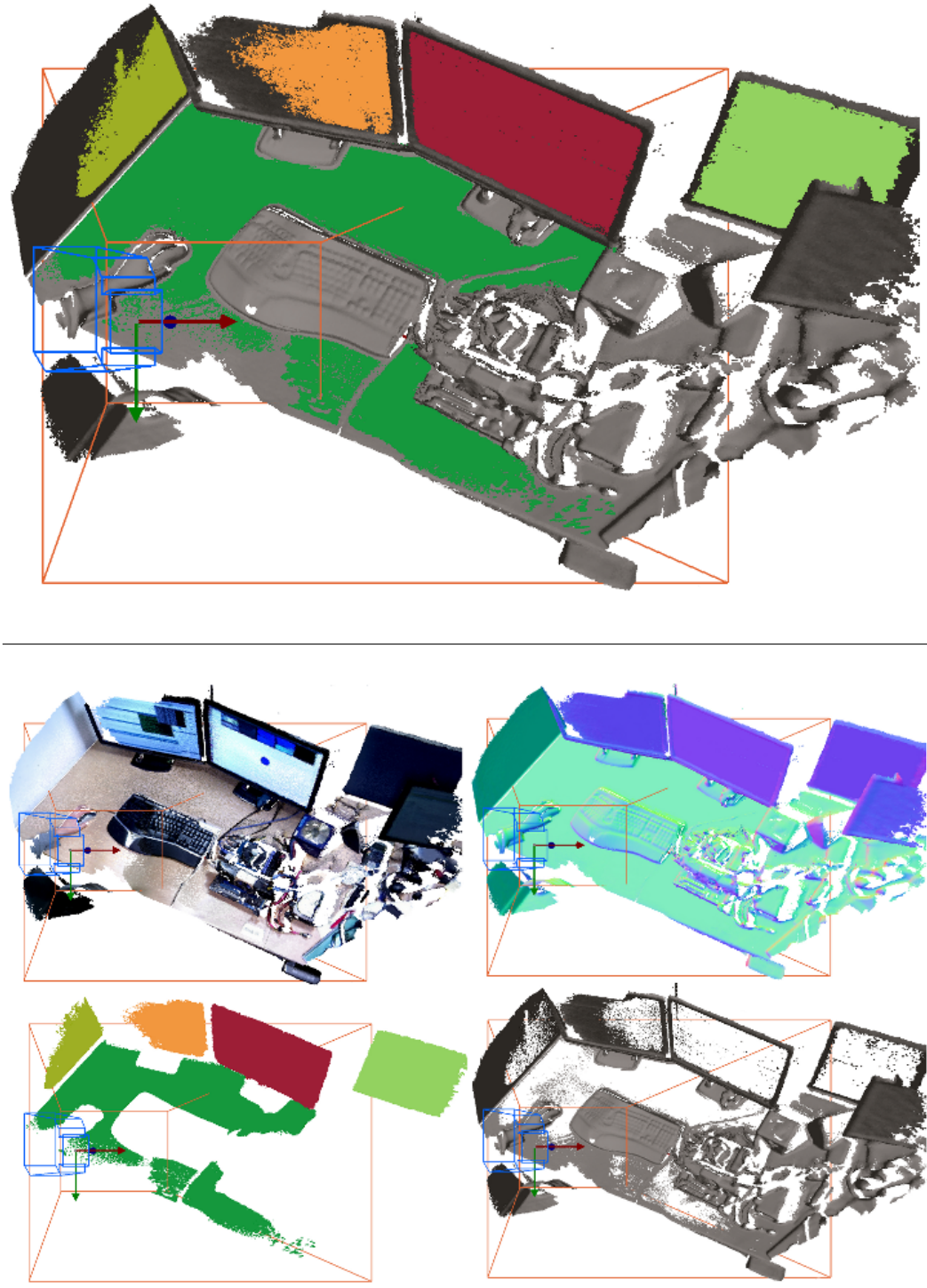


Figure 6.10: Real scene reconstruction of a desktop. **(top)** Displaying both planar and non-planar regions surfels. **(bottom)** In clockwise order: Colour output, Normal Map, Non-Planar region surfels only, Planar region surfels only.


Figure 6.11: Facebook Wall on a real wall using the Oculus Rift. The user chooses a wall from which to read his Facebook Wall.

Figure 6.12: Floor carpet change. The ground plane is selected and overlaid with a new carpet.

Chapter 7

# Conclusions

Having described the core work of our research in the past few chapters, we now summarise the novel contributions provided to the field of SLAM and discuss its current limitations and possible workarounds. Finally we provide some ideas of future avenues to explore for creating more compelling semantic SLAM systems.

## 7.1 Contributions

In Chapter 1 we defined what the theme of *Dense Semantic SLAM* is about: a system capable of identifying meaningful discrete elements in a map using all available sensory information within the loop of SLAM itself, for the purpose of self-localisation and complex interaction; we also envisaged future applications enabled by this technology. Later we described the evolution of SLAM leading to cornerstone components that we base our research on, such as dense SLAM systems like KinectFusion.

To enable semantic SLAM capabilities, in Chapter 2 we reviewed state-of-the-art approaches for object recognition and scene labelling, which revealed the limitations of the methods to deal with real-time requirements and new sensory inputs like depth sensors.

In SLAM we have requirements such as real-time processing and mobile operation. However, the practice of using the information from every pixel to enhance the robustness of SLAM challenges the capabilities of traditional CPU architectures.

Therefore, in Chapter 3 in addition to discussing the mathematical preliminaries, we also reviewed the commodity parallel processing available via GPU architectures as well as the programming models needed to control their operation that we used to design our novel algorithms.

We developed a strategy to accelerate Bundle Adjustment in Chapter 4 using a hybrid CPU/GPU solution, achieving up to 10.5x speedup compared to a CPU-only version. The architectural demands of having a discrete GPU with its own memory subsystem made the approach suitable when optimising more than 360 points but less than approximately 250K and 30 cameras due to the cost of memory transfers and lack of virtual memory support. Other architectural challenges that constrained the approach were the little shared memory available when performing reductions and also limited opportunities for memory coalescing.

We concluded the chapter by providing insights to overcome those limitations, namely by using different data structures like structure of arrays (SoA) to maximise memory coalescing, the use of another optimisation algorithm known as Linear Conjugate Gradient that would reduce the memory requirements, and the need for a domain-specific language that would simplify the algorithmic description for parallel operation while maximising opportunities for acceleration in current and new hardware architectures.

In Chapter 5 we presented our SLAM++ system, which takes advantage of using objects in the loop of SLAM itself, following the assumption that many scenes consist of repeated, domain-specific objects and structures. This offers the descriptive and predictive power of SLAM systems which perform dense surface reconstruction, but with a huge representation compression. To do so, we extended two object recognition approaches based on shape matching and Hough forests and made them operate at real-time speeds on the GPU. The recognised objects provide 6 DoF camera-object constraints which feed into an explicit graph of objects, continually refined by efficient pose-graph optimisation. The object graph enables predictions for accurate ICP-based camera to model tracking at each live frame, and efficient active search for new objects in undescribed image regions.

The enhanced object-level representation enables real-time incremental SLAM in large, cluttered environments, including loop closure, relocalisation and the detection of moved objects. Prior knowledge of objects also brings new opportunities for

augmented reality and robotics interactions, since the stored objects can be annotated with non-visual information such as purpose, weight, and grasping regions. The SLAM++ approach however requires a pre-processing stage consisting of building the database prior to execution and it will only operate if known objects can be found.

Finally, in Chapter 6 we developed a new system called Dense Planar SLAM that can directly map a new environment using bounded planes and surfels. The new method offers the every-pixel descriptive power of the latest dense SLAM approaches, but takes advantage directly of the planarity of many parts of real-world scenes via a data-driven process to directly regularize planar regions and represent their accurate extent efficiently using an occupancy approach with on-line compression. Large areas can be mapped efficiently and with useful semantic planar structure which enables intuitive and useful AR applications such as using any wall or other planar surface in a scene to display a user's content.

## 7.2 Discussion and Future Research

We have demonstrated the benefits of incorporating higher-level entities in the loop of SLAM itself such as the use of objects and planar regions. Throughout our work we have also emphasised the importance of parallel GPU architectures to enable real-time operation of algorithms such as bundle adjustment, object recognition and plane detection using all the available sensory information.

The most straightforward next steps would be to blend the contributions presented in the previous three chapters. First by unifying Dense Planar SLAM with SLAM++ we would be able to parametrise planar regions and objects such that operation of the system in a new environment is not compromised due to a limited object database. To easily extend the database, knowledge of planar regions would be beneficial as it can help to segment new objects on-the-fly if we consider that most objects of interest have a supporting planar region.

To be able to generate globally consistent maps, we could rely on the graph approach as presented in Section 5.2.6 and extend it to incorporate planar entities. Early experiments to do so however revealed some problems arising from the use of unstructured surfels, as the optimisation of planes alone generates gaps between planar and non-planar regions. Therefore, more thought should be given to the

right way to incorporate neighbouring and visibility constraints, possibly following the mass-spring approach used for physics simulation [154]. In order to optimise such large scale models of surfels not belonging to either planes or objects, the insights gained in developing GPU-accelerated bundle adjustment would be beneficial.

Another piece of future work is to address object recognition with scalability in mind. For that, development of tools to create a large database of objects, annotated with ground truth pose and other useful metadata is a requirement to explore better algorithms. We need to consider methods for near real-time learning, perhaps by the use of Extremely Randomised Trees [59] or Random Ferns [122] and combine them with Information Retrieval ideas to handle large-scale datasets. In a practical setting, we would like to see a Dense Semantic SLAM system deployed to a large number of users backed by a centralised database that is continuously being extended. It is therefore imperative to address scalability to thousands or even millions of objects without sacrificing real-time tracking. Some strategies to simplify the scalability requirement would be to incorporate contextual signals such as GPS and constraining the search space by scene types, such as office interiors compared to houses.

In addition, it will be interesting to investigate the minimal number of viewpoints required to regress object pose considering that an ICP alignment will take place afterwards. Our choice of viewpoint sampling consisting of 162 vertices on a hemisphere is somehow arbitrary but seemed to perform well using view-based ICP with a point-plane error metric [133]. Other methods such as Fast ICP [51] with a much wider basin of convergence might require less coverage and should be worth investigating.

We believe that an ideal semantic SLAM system would be best described with a scene graph, incorporating not only object instances but also hierarchical parent-child relationships between composite objects, as well other non object parametrisation like planar regions and scene lighting. A possible scene graph structure for an indoor scene is illustrated in Figure 7.1. In addition, being able to add much more metadata like grasping regions and weight would have practical applications in robotics, augmented reality and related disciplines.

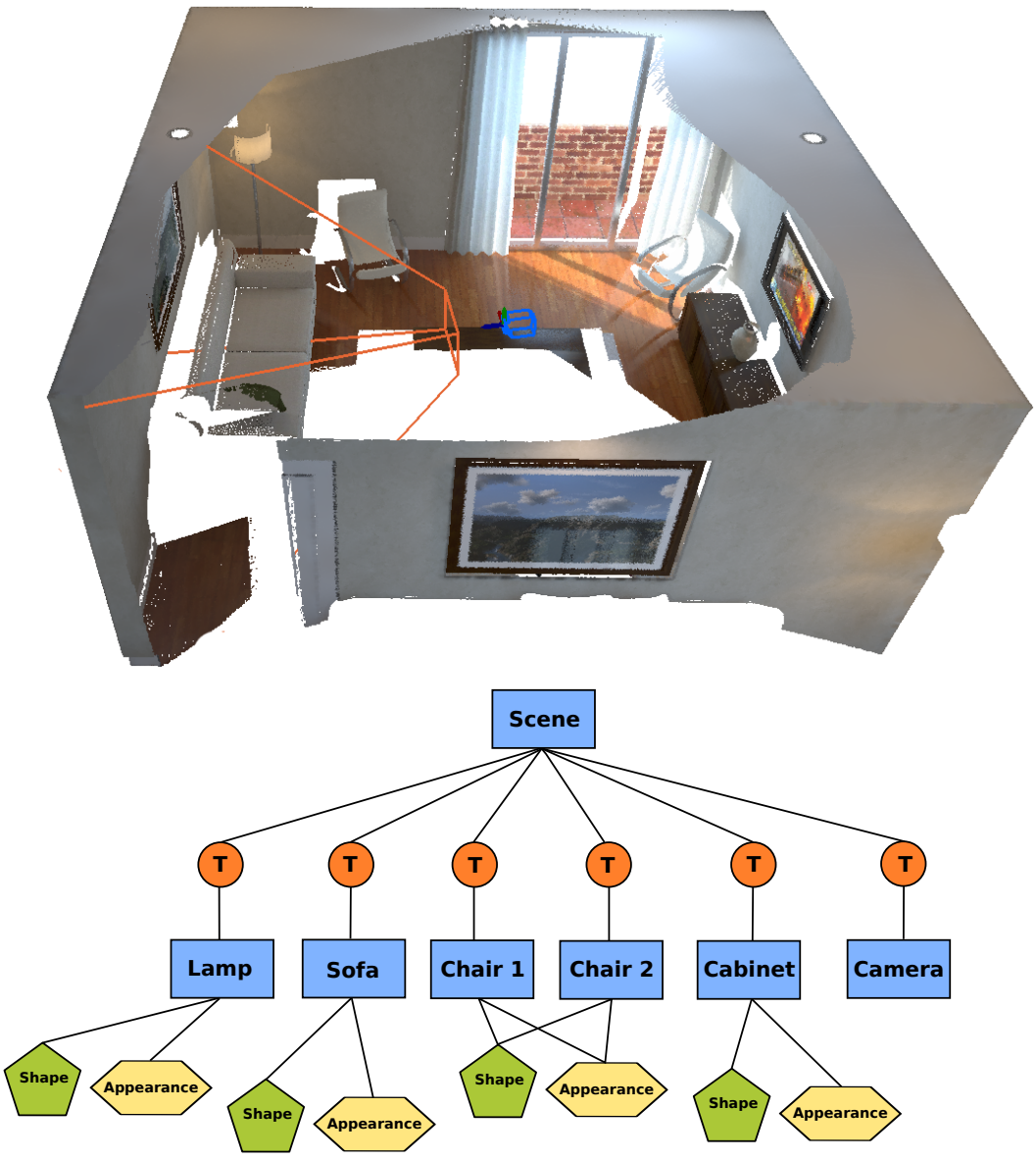



Figure 7.1: An example scene graph. **(top)** The living room scene to be described by the graph. **(bottom)** A simplified scene graph consisting of group container nodes *(blue)*, transformation nodes *(orange)*, shape nodes storing 3D geometric information *(green)* and material nodes storing texture maps *(yellow)*. Notice how the two chairs share a common shape and appearance nodes, while the camera does not need such information as it can be completely described by its transformation node.

Appendix A

# Code Listing

Listing A.1: Host side launching of Parallel Sum reduction with CUDA

```
void main(void)
{
  int elementCount = 32768;

  //Create input array on host
  std::vector<float> h_inputArr(elementCount);
  //[...] Fill input array with data

  //Create input array on device
  float *d_inputArr = NULL;
  cudaMalloc((void**)&d_inputArr, sizeof(float)*elementCount);

  //Copy input data from host to device
  cudaMemcpy(d_inputArr, &h_inputArr[0],
    sizeof(float) * elementCount, cudaMemcpyHostToDevice);

  //Organise threads into blocks and grids
  int blockSize = 256; //threads in block
  int gridSize = elementCount/blockSize; //blocks in grid

  //Create results array for each block
  //plus an additional slot for the final sum
  float *d_blockSumArr = 0;
  cudaMalloc((void**)&d_blockSumArr,sizeof(float)*(gridSize+1));

  //Execute kernel to compute partial sums per block
  cuBlockSumReduction<<<gridSize,
                        blockSize,
                        blockSize * sizeof(float)>>>(
    //output
    d_blockSumArr,
    //input
    d_inputArr, elementCount);

  //Execute kernel again with a single block
  //to complete the partial sums
  cuBlockSumReduction<<<1,
                        gridSize,
                        gridSize * sizeof(float)>>>(
    //output
    d_blockSumArr + gridSize, //pointer to additional slot
    //input
    d_blockSumArr, gridSize);

  //Copy single result data from device to host
  float sumResult = 0;
  cudaMemcpy(&sumResult, d_blockSumArr + gridSize,
    sizeof(float), cudaMemcpyDeviceToHost);

  //Release device arrays
  cudaFree(d_inputArr); cudaFree(d_blockSumArr);
}
```

Listing A.2: Parallel Sum reduction in CUDA

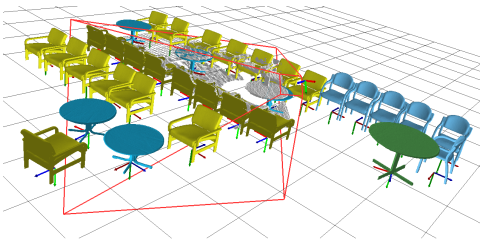
```
__global__ void cuBlockSumReduction(
  //output
  float* d_blockSumArr,
  //input
  const float* d_inputArr,
  const int elementCount)
{
  extern __shared__ float cache[];

  //Compute the Global Thread ID
  int id = threadIdx.x + blockIdx.x*blockDim.x;

  //Thread ID within a block
  int cacheID = threadIdx.x;

  //Load thread data item into shared memory
  float x = 0;
  if(id < elementCount)
  {
    x = d_inputArr[cacheID];
  }
  cache[cacheID] = x;

  //Ensure all threads in block have written to shared memory
  __syncthreads();

  int offset = blockDim.x/2;
  while(offset != 0)
  {
    if(cacheID < offset)
    {
      cache[cacheID] += cache[cacheID + offset];
    }

    //Ensure all threads in block have written to shared memory
    __syncthreads();
    offset /= 2;
  }

  //Final result of a block is at the first element of cache
  if(cacheID == 0)
  {
    d_blockSumArr[blockIdx.x] = cache[0];
  }
}
```

Listing A.3: Geometry Shader to generate a surfel from an oriented point

```
#define HEXRADIUS 1.154700538 // 2/sqrt(3)

POINT
TRIANGLE_OUT
void gsSurfel(float3 center : POSITION,
              float3 color : COLOR,
              float3 normal : TEXCOORD0,
              float inradius : TEXCOORD1,
              uniform float4x4 mvp_mat : state.matrix.mvp)
{
  float circumradius = HEXRADIUS*inradius;
  float halfcircumradius = circumradius / 2.0f;

  //Generate two axis on surfel plane
  float3 axisU =
    float3(normal.y - normal.z, -normal.x, normal.x));
  //axisV is perpendicular to normal and axisU
  float3 axisV = cross(normal, axisU);

  //Top and Bottom points in inscribed circle
  float3 pa = center + axisV*inradius; //top
  float3 pb = center - axisV*inradius; //bottom

  //End points of hexagon in world space
  float4 p0_w = float4(center + axisU*circumradius, 1);
  float4 p1_w = float4(pa + axisU*halfcircumradius, 1);
  float4 p2_w = float4(pa - axisU*halfcircumradius, 1);
  float4 p3_w = float4(center - axisU*circumradius, 1);
  float4 p4_w = float4(pb - axisU*halfcircumradius, 1);
  float4 p5_w = float4(pb + axisU*halfcircumradius, 1);

  //End points in image space
  float4 p0_i = mul(mvp_mat, p0_w);
  float4 p1_i = mul(mvp_mat, p1_w);
  float4 p2_i = mul(mvp_mat, p2_w);
  float4 p3_i = mul(mvp_mat, p3_w);
  float4 p4_i = mul(mvp_mat, p4_w);
  float4 p5_i = mul(mvp_mat, p5_w);

  //Emit amplified geometry
  emitVertex(p1_i : POSITION, color : COLOR);
  emitVertex(p2_i : POSITION, color : COLOR);
  emitVertex(p0_i : POSITION, color : COLOR);
  emitVertex(p3_i : POSITION, color : COLOR);
  emitVertex(p5_i : POSITION, color : COLOR);
  emitVertex(p4_i : POSITION, color : COLOR);
  restartStrip();
}
```

Appendix B

# Video Material



*SLAM++: Simultaneous Localisation and Mapping at the Level of Objects.*
R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison. CVPR 2013.
http://youtu.be/tmrAh1CqCRo

*SLAM++ Demo.*
System demonstration at the Imperial Festival 2013.
http://youtu.be/6IU4e8yUdis

*Dense Planar SLAM.*
R. F. Salas-Moreno, B. Glocker, P. H. J. Kelly, and A. J. Davison. ISMAR 2014.
http://youtu.be/KOG7yTz1iTA

*Dense Planar SLAM Demo.*
System demonstration running on the Oculust Rift.
http://youtu.be/1ozadUAwE5Q

# List of Figures

# Bibliography

[1] Advanced Micro Devices Inc. Bolt C++ Template Library. 2012. 66, 99, 133

[2] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building Rome in a Day. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2009. 17, 18, 71

[3] A. Agrawal, R. Raskar, and R. Chellappa. What is the Range of Surface Reconstructions from a Gradient Field. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2006. 17

[4] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. What is an object? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 28

[5] Yali Amit and Donald Geman. Shape Quantization and Recognition with Randomized Trees. *Neural Computation*, 9:1545–1588, 1997. 22, 30, 31, 38

[6] E. Ataer-Cansizoglu, Y. Taguchi, S. Ramalingam, and T. Garaas. Tracking an RGB-D Camera Using Points and Planes. In *ICCV CDC4CV Workshop*, 2013. 126, 128

[7] M. Aubry, D. Maturana, A. Efros, B. Russell, and J. Sivic. Seeing 3D chairs: exemplar part-based 2D-3D alignment using a large dataset of CAD models. In *CVPR*, 2014. 33

[8] D. H. Ballard. Generalizing the Hough Transform to Detect Arbitrary Shapes. *Pattern Recognition*, 12(2):111–122, 1981. 29, 39, 98

[9] S. Y. Bao, M. Bagra, Y.-W. Chao, and S. Savarese. Semantic Structure From Motion with Points, Regions, and Objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 97, 126

[10] J.T. Barron and J. Malik. Intrinsic Scene Properties from a Single RGB-D Image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013. 137

[11] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2006. 33

[12] N. Bell and J. Hoberock. Thrust: C++ Template Library for CUDA, 2009. 61, 133

[13] N. Bell and J. Hoberock. Thrust: A Productivity-Oriented Library for CUDA. *GPU Computing Gems: Jade edition*, 2011. 66

[14] P. Besl and N. McKay. A method for Registration of 3D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):239–256, 1992. 33, 54

[15] S. Betgé-Brezetz, P. Hébert, R. Chatila, and M. Devy. Uncertain Map Making in Natural Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1996. 15

[16] 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

[17] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother. Learning 6D Object Pose Estimation using 3D Object Coordinates. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014. 102, 113

[18] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 22, 30, 31, 38

[19] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984. 38

[20] G. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla. Segmentation and Recognition using Structure from Motion Point Clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2008. 36

[21] D. C. Brown. A solution to the general problem of multiple station analytical stereo triangulation. Technical Report Technical Report No. 43 (or AFMTC TR 58-8), Technical Report RCA-MTP Data Reduction, 1958. 17

[22] J. A. Castellanos. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. PhD thesis, Universidad de Zaragoza, Spain, 1998. 15

[23] R. O. Castle, D. J. Gawley, G. Klein, and D. W. Murray. Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007. 96

[24] 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. 126

[25] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. *Image and Vision Computing (IVC)*, 10(3):145–155, 1992. 20, 33

[26] M. Cheng, Z. Zhang, W. Lin, and P. H. S. Torr. BING: Binarized Normed Gradients for Objectness Estimation at 300fps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 28

[27] 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

[28] 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

[29] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(5):603–619, 2002. 32, 106

[30] C. Couprie, C. Farabet, L. Najman, and Y. LeCun. Indoor semantic segmentation using depth information. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013. 37

[31] A. Criminisi and J. Shotton. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer, 2013. 37

[32] M. Cummins and P. Newman. Probabilistic Appearance Based Navigation and Loop Closing. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007. 17, 34

[33] 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. 17

[34] M. Cummins and P. Newman. Highly scalable appearance-only SLAM — FAB-MAP 2.0. In *Proceedings of Robotics: Science and Systems (RSS)*, 2009. 17

[35] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005. 27, 33

[36] A. J. Davison. *Mobile Robot Navigation Using Active Vision*. PhD thesis, University of Oxford, 1998. 15, 16

[37] 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, 95, 96

[38] 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. 16

[39] M. B. Dillencourt, H. Samet, and M. Tamminen. A General Approach to Connected-component Labeling for Arbitrary Image Representations. *Journal of the ACM*, 39(2):253–280, 1992. 131

[40] M. Dou, L. Guan, J. Frahm, and H. Fuchs. Exploring High-Level Plane Primitives for Indoor 3D Reconstruction with a Hand-held RGB-D Camera. In *ACCV Workshop on Color Depth Fusion, in conjunction with 11th Asian Conference on Computer Vision (ACCV)*, 2012. 126

[41] B. Drost, M. Ulrich, N. Navab, and S. Ilic. Model globally, match locally: Efficient and robust 3D object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 29, 98, 100, 101

[42] E. Eade. *Monocular Simultaneous Localisation and Mapping*. PhD thesis, University of Cambridge, 2008. 80

[43] E. Eade and T. Drummond. Monocular SLAM as a Graph of Coalesced Observations. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2007. 15, 19

[44] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Van Gool. Random Forests for Real Time 3D Face Analysis. *International Journal of Computer Vision (IJCV)*, 101(3):437–458, February 2013. 32

[45] G. Fanelli, J. Gall, and L. Van Gool. Real Time Head Pose Estimation with Random Regression Forests. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 617–624, June 2011. 32, 39, 102, 104

[46] G. Fanelli, T. Weise, J. Gall, and L. Van Gool. Real Time Head Pose Estimation from Consumer Depth Cameras. In *Proceedings of the DAGM Symposium on Pattern Recognition*, September 2011. 32, 39

[47] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 35(8):1915–1929, 2013. 36

[48] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 32(9):1627–1645, 2010. 32, 33

[49] P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision (IJCV)*, 61(1):55–79, 2005. 32

[50] 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. 27

[51] A. W. Fitzgibbon. Robust Registration of 2D and 3D Point Sets. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2001. 28, 150

[52] 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)*, 1998. 17

[53] G. Floros and B. Leibe. Joint 2D-3D temporally consistent semantic segmentation of street scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 36

[54] J. Gall and V. Lempitsky. Class-specific Hough Forests for Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 31, 37, 39, 104

[55] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky. Hough Forests for Object Detection, Tracking, and Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33(11):2188–2202, November 2011. 31

[56] J. Gallier. *Geometric Methods and Applications: For Computer Science and Engineering*. Springer, second edition, 2011. 45, 47

[57] A.P. Gee, D. Chekhlov, W. Mayol, and A. Calway. Discovering planes and collapsing the state space in visual slam. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2007. 126

[58] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 6(6):721–741, 1984. 35

[59] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006. 113, 150

[60] R. Girshick, J. Shotton, P. Kohli, A. Criminisi, and A. Fitzgibbon. Efficient Regression of General-Activity Human Poses from Depth Images. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011. 102

[61] B. Glocker, S. Izadi, J. Shotton, and A. Criminisi. Real-Time RGB-D Camera Relocalization. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2013. 31, 130

[62] G Grisetti, G.D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi. Fast and accurate SLAM with Rao-Blackwellized particle filters. *Robotics and Autonomous Systems*, 55:30–38, 2007. 17

[63] A. Handa, T. Whelan, J. McDonald, and A. J. Davison. A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014. 137, 138

[64] C. G. Harris and J. M. Pike. 3D Positional Integration from Image Sequences. In *Proceedings of the Alvey Vision Conference*, pages 233–236, 1987. 15

[65] M. Harris, S. Sengupta, and J. Owens. Parallel Prefix Sum (Scan) with CUDA. In Hubert Nguyen, editor, *GPU Gems 3*, pages 851–876. Addison-Wesley, 2008. 66

[66] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000. 77

[67] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004. 49

[68] W. Hartmann, M. Havlena, and K. Schindler. Predicting Matchability. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 34

[69] X. He, R. S. Zemel, and M. Carreira-Perpinan. Multiscale conditional random fields for image labeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004. 35

[70] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient Response Maps for Real-Time Detection of Texture-Less Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2012. 28

[71] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolidge, N. Navab, and V. Lepetit. Multimodal Templates for Real-Time Detection of Texture-less Objects in Heavily Cluttered Scenes. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011. 27

[72] S. Hinterstoisser, V. Lepetit, S. Ilic, P. Fua, and N. Navab. Dominant Orientation Templates for Real-Time Detection of Texture-Less Objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 27

[73] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model-Based Training, Detection and Pose Estimation of Texture-less 3D Objects in Heavily Cluttered Scenes. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, 2012. 28, 102, 110, 113

[74] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. A. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. J. Davison, and A. Fitzgibbon. KinectFusion: Real-Time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proceedings of ACM Symposium on User Interface Software and Technolog (UIST)*, 2011. 123

[75] A. H. Jazwinski. *Stochastic Processes and Filtering Theory*. Academic Press, 1970. 15

[76] A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 21(1):433–449, 1999. 36

[77] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In *Proc. of Joint 3DIM/3DPVT Conference (3DV)*, June 2013. 123, 125, 127, 128, 131, 134, 138, 139, 140

[78] Y. M. Kim, N. J. Mitra, D.-M. Yan, and L. Guibas. Acquiring 3D Indoor Environments with Variability and Repetition. In *SIGGRAPH Asia*, 2012. 96, 126

[79] 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. 18, 19, 71, 95, 139

[80] K. Konolige and M. Agrawal. FrameSLAM: From Bundle Adjustment to Real-Time Visual Mapping. *IEEE Transactions on Robotics (T-RO)*, 24:1066–1077, 2008. 19

[81] R. Kümmerle, 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. 116

[82] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001. 35

[83] C.H. Lampert, M.B. Blaschko, T. Hofmann, and S. Zurich. Beyond sliding windows: Object localization by efficient subwindow search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008. 28, 34

[84] B. Leibe, A. Leonardis, and B. Schiele. Robust Object Detection with Interleaved Categorization and Segmentation. *International Journal of Computer Vision (IJCV)*, 77(1-3):259–289, May 2008. 31, 32

[85] B. Leibe and B. Schiele. Interleaved object categorization and segmentation. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 759–768, 2003. 31, 35, 36

[86] J. J. Leonard. *Directed Sonar Sensing for Mobile Robot Navigation.* PhD thesis, University of Oxford, 1990. 15

[87] J. J. Leonard and Durrant H. Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3), 1991. 15

[88] J. J. Leonard and Durrant H. Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 1991. 15

[89] V. Lepetit and P. Fua. Keypoint Recognition using Randomized Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(9):1465–1479, 2006. 30, 102, 113

[90] V. Lepetit, P. Lagger, and P. Fua. Randomized Trees for Real-Time Keypoint Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005. 26, 30, 32, 37, 102

[91] V. Lepetit, J. Pilet, and P. Fua. Point Matching as a Classification Problem for Fast and Robust Object Pose Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004. 29, 31, 102, 112

[92] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathmatics*, II(2):164–168, 1944. 52

[93] E. Lindholm, M. J. Kilgard, and H. Moreton. A user-programmable vertex engine. In *Proceedings of SIGGRAPH*, 2001. 63

[94] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. In *IEEE Micro*, 2008. 63

[95] W. E. Lorensen and H. E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proceedings of SIGGRAPH*, 1987. 98

[96] M. I. A. Lourakis and A. A. Argyros. SBA: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software*, 36(1):1–30, 2009. 72, 74

[97] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 1999. 30, 33

[98] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004. 36

[99] F. Lu and E. Milios. Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4(4):333–349, 1997. 17

[100] 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. 20

[101] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441, 1963. 52

[102] J. Martinez-Carranza and A. Calway. Unifying Planar and Point Mapping in Monocular SLAM. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2010. 126

[103] P. F. McLauchlan and D. W. Murray. A Unifying Framework for Structure and Motion Recovery from Image Sequences. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 1995. 18

[104] P. Micikevicius. CUDA Optimization. In *Supercomputing*, 2009. 89

[105] M. Milford and G. Wyeth. Hippocampal Models for Simultaneous Localisation and Mapping on an Autonomous Robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003. 17

[106] M. Milford, G. Wyeth, and D. Prasser. Simultaneous Localisation and Mapping from Natural Landmarks using RatSLAM. In *Australasian Conference on Robotics and Automation*, 2004. 17

[107] M. J. Milford and G. Wyeth. Mapping a Suburb with a Single Camera using a Biologically Inspired SLAM System. *IEEE Transactions on Robotics (T-RO)*, 24(5):1038–1053, 2008. 17

[108] J. M. M. Montiel, J. Civera, and A. J. Davison. Unified Inverse Depth Parametrization for Monocular SLAM. In *Proceedings of Robotics: Science and Systems (RSS)*, 2006. 15

[109] H. P. Moravec. Towards Automatic Visual Obstacle Avoidance. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, page 584, August 1977. 15

[110] H. P. Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. PhD thesis, Stanford University, April 1980. 15

[111] 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. 15

[112] 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. 20, 72, 90, 95, 97, 102, 113, 114, 121, 123, 127, 129, 138

[113] 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. 20, 72, 95, 125

[114] P. Newman. *On the Structure and Solution of the Simultaneous Localization and Map Building Problem*. PhD thesis, University of Sydney, 1999. 15

[115] D. Nistér, O. Naroditsky, and J. Bergen. Visual Odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004. 18

[116] D. Nister and H. Stewenius. Scalable Recognition with a Vocabulary Tree. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006. 34

[117] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, second edition, 2006. 51

[118] NVIDIA. NVIDIA Tesla GPU Computing Technical Brief. http://www.nvidia.com/docs/IO/43395/tesla_technical_brief.pdf, 2006. 59

[119] NVIDIA. Fermi: NVIDIA's Next Generation CUDA Compute Architecture. http://www.nvidia.co.uk/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, 2009. 59, 66

[120] NVIDIA. NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110. http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf, 2012. 59

[121] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast Keypoint Recognition using Random Ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2010. 31

[122] M. Ozuysal, P. Fua, and V. Lepetit. Fast Keypoint Recognition in Ten Lines of Code. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007. 30, 113, 150

[123] M. Ozuysal, V. Lepetit, F. Fleuret, and P. Fua. Feature Harvesting for Tracking-by-Detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2006. 30

[124] 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. 17

[125] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing, Second Edition*. Cambridge University Press, 2 edition, October 1992. 51

[126] J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. P. Amarasinghe, and F. Durand. Decoupling algorithms from schedules for easy optimization of image processing pipelines. In *Proceedings of SIGGRAPH*, 2012. 90

[127] F. T. Ramos, J. I. Nieto, and H. F. Durrant-Whyte. Combining object recognition and SLAM for extended map representations. In *Experimental Robotics*, volume 39 of *Springer Tracts in Advanced Robotics*, pages 55–64. Springer, 2008. 97

[128] N. Razavi, J. Gall, and L. Van Gool. Scalable Multi-class Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1505–1512, 2011. 32

[129] X. Ren, L. Bo, and D. Fox. Indoor Scene Labeling using RGB-D Data. In *Workshop on RGB-D: Advanced Reasoning with Depth Cameras, in conjunction with Robotics: Science and Systems*, 2012. 96

[130] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2006. 73

[131] H. Roth and M. Vona. Moving Volume KinectFusion. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2012. 123

[132] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut": interactive foreground extraction using iterated graph cuts. In *Proceedings of SIGGRAPH*, 2004. 22

[133] S. Rusinkiewicz and M. Levoy. Efficient Variants of the ICP Algorithm. In *Proceedings of the IEEE International Workshop on 3D Digital Imaging and Modeling (3DIM)*, 2001. 54, 106, 113, 129, 150

[134] R. F. Salas-Moreno, B. Glocker, P. H. J. Kelly, and A. J. Davison. Dense Planar SLAM. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014. 22, 23

[135] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison. SLAM++: Simultaneous Localisation and Mapping at the Level

of Objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013. 22, 23, 126, 140

[136] T. Sattler, B. Leibe, and L. Kobbelt. Fast Image-based Localization using Direct 2D-to-3D Matching. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011. 34

[137] T. Sattler, B. Leibe, and L. Kobbelt. Improving Image-based Localization by Active Correspondence Search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2012. 34

[138] C. Schmid and R. Mohr. Local Grayvalue Invariants for Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 19(5):530–535, May 1997. 33

[139] M. Segal and K. Akeley. The design of the OpenGL graphics interface. In *Silicon Graphics Computer Systems*, 1994. 62

[140] T. Sharp. Implementing Decision Trees and Forests on a GPU. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 595–608, 2008. 32, 39

[141] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 32, 37, 39, 102, 104

[142] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene coordinate regression forests for camera relocalization in RGB-D images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013. 33

[143] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008. 36, 108

[144] J. Shotton, J. Winn, C. Rother, and A. Criminisi. TextonBoost: Joint Appearance, Shape and Context Modeling for Multi-Class Object Segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2006. 36

[145] N. Silberman and R. Fergus. Indoor Scene Segmentation using a Structured Light Sensor. In *Workshop on 3D Representation and Recognition, in conjunction with International Conference on Computer Vision*, 2011. 36, 96, 108

[146] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from RGBD images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2012. 36

[147] J. Sivic and A. Zisserman. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2003. 17, 34

[148] R. C. Smith and P. Cheeseman. On the Representation and Estimation of Spatial Uncertainly. *International Journal of Robotics Research (IJRR)*, 5(4):56–68, December 1986. 15

[149] 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, 86

[150] Hauke Strasdat. *Local Accuracy and Global Consistency for Efficient Visual SLAM*. PhD thesis, Imperial College London, 2012. 116

[151] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for RGB-D SLAM evaluation. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2012. 138

[152] Y. Taguchi, Y. Jian, S. Ramalingam, and C. Feng. Point-Plane SLAM for Hand-Held 3D Sensors. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013. 126, 128

[153] J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon. The Vitruvian Manifold: Inferring Dense Correspondences for One-Shot Human Pose Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2012. 32, 113

[154] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proceedings of SIGGRAPH*, 1987. 150

[155] C. Tomasi and R. Manduchi. Bilateral Filtering for Gray and Color Images. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 1998. 128

[156] A.J.B. Trevor, S. Gedikli, R.B. Rusu, and H.I Christensen. Efficient Organized Point Cloud Segmentation with Connected Components. In *3rd Workshop on Semantic Perception Mapping and Exploration (SPME)*, 2013. 131, 132

[157] A.J.B. Trevor, J.G. Rogers III, and H.I Christensen. Planar Surface SLAM with 3D and 2D Sensors. 2012. 126, 128

[158] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle Adjustment — A Modern Synthesis. In *Proceedings of the International Workshop on Vision Algorithms, in association with ICCV*, 1999. 17

[159] Z. Tu, X. Chen, A. L. Yuille, and S. Zhu. Image parsing: unifying segmentation, detection, and recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2003. 35

[160] V. Varadarajan. *Lie Groups, Lie Algebras and their Representations*. Springer-Verlag, 1974. 45

[161] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001. 27

[162] T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. J. Leonard. Kintinuous: Spatially Extended KinectFusion. In *Workshop on RGB-D: Advanced Reasoning with Depth Cameras, in conjunction with Robotics: Science and Systems*, 2012. 95, 123

[163] J. Winn and N. Jojic. Locus: Learning object classes with unsupervised segmentation. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2005. 35

[164] C.M. Wittenbrink, E. Kilgariff, and A. Prabhu. Fermi GF100 GPU Architecture. *Micro, IEEE*, 31(2):50–59, March 2011. 60, 62

[165] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz. Multicore Bundle Adjustment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 90