Imperial College London

Department of Computing

Learned Representations for Real-Time Monocular SLAM

Jan Czarnowski

27th July 2020

Supervised by Prof. Andrew Davison

Co-supervised by Dr. Stefan Leutenegger

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

Pracę dedykuję Mamie i Tacie

— *moim pierwszym nauczycielom*

Abstract

The presence of robotic products in our lives is steadily increasing, with the first commercial home and industrial robots becoming available on the market. While existing systems mostly rely on robust spatial navigation in order to fulfil their roles, one can easily imagine more useful and harder tasks to automate.

In order to perform these tasks autonomously, robots need to be able to understand the geometry and the structure of their surroundings. While this can be achieved using a range of different sensors, monocular cameras are the most cost-effective, ubiquitous and interesting from the research perspective.

The most popular technique used for recovering 3D geometric information from monocular imagery is Visual Simultaneous Localisation and Mapping (SLAM). While traditional landmark-based SLAM systems have demonstrated high levels of accuracy and robustness for trajectory estimation, the maps produced by them do not provide enough information about the environment to enable interactive robotics.

State of the art dense SLAM methods allow obtaining rich scene reconstructions but are fragile and suffer from a number of limitations. A particular problem is the lack of an efficient representation for the observed scenes. Current methods use over-parametrised and geometry-focused representations of the environment, which do not make use of the inherent semantic structure of man-made environments so easily discovered and exploited by human brains.

This thesis aims to explore the recent advances in the field of deep learning in order to address these issues with dense SLAM. New representations based on neural networks are proposed in order to enable more efficient and robust reasoning about the observed scenes. We also demonstrate novel SLAM systems that make use of these representations, pointing towards the next generation of SLAM research.

Acknowledgements

Over the course of studies, I have received support from various individuals that helped to bring this work to life. Here I wish to acknowledge these friends and colleagues for their efforts and contributions.

First and foremost, I would like to thank my wife for her unconditional love and unyielding support during this long adventure. I am grateful for your continuous selfless help through my most difficult times, allowing me to return to the fight with renewed spirit. Thank you for supporting me in my, most often unwise, decisions, and making countless sacrifices that got me to where I am now. You are a big part of why this thesis exists, and for that I shall be eternally grateful.

This thesis would not have been possible without the support of my supervisor, Prof. Andrew Davison, and my co-supervisor Dr. Stefan Leutenegger, who have managed to guide me through the confusing and meandering ways of PhD research. Andy has taught me everything about the academic life and allowed me to grow as a researcher. His drive towards pursuing the most important questions regardless of inconvenience and his passion for live demonstrations have inspired and influenced my career. The encouraging environment created by Andy and his trust in my abilities gave me the freedom to pursue my research interests and have made for a pleasant experience during my years in the lab.

I am forever indebted and grateful to my parents, Iwona and Piotr, for their support since day one, who have taught me about important things in life and who have shaped me into the person I am now. You have created a stable household, which was always there for me when I needed an escape from the research world. I would also like to thank my sister, Julia, for putting up with her older brother and allowing me to feel like a role model to someone.

To Michał Kręczkowski, my closest friend, I send my warmest regards. You have provided continued emotional support during over more than a decade of our friendship. Thank you for our innumerable conversations, which allowed me to unwind and forget my woes.

I wish to state my gratitude to the other two members of my cohort — Tristan Laidlow and Andrea Nicastro, whom I have been lucky to have suffered alongside with. I have learned a lot from you two and most of my research ideas originated from our discussions and whiteboard brainstorming sessions. The countless coffee breaks and desk banter brightened even the most rainy London days. Thank you for your advice, support and encouragement!

I thank numerous colleagues and friends I met during my time in the Dyson Robotics Lab: Ankur Handa, for his mentorship, discussions and continued friendship across time-zones; Robert Lukierski, for teaching me a lot about code, organising amazing parties and providing an undending stream of internet memes; Michael Blösch, who has shown me the academic way by example and with whom I have spent hours gaming online; Ronald Clark, for being the source of many interesting ideas and for your refined sense of aesthetics which I have made use of on several occasions; Iosifina, for running the lab operations incredibly smoothly, dealing with any problems I have managed to come up with and for constant friendly banter.

I would also like to thank all past and present members of the Dyson Robotics Lab, who have provided me with interesting research discussions and much needed inspiration, which had great influence on my research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Contents

1.1 Interactive Robotics and Vision

The presence of robotic products in our lives is steadily increasing and is projected to continue doing so. While solutions like robotic manipulators are widespread in manufacturing, their control algorithms are mostly open loop and work in constrained and adapted environments with very little to no perception involved. If we want to automate more and more tasks outside factories, it is imperative to make robots perform well in a dynamic, uncertain environment. The most basic requirement for the robot is to be able to traverse and recognise the environment it is located in. Some tasks can be enabled by primitive navigation strategies, such as the early versions of the iRobot Roomba vacuuming the room by "bouncing" off its walls and thus, statistically covering the majority of the area over time. In that case the robot only has to perceive the environment through the touch sensors to sense contact with the walls. While simpler robotic jobs can be solved using such straightforward approaches, most of the tasks require navigation to a set location. For this, a map needs to be created that allows to plan an initial path to the desired goal and the immediate scene in front of the robot needs to be perceived in order to avoid any obstacles in its way.

The first commercial mobile solutions are starting to appear on the home robotics market, with the last years seeing a number of new autonomous vacuum cleaning solutions. These products incorporate increasing perception capabilities in order to add useful features such as returning to base for charging, emptying the dust compartment or performing cleaning in certain patterns (e.g. the Dyson 360 Eye). Mobile robots are also starting to appear on the industrial market for scheduled inspection, with the highly agile Spot — the first commercial legged robot released by Boston Dynamics. All of these robotic applications rely on the existence of a robust perception pipeline allowing the robot to construct a map of its surroundings and localise itself within that map. This problem is known in the literature as *Simultaneous Localisation and Mapping* (SLAM).

While existing commercial systems mostly rely on robust spatial navigation in order to fulfil their roles, one can easily imagine more advanced tasks to automate. An immediate example is a domestic robot that is not only limited to vacuuming the floors but is also capable of tidying up objects by returning them to predefined locations or performing chores like washing the dishes or dusting. Especially important are the tasks that can relieve humans from working in dangerous and harmful environments, which were the target of the DARPA Robotics Challenge — a prize competition funded by the US Defense Advanced Research Projects Agency held between 2012 and 2015. The challenge, designed to spur innovation in autonomous robotics, required the robots to perform tasks that could be useful in a real disaster scenario, such as the Fukushima Daiichi nuclear power plant disaster in 2011. The tasks involved human activities such as driving a utility vehicle, opening a valve, entering a building through a door or traversing rubble.

These advanced tasks require machines to interact with and understand much more about their surroundings, which puts more demands on the perception pipeline. Beyond localisation, geometry of the immediate environment has to be estimated in order to execute grasping and manipulation in a safe and efficient manner. Objects have to be recognised in the sensor stream for the robot to reason about them and plan what actions need to be performed. A more general problem emerges here which is appropriately captured by the term *Spatial AI* coined by Davison in [Davison, 2018]. It describes the online task of using vision and possibly other sensors to allow embodied devices, either classical autonomous agents or other devices, to intelligently and usefully interact with their environment.

An example of embodied devices that are not considered robots in the classical sense and in which Spatial AI plays a fundamental role are Augmented Reality (AR) devices, which aim to enhance human visual perception with computer generated information in a realistic manner. The applications of AR range from utility to entertainment, with examples including inserting virtual objects into the scene (e.g. furniture in a room) or displaying art or media on flat surfaces. This is typically achieved by making the user observe the world through a device that captures, modifies and presents images on a digital display. The hardware currently used for that purpose are smartphones, smart-glasses or custom head-mounted displays (HMDs). Examples of dedicated AR products include Microsoft HoloLens and the Magic Leap 1 headset developed by Magic Leap. In order to achieve a realistic outcome the program has to estimate a 3D geometric model of the visible scene, which is required to simulate physical and visual interactions between the virtual objects and reality. Estimating additional parameters like light sources and surface properties allows colouring the inserted virtual objects in a way that simulates reflections and shadows.

The general SLAM problem can be solved by using different sensors — various RGB camera configurations (monocular, stereo pairs, multiple cameras), depth sensors (RGB-D), laser ranging (LIDARs), GPS, touch sensors, inertial sensors (IMUs) or odometry. In a typical SLAM system, information originating from multiple of these modalities is probabilistically fused in order to infer location and map estimates. While such multi-sensor systems are robust and using additional information that is often already available on the robotic platform is always beneficial, this thesis aims to explore the limits of what can be achieved using pure monocular vision. There are many reasons why focusing on a single camera is interesting. Monocular sensors are extremely cost-effective compared to e.g. LIDAR. They are also ubiquitous, with every consumer mobile phone equipped with a single fixed lens camera, which opens up many possibilities for applications that can be deployed globally without requiring any additional hardware. In some situations the type of sensing is not controlled — there exist vast amounts of already recorded footage which can be leveraged only using monocular methods. Estimating geometry and motion from a single camera only is also an interesting research direction in itself. Despite the fact that there is no depth information in purely monocular imagery, humans are still able to retain depth perception and navigate the world, interacting with it with one eye closed. This suggests that it is possible to achieve robust spatial perception even if limited to a single view perspective.

For the reasons presented above, the work presented in this thesis focuses solely on a class of methods named *dense monocular SLAM*, which concern estimating a rich geometric reconstruction together with the sensor trajectory from a stream of RGB images coming from a monocular camera browsing an unknown environment.

1.2 Brief History of Visual SLAM

Sparse SLAM

The early approaches for visual SLAM focused on detecting and tracking a set of landmarks present in the environment – e.g. points in the 3D space. The landmarks are typically created based on their appearance in the camera images and chosen to lay in the areas with strong gradient in both directions (e.g. [Shi and Tomasi, 1994]). These high gradient locations detected in the image are called *keypoints* or *features*. Correspondence between between features originating from different images is established based on their appearance. This is usually achieved by comparing *descriptors* that aim to uniquely characterise the patch around the feature location within the image. Keypoint detection and description has received a lot of attention from the research community which has resulted in a big array of different methods such as Harris Corner Detection [Harris and Stephens, 1988], SIFT [Lowe, 2004], SURF [Bay et al., 2008], FAST [Rosten and Drummond, 2006], BRIEF [Calonder et al., 2010], ORB [Rublee et al., 2011], and BRISK [Leutenegger et al., 2011]. Due to a relatively small number of estimated 3D landmarks, systems based on keypoint detection and tracking are referred to as *sparse SLAM*.

One of the earliest works on incremental visual estimation was done by Harris and Pike in [Harris and Pike, 1988]. The authors presented a pioneering system that sequentially built visual maps using input from a single camera. The system was based on detecting Harris corner features and estimating the location of associated landmarks and was able to create sparse 3D maps from long image sequences. A major issue with that work was that each landmark was being treated as a separate estimation problem, neglecting the correlations that are introduced by co-observing the landmarks from the same camera perspective. The other systems released at that time suffered from the same problem.

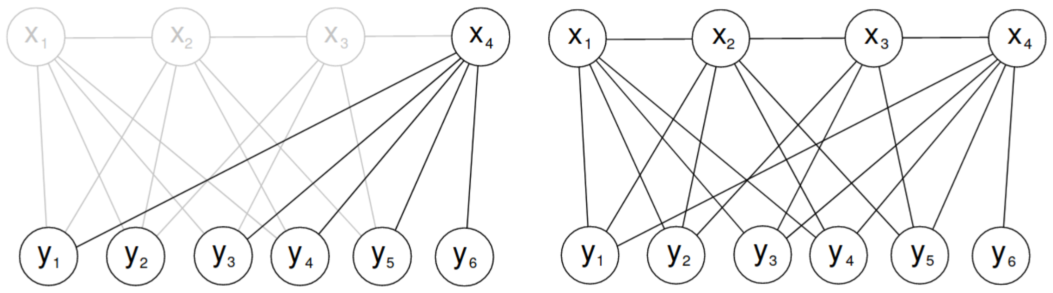
Smith et.al [Smith et al., 1987] and Moutarlier and Chatila [Moutarlier and Chatila, 1989] have later proposed tracking the correlations between quantities in general robot localisation and mapping by estimating a single state vector with an associated covariance matrix. In the proposed method, the landmark 3D positions and the robot pose are jointly estimated in an Extended Kalman Filter (EKF) framework which explicitly represents variable correlations and uncertainty. This approach of estimating all quantities with a single EKF has become the foundation of many future sparse SLAM systems and is still used to this date.

A breakthrough monocular SLAM system called MonoSLAM was introduced in [Davison et al., 2003], which brought the developments from general SLAM research described in the previous paragraph into the pure vision domain. The system used a single EKF to estimate both the 3D positions of planar patch landmarks and the current sensor pose. The 3D ladmark positions were measured in new images by searching for maximum cross-correlation between the RGB intensities of landmark patch and the image. New landmarks were spawned from keypoints detected with [Shi and Tomasi, 1994] and added to the map.

This work was later followed by PTAM [Klein and Murray, 2007], which tracked a larger number of keypoints (e.g. thousands) and broke the joint probabilistic inference by separating the tracking and mapping steps in order to maintain real-time performance. Similar to MonoSLAM, planar patches were initially localised within images using template matching but the mapping backed involved solving the Bundle Adjustment [Triggs et al., 1999] problem rather than updating an EKF.

Many sparse SLAM systems have been released since. A notable example of a modern sparse SLAM system is ORB-SLAM [Mur-Artal et al., 2015], which builds and maintains a landmark map together with a keyframe co-visibility graph. Features are extracted from each camera image and matched against a relevant subset of visible landmarks based on the ORB descriptor [Rublee et al., 2011]. The camera location is estimated by projecting the landmarks onto the current frame and comparing their pixel locations with their detected ones. The system is able to close large loops and perform global relocalisation in real-time and from wide baselines and includes an automatic and robust initialisation from planar and non-planar scenes.

An alternative approach to the SLAM problem is based on optimisation and was championed by Dellaert *et al.* in [Dellaert and Kaess, 2006], which follows the same probabilistic formulation but obtains the solution in a different way. The authors propose the *smoothing* approach, which involves solving for not only the most current robot location, but for the entire robot trajectory up to the current time. While there existed a large number of offline methods that solved for the whole trajectory (e.g. traditional Bundle Adjustment [Brown, 1958, Triggs et al., 1999]), the novelty lied in demonstrating a method for performing this in real-time during robot operation. The



(a) Filtering (b) Smoothing

Figure 1.1: Comparison of graphical models of different approaches to the SLAM problem adapted from [Newcombe, 2012]. In four time steps, the robot moves through positions x_1 to x_4, observing landmarks y_1 to y_6. Filtering (a) estimates the current robot location only, marginalising the previous pose at each update step. Smoothing (b) solves for the whole robot trajectory at each step.

method shows that with careful matrix column ordering, optimised decomposition algorithms and exploiting the sparsity patterns typical for SLAM leads to speed and accuracy improvements over the traditional EKF framework. This is due to the marginalisation performed at each EKF update step that results in irreversible linearisation choices and developments in general matrix operations. A graphical comparison of filtering and smoothing approaches has been presented in Figure 1.1. As one of the methods presented in this thesis is based on the smoothing approach, we will describe it in more detail in later chapters.

Dense Methods

A fundamental problem with the sparse approach to SLAM is that the model of the environment that is built does not provide rich geometric information. The information stored in the map in the form of a set of 3D landmarks can be considered a by-product of tracking rather than a product itself. A sparse cloud of points does not allow to infer much about shapes and surfaces of the observed scene. This is a major hurdle on the way to general interactive robotics, which requires geometric information for action planning or grasping.

A piece of hardware that is intimately connected to 3D computer graphics is the GPU (Graphics Processing Unit). GPUs were originally invented as a way to replace

CPU based ray-tracing with an optimised 3D rasterisation pipeline. To perform this task efficiently they have been designed for mass parallelism, embracing the Single Instruction, Multiple Data (SIMD) paradigm. Since their inception, it has been discovered that the GPUs can be re-purposed to solve other tasks that were previously the domain of high power CPU clusters, for example: physical simulations, scientific computing, video decoding/encoding, cryptography or cryptocurrency mining.

The increased presence of graphics hardware in general computation and dedicated frameworks such as CUDA or OpenCL have inspired a new set of methods called *dense SLAM* that utilise information from all of the pixels in the image and create rich geometric reconstructions. Due to the increased computational cost, these methods typically break the joint probabilistic formulation and rely on interleaving tracking and mapping steps in order to maintain real-time performance. In contrast to sparse methods, correlations between the estimated map (depth) and camera poses are not modelled. A more detailed description of dense SLAM methods can be found in chapter 2.4.

A notable example of a monocular dense system is DTAM [Newcombe et al., 2011], which estimates dense depth by integrating data from multiple frames into a photometric cost volume and optimising it with hand crafted regularisation terms. Tracking the camera location is based on the Lucas-Kanade template matching method [Lucas and Kanade, 1981]. DTAM was later followed by more modern systems like MonoFusion [Pradeep et al., 2013] or REMODE [Pizzoli et al., 2014].

The appearance of commodity depth sensors like Microsoft Kinect [Microsoft Corp, 2010] had a big impact on the popularity of dense SLAM. RGB-D based methods such as KinectFusion [Izadi et al., 2011] produced impressive high quality and accurate 3D maps and ran on a standard desktop computer. Due to the use of a truncated signed distance (TSDF) volume, the size of the reconstructed scene was limited. In ElasticFusion [Whelan et al., 2015] the authors used a surfel-based map representation which, together with loop closure capabilities and map corrections via deformations allowed for greater scalability.

Worth mentioning are also *semi-dense* methods such as LSD-SLAM [Engel et al., 2014] that estimate a larger number of points than sparse SLAM, but typically restrict the computation to high gradient areas of the image yielding considerably sparser maps than dense SLAM. In this thesis we focus on fully dense approaches that we argue are fundamental to future robotics applications.

Shortcomings of Dense SLAM

Even though dense SLAM was originally proposed to increase robustness [Newcombe, 2012], there are fundamental issues that make it fragile and impractical to use. The photometric error widely used in dense methods is measured in the difference between RGB intensities; for example, given two corresponding locations $\mathbf{x}_0$ and $\mathbf{x}_1$ in images I_0 and I_1:

$$\rho_{pho} = ||I_0(\mathbf{x}_0) - I_1(\mathbf{x}_1)||. \tag{1.1}$$

This dependence on scene appearance makes this metric very sensitive to lighting changes and other influences like camera exposure. In the case of using different cameras, there might be other factors impacting the final image intensities, like the gamma response of the cameras. There exist methods that aim to alleviate these problems like estimating an affine transformation or using gradient based metrics (Normalised cross correlation). Later in the thesis, we propose a more robust learned photometric-style consistency metric.

Another issue with the photometric error is that depth is not observable in the textureless regions of images. This allows for reliable estimates only in the regions with sufficient gradient and gives noisy depth estimates elsewhere. This is usually remedied by using simple handcrafted priors on the geometry in the total variation optimisation, e.g. assuming local smoothness — neighbouring pixels ought to have similar depth values. These priors are limited and do not always model reality well. To obtain better reconstructions, it might be beneficial to use more complicated priors that are learned directly from data.

Dense geometry can be represented using point clouds, meshes, occupancy voxel grids or TSDF volumes. This causes the surfaces to be over-parametrised with thousands of points or vertices. We argue that there must exist a better representation of geometry that allows for more efficient inference. We hypothesise that there must exist a perfect semantic compression that utilises a minimal number of parameters to describe geometric entities. This solution would have to involve deconstructing all possible geometry into minimally parametrised "semantic" classes and build the geometric maps with them. As an example, a 3D planar surface like a wall could be described by 4 parameters (plus boundary) instead of regular sampling with points or vertices. Even though the number of entities stored in the SLAM map is significantly smaller in this representation, we still consider it "dense" as it is possible to re-generate the original full 3D geometry from it.

Dense SLAM also suffers from "correlation problem" which was brought up by the research community in the late 80's: "*(...) if the robot uses an observation of an imprecisely known target to update its position, the resulting vehicle position estimate becomes correlated with the feature location estimate.*" [Leonard and Whyte, 1991]. This was solved by the use of a probabilistic representation of the map and robot state within an EKF framework. Due to the fact that optimisation of dense map representations involves a significantly larger number of parameters (for reference: 480'000 for a 800×600 depth map) dense methods typically resort to interleaving tracking and mapping steps effectively choosing not to model the correlations. During camera tracking, the constructed world model is assumed to be correct and similarly, during mapping the estimated camera positions are fixed. Moreover, the large number of parameters make it difficult to apply standard probabilistic methods like the EKF. It is also not clear what probabilistic model could be used to densely represent the geometry of the world, as it is likely not to fit the commonly used Gaussian noise assumption.

As we demonstrate with the work presented in this thesis, these problems can be addressed by introducing deep learning techniques into the SLAM pipeline.

1.3 Deep Learning and SLAM

An example of the early machine learning models is the "perceptron" proposed in [Rosenblatt, 1958]. This work and many others that followed later suggested a different paradigm for creating environment models. Rather than attempting to deconstruct the problem into simple and human understandable principles they acknowledge its complexity and use a generic and biology-inspired model — a parametric hierarchical model of increasingly abstract layers (a *neural network*) is learned directly from data. In *supervised learning*, the network is presented a with a set of examples consisting of an input paired with the desired output and the parameters of the model are optimised to fit the input data set. In contrast, *unsupervised learning* aims to learn to discover alternative data representations and predict desired quantities without requiring explicit ground-truth labels.

Convolutional Neural Networks (CNNs) have their origin in the "neocognitron" described in [Fukushima and Miyake, 1982], where the authors proposed a network built out of two types of layers: convolutional and downsampling. The convolutional layer consists of neurons that compute their activation by applying a filter to corresponding patches in the previous layer. The outputs of convolutional layers are successively downsampled through averaging to reduce the spatial resolution. This architecture allowed the learning of shift-invariant filters for detecting shapes or patterns and drastically reduced the number of parameters in comparison to fully connected networks.

An efficient way of training CNNs was first proposed in [LeCun et al., 1989], which used *back-propagation* in order to learn the model parameters directly from the images to recognise hand-written ZIP Code numbers. In contrast to previously manually designed weights, this method was fully automatic and allowed for easy application to other computer vision tasks. This work was later followed by [LeCun et al., 1998], which proposed the LeNet-5 – a seven layer convolutional network architecture that was trained to classify handwritten digits.

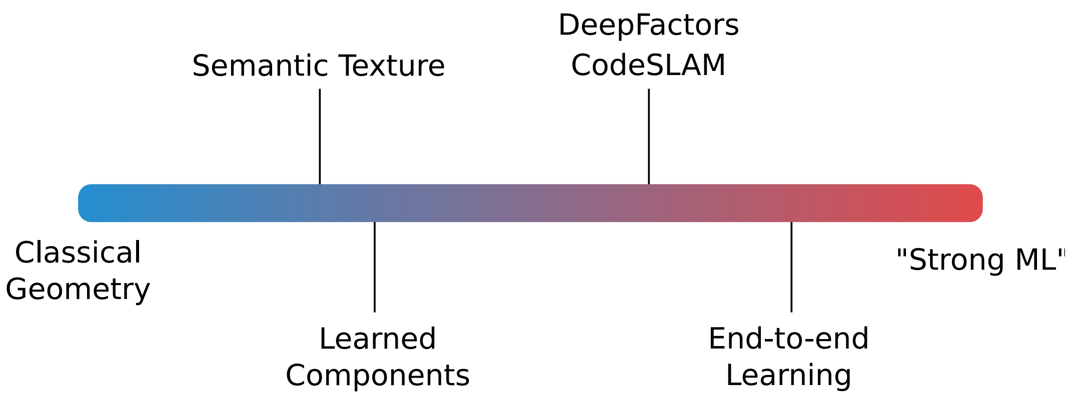


Figure 1.2: A spectrum of different methods of uniting deep learning and SLAM, spanned between classical geometry methods and the "strong ML" approach. The top part of the diagram indicates where each piece of work presented in this thesis lies on the spectrum. The bottom part displays dominant methods found in the literature.

Modern GPUs and GPGPU frameworks such as CUDA have played a major role in the development of this field. Massive parallelisation enabled a significant increase in both the amount of training data and the model complexity, with the number of parameters growing from thousands to tens of millions. The ability to train bigger models led to the realisation that *deeper* networks consisting of a larger number of layers perform better. This can be observed in the network for general image classification presented in [Krizhevsky et al., 2012], which had a similar architecture to LeNet-5, but used more convolutional layers and had a significantly larger number of parameters. This trend was later followed by [Szegedy et al., 2015, Simonyan and Zisserman, 2015, He et al., 2016] and gave rise to the field of *Deep Learning*.

As discussed in the previous section, SLAM could benefit significantly from Deep Learning as many issues connected with dense methods could be remedied by the using priors extracted from data. Conversely, SLAM can potentially aid deep learning through providing data for network supervision, e.g. estimating camera poses or 3D reconstructions from a video.

SLAM and deep learning can be united in a range of different ways which can be arranged on a spectrum spanning between classical methods and pure learning.

Figure 1.2 presents the dominant approaches that will be described below as well as where the systems developed for this thesis fall on the spectrum.

In the simplest form, learned blocks can replace specific components in a standard SLAM pipeline such as feature detectors (e.g. LIFT [Yi et al., 2016]) or optimisation [Clark et al., 2018, Zhou et al., 2018]. Priors learned directly from data enabled solving difficult tasks such as depth prediction from monocular imagery [Eigen et al., 2014, Ummenhofer et al., 2017, Liu et al., 2015, Garg et al., 2016]. Such solutions are often treated as "sensors" and are fused into standard algorithms to increase robustness or densify the reconstructions [Laidlow et al., 2019, Laidlow et al., 2020, Weerasekera et al., 2017]

A different approach, higher on the spectrum, is to have the network directly predict the quantities of interest (*end-to-end* learning). The model is required to learn the necessary intermediate representations/algorithm in order to successfully generalise across data. For example, if regressing camera poses directly from the input image stream, the network is expected to implicitly learn to solve SLAM as an intermediate task. Examples of end-to-end methods include VINet [Clark et al., 2017b], GeoNet [Yin and Shi, 2018] and [Zhou et al., 2017].

With the initial wave of enthusiasm for deep learning came an approach that we call "strong ML", which constitutes the far end of the spectrum. It encapsulates the idea that it is possible to train an agent using, e.g. *Reinforcement Learning* techniques to navigate an environment and carry out tasks based on a predetermined reward [Chaplot et al., 2020, Banino et al., 2018]. This approach is promising and has many advantages: the user has to only specify the desired goal in the form of a reward function and the algorithm discovers the required intermediate representations and actions. Manually crafted models need not to be created and carefully tuned, risking over-fitting the method to the particular task or making simplified assumptions about the environment. This method suggests that SLAM would simply become a by-product, an intermediate task required for the agent to solve in order to obtain the reward.

Although deep learning currently might be seen as the gold standard for many computer vision tasks such as semantic segmentation, object detection, denoising and document parsing, the rate of adoption of learned methods has been slower in SLAM. The general 6 DoF SLAM in the real world still remains an elusive target and the research field has not been as overwhelmed by learning as the other computer vision categories. We believe this is caused by the fact that many aspects of the SLAM problem have been well understood by the research community, such as multi-view geometry, the imaging process or calibration. At the heart of classical methods often lies an effective principle of Bayesian combination of uncertain information. This stands in stark contrast with the models for the human recognition system, whose processes are not fully discovered. The methods for pattern recognition and image classification developed before the learning era were limited and based on primitive modelling. The arrival of Deep Learning allowed training generic template models directly from annotated data and avoiding creating hand-crafted models, which lead to rapid success and increase in performance.

In this thesis, we aim to build towards long term mapping systems that leverage learned representations of scenes that mimic the way humans understand them — not by estimating a geometrically perfect grid of millions of points or voxels but through an explicit understanding of objects, relative to things observed before. We are seeking a representation for 3D geometry that is both useful and efficient in storage, capturing the hierarchical priors that humans make use of. For example, an entity identified as a 'cup' will have limited variations in shape and colour that can be minimally parametrised to succinctly describe and associate any instance that is later observed in sensory data. Such minimal and efficient decompositions of the world can drastically reduce the complexity of estimating 3D geometry, as we will show in Chapter 4. While we did not necessarily focus directly on fixing the shortcomings of dense SLAM presented in Section 1.2, it will be shown that our work does improve them. Instead, we are particularly interested in using novel neural representations to drive fundamental changes to the SLAM pipeline as a whole and point towards a new generation of systems.

An important characteristic of the methods developed within this thesis is retaining the standard iterative optimisation approach of classical SLAM at test-time. We argue that using networks in a simple feed-forward manner can produce promising results in challenging conditions but fails to generalise to real-life scenarios. In that setting, the training set is trusted to encompass all possible variations, a hard condition to meet in the real world. Instead of performing iterative refinement prevalent in classical SLAM those methods produce a one-shot estimate. Work presented in this thesis proposes different ways to incorporate priors learned from data while preserving a standard iterative optimisation scheme.

1.4 Contributions

The contributions made in this thesis resulted in three separate publications. A full list of publications done in conjunction with thesis is provided in the next section. The contribution of each paper is briefly discussed below.

Paper I: SemTex – Semantic Texture for Robust Tracking

This paper argues that robust dense SLAM systems can make valuable use of the layers of features coming from a standard CNN as a pyramid of "semantic texture" which is suitable for dense alignment while being much more robust to nuisance factors such as lighting than raw RGB values. A straightforward Lucas-Kanade formulation of image alignment is used, with a schedule of iterations over the coarse-to-fine levels of a pyramid. The usual image pyramid is replaced by the hierarchy of convolutional feature maps generated from a pre-trained CNN. The resulting dense alignment performance is much more robust to lighting and other variations, as demonstrated by camera rotation tracking experiments on time-lapse sequences captured over many hours. Looking towards the future of scene representation for real-time visual SLAM, it is further demonstrated that a selection using simple criteria of a small number of the total set of features output by a CNN gives just as accurate but much more efficient tracking performance.

SemTex will be described in detail in Chapter 3.

Paper II: CodeSLAM – Learning a Compact Optimisable Depth Representation

The representation of geometry in real-time 3D perception systems continues to be a critical research issue. Dense maps capture complete surface shape and can be augmented with semantic labels, but their high dimensionality makes them computationally costly to store and process, and unsuitable for rigorous probabilistic inference. Sparse feature-based representations avoid these problems, but capture only partial scene information and are mainly useful for localisation only. We present a new compact but dense representation of scene geometry which is conditioned on the intensity data from a single image and generated from a code consisting of a small number of parameters. We are inspired by work both on learned depth from images, and auto-encoders. Our approach is suitable for use in a keyframe-based monocular dense SLAM system: While each keyframe with a code can produce a depth map, the code can be optimised efficiently jointly with pose variables and together with the codes of overlapping keyframes to attain global consistency. Conditioning the depth map on the image allows the code to only represent aspects of the local geometry which cannot directly be predicted from the image. We explain how to learn our code representation, and demonstrate its advantageous properties in monocular SLAM. In this paper, I have contributed to the initial idea of optimising a learned code representation to fit observations and contributed equally to development and experiments, with a focus on the SLAM system.

CodeSLAM will be described in detail in Chapter 4.

Paper III: DeepFactors – Dense Probabilistic Monocular SLAM

This paper presents *DeepFactors*, a real-time SLAM system that builds and maintains a dense reconstruction but allows for probabilistic inference and combines the advantages of different SLAM paradigms. It also presents a tight integration of learning and model based methods through a learned compact dense code representation that drives significant changes to the core mapping/tracking components of the SLAM pipeline. The system achieves greater robustness and precision, which is demonstrated in trajectory and reconstruction experiments. The use of a standard framework allows it to be easily extended with different sensor modalities, which was not previously possible in the context of purely dense SLAM. An efficient C++ implementation and careful choices in the SLAM design enable real-time performance.

DeepFactors will be described in detail in Chapter 5.

1.5 Publications

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

- Czarnowski, J., Leutenegger, S. and Davison, A. J. (2017), **Semantic Texture for Robust Dense Tracking**. *ICCV Workshop, Geometry Meets Deep Learning.* [Czarnowski et al., 2017]

- Bloesch, M., Czarnowski, J., Clark, R., Leutenegger, S., and Davison, A. J. (2018), **CodeSLAM - Learning a Compact, Optimisable Representation for Dense Visual SLAM**. *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* [Bloesch et al., 2018]

- Czarnowski, J., Laidlow, T., Clark, R., and Davison, A. J. (2020), **DeepFactors: Real-Time Probabilistic Dense Monocular SLAM**. *IEEE Robotics and Automation Letters (RA-L).* [Czarnowski et al., 2020]

While not described directly, the following publications were done in conjunction with this thesis:

- Clark, R., Bloesch, M., Czarnowski, J., Leutenegger, S., Davison, A. J. (2018), **LS-Net: Learning to Solve Nonlinear Least Squares for Monocular**

Stereo. *Proceedings of the European Conference on Computer Vision (ECCV)* [Clark et al., 2018]

- Laidlow, T., Czarnowski, J., and Leutenegger, S. (2019), **Towards the Probabilistic Fusion of Learned Priors into Standard Pipelines for 3D Reconstruction**. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. [Laidlow et al., 2019]

- Laidlow, T., Czarnowski, J., Nicastro, A., Clark, R., and Leutenegger, S. (2020), **DeepFusion: Real-Time Dense 3D Reconstruction for Monocular SLAM using Single-View Depth and Gradient Predictions**. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. [Laidlow et al., 2020]

The following video materials provide a visualisation of the algorithms developed in this thesis:

- Semantic Texture for Robust Dense Tracking, https://youtu.be/SkpHccE1eTQ,
- CodeSLAM - Learning a Compact, Optimisable Representation for Dense Visual SLAM, https://youtu.be/PbSggzaZWAQ,
- DeepFactors: Real-Time Probabilistic Dense Monocular SLAM, https://youtu.be/htnRuGKZmZw

The source code for the DeepFactors system has been released and is available under the following link: https://github.com/jczarnowski/DeepFactors

1.6 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 introduces basic notation and the concepts used in dense SLAM and throughout this thesis.

Chapter 3 describes a learned pyramid of 'Semantic Texture' which increases robustness to nuisance factors (such as lighting). The Increased performance of a dense alignment application is shown by camera rotation tracking experiments on time-lapse sequences captured over many hours and some properties of the learned texture are discussed.

Chapter 4 presents CodeSLAM, a new compact but dense representation of scene geometry which is conditioned on the intensity data from a single image and generated from a code consisting of a small number of parameters. A method for learning the code representation is described, and its advantageous properties in monocular SLAM are presented.

Chapter 5 describes DeepFactors, which explores the impact of the compact code representation on dense SLAM pipelines. A new SLAM system is presented that unifies different SLAM approaches in a probabilistic framework while still maintaining real-time performance. The system is evaluated on trajectory estimation and depth reconstruction on real-world sequences.

Chapter 6 concludes the thesis with a discussion of the research presented and suggestions for future work.

Chapter 2

Preliminaries

Contents

In this chapter we review some of the methods and models that will be used in the remainder of the thesis. It begins with introducing our chosen mathematical notation. Section 2.2 describes rigid body transformations in 3D space as well as their minimal parametrisation. Next, the pinhole camera model is described, as well as the mathematical functions abstracting the imaging process. This is followed by a presentation of the fundamental machinery of dense image alignment, a crucial building block of every system presented in this thesis. Section 2.5 introduces Factor Graphs, a graphical representation often found in SLAM and used as the backbone of our DeepFactors system (Chapter 5). Key methods for nonlinear optimisation employed in our work are presented in Section 2.6. We also provide a short introduction to Convolutional Neural Networks and the U-Net architecture. The chapter ends with a description of some of the implementation details of dense SLAM systems.

2.1 Notation

This thesis makes use of the following notation:

2.1.1 General Notation

a	This font is used for scalars.

$\mathbf{a}$	This font is used for M-dimensional column vectors, where a_i is the i^{th} element of the vector:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_M \end{bmatrix}, \quad \mathbf{a}^T = \begin{bmatrix} a_1 & a_2 & \dots & a_M \end{bmatrix}. \tag{2.1}$$

$\mathbf{A}$	This font is for $M \times N$-dimensional matrices, where a_{ij} is the matrix element at the i^{th} row and j^{th} column:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{bmatrix}. \tag{2.2}$$

$\mathbf{1}$	represents the identity matrix.
$\mathbf{0}$	represents the zero matrix.
$(\cdot)_\times$	denotes the cross-product operator that produces a skew-symmetric matrix from a 3-dimensional vector, such that $\mathbf{a} \times \mathbf{b} = \mathbf{a}_\times \mathbf{b}$:

$$\mathbf{a}_\times = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}. \tag{2.3}$$

2.1.2 Probability

$p(\mathbf{x})$	represents the probability density of $\mathbf{x}$.
$p(\mathbf{x}\vert\mathbf{y})$	represents the probability density of $\mathbf{x}$ given $\mathbf{y}$.

2.1.3 Spaces and Manifolds

$\mathbb{R}$	denotes the set of real numbers.
$\mathbb{R}^M$	denotes the vector space of real M-dimensional vectors.
$\mathbb{R}^{M\times N}$	denotes the vector space of real $M \times N$-dimensional matrices.
$SO(3)$	denotes the 3D rotation group.
$SE(3)$	denotes the Special Euclidean group.
$\exp(\cdot)$	denotes the exponential map from $\mathbb{R}^3$ to $SO(3)$.
$\boxplus$	denotes the *box-plus* operator that applies a small perturbation expressed in the tangent space to a manifold state.
$\boxminus$	denotes the *box-minus* operator that determines the difference between two manifold states in the tangent space.

2.1.4 Frames and Transformations

${}_A\mathbf{a}$	The represents the vector **a** expressed in coordinate frame A.
$\mathbf{R}_{AB}$	represents a 3D rotation from coordinate frame B to A, expressed as a rotation matrix (i.e. $\mathbf{R}_{AB} \in SO(3)$).
$\mathbf{T}_{AB}$	represents the homogeneous transformation matrix that transforms homogeneous points from coordinate frame B to A.

2.1.5 Camera Models and Images

f_x	represents the horizontal focal length of the camera, in pixels.
f_y	represents the vertical focal length of the camera, in pixels.
c_x	represents the horizontal coordinate of the camera centre, in pixels.
c_y	represents the vertical coordinate of the camera centre, in pixels.
$\mathbf{K}$	represents the intrinsic camera matrix
$\pi(\cdot)$	denotes the perspective projection function
$\pi^{-1}(\mathbf{x}, d)$	denotes the back-projection function, which maps a 2D pixel coordinate to a 3D Euclidean coordinate using the depth and camera parameters.
$I(\mathbf{x})$	represents the intensity of image I at pixel coordinate $\mathbf{x}$. Images are treated as functions, e.g. $I : \mathbb{R}^2 \longrightarrow \mathbb{R}$ for a grayscale image.

2.2 Transformations

Rigid body rotations of three-dimensional Euclidean space $\mathbb{R}^3$ form a group under the operation of composition. This group, regardless of the actual representation of the rotations is often denoted $\mathbf{SO}(3)$. Since rotations are linear transformations of $\mathbb{R}^3$ that preserve the origin, distance and orientation, the $\mathbf{SO}(3)$ group can be identified with 3×3 orthogonal matrices with unit determinant under matrix multiplication:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \in \mathbb{R}^{3\times3}. \tag{2.4}$$

While this representation has 9 parameters, any rotation can be described using only three degrees of freedom. As rotation matrices live on a manifold optimising the matrix coefficients directly would quickly lead to losing the orthogonality constraint and would yield matrices not representing rotations. Lie groups can be used to optimise a minimal representation $\mathfrak{so}(3)$ of elements living on the tangent space around the identity of the manifold $\mathbf{SO}(3)$. Elements of Lie algebra $\mathfrak{so}(3)$ are anti-symmetric 3×3 matrices. Each skew matrix can be associated with an $\mathbb{R}^3$ vector using the $()_\times$ operator:

$$\boldsymbol{\omega}_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \in \mathfrak{so}(3). \tag{2.5}$$

An exact mapping between elements of $so(3)$ and their $SO(3)$ manifold counterparts is given by the exponential map:

$$\exp : \mathfrak{so}(3) \longrightarrow \mathbf{SO}(3) \tag{2.6}$$

$$\exp(\boldsymbol{\omega}_\times) = \sum_{k=0}^{\infty} \frac{1}{k!}\boldsymbol{\omega}_\times^k = I + \boldsymbol{\omega}_\times + \frac{\boldsymbol{\omega}_\times^2}{2!} + \frac{\boldsymbol{\omega}_\times^3}{3!} + \dots, \tag{2.7}$$

where powers denote *matrix powers*. This can be seen as adding increasingly higher order terms to bring the $\mathfrak{so}(3)$ element onto the manifold. Throughout the thesis, the following notation is alternatively used to represent the exponential map:

$$\mathbf{R}(\boldsymbol{\omega}) = \exp(\boldsymbol{\omega}_\times). \tag{2.8}$$

The exponential map has an analytical closed form solution, called the *Rodrigues formula*:

$$\theta = \sqrt{\boldsymbol{\omega}^T\boldsymbol{\omega}} \tag{2.9}$$

$$\mathbf{R}(\boldsymbol{\omega}) = I + (\frac{sin\theta}{\theta})\boldsymbol{\omega}_\times + (\frac{1 - cos\theta}{\theta^2})\boldsymbol{\omega}_\times^2. \tag{2.10}$$

During optimisation, the boxplus operator $\boxplus$ is used to perturb the rotation using a vector in the minimal representation:

$$\boxplus : \mathbf{SO}(3) \times \mathbb{R}^3 \longrightarrow \mathfrak{so}(3) \tag{2.11}$$

$$\mathbf{C} \boxplus \omega = \mathbf{R}(\boldsymbol{\omega})\mathbf{C} = \exp(\boldsymbol{\omega}^\times)\mathbf{C}. \tag{2.12}$$

The boxminus operator $\boxminus$ can be used to obtain the vector space distance between two rotations:

$$\boxminus : \mathbf{SO}(3) \times \mathbf{SO}(3) \longrightarrow \mathbb{R}^3 \tag{2.13}$$

$$\mathbf{R}_1 \boxminus \mathbf{R}_2 = \log(\mathbf{R}_2^T\mathbf{R}_1). \tag{2.14}$$

In the neighbourhood of the identity element, elements of $\mathbf{SO}(3)$ can be approximated with:

$$\mathbf{R}(\boldsymbol{\omega}) \approx \mathbf{I} + \boldsymbol{\omega}_\times. \tag{2.15}$$

General six degrees of freedom rigid body motion is represented with 4×4 matrices belonging to the $\mathbf{SE}(3)$ group and consisting of rotation and translation:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0^T & 1 \end{bmatrix} \in \mathbf{SE}(3), \tag{2.16}$$

where $\mathbf{R} \in \mathbf{SO}(3)$ and $\mathbf{t} \in \mathbb{R}^3$.

The following notation is used to indicate a transformation that transforms homogeneous points $\mathbf{p} = (x, y, z, 1)^T$ from frame A to frame B:

$$_A\mathbf{p} = \mathbf{T}_{AB}\, {_B\mathbf{p}}. \tag{2.17}$$

Transformations can be chained in the following manner:

$$\mathbf{T}_{AC} = \mathbf{T}_{AB}\mathbf{T}_{BC}, \tag{2.18}$$

and are invertible:

$$\mathbf{T}_{BA} = \mathbf{T}_{AB}^{-1} = \begin{bmatrix} \mathbf{R}_{AB}^T & -\mathbf{R}_{AB}^{\mathbf{T}}\, {_A\mathbf{t}_B} \\ 0^T & 1 \end{bmatrix} \in \mathbf{SE}(3). \tag{2.19}$$

2.3 Camera Model

In the work presented in this thesis, the *pinhole camera model* (Figure 2.1) is used to describe the relationship between 3D point coordinates and their projections onto the image plane. It models the imaging process of an ideal *pinhole camera* which has an infinitesimally small, single point aperture.

Figure 2.1 illustrates the model. The camera coordinate frame has its origin at point $\mathbf{O}$, which is where its aperture is located. Axis z, which points in the viewing direction, is referred to as the *optical axis*. In a physical pinhole camera, the image plane is located behind $\mathbf{O}$, intersecting the optical axis at $-f$ from the origin $\mathbf{O}$, where f is the *focal length*. This causes the resulting projection of the 3D world to be inverted. We simplify the model by placing the image plane at $+f$ and modifying the formulae accordingly, resulting in an equivalent but correctly oriented image. Point $\mathbf{P} \in \mathbb{R}^3$ in Euclidean space is projected through the aperture onto the image plane, giving raise to point $\mathbf{Q} = (x_q, y_q, f)^T$. The specific image location can be calculated by noticing the two similar triangles created by the optical axis and the ray along which the point $\mathbf{P}$ is projected. It therefore follows that:

$$x_q = f\frac{x_p}{z_p} \tag{2.20}$$

$$y_q = f\frac{y_p}{z_p}. \tag{2.21}$$

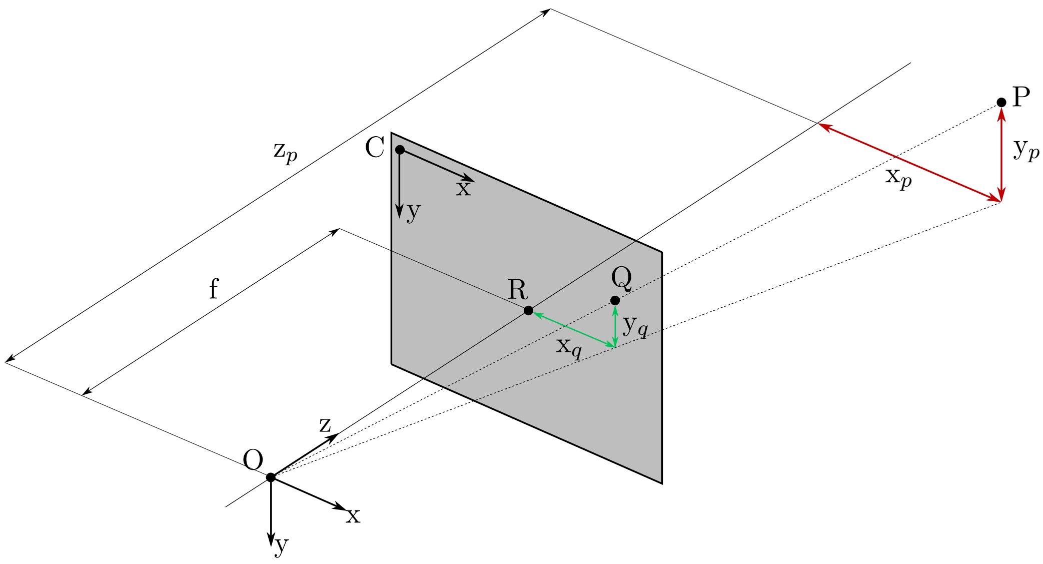


Figure 2.1: An illustration of the pinhole camera model, which describes the relationship between points in the 3D Euclidean space and their projections onto the camera image.

To calculate the coordinates within the actual image, an offset (c_x, c_y) called the *principal point* needs to be applied. In practice, camera lenses also have different focal lengths f_x, f_y for x and y dimensions. We combine all the processes described above into a single *camera projection function* $\pi : \mathbb{R}^3 \longrightarrow \mathbb{R}^2$, which maps a point $\mathbf{x} \in \mathbb{R}^3$ to its corresponding pixel location in the image coordinates:

$$\pi(\mathbf{x}) = \frac{1}{x_3} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \frac{1}{x_3}\mathbf{K}\mathbf{x}, \tag{2.22}$$

where matrix $\mathbf{K}$ is called the *camera intrinsic matrix*. Similarly, if depth d of pixel $\mathbf{x} \in \mathbb{R}^2$ is known, the corresponding location in euclidean space can be recovered using the *unprojection function* $\pi^{-1} : \mathbb{R}^2 \times \mathbb{R} \longrightarrow \mathbb{R}^3$:

$$\pi^{-1}(\mathbf{x}, d) = d \begin{bmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = d\,\mathbf{K}^{-1}\dot{\mathbf{x}}, \tag{2.23}$$

where $\dot{\mathbf{x}} = (x_1, x_2, 1)^T$.

Many effects are not modelled by the pinhole camera model but can be compensated for with coordinate transformations or neglected when dealing with good quality cameras/lenses. In order to deal with non-linear lens distortion not explained by the pinhole camera model, we compensate for it by warping the incoming images using a spherical distortion model at the start of the processing pipeline, which removes the non-linear effect of camera lenses.

2.4 Dense Image Alignment

This section contains a detailed description of dense image alignment, which is a fundamental method that lies at the heart of many dense SLAM methods and will be used in all work presented in this thesis. It is particularly important for the work on Semantic Texture presented in Chapter 3, which will focus on analysing the photometric error and it's convergence basins.

2.4.1 Direct Per-Pixel Cost Function

Image alignment (registration) requires moving and deforming a constant template image T to find the best match with a reference image I. To assess correspondence we must define a measure of image similarity. The sum of squared differences (SSD) of pixel intensities (also called the *photometric error*) is a typical measure used in dense alignment, which is derived from minimising the negative log of a Gaussian likelihood model. This gives the following final objective function:

$$\rho(\mathbf{x};\mathbf{p}) = \frac{1}{2}\sum_{\mathbf{x}} \| I(W(\mathbf{x};\mathbf{p})) - T(x) \|^2. \tag{2.24}$$

The reference image is warped using a parametric warp $W(\mathbf{x};\mathbf{p})$, where $\mathbf{p} = (p_1, p_2, ..., p_n)$ is a vector of warp parameters. This warp relates pixels of the template to the corresponding ones in the reference image. This might also be viewed as warping of the reference image, or predicting its "view" given a particular movement. Movement models popularly used are 2D and 3D rigid body motion models or their pure rotational versions.

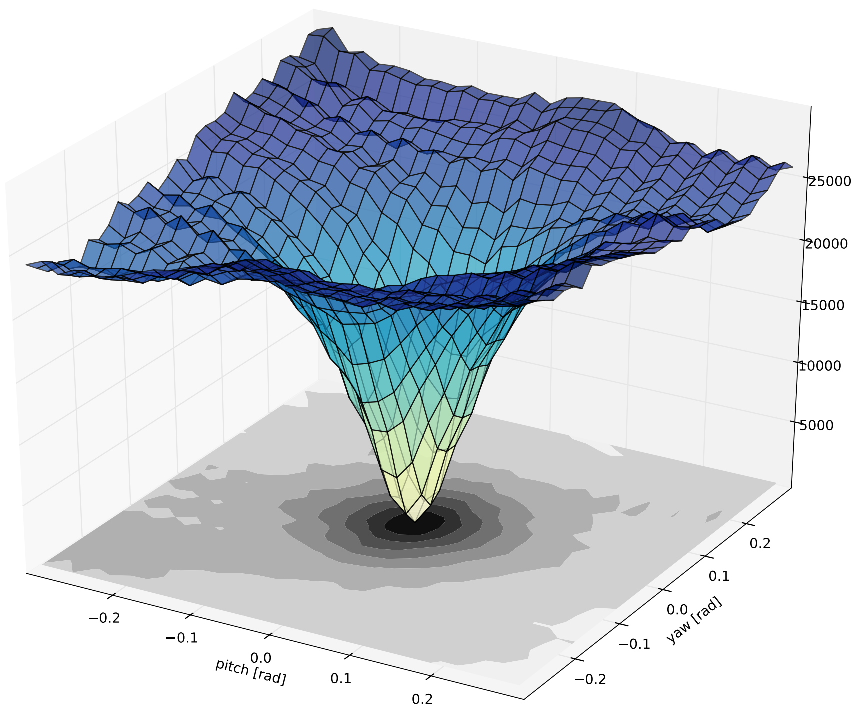


Figure 2.2: Example 2DoF image alignment cost function for pan-tilt camera rotation, with a clear minimum and smooth bowl.

One way to find the optimal parameters is simply ‘exhaustive search’, evaluating the cost function over a full range of quantised parameter combinations and selecting the minimum. Visualising the cost surface produced is insightful; Figure 2.2 presents a 2 degree-of-freedom example for images related by pan-tilt camera rotation. The clear bowl shape of this surface shows that we can do something much more efficient than exhaustive search as long as we start from a set of parameters close enough to correct alignment by following the gradient to the minimum.

2.4.2 Lucas-Kanade Image Alignment

An efficient method for dense alignment was presented in [Lucas and Kanade, 1981] and named after its two authors. The revolutionary contribution of this algorithm is that instead of calculating a global correlation by sweeping through the whole warp parameter space, the algorithm proceeds by iteratively finding better sets of parameters to warp the images into correspondence. This is achieved by using a gradient descent based approach to minimise the total mismatch error. Different optimisation methods can be used to find the minimum, for example: *steepest-descent*, *Gauss-Newton*, *Newton*, or *Levenberg-Marquardt*. The shape of the convergence basin heavily depends on the image content: the amount and type of texture and ambiguities present in the image. Since the cost landscape is usually locally convex in the vicinity of the real translation, a good initialisation is required for the optimisation to successfully converge to the correct solution.

To use the algorithm for camera tracking we minimise the error between the live image I_l and the previous, reference frame I_r.

$$\rho(\mathbf{x};\mathbf{p}) = \frac{1}{2}\sum_{\mathbf{x}} \| I_r(W(\mathbf{x};\mathbf{p})) - I_l(\mathbf{x}) \|^2. \tag{2.25}$$

This optimisation task is non-linear because images are in general not linear in $\mathbf{x}$. Therefore, in order to derive the Lucas-Kanade algorithm, we first linearise Equation 2.25 by expanding the expression $I_r(W(\mathbf{x};\mathbf{p}))$ into a Taylor series around the current warp parameter estimate p, dropping the higher-order terms and assuming the images are grayscale:

$$\rho(\mathbf{x};\Delta\mathbf{p}) = \frac{1}{2}\sum_{\mathbf{x}} \left[I_r(W(\mathbf{x};\mathbf{p})) + \nabla I_r|_{W(\mathbf{x};\mathbf{p})} \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p} - I_l(\mathbf{x}) \right]^2, \tag{2.26}$$

where $\nabla I_r = (\frac{\partial I_r}{\partial x}, \frac{\partial I_r}{\partial y})$ is the reference image gradient. It is evaluated at a warped location $W(\mathbf{x};\mathbf{p})$ which requires sub-pixel interpolation.

To find the minimum, we take the partial derivative of Equation 2.26 and set it to zero:

$$\sum_{\mathbf{x}} \left[I_r(W(\mathbf{x};\mathbf{p})) + \nabla I_r|_{W(\mathbf{x};\mathbf{p})} \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p} - I_l(\mathbf{x}) \right] \left[\nabla I_r \frac{\partial W}{\partial \mathbf{p}} \right] = 0. \tag{2.27}$$

After rearranging the terms and solving for $\Delta\mathbf{p}$ we arrive at the final closed form for the warp parameter update:

$$\Delta\mathbf{p} = \left(\sum_{\mathbf{x}} \mathbf{J}^T \mathbf{J} \right)^{-1} \left(\sum_{\mathbf{x}} \mathbf{J}^T r \right), \tag{2.28}$$

where:

$$\mathbf{J} = \nabla I_r|_{W(\mathbf{x};\mathbf{p})} \frac{\partial W}{\partial \mathbf{p}}$$

$$r = I_l(\mathbf{x}) - I_r(W(\mathbf{x};\mathbf{p})).$$

$\mathbf{H} = \left(\sum_{\mathbf{x}} \mathbf{J}^T \mathbf{J}\right)$ is the Gauss-Newton approximation of the Hessian: a 3×3 matrix. Given $\Delta\mathbf{p}$, the current parameter estimate $\mathbf{p}$ is updated as follows:

$$\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}. \tag{2.29}$$

The approach described in this section is not computationally efficient. As both the Jacobian and the approximate Hessian depend on $\mathbf{p}$, they need to be recomputed at each iteration.

2.4.3 The Inverse Compositional method

A more efficient version of this algorithm, called the *Inverse Compositional method*, which allows for pre-computation of the system Jacobian and Hessian was proposed in [Baker and Matthews, 2004]. This is achieved by swapping the roles of the reference and template image and optimising for an update warp composed with the current warp estimate. This results in the following modified cost function:

$$\rho(\mathbf{x};\mathbf{p}) = \frac{1}{2}\sum_{x} \| I_l(W(\mathbf{x};\Delta\mathbf{p})) - I_r(W(\mathbf{x};\mathbf{p})) \|^2. \tag{2.30}$$

In this approach, at each iteration we are looking for an incremental warp $W(\mathbf{x};\Delta\mathbf{p})$ of the live image and compose it with the current estimate in the following way:

$$W(\mathbf{x};\mathbf{p}) \leftarrow W(\mathbf{x};\mathbf{p}) \circ W(\mathbf{x};\Delta\mathbf{p})^{-1}. \tag{2.31}$$

To derive the warp update we start by expanding the non-linear term of Equation 2.30 into a Taylor series around $\Delta\mathbf{p} = 0$, dropping the higher order terms. We also assume the images are grayscale:

$$\frac{1}{2}\sum_{x} \left[I_l(W(\mathbf{x};0)) + \nabla I_l|_{W(\mathbf{x};0)} \left.\frac{\partial W}{\partial \mathbf{p}}\right|_{\mathbf{p}=0} \Delta\mathbf{p} - I_r(W(\mathbf{x};\mathbf{p})) \right]^2.$$

Assuming that $W(\mathbf{x}; 0)$ is the identity warp, we have:

$$\frac{1}{2}\sum_{\mathbf{x}}\left[I_l(\mathbf{x}) + \nabla I_l \left.\frac{\partial W}{\partial \mathbf{p}}\right|_{\mathbf{p}=0} \Delta\mathbf{p} - I_r(W(\mathbf{x}; \mathbf{p}))\right]^2 .$$

After rearranging terms and solving for $\Delta\mathbf{p}$, we arrive at the closed form solution for the warp parameter update:

$$\Delta\mathbf{p} = \left(\sum_{\mathbf{x}} \mathbf{J}^T\mathbf{J}\right)^{-1}\left(\sum_{\mathbf{x}} \mathbf{J}^T r\right), \tag{2.32}$$

where:

$$\mathbf{J} = \nabla I_l \left.\frac{\partial W}{\partial \mathbf{p}}\right|_{\mathbf{p}=0}$$
$$r = I_r(W(\mathbf{x}; \mathbf{p})) - I_l.$$

This time, the Jacobian $\mathbf{J}$ does not depend on $\mathbf{p}$. Because of this, the Hessian can be precalculated and an iteration consists only of computing the photometric error r and multiplying it with the Jacobian. Moreover, the Jacobian of the warp $\frac{\partial W}{\partial \mathbf{p}}$ is evaluated at $\mathbf{p} = 0$, which often simplifies its computation.

2.4.4 Computational complexity

This section will analyse the computational complexity of the inverse compositional Lucas-Kanade registration algorithm. To summarise the previous section, the following steps need to be performed in the algorithm:

Precompute:

1. Calculate live image gradients ∇I_l
2. Evaluate the warp Jacobian $\frac{\partial W}{\partial \mathbf{p}}$ at $\mathbf{p} = 0$
3. Calculate the approximate Hessian $\mathbf{H} = \sum_{\mathbf{x}} \mathbf{J}^T\mathbf{J}$
4. Calculate the inverse of Hessian $\mathbf{H}^{-1}$

Table 2.6: Computational complexity of the precomputation stage of the Inverse Compositional Lucas-Kanade algorithm

Precomputation step	1	2	3	4	Total
Complexity	$O(N)$	$O(n)$	$O(n^2N)$	$O(n^3)$	$O(n^2N)$

Per iteration:

1. Warp the reference image with current warp estimate $I_r(W(\mathbf{x};\mathbf{p}))$
2. Calculate the residual error image $r = I_r(W(\mathbf{x};\mathbf{p})) - I_l$
3. Calculate $\sum_{\mathbf{x}} \mathbf{J}^T r$
4. Solve $\Delta\mathbf{p} = \mathbf{H}^{-1}(\sum_{\mathbf{x}} \mathbf{J}^T r)$
5. Update the warp parameters $W(\mathbf{x};\mathbf{p}) \leftarrow W(\mathbf{x};\mathbf{p}) \circ W(\mathbf{x};\Delta\mathbf{p})^{-1}$

Assume that the number of pixels in the image is N and the number of warp parameters n. In the precompute stage, the first step is to calculate gradients of the live image. This consists of convolving a kernel with the image. Since this is a per-pixel operation and the kernel sizes are usually small the complexity of this operation is $O(N)$. Step 2 involves evaluating a closed form of the warp Jacobian. Its cost depends on the warp. For the popularly used warps evaluated at $\mathbf{p} = 0$ we can assume cost of $O(n)$. In step 3 the Hessian $\mathbf{H}$ is computed. For this, the Jacobian $\mathbf{J} = \nabla I_l \left.\frac{\partial W}{\partial \mathbf{p}}\right|_{\mathbf{p}=0}$ is calculated for each pixel. This involves multiplying a 1×2 matrix with a $2 \times n$ matrix, which costs $O(2n) = O(n)$. Next, $\mathbf{J}^T\mathbf{J}$ is calculated which also costs $O(n)$ for multiplication and $O(n^2)$ for transposition, composing to a total $O(n^2)$. The total cost of step 3 is $O(n^2N)$. Inverting the Hessian in step 4 costs $O(n^3)$. Table 2.6 presents a summary of the computational cost of the precompute stage.

To perform a single iteration of the algorithm, the reference image I_r needs to be first warped with the current warp estimate $W(\mathbf{x};\mathbf{p})$. For each pixel of the live image I_l we compute $W(\mathbf{x};\mathbf{p})$ and sample I_r at this location. The cost of calculating

Table 2.7: Computational complexity of a single iteration of the Lucas-Kanade algorithm

Iteration step	1	2	3	4	5	Total
Complexity	$O(nN)$	$O(N)$	$O(nN)$	$O(n^2)$	$O(n^2)$	$O(nN + n^2)$

$W(\mathbf{x}; \mathbf{p})$ depends on the warp itself, but for the popularly used ones is usually $O(n)$. Therefore, the total cost of step 1 is $O(nN)$. In step 2, a simple image difference is performed, which costs $O(N)$. Next, using the precalculated $\mathbf{J}$ we calculate $\sum_{\mathbf{x}} \mathbf{J}^T r$ in $O(nN)$. Solving the equation in step 4 involves multiplication of matrices $n \times n$ and $n \times 1$ ($O(n^2)$). Updating the warp parameters consists of inverting $W(\mathbf{x}; \Delta\mathbf{p})$ and matrix multiplication, complexity of which can vary. We will use the cost for an affine warp, which is $O(n^2)$. Table 2.7 presents a summary of the computational complexity of a single iteration of the registration algorithm.

2.4.5 Coarse-to-fine approach

To increase the rate of convergence, low-pass filtering can be used on the images in order to skip higher frequencies, effectively smoothing the cost function, making it possible to jump over local minima [Lucas and Kanade, 1981] and broadening the basin of convergence. As low-pass filtered images can be sub sampled without loss of information, this approach also significantly reduces the computation needed to perform an iteration of the algorithm. The trade off is that the smoothing discards small details in the image, which leads to reduced accuracy of the match. In an extreme case, the features available in the image might be completely suppressed, making it impossible to align the images correctly.

An iterative coarse-to-fine strategy can help to mitigate this problem. A low-resolution, Gaussian-smoothed version of the image can be used to find an approximate solution first. This approximation can be then used in optimisation with a more detailed image. In order to find the final alignment, a few iterations of the alignment algorithm are ran on each level of an image pyramid, in which the images are successively smoothed and downsampled. An example of such Gaussian pyramid is presented in Fig. 2.3.

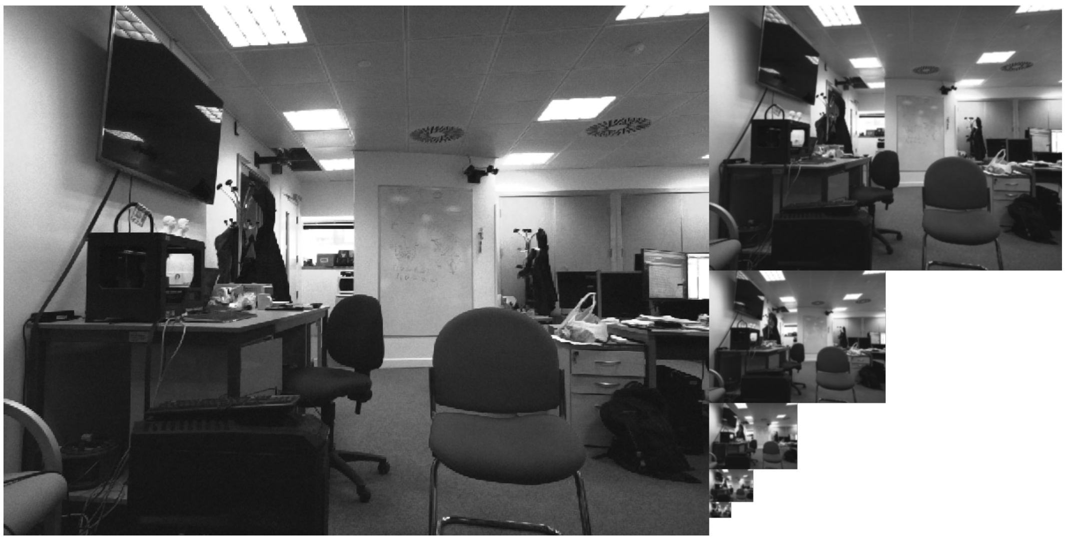


Figure 2.3: A Gaussian pyramid with 6 levels created from the live image for tracking

Since during registration a number of values needs to be calculated for each pixel, the cost of the whole operation is mostly defined by the image size. Assuming the number of pixel in the original image (0^{th} pyramid level) is n, pyramid level i has $\frac{n}{2^{2i}}$ pixels. This leads to significant computational savings, in which the most detailed and costly registration is done at the end to increase the precision. Typically, 4-5 pyramid levels are used in tracking.

In order to create the smoothed image I^*, the original image I is convolved with a Gaussian smoothing kernel:

$$\mathbf{G} = \begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix}.$$

When registering images at lower resolution, special care must be taken to scale the intrinsic matrix $\mathbf{K}$ appropriately. Given the original focal lengths f_x, f_y and centre location u_0, v_0, the intrinsic matrix for pyramid level n has the following form:

$$\mathbf{K} = \begin{bmatrix} \frac{1}{2^n} f_x & 0 & \frac{1}{2^n} u_0 \\ 0 & \frac{1}{2^n} f_y & \frac{1}{2^n} v_0 \\ 0 & 0 & 1 \end{bmatrix}.$$

In the following section, we describe the machinery that can make use of dense image alignment presented in this section in order to estimate the camera localisation jointly with a 3D map of the observed environment.

2.5 Factor Graphs for SLAM

In this section we will describe Factor Graphs — an important graphical representation used in mapping back-ends of many SLAM systems, including DeepFactors presented in Chapter 5. The detailed description and derivation from probabilistic models found below is directly relevant to that work, in particular due to the use of ISAM2, an incremental mapping method.

Factor graphs are a type of graphical models for representing function factorisations [Koller and Friedman, 2009]. They are commonly used to represent probability distribution functions and are well suited to model complex estimation problems such as SLAM or Structure From Motion. Following the example presented in [Dellaert and Kaess, 2017], Figure 2.4 illustrates a toy SLAM problem in a graphical manner. A robot moving in the environment takes bearing measurements of landmarks l_1, l_2 from three different positions: x_1, x_2, x_3. We also assume that there is a measurement anchoring the first robot position in space.

As standard in the SLAM literature, the measurements are modelled to be independent of each other and the robot states are made to depend only on the directly preceding state (the Markov Chain assumption). Figure 2.5 presents a Bayes Net representation of the toy SLAM example. A Bayes Net is a probabilistic graphical model that represents a set of random variables and the conditional dependencies between them as a directed acyclic graph. They are particularly useful to intuitively represent the structure of the joint probability density function. The joint probab-

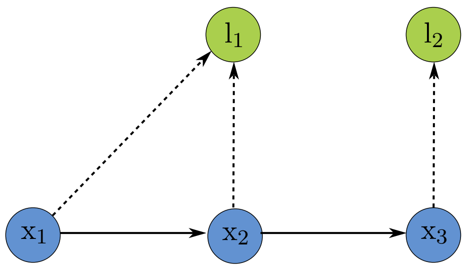


Figure 2.4: An example SLAM problem with a robot taking bearing measurements of two landmarks from three positions. The arrows represent robot motion and the dashed arrows indicate landmark observations. Inspired by [Dellaert and Kaess, 2017].

ility density function $p(X, Z) = p(x_1, x_2, l_1, l_2, l_3, z_1, z_2, z_3, z_4)$ of the robot poses x_i, landmark positions l_j and landmark observations z_k can be obtained as a product of conditional densities of each variable conditioned on their parent nodes. This results in the following factorisation, which can be broken into four distinct types of factors:

$$p(X, Z) = \underbrace{p(x_1)p(x_2|x_1)p(x_3|x_2)}_{a} \cdot \underbrace{p(l_1)p(l_2)}_{b} \cdot \underbrace{p(z_1|x_1)}_{c} \cdot \underbrace{p(z_2|x_1, l_2)p(z_3|x_2, l_1)p(z_4|x_3, l_2)}_{d}$$

a) The Markov Chain $p(x_1)p(x_2|x_1)p(x_3|x_2)$. The conditional densities represent relative motion measurements from wheel odometry.

b) Prior densities on landmarks $p(l_1), p(l_2)$. Typically not known in standard SLAM settings.

c) Absolute measurement on the first pose $p(z_1|x_1)$

d) Bearing measurements on landmarks: $p(z_2|x_1, l_2)$, $p(z_3|x_2, l_1)$ and $p(z_4|x_3, l_2)$.

Based on this model, we can find the unknown state variables X (robot poses and landmark locations) given the measurements Z. In probabilistic terms, this translates to maximisation of the posterior $P(X|Z)$ and is known in literature as

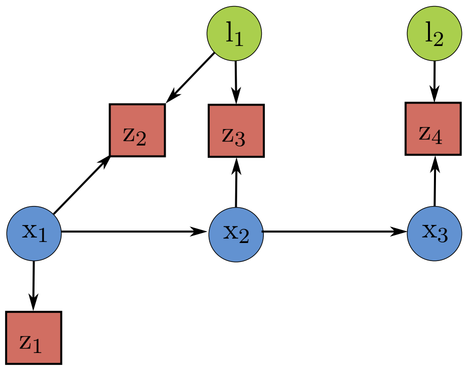


Figure 2.5: Bayes Net corresponding to the toy SLAM example. Nodes represent different random variables from the model: landmarks, robot states and measurements. Measurement variables are illustrated with boxes to indicate that they are observed quantities. Adapted from [Dellaert and Kaess, 2017].

Maximum a Posteriori (MAP) estimation:

$$X^* = \underset{X}{\operatorname{argmax}}\, p(X|Z) = \underset{X}{\operatorname{argmax}}\, \frac{p(Z|X)p(X)}{p(Z)}. \tag{2.33}$$

The $p(Z|X)$ term is called the *likelihood* of the measurements. From the perspective of the optimisation problem the likelihood $p(Z|X)$ is treated as a function of X, parametrised by the given measurements Z. This is opposite to the original formulation of this term and most often does not follow the same distribution. To indicate this, we use $l(X;Z)$ to represent the likelihood term. Moreover, since the measurements are given, the normalising term $p(Z)$ can be dropped, leading to:

$$X^* = \underset{X}{\operatorname{argmax}}\, \frac{p(Z|X)p(X)}{p(Z)} = \underset{X}{\operatorname{argmax}}\, l(X;Z)p(X). \tag{2.34}$$

Due to the change of parametrisation of the likelihood and reformulation of the minimisation problem, the models created by Bayes Nets become mismatched with the actual structure of the inference problem. *Factor graphs* are more suited to model these types of problems, because they can represent factorisation of any function $\psi(X)$ over a set of variables X. Figure 2.6 presents a factor graph representation of the previously introduced toy SLAM example. All model variables are illustrated as nodes of the graph. The measurement variables omitted because they are given and not of interest in the optimisation. Rather than associating each graph node with a

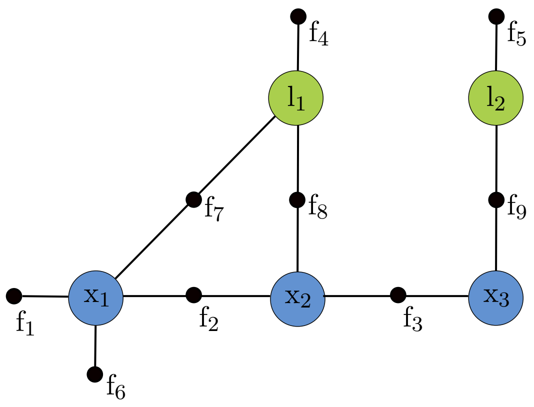


Figure 2.6: Factor graph representation of the toy SLAM inference problem. Black dots represent factor nodes and variables are represented in circles. Adapted from [Dellaert and Kaess, 2017]

conditional density, factor graphs explicitly represent factors with separate nodes in the graph with each factor being connected with an edge to all the variables it is a function of. Similar to Bayesian Networks, the function can be obtained by taking a product of all the factors involved in the graph:

$$f(X) = \prod_i f_i(X_i), \tag{2.35}$$

which, for the example graph leads to:

$$p(X,Z) = \underbrace{f_1(x_1)f_2(x_1,x_2)f_3(x_2,x_3)}_{\text{a}} \cdot \underbrace{f_4(l_1)f_5(l_2)}_{\text{b}} \cdot \underbrace{f_6(x_1)}_{\text{c}} \cdot \underbrace{f_7(x_1,l_2)f_8(x_2,l_1)f_9(x_3,l_2)}_{\text{d}}$$

The correspondence to the previously described probability term types is given by a,b,c,d.

In the case that all random variables are modelled by Gaussian distributions, the inference equation 2.34 can be transformed further by changing it into a minimisation problem and taking the logarithm:

$$\underset{X}{\text{argmax}}\, l(X;Z)p(X) = \underset{X}{\text{argmin}}[-\log(l(X;Z)p(X))] \tag{2.36}$$

$$= \underset{X}{\text{argmin}} \sum_i ||h_i(X_i) - \mathbf{z}_i||^2_{\Sigma_i}, \tag{2.37}$$

where $h_i(X_i)$ is the measurement model and $||\cdot||_{\Sigma_i}$ denotes the Mahalanobis distance. The resulting equation represents a Nonlinear Least Squares problem. In order to solve it, the measurement functions h_i have to be linearised using first-order expansion into Taylor series:

$$h_i(X_i) = h_i(X_i^0 + \Delta_i) \approx h_i(X_i^0) + \mathbf{H}_i \Delta_i, \tag{2.38}$$

where the measurement Jacobian $\mathbf{H}_i$ is the derivative of h_i evaluated at the linearisation point X_i^0. Substituting Equation 2.38 into Equation 2.36 gives:

$$\begin{aligned} \Delta^* &= \underset{\Delta}{\operatorname{argmin}} \sum_i ||h_i(X_i^0) + \mathbf{H}_i\Delta_i - z_i||^2_{\Sigma_i} && (2.39)\\ &= \underset{\Delta}{\operatorname{argmin}} \sum_i ||\mathbf{H}_i\Delta_i - \{z_i - h_i(X_i^0)\}||^2_{\Sigma_i} && (2.40)\\ &= \underset{\Delta}{\operatorname{argmin}} \sum_i ||\mathbf{A}_i\Delta_i - \mathbf{b}_i||^2, && (2.41) \end{aligned}$$

where $\mathbf{A}_i = \Sigma_i^{-1/2}\mathbf{H}_i$ and $\mathbf{b}_i = \Sigma_i^{-1/2}(z_i - h_i(X_i^0))$. By further grouping the individual equations into matrix and vector forms we arrive at the canonical form of Linear Least Squares:

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} ||\mathbf{A}\Delta - \mathbf{b}||^2. \tag{2.42}$$

The update vector Δ^* can be then obtained by solving the *normal equations*:

$$(\mathbf{A}^T\mathbf{A})\Delta^* = \mathbf{A}^T\mathbf{b}. \tag{2.43}$$

2.6 Nonlinear Optimisation

This section presents a very brief description of Nonlinear Optimisation methods, especially focusing on the Nonlinear Least Squares problem that can be found within almost all classical SLAM methods.

The general Nonlinear Least Squares objective function has form:

$$\rho(X) = \sum_i ||f(X_i) - z_i||^2 = ||f(X) - \mathbf{z}||^2. \tag{2.44}$$

This problem cannot be solved directly and requires an iterative method that finds the solution by successively refining the parameters that better satisfy the objective function. This is typically performed by linearising the residuals at the current estimate and calculating an incremental parameter update that minimises the cost function. At each iteration, the objective function $\rho(X)$ is linearised at the current parameter estimate X^*:

$$\bar{\rho}(\Delta X) = ||\mathbf{J}\Delta X - \mathbf{r}||^2, \tag{2.45}$$

where $\mathbf{J} = \frac{\partial f}{\partial X}|_{X^*}$ and $\mathbf{r} = \mathbf{z} - f(X^*)$.

The *Gradient Descent* method calculates the update step by using the direction of the steepest descent at the current estimate X^*:

$$\Delta X = -\alpha \frac{\partial \rho(X)}{\partial X}\Big|_{X^*} = -2\alpha \mathbf{J}^T \mathbf{r} \tag{2.46}$$

where α is called *step size*. Larger step sizes increase the speed of convergence but can also lead to instability. A decreasing schedule is often used for α in order to balance speed and precision of convergence.

The *Gauss-Newton* (GN) method is a second-order method that improves on Gradient Descent by approximating the Hessian of the function. It provides faster convergence but the quadratic approximation can be invalid, which might lead to diverging from the optimum. The update in this method is obtained by solving the normal equations:

$$\Delta X = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{r}. \tag{2.47}$$

A more robust version of GN called *Levenberg-Marquardt* (LM) has been proposed in [Levenberg, 1944] and later modified in [Fletcher, 1971]. This method combines the advantages of the Gauss-Newton and Gradient descent algorithms by interpolating between the two. Because of that, it is also known as a *trust-region* method. To obtain the parameter update, a modified (*damped*) version of the normal equations is used:

$$[\mathbf{J}^T\mathbf{J} + \lambda diag(\mathbf{J}^T\mathbf{J})]\Delta X = \mathbf{J}^T\mathbf{r}. \tag{2.48}$$

The damping factor λ is adjusted at each iteration depending on the quality of the quadratic fit. For larger values of λ, the step will be taken approximately in the direction of the gradient. The Levenberg-Marquardt method tends to be slightly slower than Gauss-Newton for well-behaved functions, but is more robust in other cases.

2.7 Deep Neural Networks

In this section we introduce the basic definitions and architectures of Deep Neural Networks, which are fundamental to the work presented in this thesis. We begin with a brief description of Convolutional Neural Networks, a network structure that has enabled the success of machine learning in image-based tasks. This is then followed by an introduction of the U-Net architecture, used extensively in our work.

Neural Networks are trained to approximate a function $f(\mathbf{x})$ using a parametric generic model $f(\mathbf{x};\theta)$. Due to the large number of parameters in the Deep Neural Network models, this is usually achieved by using a first-order optimisation method to minimise the prediction error over a dataset of examples. To generalise outside the training data, the dataset must encompass all possible variations of inputs. Because of that, *stochastic gradient descent* is typically used which performs the optimisation on *mini-batches* rather than on all examples at once [Goodfellow et al., 2016].

2.7.1 Convolutional Neural Networks

Currently, the majority of neural networks used in computer vision are built using convolutional layers. Convolutional networks draw inspiration from biology, with their connectivity patterns following the organisation of the animal visual cortex [Fukushima, 1980, Hubel and Wiesel, 1968, Matsugu et al., 2003]. A typical network is built out of a series of layers: for each, a set of convolutional filters is learned that is convolved with the input spatial data, and the result is passed to the next layer [Goodfellow et al., 2016]. Convolutional layer outputs are often followed by a non-linear *activation function*, which is to simulate the activation of neurons in the brain.

The filters are each applied to the input with a specified stride S by computing the dot product of the target region with their coefficients. Figure 2.7 presents a one

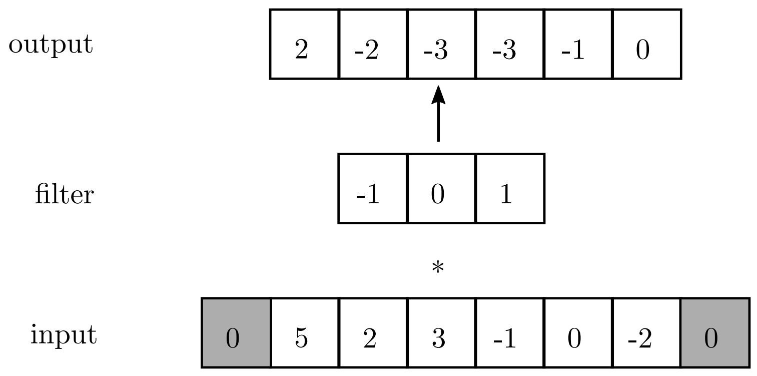


Figure 2.7: A one dimensional example of a convolutional layer applying a gradient filter of size 3 with stride 1. Padding the input tensor with 0's allows to achieve the same dimension in the output tensor.

dimensional example of applying a convolutional layer with a filter (or *kernel*) of size $K = 3$ using a stride $S = 1$. Compared to the pure mathematical definition of convolution, this method does not mirror the kernel coefficients. The filter is applied at each pixel of the input, producing an output activation. Padding the input with zeros allows the filter to be applied at the data boundaries and to preserve the input dimensions. The size of the output tensor can be calculated using the following formula:

$$M = \frac{N - K + 2P}{S} + 1, \tag{2.49}$$

where N is the input dimension and P is the size of padding. It can be seen that with stride 1, for the input and output array dimensions to be the same, the input padding needs to be $P = (K - 1)/2$.

Figure 2.8 presents example activations produced by a CNN trained for image classification. Typically, the first CNN layers extract small features such as edges and coloured blobs, followed by general shapes and partial objects in the deeper layers [Qin et al., 2018].

2.7.2 The U-Net Architecture

One of the most popular network models is the U-Net, proposed in [Ronneberger et al., 2015] for biomedical image segmentation. Figure 2.9 illustrates its architecture

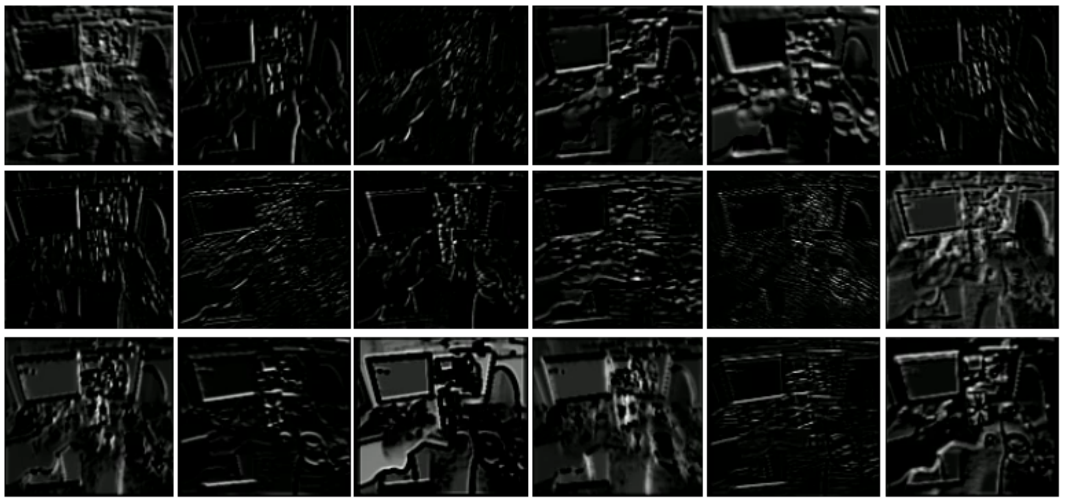


Figure 2.8: Example feature activation maps produced at the fourth layer of Convolutional Neural Network trained for image classification (VGG-16). Each feature map is the result of convolving a learned filter with the preceding layer. The filters of early layers typically learn to detect coloured edges or corners.

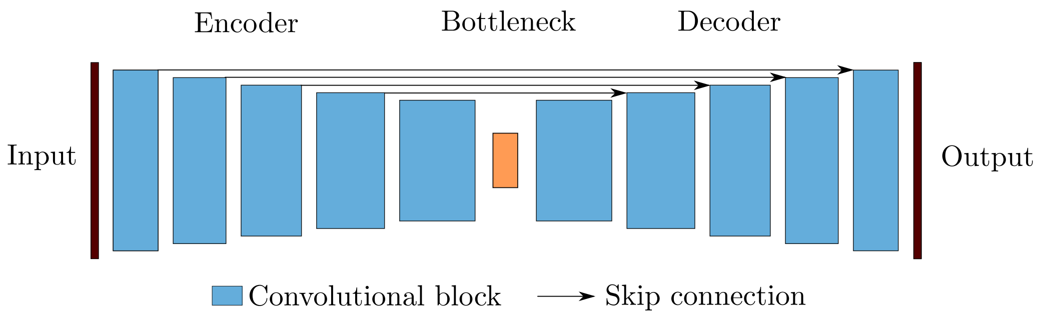


Figure 2.9: Architecture of a U-Net. The input is transformed by convolutional blocks consisting of multiple dimension preserving convolutional layers followed by activation functions. Strided convolutions reduce the *tensor* dimensions after each block. Skip connections allow the decoder part of the network to access high dimensional spatial information.

– an encoder-decoder structure with a bottleneck in between. Autoencoders are a type of a neural networks that are used to learn efficient encodings of data domains in an unsupervised manner. They are typically trained to significantly reduce the dimensions of the input space, transforming it into a compact 'code'. The input is successively transformed with convolutional blocks into features with reduced spatial resolution and increased depth. Each convolutional block consists of multiple convolutional layers followed by an activation function. Resolution reduction is achieved by using an increased stride. Upon reaching the minimum spatial resolution, the bottleneck, the process is reversed. The decoder uses sequential convolutions and up sampling to produce the desired output resolution. Since the decoder often has similar but mirrored structure to the encoder, the network resembles an hourglass or a U-shape, depending on the visualisation method. Forcing the data through a small bottleneck ensures that the network utilises global image context but also causes the detailed spatial information to be lost, degrading the prediction quality. Skip connections, either concatenating or adding the outputs of two layers, between the convolutional layers of the encoder and the decoder aim to remedy this by giving the decoder access to the original information at higher resolutions. Concatenation based skip connections were used in all the networks used in this thesis.

2.7.3 Variational Autoencoders

Variational Autoencoders (VAEs) have been used in both CodeSLAM (Chapter 4) and DeepFactors (Chapter 5) and are briefly described in this section. They are a class of networks used to learn generative models of data and have demonstrated impressive results, e.g. producing realistic images of human faces and allowing to interpolate between them [Kingma and Welling, 2014].

Given a dataset $X = \{x^{(i)}\}_{i=1}^{N}$ of N samples of some random variable x, the method assumes that the data is generated by a certain unknown process that involves an uncertain unobserved latent variable $z \in \mathbb{R}^M$. For example, if x is an image from the set of all possible images of human faces, z might represent features not observed during training, such as race, age, pose, gender or expression. In this

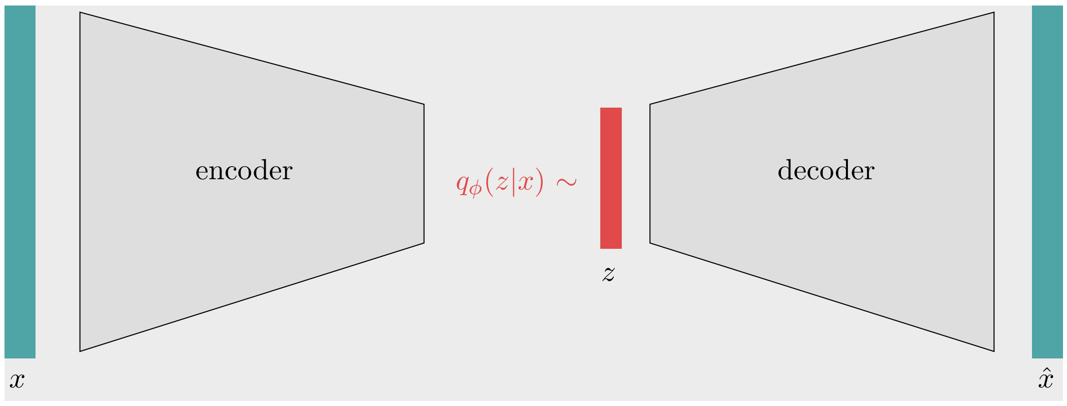


Figure 2.10: Architecture of a Variational Autoencoder network. The parameters of the approximate posterior $q_\phi(z|x)$ are produced for the input x by the encoder. This distribution is then used to sample a latent variable z, which is transformed by the decoder $p_\theta(x|z)$ to generate the reconstruction of the input $\hat{x}$.

model, a value $z^{(i)}$ is generated from a prior distribution $p(z)$ and $x^{(i)}$ is generated from some conditional distribution $p_\theta(x|z)$, parametrised with θ.

Finding the ML or MAP estimate for the parameters θ allows mimicking the unknown random process and generating new, artificial data that resembles the original data. Since the true posterior $p_\theta(z|x)$ is typically intractable, the method introduces a recognition model $q_\phi(z|x)$ as its approximation. This recognition model allows to identify the hidden variables $z^{(i)}$ (also called the 'code') for a particular data example $x^{(i)}$ and is called the *encoder*. Similarly, the likelihood $p_\theta(x|z)$ is called the *decoder*, since it produces a distribution of x given a latent code z. The prior is assumed to be a centred isotropic multivariate Gaussian distribution $p(z) = N(z; O, I)$ and the approximate posterior $q_\phi(z|x)$ and the likelihood $p_\theta(x|z)$ are modelled as a multivariate Gaussian with diagonal covariance whose parameters μ and σ are computed from a neural network.

Figure 2.10 presents the architecture of the network. During training, the latent variable $z^{(i)}$ is sampled from the approximate posterior $q_\phi(z|x)$ determined by the network for each data point $x^{(i)}$ and is then used by the decoder $p_\theta(x|z)$ to generate the reconstructed version of the data $\hat{x}$. The network is trained using the following variational loss:

$$L(x, \theta, \phi) = ||x - \hat{x}||^2 + KL[q_\phi(z|x)||p(z)], \tag{2.50}$$

where KL denotes the Kulback-Leibler divergence between the two distributions. The additional loss term fulfils the role of a regulariser, ensuring that the distributions of the latent variable are continuous and complete (produce a meaningful reconstruction for each point sampled from a given distribution).

2.8 System Building

Building any kind of SLAM system is a challenging task, especially one that integrates multiple components such as dense tracking, mapping or neural networks. A great part of the novelty of the work presented in this thesis is the knowledge gained by building real-time Dense SLAM systems that utilise deep neural networks. This section provides an overview of the common structures of software developed for this thesis.

2.8.1 Software Architecture

When writing SLAM systems, we follow a practice inspired from the Model-View-Controller (MVP) software design pattern and encapsulate the core algorithm into a single library component, making it independent of the user interface and other system-specific choices. This allows exchanging the visualisation easily and allows to use our SLAM as a sub-component of a bigger system or during functional testing. The core library implements an API interface that accepts new camera frames and informs the caller about new pose or geometry estimates, usually through invoking preconfigured callback functions or direct query.

The structure of an example core library of a dense, deep learning enabled SLAM system is presented in Figure 2.11. The main algorithm logic, such as deciding when to create a new keyframe, run relocalisation or close loops is contained within the ‘system’ group. The task-specific implementation is encapsulated within components

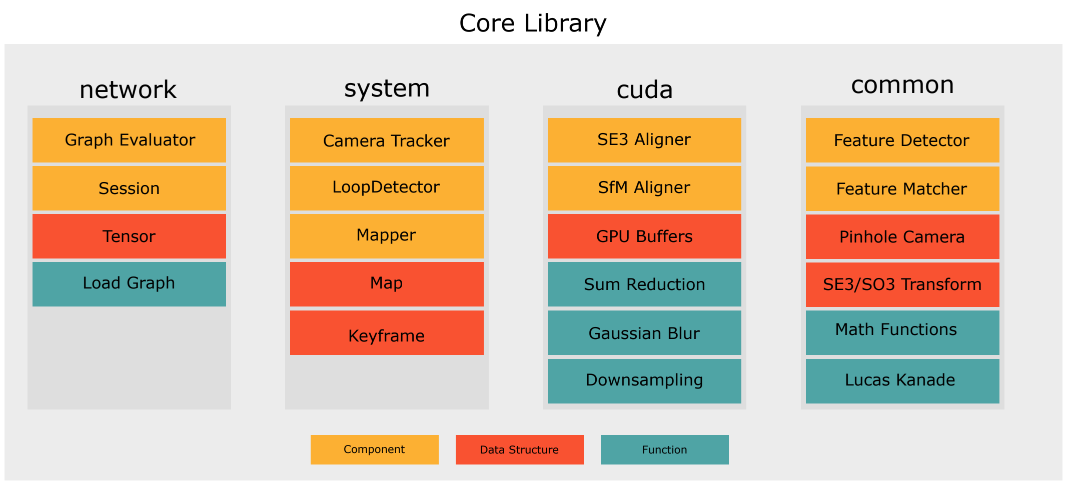


Figure 2.11: Subsystems and components of an example dense SLAM library divided by their functionality. The 'system' group contains the main algorithm (e.g. keyframe creation/management) which delegates work to its various sub-components: Camera Tracker, Loop Detector and Mapper. These components make use of items from other groups: network, for evaluating trained models; cuda, for fast parallel implementations of image operations; common, for general abstract mathematical functions and algorithms.

of this group, e.g. a Camera Tracker estimating the current camera position against the latest keyframe or a Mapper that optimises the keyframes for global geometrical consistency. To fulfil their tasks, these classes use components from other groups. The network group contains functionality that allows to load a trained network graph and evaluate it on a GPU Session to obtain predictions in form of Tensors. The cuda components include parallel implementations of various image processing tasks, such as downsampling or computing gradients. The dense image alignment method that forms the basis of camera tracking (SE3 Alignment) or mapping (SfM Alignment) is typically also performed on the GPU, by first calculating per-pixel Jacobians of the photometric error with respect to the quantity of interest and later reducing the matrices into a single system of normal equations. Often, there are many various functions and components are reused across the whole SLAM system. These are separated in a 'common' group that contains, for example, mathematical function implementations and classes representing rigid body transformations or the camera model.

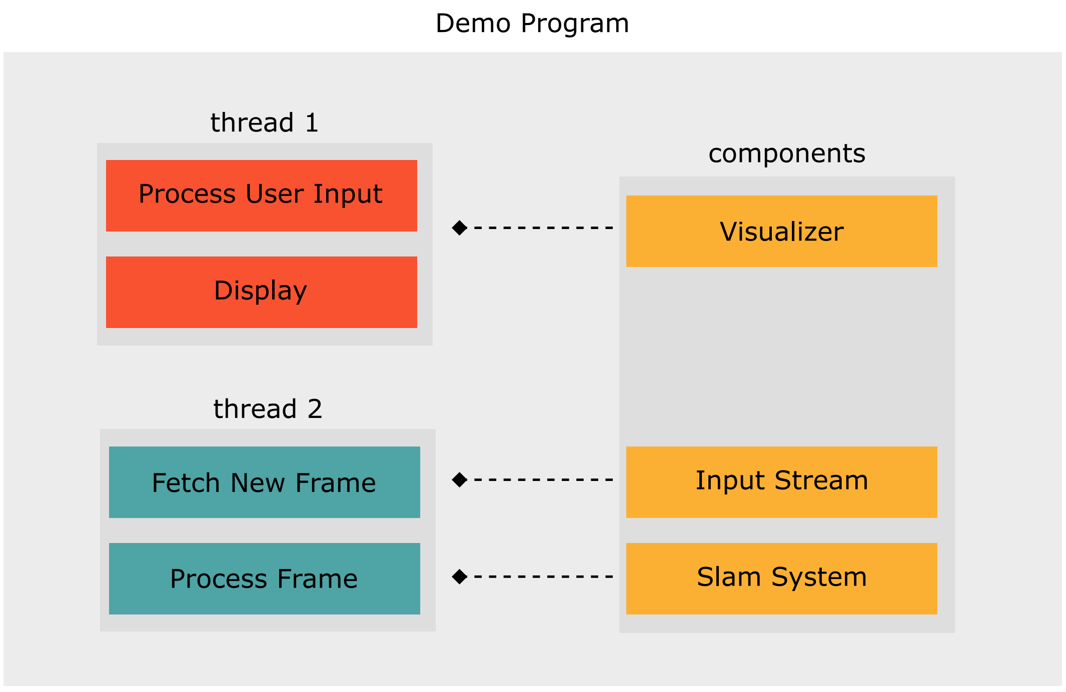


Figure 2.12: An example architecture of a live SLAM demo. The program, separated from the core SLAM library, consists of two threads. The first thread handles interaction with the user and displays a visualisation of the current state of the algorithm using the Visualizer component. The second thread fetches new images from hardware using the Input Stream and feeds them to the Slam System.

In the research setting, the most common program that will use the core SLAM library is the live demo. In our architecture, the demo program serves as an application layer responsible for integrating a specific form of data visualisation, a source of data and the SLAM algorithm. Figure 2.12 presents an overview of an example live demo program, which consists of two parallel threads. The first thread runs the chosen Visualiser allowing the user to interact with the system and inspect the results. The second, main thread, handles fetching latest camera frames using the Image Stream component and feeds them to the SLAM system. Delegating the visualisation loop to a separate thread allows uninterrupted interaction with the GUI, independent of the SLAM processing rate.

In our software, the Image Stream component was designed using the Abstract Factory pattern. Various sources of image sequences implement a single common abstract interface, which allows to seamlessly run the program on various different

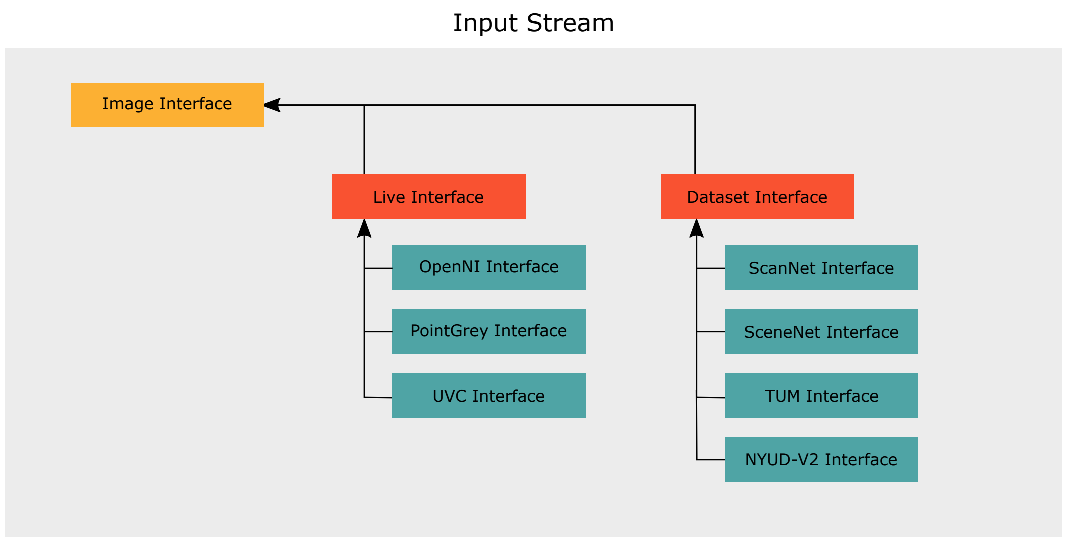


Figure 2.13: Class hierarchy for the Input Stream component. The leaf nodes in the graph implement the two abstract interfaces: Live Interface and Dataset Interface, both of which inherit from the generic Image Interface. Arrows denote class inheritance.

datasets or cameras. This pattern is nicely extensible as adding a new implementation does not require changes to any other parts of the system — each concrete implementation registers itself with the abstract factory. Since each interface is registered using a short identifier (e.g. 'tum' for the TUM Interface) a function *CreateInterfaceFromUrl* can be implemented that takes in a URL string in the form of *id://params* and returns the appropriate interface. This can be used to let the user specify input sources through command line arguments. The inheritance diagram of the Input Stream component has been presented in Figure 2.13.

2.8.2 Deep Learning Integration

The typical workflow for developing programs combining deep learning and SLAM is to perform training in python and afterwards deploy the model for use in a high performance C++ implementation. We have used the TensorFlow framework for training all the networks used in this thesis. At the time of writing the software for this thesis, there were limited options for deploying a trained model for use in a different language. Our TensorFlow networks weights were saved during training using 'tf.Saver', which stores the learnable parameters in a binary format together with a description of the computational graph in the form of a Protocol Buffers file. To be used in a C++ program, the saved model has to be 'frozen', reading the weights from the binary file and store them within the computational graph description as constants. A model processed in such way can be loaded and evaluated using the TensorFlow C API. Since this API only supports loading a single computational graph, all networks to be used in the SLAM system have to be merged into a single graph which can later be selectively evaluated to obtain desired quantities.

Both deep learning and dense SLAM rely on the GPU for carrying out critical computation. TensorFlow abstracts the hardware used for parallel computation using Google's StreamExecutor library, which serves as a wrapper for CUDA and OpenCL and allows running the same computational kernels using either of these technologies. The StreamExecutor library is written directly on top of the low level CUDA Driver API and assumes sole control of the device. This causes problems with sharing the GPU between the CUDA kernels used e.g. for calculating per-pixel Jacobians of the photometric error and the TensorFlow network. Each GPU process is connected to a separate CUDA context, which is similar to the context of a standard operating system process and contains essential information such as addresses of the allocated memory blocks. In order for TensorFlow and our CUDA-based SLAM optimisation to coexist we have separated the two contexts using a feature of the CUDA API called the 'context stack', essentially isolating TensorFlow in its own context, giving it the impression it has sole control over the device. This was achieved by initialising the main CUDA context for SLAM computation first, removing it from the stack, initialising the TensorFlow library and finally returning the original context on top of the stack. The SLAM context is then 'popped' off the stack using a convenience class (Listing 2.1) for network evaluation, exposing the context that was used to initialise TensorFlow.

```
class ScopedContextPop {
  CUcontext ctx_;

public:
  ScopedContextPop() {
    // pop context on construction
    cuCtxPopCurrent(&ctx_)
  }

  ~ScopedContextPop() {
    // push context on scope exit
    cuCtxPushCurrent(ctx_);
  }
};

// Example usage:
void FunctionUsingTensorFlow() {
  ScopedcontextPop pop;
  // Run the network
}
```

Listing 2.1: The ScopedContextPop class, used to manage the CUDA context stack to enable sharing the GPU between TensorFlow and CUDA computation

Chapter 3

Semantic Texture for Robust Dense Tracking

Contents

3.1 Introduction

As described in Chapter 2.4, dense SLAM relies on whole image alignment of live images with the reprojected dense texture of the reconstruction. However, using raw RGB values in persistent dense scene representations over long time periods is problematic because of their strong dependence on lighting and other imaging factors, causing image to model alignment to fail. While one thread of research to mitigate this involves getting closer to the real physics of the world by modelling lighting and surface reflectance in detail [Jachnik et al., 2012, Whelan et al., 2016], this is very computationally challenging in real-time. The alternative, which is described in this chapter, is to give up on representing light intensity directly in scene representations, and instead to use transformations which capture the information important for tracking but are invariant to nuisance factors such as lighting.

A long term goal in SLAM is to replace the raw geometry and appearance information in a 3D scene map by high level semantic entities such as walls, furniture, and objects. This is an approach being followed by many groups who are working on semantic labelling and object recognition within SLAM [Hermans et al., 2014, Valentin et al., 2013, McCormac et al., 2017a], driving towards systems capable of scene mapping at the level of nameable entities (a direction pointed to by the SLAM++ system [Salas-Moreno et al., 2013]).

What we argue here, and make the first experimental steps to demonstrate, is that there is a range of very useful levels of representation for mapping and tracking in between raw pixel values and object level semantics. The explosion of success in computer vision by Convolutional Neural Networks, and work on investigating and visualising the levels of features they generate in a variety of vision tasks (e.g. [Simonyan et al., 2013]), has revealed a straightforward way to get at these representations as the outputs of successive levels of convolutional feature banks in a CNN.

This chapter demonstrates that dense alignment, the most fundamental component of dense SLAM, can be formulated simply to make use not of a standard image pyramid, but the responses of the layers of a standard CNN trained for classification; and that this leads to much more robust tracking in the presence of difficult conditions such as extreme lighting changes. We demonstrate our results in a pure rotation SLAM system, where long term tracking against keyframes is achieved over the lighting variations during a whole day. Detailed experiments on the convergence basins of alignment at all pyramid levels are performed, comparing CNN pyramids against raw RGB and dense SIFT. It is also shown that just as good performance can be achieved with small percentages of CNN features chosen using simple criteria of persistence and texturedness, pointing to highly efficient real-time solutions in the near future.

A completely new real-time dense spherical SLAM has been developed and used as the basis of experimentation in this chapter. While it was not necessary for proving the main concept of semantic texture, it served not only as a means to investigating the concept, but also to prototype and learn about building dense SLAM systems that use a neural network in real-time. The know-how gained from this project later became the foundations to our next work, described in the remainder of the thesis. We also wanted to prove that Semantic Texture could be used as straightforward drop-in replacement for RGB textures in existing SLAM pipelines.

There are two characteristic features which make spherical mosaicing a simpler problem yet representative of SLAM in general: the camera movement model is simple, having three degrees of freedom (rotation) compared to six for general rigid body motion (rotation and translation). The other feature is that the maximum map size is limited to a view sphere, which means that the maximum distance travelled without loop closure is bounded.

3.1.1 Replacing the Pyramid with CNN Feature Maps

The performance of dense alignment can be measured along several axes. Two are the accuracy of the final alignment and the speed of convergence, but generally more important are the size of the basin of convergence and the robustness to unmodelled effects such as lighting changes. Both the speed and basin of convergence of LK alignment are improved by the use of image pyramids. Before alignment, both images are successively downsampled to produce a stack of images with decreasing resolution. Alignment then proceeds by performing a number of iterations at each level, starting with the lowest resolution versions which retain only low frequency detail but allow fast computation, and ending back at the original versions where only a few of the most expensive iterations are needed.

This chapter proposes replacing the downsampling pyramid in LK by the output of the convolutional layers of an off-the-shelf VGG Convolutional Neural Network trained for classification. The early layers of a CNN are well known to be generic, regardless of the final task it was trained for, as shown in [Agrawal et al., 2015]. The later layers respond to features which are increasingly complex and semantically meaningful, with more invariance to transformations including illumination. Now that the convolutional layers of a CNN can comfortably be run in real-time on video input on desktop or mobile GPUs, the simplicity of using them to build pyramids for alignment and scene representation is very appealing.

Even though CNNs turn raw images into feature maps, we argue that aligning their hierarchical responses is still a 'dense' method, which is much more akin to standard Lucas-Kanade alignment than image matching using sparse features. The convolutions produce a response at every image location. As we move towards powerful real-time scene understanding systems which can deal with complex scenes whose appearance and shape changes over short and long timescales, we will need dense and fully generative representations, and the ability to fuse and test live data against these at every time-step. We believe that there are many interesting levels of representation to find between raw pixel values and human annotated 'object-level' models, and that these representations should be learned and optimised depending on what task needs to be achieved, such as detecting whether something has changed in a scene.

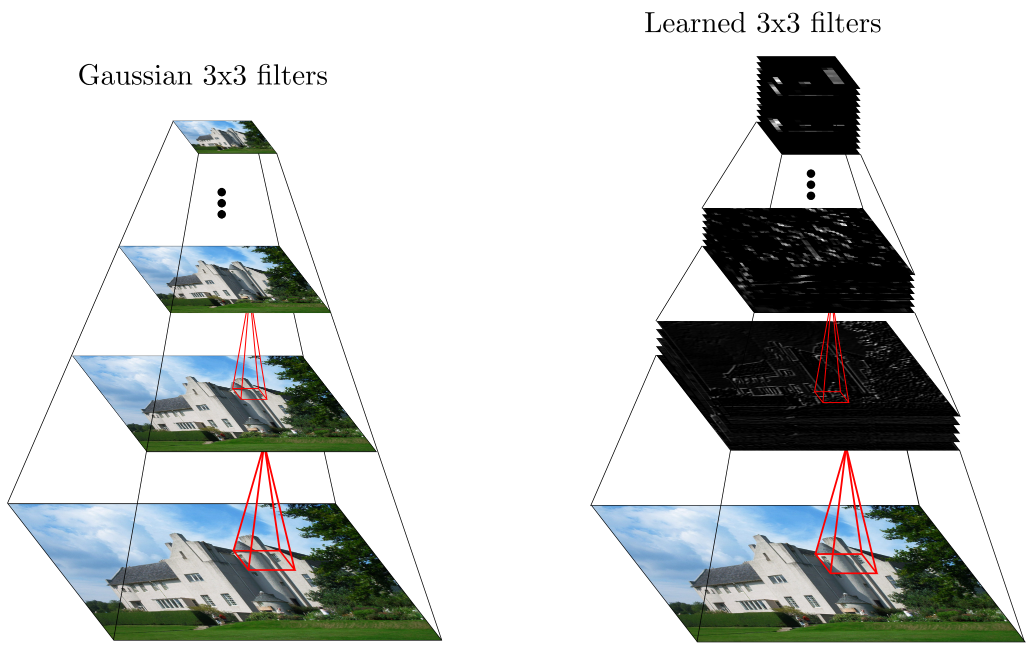


Figure 3.1: A comparison between a Gaussian pyramid (left) and the proposed semantic pyramid (right).

Outputs of the convolutions in standard classification CNN's form a pyramid similar to the ones used in RGB based LK image alignment (Figure 3.1). Consecutive layers of such pyramid encode increasingly semantic information, starting from simple geometrical filters in the first layers, leading to more complex concepts. Each level of the pyramid takes the form of a multi-channel image (tensor), where each channel contains responses to a certain learned convolutional filter. We propose to align the volumes in a coarse-to-fine manner, starting from the highest, low resolution layers and progressing down to the lower, geometrical layers to refine alignment.

The pyramid is created through applying successive convolutions followed by a nonlinear activation function, with occasional downsampling, very much like in standard CNN's. The weights for convolutions are trained for an image classification task. While in this work we have used weights from a trained VGG-16 classification model from [Simonyan and Zisserman, 2015] we believe that any classification CNN could be used instead due to the fact that they have been shown to learn mostly similar filters. The VGG network network consists of 13 layers of convolutions and a number of top level layers which we do not make use of in our work, and which compute the final classification vector. Figure 3.2 presents the architecture of this network.

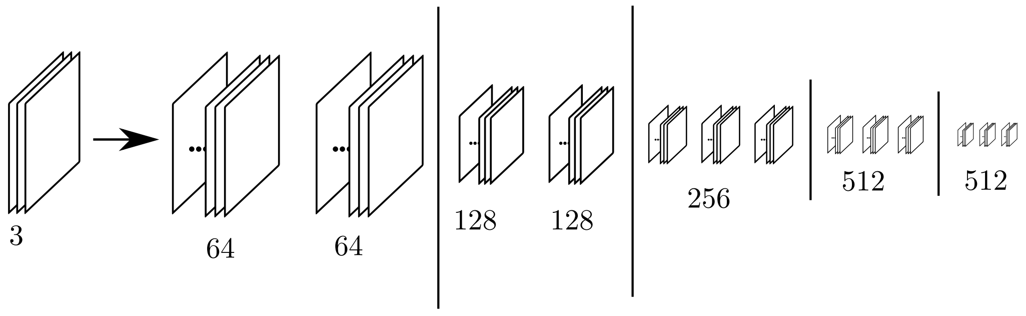


Figure 3.2: A pyramid is created using successive learned convolutions, just as in a standard CNN classification network. We have used VGG-16 for our experiments

We recommend watching the associated video material that visualises the CNN features and shows experiments described in this chapter. It can be found under the following link: https://youtu.be/SkpHccE1eTQ.

3.1.2 Related Work

Other non-CNN transformations of RGB have been attempted to improve the performance of correspondence algorithms. There are many advantages to generating a scale-space pyramid using non-linear diffusion [Perona and Malik, 1990], where edges are preserved while local noise is smoothed away, although it is not clear how much this would help Lucas-Kanade alignment. In stereo matching, there has been work such as that by Hirschmüller *et al.* [Hirschmüller, 2007] which compared three representations: Laplacian of Gaussian (LoG), rank filter and mean filter. All of these approaches help with viewpoint and lighting changes. In face fitting using Lucas-Kanade alignment, Antonakos *et al.* [Antonakos et al., 2015] evaluated nine different dense hand-designed feature transformations and found SIFT and HOG to be the most powerful.

There is a growing body of literature that uses convolutional networks to compare image patches, register camera pose and compute optical flow between images [Fischer et al., 2015, Zagoruyko, S., and Komodakis, N., 2015, Luo et al., 2016, Kendall et al., 2015]. Such methods use networks trained end-to-end, and can output robust estimates in challenging conditions but fail to deliver the accuracy of model-based approaches. Instead of performing iterative refinement those methods produce a one-shot estimate. In FlowNet [Fischer et al., 2015], in particular, the optical flow result must still be optimised with a standard variational scheme (TV-L1). Our approach bridges the gap between these two paradigms, delivering both the accuracy of online optimisation and the robustness of learned features.

Since the time the method presented in this chapter was developed, the research community has proposed several methods for directly learning a set of features specialised for camera tracking. In [Lv et al., 2019], the authors remodel the Inverse Compositional Lucas-Kanade pipeline using neural networks and train it end-to-end, including a neural optimiser, coarse-to-fine alignment of learned features and a predicted M-estimator. GN-Net, a network by Stumberg et al. [Stumberg et al., 2019] learns a nonlinear transformation of RGB images that produces a robust N-channel texture that is designed to promote robustness to strong lighting and weather changes while enforcing a maximal basin of convergence for the respective SLAM algorithm. In [Tang and Tan, 2019] the authors propose BA-Net, an end-to-end trained network for solving the two-view dense bundle adjustment problem. The depth and camera poses are recovered by minimising a direct per-pixel error on features predicted by the network.

While directly training a network to learn a representation for tracking might improve performance, we argue that exploring using off-the-shelf pre-trained CNNs is a worthwhile endeavour. A real-time autonomy system for a robot might require solving multiple different problems such as tracking, estimating optical flow or predicting depth. Since it has been demonstrated that early features of CNNs trained for different tasks learn similar filters [Agrawal et al., 2015], it might make sense to share the common preprocessing step between the various tasks, instead of training multiple networks. Moreover, on such platforms, it is very likely that a neural network will already be used in some capacity, thus producing the 'Semantic Texture' representation. Different types of on-board methods, including other neural networks could share this alternative representation of the image to calculate their desired quantities.

3.2 Real-Time Dense Mosaicing SLAM

This section describes the real-time mosaicing SLAM system developed as a means to investigate the proposed Semantic Texture representation. Inspiration for the principle of operation of the system was drawn from the work of Lovegrove and Davison [Lovegrove and Davison, 2010]. We will begin with Section 3.2.1, where we will derive the pixel warp function for purely rotational movement by first assuming the camera is observing a planar scene (plane induced homography) and then showing that in the case of pure camera rotation the knowledge of the true scene geometry is not required.

We then show in Section 3.2.2 how to use the derived warp to estimate the camera motion using the method previously described in Section 2.4. In Section 3.2.3, we demonstrate how to modify the system to make use of the proposed Semantic Texture. We finish with a description of two methods for displaying a spherical mosaic and information on the implementation of our system and its performance.

3.2.1 Purely Rotational Motion Model

In order to relate image points in one camera view to corresponding image points in a second view, knowledge of scene geometrical structure is required. A *Plane induced homography* is such a transformation, which uses a plane to relate coordinate frames of two cameras co-observing it (Figure 3.3). The homography has a form of a 3×3 matrix and can be determined by intersecting the ray corresponding to an image location $\mathbf{x}_1$ with the plane π and then projecting the intersection point $\vec{\pi}$ onto the second camera image plane.

Assume that $\mathbf{K}_1$ and $\mathbf{K}_2$ are the intrinsic matrices of both cameras, the first

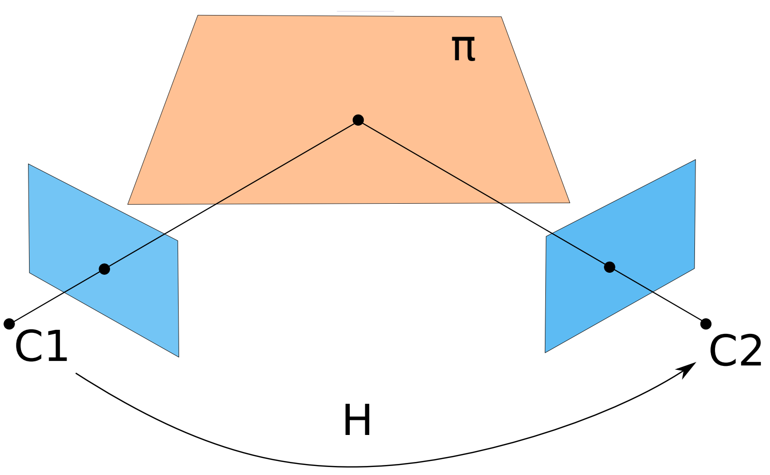


Figure 3.3: Homography $\mathbf{H}$ induced by the plane π relating the image coordinates of cameras C1 and C2.

camera is located at world origin and the $\mathbf{R}$, $\mathbf{t}$ are the rotation and translation between two cameras. Projection matrices corresponding to the cameras are then:

$$\mathbf{P}_1 = \mathbf{K}_1[\mathbf{I}|\mathbf{0}]$$

$$\mathbf{P}_2 = \mathbf{K}_2[\mathbf{R}|\mathbf{t}].$$

Given a plane $\pi = (\mathbf{v}^T d)^T$ where v is a unit vector orthogonal to the plane and d is the distance from the origin, we can write the plane equation as:

$$\pi^T \mathbf{x} = 0. \tag{3.1}$$

The intersection of a point from the first camera $\mathbf{x} = \mathbf{K}^{-1}\mathbf{x}_1$ with the plane π can be calculated with:

$$\mathbf{x}_\pi = (\mathbf{x}^T, -\mathbf{v}^T\mathbf{x})^T. \tag{3.2}$$

Projecting the point $\mathbf{x}_\pi$ onto the second camera image plane:

$$\mathbf{x}_2 = \mathbf{P}\mathbf{x}_\pi = \mathbf{K}_2[\mathbf{R}|\mathbf{t}]\mathbf{x}_\pi = \mathbf{R}\mathbf{x} - \mathbf{t}\mathbf{v}^T\mathbf{x} = \mathbf{K}_2(\mathbf{R} - \mathbf{t}\mathbf{v}^T)\mathbf{K}_1^{-1}\mathbf{x}_1 \tag{3.3}$$

$$\mathbf{H} = \mathbf{K}_2(\mathbf{R} - \mathbf{t}\mathbf{v}^T)\mathbf{K}_1^{-1}. \tag{3.4}$$

In case of pure rotational movement the homography becomes independent of the scene altogether:

$$\mathbf{H} = \mathbf{K}_2\mathbf{R}\mathbf{K}_1^{-1}. \tag{3.5}$$

3.2.2 Dense Rotational Tracking

Using the pure rotational homography derived in the previous section (Equation 3.5), we can construct a per-pixel photometric error cost function and solve the optimisation problem for minimising the warp between two overlapping frames using the *inverse compositional* approach (described in Section 2.4):

$$\rho(\mathbf{x};\boldsymbol{\omega}) = \sum_{\mathbf{x}} [I_l(\pi(\mathbf{K}\mathbf{R}(\boldsymbol{\omega})\mathbf{K}^{-1}\mathbf{x})) - I_r(\pi(\mathbf{K}\mathbf{R}\mathbf{K}^{-1}\mathbf{x})]^2, \tag{3.6}$$

where:

$$\pi(x,y,z) = \begin{bmatrix} x/z \\ y/z \end{bmatrix} \tag{3.7}$$

is the dehomogenising function projecting the points onto the image plane.

To solve, we first perform Taylor expansion around $\boldsymbol{\omega} = 0$ and drop higher order terms:

$$\rho(\mathbf{x};\Delta\boldsymbol{\omega}) = \sum_{\mathbf{x}} [I_l(\pi(\mathbf{K}\mathbf{R}(0)\mathbf{K}^{-1}\mathbf{x})) + \mathbf{J}\Delta\boldsymbol{\omega} - I_r(\pi(\mathbf{K}\mathbf{R}\mathbf{K}^{-1}\mathbf{x})]^2, \tag{3.8}$$

where:

$$\mathbf{J} = \frac{\partial}{\partial\boldsymbol{\omega}}\left(I_l(\pi(\mathbf{K}\mathbf{R}(\boldsymbol{\omega})\mathbf{K}^{-1}\mathbf{x}))\right)\Big|_{\boldsymbol{\omega}=0}. \tag{3.9}$$

Since $\mathbf{R}(0)$ is the identity rotation $\mathbf{R}(0) = \mathbf{1}$, we have:

$$\rho(\mathbf{x};\Delta\boldsymbol{\omega}) = \sum_{\mathbf{x}} [I_l(\mathbf{x}) + \mathbf{J}\Delta\boldsymbol{\omega} - I_r(\pi(\mathbf{K}\mathbf{R}\mathbf{K}^{-1}\mathbf{x})]^2. \tag{3.10}$$

Next, we calculate the Jacobian $\mathbf{J}$:

$$\begin{aligned} \frac{\partial}{\partial\boldsymbol{\omega}}\left(I_l(\pi(\mathbf{K}\mathbf{R}(\boldsymbol{\omega})\mathbf{K}^{-1}\dot{\mathbf{x}}))\right) &= \frac{\partial I_l}{\partial\pi}\frac{\partial\pi}{\partial\mathbf{x}}\frac{\partial}{\partial\boldsymbol{\omega}}\left(\mathbf{K}\mathbf{R}(\boldsymbol{\omega})\mathbf{K}^{-1}\mathbf{x}\right)\Big|_{\boldsymbol{\omega}=0} = \\ &= \nabla I_l \mathbf{J}_\pi \mathbf{K}\frac{\partial}{\partial\boldsymbol{\omega}}\left(\mathbf{R}(\boldsymbol{\omega})\mathbf{K}^{-1}\mathbf{x}\right), \end{aligned} \tag{3.11}$$

where

$$\mathbf{J}_\pi = \begin{bmatrix} 1 & 0 & -\frac{x}{z^2} \\ 0 & 1 & -\frac{y}{z^2} \end{bmatrix} \tag{3.12}$$

is the Jacobian of the π function and $\nabla I_l = \begin{bmatrix} g_x & g_y \end{bmatrix}$ is the image gradient.

$\mathbf{R}(\omega)$ is the exponential map taking elements from $\mathfrak{so}(3)$ to their corresponding lie group elements. To calculate the derivative of the exponential map, we use the following approximation:

$$\exp(\boldsymbol{\omega}_\times) \approx \mathbf{1} + \omega_\times. \tag{3.13}$$

Substituting for $\exp(\boldsymbol{\omega}_\times)$ in Equation 3.11 and using the anti-commutative property of the cross product, we have:

$$\begin{aligned} \nabla I_l \mathbf{J}_\pi \mathbf{K} \frac{\partial}{\partial \boldsymbol{\omega}} \left((\mathbf{I} + \boldsymbol{\omega}_\times) \mathbf{K}^{-1} \mathbf{x} \right) &= \nabla I_l \mathbf{J}_\pi \mathbf{K} \frac{\partial}{\partial \boldsymbol{\omega}} \left(\boldsymbol{\omega}_\times \mathbf{K}^{-1} \mathbf{x} \right) = \\ &= \nabla I_l \mathbf{J}_\pi \mathbf{K} \frac{\partial}{\partial \boldsymbol{\omega}} \left(-(\mathbf{K}^{-1}\mathbf{x})_\times \boldsymbol{\omega} \right) = \\ &= -\nabla I_l \mathbf{J}_\pi \mathbf{K} (\mathbf{K}^{-1}\mathbf{x})_\times. \end{aligned} \tag{3.14}$$

The warp parameter updates can be then calculated using equation 2.28:

$$\Delta \boldsymbol{\omega} = \left(\sum_{\mathbf{x}} \mathbf{J}^T \mathbf{J} \right)^{-1} \left(\sum_{\mathbf{x}} \mathbf{J}^T r \right), \tag{3.15}$$

where

$$\begin{aligned} r &= I_r(\pi(\mathbf{KRK}^{-1}\mathbf{x})) - I_l(\mathbf{x}) \\ \mathbf{J} &= -\nabla I_l \mathbf{J}_\pi \mathbf{K} (\mathbf{K}^{-1}\mathbf{x})_\times. \end{aligned}$$

The current camera pose estimate R is then updated by composing it with the inverse of the calculated warp:

$$\mathbf{R} \leftarrow \mathbf{R} \exp(\Delta \boldsymbol{\omega}_\times)^{-1}. \tag{3.16}$$

3.2.3 Volume Alignment

To apply the proposed semantic texture to our dense mosaicing system, we use the same purely rotational Lucas-Kanade formulation from Equation 3.6 and simply replace the images with feature volumes $V : \mathbb{R}^2 \to \mathbb{R}^N$:

$$\sum_{\mathbf{x}} ||V_t(\pi(\mathbf{KR}(\boldsymbol{\omega})\mathbf{K}^{-1}\mathbf{x}) - V_r(\mathbf{KRK}^{-1})||^2 \ . \tag{3.17}$$

In order to solve for the parameter update $\Delta\mathbf{p}$, the Jacobian and Hessian of this nonlinear Least Squares System need to be calculated. Similarly to contributions from each pixel, terms from different filters can be summed together:

$$\mathbf{J}_{\mathbf{x}} = \frac{\partial \mathbf{e}_{\mathbf{x}}}{\partial \Delta\mathbf{p}} = \begin{bmatrix} \frac{\partial e_{\mathbf{x},1}}{\partial \Delta\mathbf{p}} \\ \vdots \\ \frac{\partial e_{\mathbf{x},N}}{\partial \Delta\mathbf{p}} \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{\mathbf{x},1} \\ \vdots \\ \mathbf{J}_{\mathbf{x},N} \end{bmatrix} , \tag{3.18}$$

$$\begin{aligned} \mathbf{H} := \sum_{\mathbf{x}} \mathbf{J}_{\mathbf{x}}^T \mathbf{J}_{\mathbf{x}} = \sum_{\mathbf{x}} \left[\mathbf{J}_{\mathbf{x},1}^T \dots \mathbf{J}_{\mathbf{x},N}^T\right] \begin{bmatrix} \mathbf{J}_{\mathbf{x},1} \\ \vdots \\ \mathbf{J}_{\mathbf{x},N} \end{bmatrix} = \\ = \sum_{\mathbf{x}} \sum_{c=1}^{N} \mathbf{J}_{\mathbf{x},c}^T \mathbf{J}_{\mathbf{x},c} \end{aligned} , \tag{3.19}$$

$$\begin{aligned} \mathbf{b} := \sum_{\mathbf{x}} \mathbf{J}_{\mathbf{x}}^T \mathbf{e}_{\mathbf{x}} = \sum_{\mathbf{x}} \left[\mathbf{J}_{\mathbf{x},1}^T \dots \mathbf{J}_{\mathbf{x},N}^T\right] \begin{bmatrix} \mathbf{e}_{\mathbf{x},1} \\ \vdots \\ \mathbf{e}_{\mathbf{x},N}^T \end{bmatrix} = \\ = \sum_{\mathbf{x}} \sum_{c=1}^{N} \mathbf{J}_{\mathbf{x},c}^T \mathbf{e}_{\mathbf{x},c} \end{aligned} . \tag{3.20}$$

3.2.4 Spherical Mosaic Rendering

The map constructed and maintained by the SLAM system consists of a collection of keyframes containing the associated image and its estimated pose. In order to present the panorama to the user, the map has to be transformed into a single image. Overlapping parts of keyframes also need to be properly blended to avoid ghosting and visible seams.

In order to render a panorama from a set of registered images, pixels from each need to be projected onto a compositing surface which is then mapped to a flat 2D output image. To ensure real-time performance the implementation utilises parallel GPU kernels. For each keyframe, a separate kernel is launched which processes pixels of the output mosaic. Each pixel location is mapped to coordinates in a system appropriate to the selected compositing surface (spherical, cylindrical, etc). These coordinates represent a ray r specified in the world frame, which is transformed into the keyframe frame of reference using its associated pose estimation R_{wk}. The ray is next intersected with the keyframe image plane. If the intersection lies within the image bounds, the image is sampled at that location and added to the original processed mosaic pixel. This approach allows rendering panoramas at arbitrarily selected resolutions, which can be higher than the resolution of the input image, enabling super-resolution.

A separate counter is stored for each output mosaic pixel, which is incremented every time a keyframe is successfully sampled and contributes intensity to this specific pixel. After processing all keyframes a second normalisation kernel is launched which iterates over the output mosaic pixels and divides the accumulated intensity by the counter value. This results in simple averaging of keyframes at overlap regions.

One of the surfaces used for composition is a sphere centred in the camera centre of rotation. This projection is pictured in Figure 3.4. To render a spherical panorama, we first map the mosaic image coordinates $\mathbf{x} = (u, v)^T \in \Omega = [0, w] \times [0, h]$ to spherical coordinates: yaw $\psi \in [-\pi, \pi)$ and pitch $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2})$. The corresponding ray can be calculated as follows:

$$\mathbf{r}_{sph}(\psi, \theta) = (\cos\theta\cos\psi, \sin\theta, \cos\theta\sin\psi)^T. \tag{3.21}$$

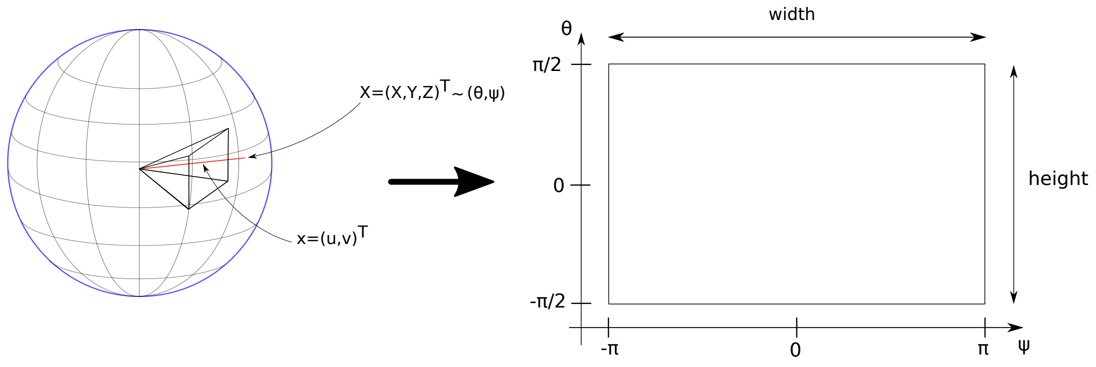


Figure 3.4: Spherical projection of keyframe pixels and mapping

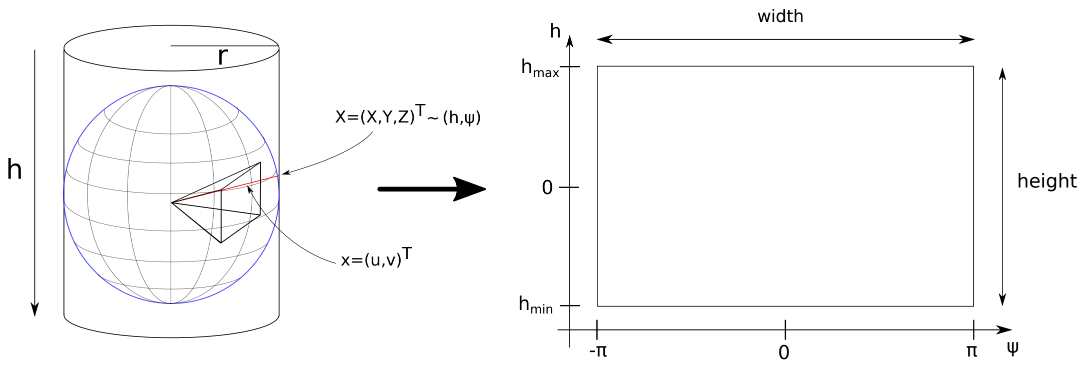


Figure 3.5: Cylindrical projection of keyframe pixels and mapping

The formula for the resulting mosaic is:

$$I_{sph}(\psi, \theta) = \frac{1}{n}\sum_{i=1}^{n} I_i(\pi(\mathbf{K}\mathbf{R}_{wk}^{-1}\mathbf{r}_{sph}(\psi, \theta))). \tag{3.22}$$

A cylinder can be also used for composition. This type of projection is presented in Figure 3.5. For this, we map mosaic image coordinates to cylindrical coordinates: yaw $\psi \in [-\pi, \pi]$ and height $h \in [h_{min}, h_{max}]$. The corresponding ray has the form:

$$\mathbf{r}_{cyl}(\psi, h) = (\cos\psi, h, \sin\psi)^T. \tag{3.23}$$

The formula for the resulting mosaic is:

$$I_{cyl}(\psi, h) = \frac{1}{n}\sum_{i=1}^{n} I_i(\pi(\mathbf{K}\mathbf{R}_{wk}^{-1}\mathbf{r}_{cyl}(\psi, h))). \tag{3.24}$$

3.2.5 System Implementation

On start, the system creates an initial keyframe from the first image and initialises its frame of reference. As the user moves the camera, its rotation is tracked using dense image alignment against the first keyframe. Once the overlap between the current and the initial view falls below a certain threshold, a new keyframe is spawned. In the case when tracking is lost, the system attempts relocalisation by trying to track against small resolution versions of each keyframe. A spherical or cylindrical mosaic is rendered in real-time live from the keyframe map, together with the current camera frustum. An example panorama created by the system is presented in Figure 3.6.

Our system has been implemented from scratch in C++, using the Caffe framework for neural network evaluation. In order to perform alignment in real-time over big volumes the optimisation has been implemented to run on the GPU using NVIDIA CUDA. Each computational thread calculates per-pixel values, which are later reduced to single matrices using Equations 3.19 and 3.20.

The system runs in interactive real-time at 15-20 frames per second when using the whole pyramid, although as shown in Section 3.3.2, this could be significantly increased by selecting a small subset of features to track. Extraction of full 13 layers of convolutional features takes approximately 1 ms per frame. We use an image of size 224×224 to extract features and align all of the levels. The Caffe framework did not allow to access the GPU pointers of pyramid data, therefore our software is forced to make copies. This takes 10ms, which could be avoided by further modifications to the software. Time spent on performing the alignment depends on the number of iterations performed at different levels and for the settings used in our experiments, usually takes approximately 40–60ms.

In all tests, a tripod setup with a PointGrey Flea3 FL3-U3-20E4C-C camera with wide angle $82.4° \times 66.9°$ lens has been used to capture the image streams.

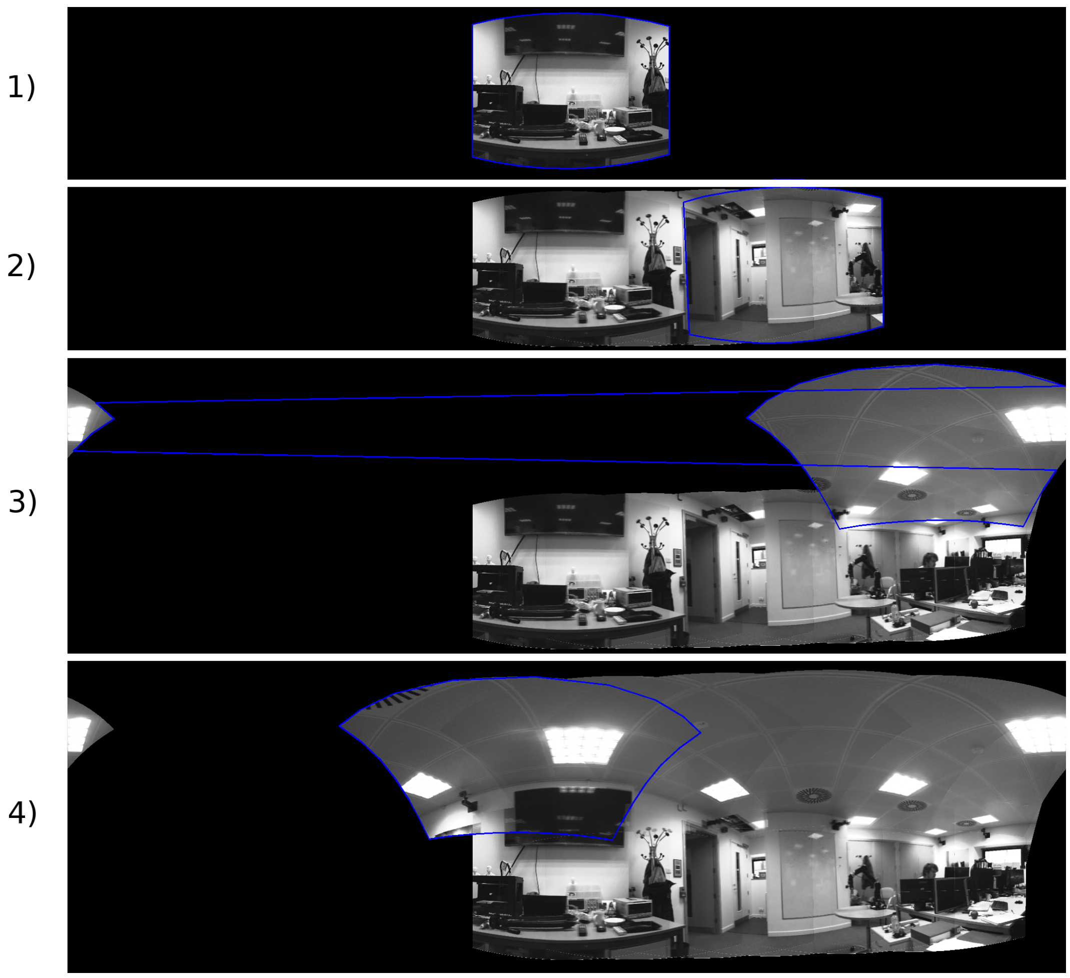


Figure 3.6: This figure presents the spherical mosaic incrementally built by our spherical SLAM system as the camera browses the scene. The current camera location estimate is drawn in blue. On start, the system creates an initial keyframe from the first image and initialises its frame of reference (1). As the user moves the camera, its rotation is tracked using dense image alignment against the first keyframe. Once the overlap between the current and the initial view falls below a certain threshold, a new keyframe is spawned (2-3). The bottom image presents the final mosaic consisting of 8 keyframes (4).

3.3 Experimental Results

We present experiments which compare the performance of image alignment for our CNN pyramid with raw RGB and dense SIFT pyramids. We consider that robustness is the most important factor in camera tracking, and therefore use the size of basin of convergence as our performance measure in the main results in Section 3.3.1. We also investigate the possibility of improving the results and reducing the computational overhead through selecting the most valuable feature maps in Section 3.3.2.

3.3.1 Robust, Long-Term Tracking

In order to test the robustness of the proposed representation in long-term tracking scenarios, three time-lapse sequences from a static camera have been captured showcasing real-world changing lighting conditions. The videos cover 8–10 hours of outdoor scenes, and have been sub-sampled into 2 minute clips.

In our tests we focus on two of the captured sequences — *window* and *home*. The first features a highly specular scene which undergoes major and irregular lighting changes throughout the day with no major outliers in the form of moving objects. The *home* sequence shows a relatively busy street with frequent outliers. This sequence contains less specular surfaces, the observed illumination changes have more a global character. We have selected three snapshots from each of these sequences containing different lighting conditions, and evaluated the area of the convergence basin while trying to align each possible pair of frames. To serve as a baseline comparison, we also present the results of alignment using RGB (RGB) and dense SIFT features (SIFT), as they have been shown to be very robust to lighting nuisance when used for Lucas-Kanade based face tracking in [Antonakos et al., 2015].

In order to measure the size of the convergence basin, we segment the parameter space into a regular grid and initialise the Lucas-Kanade algorithm at each point. Next, we perform the optimisation for 1000 iterations and inspect the final result. We find that when convergence is successful, the final accuracy is generally good, and therefore define that if the tracking result is within 0.07 radians of ground truth, we mark the tested point as belonging to the convergence basin. The marked points are next used to calculate the total area of the convergence basin.

The results are presented in Figures 3.7 and 3.8. Five pyramids levels of SIFT and RGB have been used in the tests to compare with the proposed CNN pyramid. The missing values were duplicated in the plots so that it is possible to easily compare the convergence basin areas of the corresponding image resolutions (e.g. RGB/SIFT level 5 corresponds to CONV levels 11–13). Note that the LK alignment results are not symmetric with regard to which image is used as the template and which as the reference, as only the gradient of the template image is used, which might have impact on performance under varying lighting conditions and blur.

For frames from the *window* dataset (Figure 3.7), all of the methods have a sensible basin of convergence on the diagonal, where the images used for alignment are identical. For more challenging, off-diagonal pairs, such as A2, A3 or B3, the RGB cost function fails to provide a minimum in the correct spot, while SIFT and CONV still have wide, comparable convergence basins. One of the failure cases is showcased in more detail in Figure 3.9. The proposed CONV method seems to excel at higher levels, which are believed to have more semantic meaning.

In the second, *home* sequence (Figure 3.8) RGB performs better, possibly due to the global character of the illumination changes. It still fails when the lighting changes significantly (pair C2,B3). Similar to the previous sequence, the proposed method performs at least as well as SIFT and better than RGB at the highest levels of pyramid. To evaluate how the proposed solution handles image blurring, we have tested it with an alignment problem of an image with a heavily blurred version of itself. Figure 3.10 presents the reference and template images used in this test, selected cost landscape plots of RGB, CONV and SIFT and a comparison of basin sizes. We can see that all methods are robust to blur, with our proposed method providing the best results at the highest pyramid levels. It provides a steady, wide basin of convergence at the top pyramid levels regardless of the lighting conditions and blur.

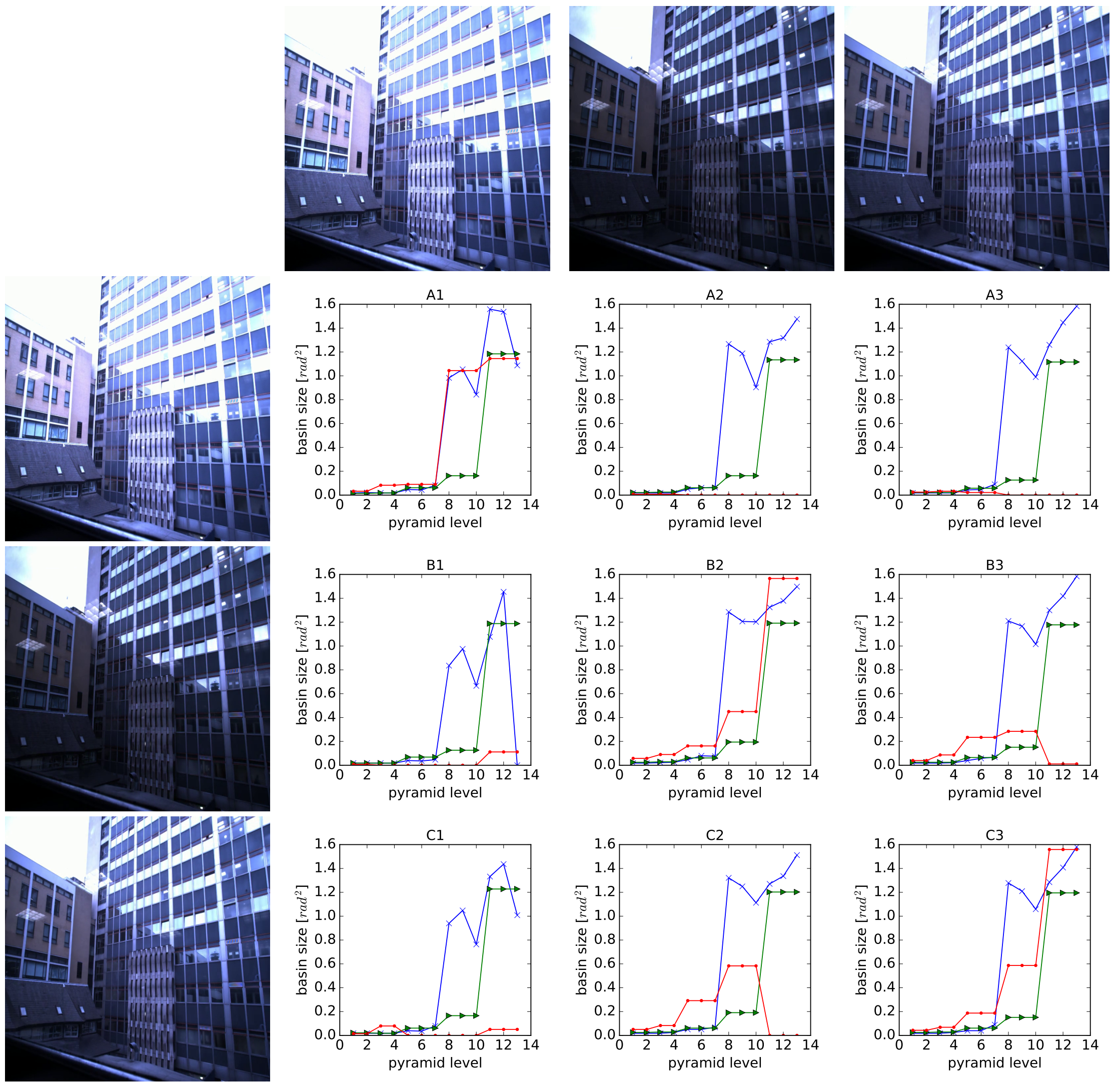


Figure 3.7: Comparison of sizes of convergence basins for aligning pairs of images with different lighting conditions (sampled from the *window* sequence). Each array cell presents convergence basins areas of RGB (red), SIFT (green) and CONV (blue) at different pyramid levels. The left column and top row images are used in LK as template and reference, respectively.

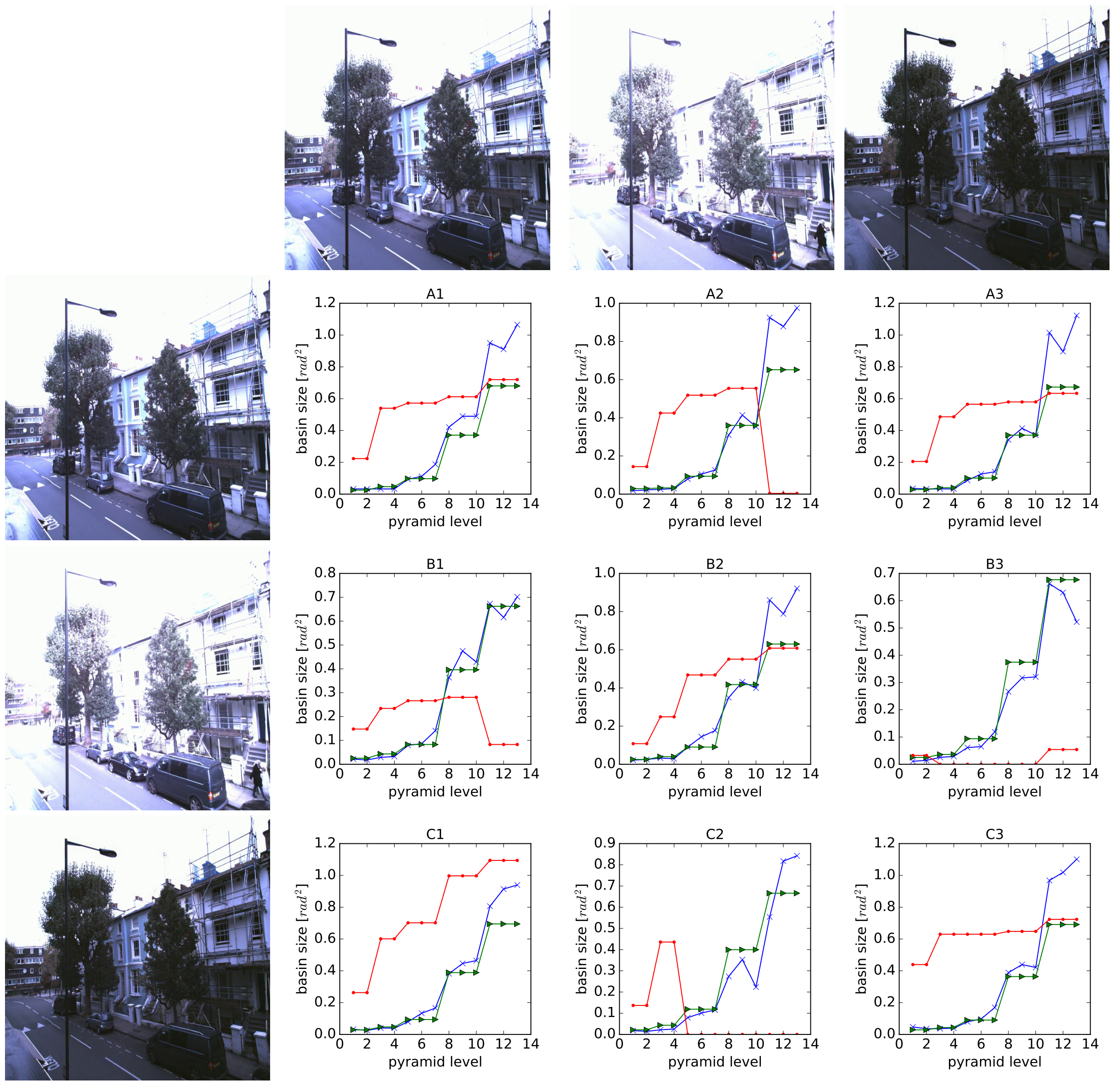


Figure 3.8: Comparison of sizes of convergence basins for aligning pairs of images with different lighting conditions (sampled from the *home* sequence). Each array cell presents convergence basins areas of RGB (red), SIFT (green) and CONV (blue) at different pyramid levels. Left and top images are used in LK as template and reference, respectively.

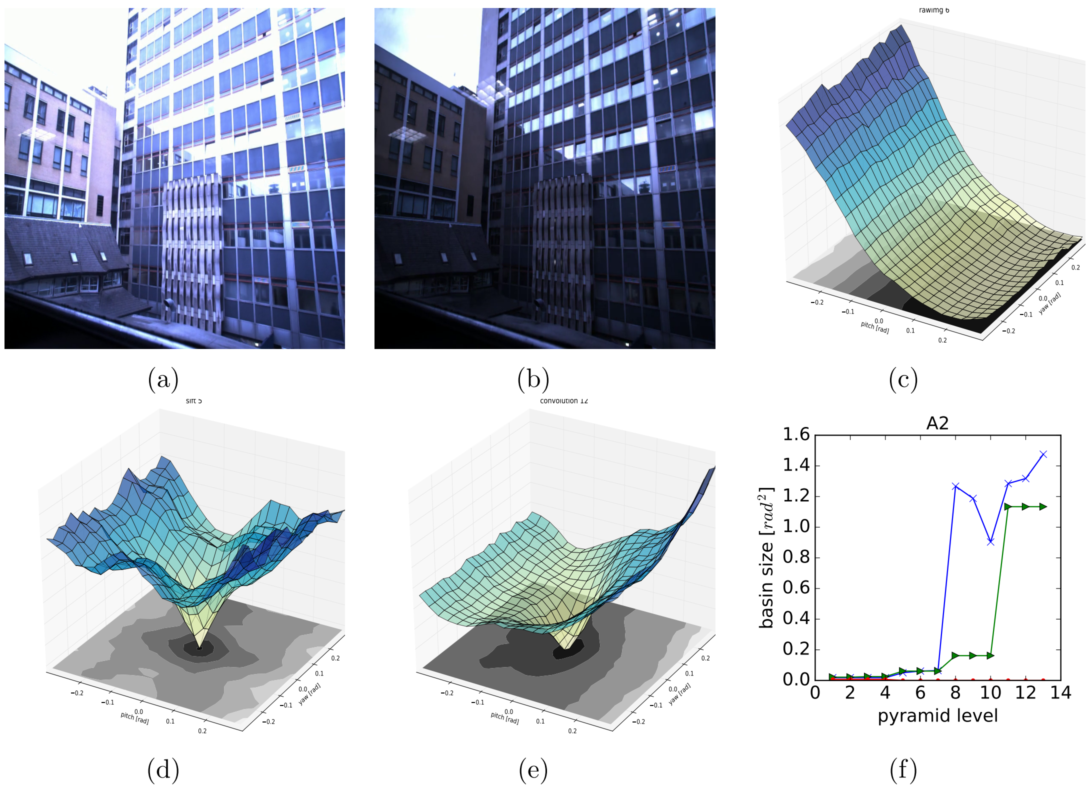


Figure 3.9: Cost landscape plots and convergence basin areas of different methods for aligning two images of the same scene in vastly different lighting conditions. (a) Template image (b) Reference image (c) RGB cost landscape (d) SIFT cost landscape (e) CONV cost landscape (f) Comparison of convergence basin areas (RGB: red, SIFT: green, CONV: blue).

3.3.2 Reducing the Number of Features

The results presented so far are based on aligning all of the CNN feature maps produced at each pyramid level jointly. One clear disadvantage of this approach is computational cost: the amount of work which needs to be done at each alignment iteration to calculate a difference is proportional to the number of features. It is apparent from any inspection of the features generated by a CNN that there is a lot of redundancy in feature maps, with many looking very similar, and therefore it seemed likely that it is possible to achieve similar or better results through selection of a percentage of features. We perform experiments to compare a random selection with simple criteria based on measures of texturedness and stability. An other option would be to use a different, smaller CNN with a smaller number of features, but it

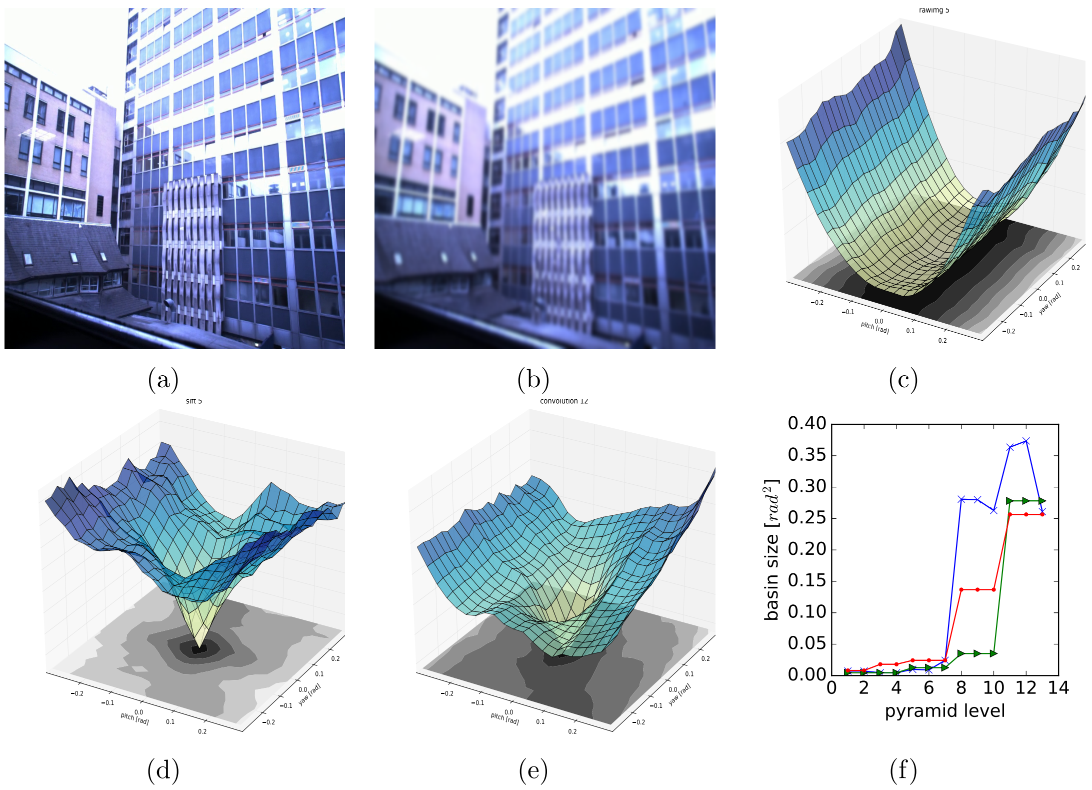


Figure 3.10: Cost landscape plots and calculated convergence basin areas of different methods for aligning an image with a blurred version of itself. (a) Template image (b) Reference image (c) RGB cost landscape (d) SIFT cost landscape (e) CONV cost landscape (f) Comparison of convergence basin areas (RGB: red, SIFT: green, CONV: blue).

could be benefit from the techniques presented in this section. This approach could also be replaced by learning a small subset of features that are most important for tracking, but as we have outlined in the introduction of this chapter, we believe it is still worthwhile to explore using features from a network trained for a different task.

In standard Lucas-Kanade, the size of the convergence basin depends highly on image content. For example, a vertical stripe can only be used to detect horizontal translation; the lack of vertical gradient makes it impossible to determine the movement in this direction. In order to correctly regress the camera pose, a strong gradient in both feature directions is required. As in Shi and Tomasi's classic 'Good Features to Track' paper [Shi and Tomasi, 1994], we measure the texturedness of a feature based on its structure tensor:

$$\mathbf{G}_f = \sum_{\mathbf{x}\in\mathbf{I}_f} \begin{bmatrix} g_x^2 & g_x g_y \\ g_x g_y & g_y^2 \end{bmatrix},$$

where I_f is the activation map of the feature in response to a certain input image. We use the smallest eigenvalue of matrix $\mathbf{G}_f$ as a comparable single score:

$$\lambda_f = \frac{1}{N}\sum_{i=1}^{N} \min({}_1\lambda_f^i, {}_2\lambda_f^i),$$

where ${}_1\lambda_f^i$ and ${}_2\lambda_f^i$ are the two eigenvalues of matrix $\mathbf{G}_f^i$.

The other factor to consider is stability. Most valuable features provide stable activations that change only due to changes in camera pose. Features that react to objects that are likely to change their location, or lighting changes, are undesirable. We measure the instability of a feature f by calculating the average sum of squared differences (SSD) between activations obtained from images of the test sequence to the ones extracted from the first image and warped to the current camera frame:

$$s_f = \frac{1}{N-1}\sum_{i=2}^{N}\sum_{\mathbf{x}} \|(I_f^1(\mathbf{W}(\mathbf{x};\boldsymbol{\omega})) - I_f^i(\mathbf{x})\|^2.$$

Averages of these two scores have been calculated across frames of three video sequences — two time-lapses and one hand tracking sequence. We have selected and evaluated several subsets of features of varying size that have the optimal texturedness and stability. To assess their robustness, we again use the size of convergence basin as a measure. Twelve challenging (outliers, varying lighting conditions) image pairs sampled from our recorded sequences have been used in the evaluation.

Figure 3.12 compares the average convergence basin size achieved with the features selected with our proposed approach with a baseline of using random selections. Random tests were performed in a range of 5–100% sub-sampling, with a step of 5%. Each sampled subset was tested 20 times against all twelve image pairs. We see very strongly that the features selected using our simple criteria give excellent tracking performance when they are much fewer than the whole set; while the performance of the randomly chosen features tails off much faster.

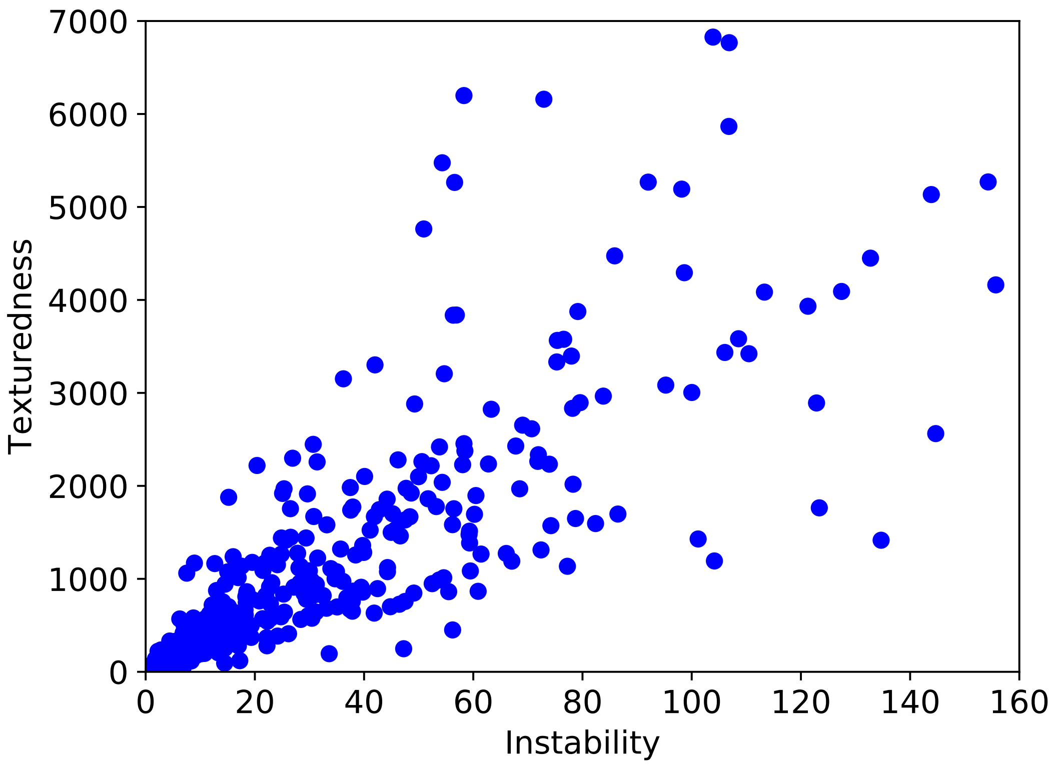


Figure 3.11: Feature texturedness versus instability. Each point represents one of the 512 features from the last convolutional layer. We pick features with maximum *texturedness* and minimum *instability* (top left corner of the plot).

3.4 Conclusions

We have shown that substituting the image pyramid in standard Lucas Kanade dense alignment with a hierarchy of feature maps generated by a standard VGG trained for classification straightforwardly gives much improved tracking robustness with respect to large lighting changes, often outperforming SIFT, an image transformation shown to be superior for template matching at that time [Zagoruyko, S., and Komodakis, N., 2015]. Our tests have proven that it is possible to make use of the generic filters learned by Convolutional Neural Networks, without training them directly for camera tracking. This allows us to take advantage of the fact that an object recognition or other type network will most likely be already present on any intelligent robotic system, providing us with the required Semantic Texture.

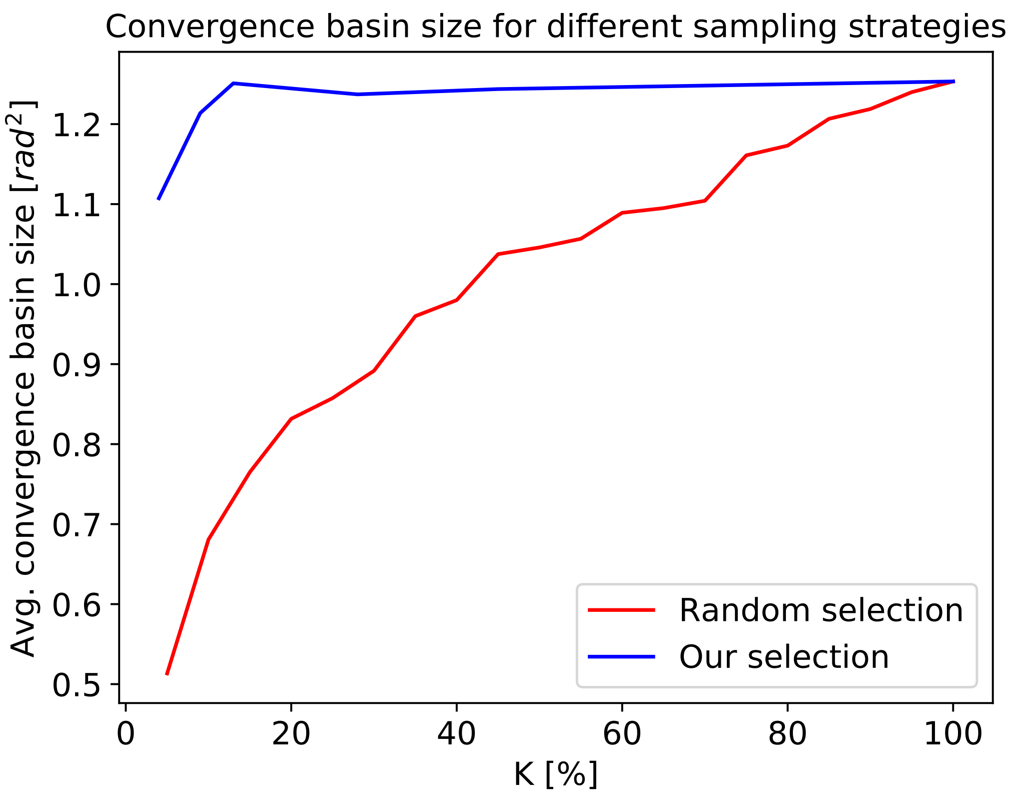


Figure 3.12: Average convergence basin size obtained by using our proposed selection process (blue) and random sampling (red), for different sampling percentage. Results were evaluated using twelve image pairs representing the most challenging conditions (outliers, lighting variations).

While our method induces additional computation, it offers a trade-off between performance and robustness which could be tuned to each specific application. We have demonstrated a simple yet effective feature selection procedure that could be performed as a pre-processing step for each specific network, obtaining the desired increase in robustness and computational complexity.

Despite the fact that other methods [Lv et al., 2019, Stumberg et al., 2019, Tang and Tan, 2019] have taken this approach further, it is still interesting to observe how much the performance of tracking can be improved using non-specific features. We expect the learned methods to perform better than our proposed approach, but it has not been determined exactly how much is gained by performing specific training. Developing the work in this thesis has taught us that incorporating more learned parts in the processing pipeline causes problems with generalisation across different datasets. This might indicate that not relying on networks being specifically trained for the task we are trying to solve might prove to be a more robust approach. We continue this trend in our next work, which also retains the classical iterative optimisation scheme and does not fully rely on the network predicting the correct values in a feed-forward manner.

A different but important lesson learned from developing the work for this chapter is the know-how of building real-time dense SLAM systems that use the GPU for both processing images with a neural network and performing image alignment with CUDA kernels. This knowledge was extensively used for developing the DeepFactors system, which will be described in Chapter 5.

While Semantic Texture is immediately a neat and convenient method for tracking, we hope that it opens the door to further research on new representations for dense mapping and tracking which lie in between raw pixel values and object-level models. What levels of representation are needed in dense SLAM in order actually to achieve tasks such as detecting a change in an environment? A clear next step would be to investigate the performance of m-estimators on the CNN features to gate out outliers. Might we expect to see that whole semantic regions which had moved or changed could be masked out from tracking? We imagine that a dense 3D model will be painted with smart learned feature texture for tracking and updating, rather than the raw pixel values of systems like DTAM [Newcombe et al., 2011].

For the remainder of the work presented in this thesis, rather than continuing research from this chapter, we have moved towards more general 3D representations that impact the dense SLAM pipeline in a more fundamental way, as we were interested in working towards a new generations of systems that are united with learning in a more elegant and coherent way.

Chapter 4

Learned Compact Depth Map Representation

Contents

4.1 Introduction

The underlying representation of scene geometry is a crucial element of any localisation and mapping algorithm. Not only does it influence the type of geometric qualities that can be mapped, but also dictates what algorithms can be applied. In SLAM in general, but especially in monocular vision, where scene geometry cannot be retrieved from a single view, the representation of geometrical uncertainties is essential. However, uncertainty propagation quickly becomes intractable for large degrees of freedom. For example, to represent the covariance of the pixels of a single 800×600 depth map $2.3 \cdot 10^{11}$ parameters are required. This difficulty has split mainstream SLAM approaches into two categories: *sparse* SLAM [Davison, 2003, Klein and Murray, 2007, Mur-Artal et al., 2015] which represents geometry by a sparse set of features and thereby allows joint probabilistic inference of structure and motion (which is a key pillar of probabilistic SLAM [Durrant-Whyte and Bailey, 2006]) and *dense* or semi-dense SLAM [Newcombe et al., 2011, Engel et al., 2014] that attempts to retrieve a more complete description of the environment at the cost of approximations to the inference methods (often discarding cross-correlation of the estimated quantities and relying on alternating optimisation of pose and map [Platinsky et al., 2017, Engel et al., 2017]).

However, the conclusion that a dense representation of the environment requires a large number of parameters is not necessarily correct. The geometry of natural scenes is not a random collection of occupied and unoccupied space but exhibits a high degree of order. In a depth map, the values of neighbouring pixels are highly correlated and can often be accurately represented by well known geometric smoothness primitives. But more strongly, if a higher level of understanding is available, a scene could be decomposed into a set of semantic objects (e.g. a chair) together with some internal parameters (e.g. size of chair, number of legs) and a pose, following a direction indicated by the SLAM++ system [Salas-Moreno et al., 2013] towards representation with very few parameters. Other more general scene elements which exhibit simple regularity such as planes can be recognised and effi-

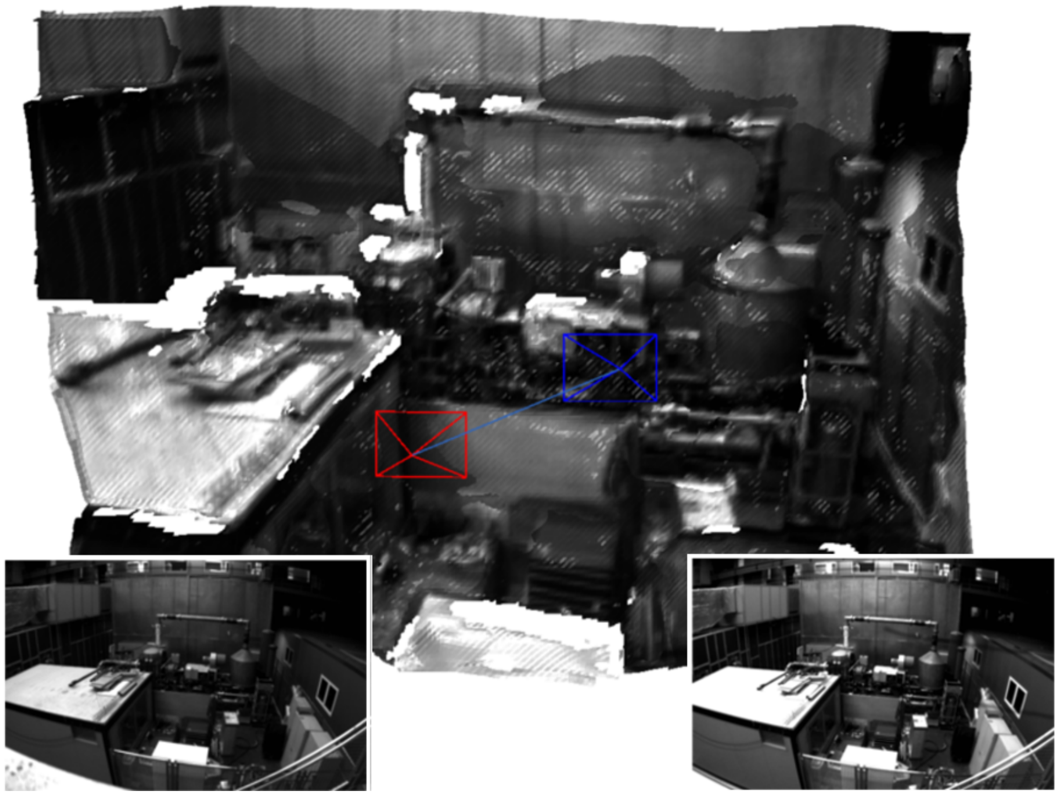


Figure 4.1: Highlights of reconstruction created by our CodeSLAM system from pairs of images selected frames from the EuRoC dataset. The proposed compact representation of 3D geometry enables joint optimisation of the scene structure and relative camera motion without explicit priors and a relatively fast performance.

ciently parametrised within SLAM systems (e.g. [Salas-Moreno et al., 2014, Kaess, 2015]). However, such human-designed dense abstractions are limited in the fraction of natural, cluttered scenes which they can represent.

In this work we aim at a more generic compact representation of dense scene geometry by training an auto-encoder on depth images. While a straightforward auto-encoder might over-simplify the reconstruction of natural scenes, the key novelty is to condition the training on intensity images. Our approach is planned to fit within the common and highly scalable keyframe-based SLAM paradigm [Klein and Murray, 2007, Engel et al., 2014], where a scene map consists of a set of selected and estimated historical camera poses together with the corresponding captured images and supplementary local information such as depth estimates. The intensity images are usually required for additional tasks, such as descriptor matching for place recognition or visualisation, and are thus readily available for supporting the depth encoding.

The depth map estimate for a keyframe thus becomes a function of the corresponding intensity image and an unknown compact representation (henceforth referred to as 'code'). This allows for a compact representation of depth without sacrificing reconstruction detail. In inference algorithms the code can be used as a dense representation of the geometry and, due to its limited size, this allows for full joint estimation of both camera poses and dense depth maps for multiple overlapping keyframes. We might think of the image providing local details and the code as supplying more global shape parameters which are often not predicted well by 'depth from single image' learning. Importantly though, these global shape parameters are not a designed geometric warp but have a learned space which tends to relate to semantic entities in the scene, and could be seen as a step towards enabling optimisation in general semantic space.

This work, published in [Bloesch et al., 2018] came at a time when many authors were combining techniques from deep learning with estimation-based SLAM frameworks, and there is an enormously fertile field of possibilities for this. Some particularly eye-catching pieces of work over the past year have focused on supervised and self-supervised training of surprisingly capable networks which are able to estimate visual odometry, depth and other quantities from video [Garg et al., 2016, Ummenhofer et al., 2017, Wang et al., 2017, Clark et al., 2017b, Zhou et al., 2017, Yin and Shi, 2018]. These methods run with pure feed forward network operation at runtime, but rely on geometric and photometric formulation and understanding at training time to correctly formulate the loss functions which connect different network components. Other systems are looking towards making consistent long-term maps by constantly refining geometric estimates, and this is the domain in which we are more interested here. In CNN-SLAM [Tateno et al., 2017] single image depth prediction and dense alignment are used to produce a dense 3D map and this gives a promising result, but it is not possible to optimise the predicted depth maps further for consistency when multiple keyframes overlap as it is in our approach.

To summarise, the two key contributions of the work presented in this chapter are:

- The derivation of a compact and optimisable representation of dense geometry by conditioning a depth auto-encoder on intensity images.
- The implementation of a novel and preliminary monocular visual odometry system that achieves a tight joint optimisation of motion and dense geometry.

In the rest of this chapter, we will first explain our method for depth learning and prediction, and then show the applicability of this approach in a SLAM setting.

4.2 Intensity Conditioned Depth Auto-Encoding

Two important qualities of geometry representations are *accuracy* and *practicality*. While the *accuracy* of a representation simply relates to its ability to reproduce the geometry, the *practicality* describes how well the representation can be used in an overall system. For inference-based SLAM systems, the latter typically requires the representation to lead to an optimisable loss function. For a representation G of the geometry a loss function $L(G)$ should be differentiable and have a clear minimum. Additionally, the size of the representation G should be limited in order to allow the estimation of second-order statistical moments (a covariance matrix) as part of more powerful inference methods.

In order to come up with a compact representation of the scene geometry we explore auto-encoder-like network architectures. Auto-encoders are networks which attempt to learn an identity mapping while being subject to an information bottleneck which forces the network to find a compact representation of the data [Rumelhart et al., 1986]. In a naive attempt to auto-encode depth this would lead to very blurry depth reconstruction since only the major traits of the depth image can make

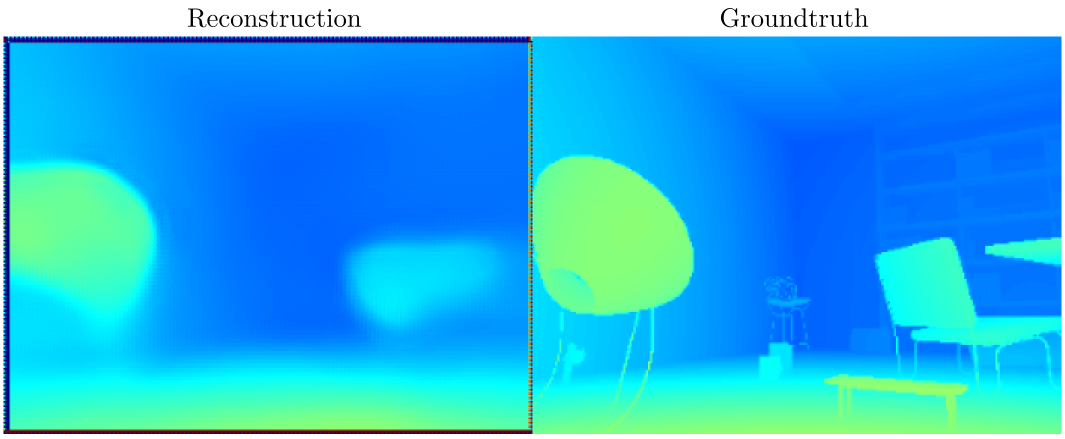


Figure 4.2: Depth auto-encoder without the use of image intensity data. Due to the bottleneck of the auto-encoder only major traits of the depth image can be captured.

it through the bottleneck (see Figure 4.2). In a monocular vision setup, however, we have access to the intensity images, which we are very likely to store alongside every keyframe. This can be leveraged to make the encoding more efficient: the full depth information does not need to be encoded, only the part of the information which cannot be retrieved from the intensities has to be retained. The depth D thus becomes a function of image I and (unknown) code $\mathbf{c}$:

$$D = D(I, \mathbf{c}) \; . \tag{4.1}$$

The above equation also highlights the relation to depth-from-mono architectures [Eigen et al., 2014, Liu et al., 2015, Garg et al., 2016, Zhou et al., 2017] which solve a code-less version of the problem, $D = D(I)$. Essentially, our architecture is a combination of the depth-from-mono-architecture of Zhou et al. [Zhou et al., 2017] and a variational auto-encoder for depth. We have chosen a variational auto-encoder network [Kingma and Welling, 2014] in order to increase the smoothness of the mapping between code and depth: small changes in the code should lead to small changes in the depth. While the *practicality* of our representation is thus addressed by the smoothness and the limited code size, the *accuracy* is maximised by training for the reconstruction error.

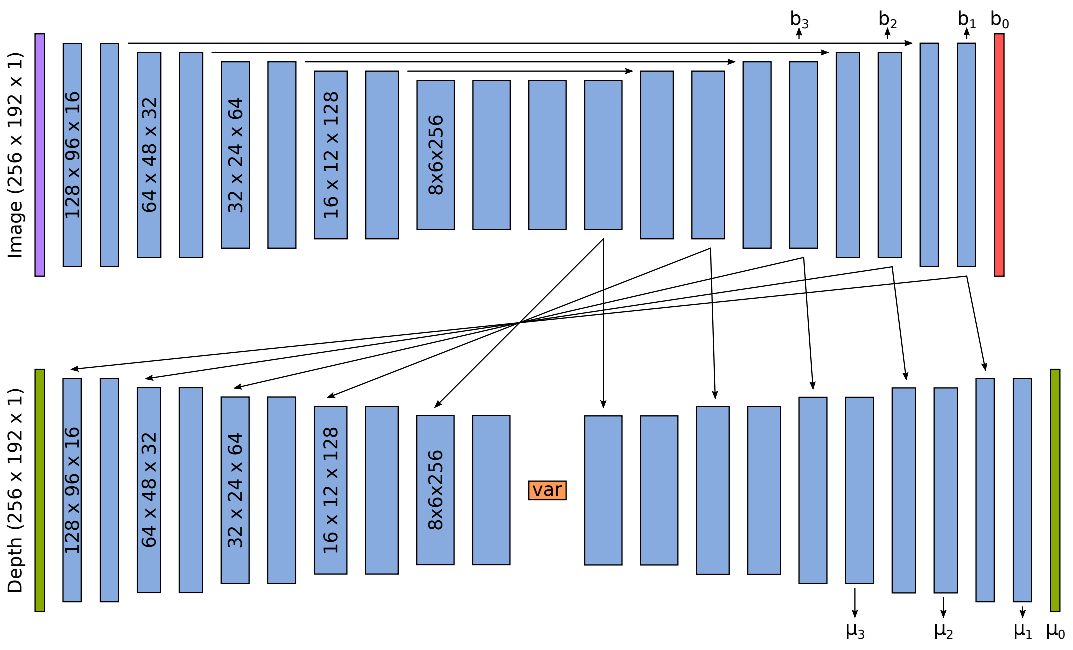


Figure 4.3: Network architecture of the variational depth auto-encoder conditioned on image intensities. We use a U-Net to decompose the intensity image into convolutional features (the upper part of the figure). These features are then fed into the depth auto-encoder by concatenating them after the corresponding convolutions (denoted by arrows). Down-sampling is achieved by varying stride of the convolutions, while up-sampling uses bilinear interpolation (except for the last layer which uses a deconvolution). A variational component in the bottleneck of the depth auto-encoder is composed of two fully connected layers (512 output channels each) followed by the computation of the mean and variance, from which the latent space is then sampled. The network outputs the predicted mean μ and uncertainty b of the depth at four pyramid levels.

4.2.1 Detailed Network Architecture

An overview of the network architecture is provided in Figure 4.3. The top part illustrates a U-Net [Ronneberger et al., 2015] applied to the intensity image, which first computes an increasingly coarse but high-dimensional feature representation of the input image. This is followed by an up-sampling part with skip-layers. The computed intensity features are then used to encode and decode the depth in the lower part of the figure. This part is a fairly standard variational auto-encoder architecture with again a down-sampling part and an up-sampling part. Embedded in the middle are two fully connected layers as well as the variational part, which samples the code from a Gaussian distribution and is subject to a regularisation cost (KL-divergence, see [Kingma and Welling, 2014] and Section 2.7). The conditioning of the auto-encoder is achieved by simply concatenating the intensity features of the corresponding resolution.

Instead of predicting just raw depth values, we predict a mean μ and an uncertainty b for every depth pixel. The uncertainty is predicted from intensity only and thus is not directly influenced by the code. Subsequently, we derive a cost term by evaluating the negative log-likelihood of the observed depth $\tilde{d}$. This allows the network to attenuate the cost of difficult regions and to focus on reconstructing parts which can be well explained. At test time, the learned uncertainties can also serve to gauge the reliability of the reconstruction. In the present work we employ a Laplace distribution which has heavier tails than the traditional Gaussian distribution:

$$p(\tilde{d}|\mu, b)) = \frac{1}{2b} \exp\left(-\frac{|\tilde{d} - \mu|}{b}\right) \ . \tag{4.2}$$

Discarding a constant offset, the negative log-likelihood thus becomes:

$$-\log(p(\tilde{d}|\mu, b)) = \frac{|\tilde{d} - \mu|}{b} + \log(b) \ . \tag{4.3}$$

Intuitively, the network will tune the pixel-wise uncertainty b such that it best attenuates the reconstruction error $|\tilde{d} - \mu|$ while being subject to a regularisation term $\log(b)$. Using likelihoods as cost terms is a well-established method and has previously been applied to deep learning problems in computer vision [Kendall and Gal, 2017, Clark et al., 2017a].

The b parameter could alternatively be predicted alongside μ, as an additional channel of the output of the code VAE. In our experiments, we have observed slightly better results when predicting the uncertainty from the intensity image and have not explored this further.

In analogy to previous work, we evaluate the error at multiple resolutions [Zhou et al., 2017]. To this end, we create a depth image pyramid with four levels and derive the negative log-likelihood for every pixel at every level. We increase the weight on every level by a factor of 4 in order to account for the lower pixel count.

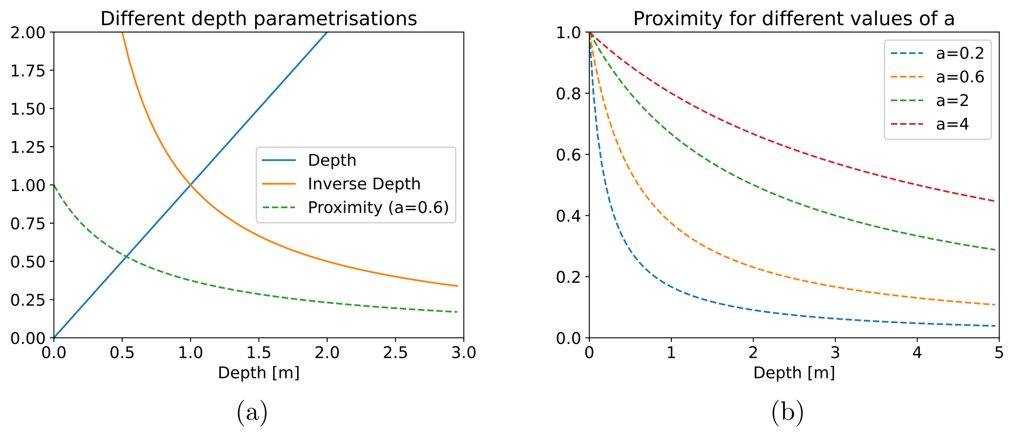


Figure 4.4: (a) Comparison of different depth parametrisations (b) proximity encoding for different values of the average depth parameter a. Best viewed on a computer screen.

Except for the computation of the latent distribution and the output channels, the activations are all set to ReLu. Furthermore, for allowing pre-computation of the Jacobians (see Section 4.4.1), we explore identity activations for the depth decoder, making the depth a linear function of the code:

$$D(c, I) = Ac + b = J(I)c + D_0(I). \tag{4.4}$$

Using concatenations for the conditioning connections between the top U-Net and the depth decoder would cause the Jacobian of the decoder to be also linear with respect to the intensity image: $J(I) = Dc + e$. Since we would like to retain a nonlinear relation between the decoder Jacobian $J(I)$ and the intensity image I, we add the element-wise multiplication of every concatenation to the concatenation itself. I.e., we increment every concatenation $[L1, L2]$ of layers $L1$ (from the previous layer of the decoder) and $L2$ (from the top, conditioning U-Net) to $[L1, L2, L1 \odot L2]$.

4.2.2 Training Setup

The depth values of the dataset are transformed to the range $[0, 1]$. We do this by employing a hybrid depth parametrisation which we call *proximity*:

$$p = \frac{a}{d + a} \; . \tag{4.5}$$

A comparison of the proposed proximity parametrisation with other parametrisations found in the literature is presented in Fig. 4.4a. Given an average depth value a, it maps the depth in $[0, a]$ to $[0.5, 1.0]$ (similar to regular depth) and maps the depths in $[a, \infty]$ to $[0, 0.5]$ (similar to inverse depth). This parametrisation is differentiable and better relates to the actual quantity that is observed in 3D reconstruction, as it is proportional to disparity measured between elements of the images taken from different viewpoints (see inverse depth parametrisation [Montiel et al., 2006]). Moreover, it allows to efficiently encode depth for a chosen value range by adjusting the parameter a, which controls the inflection point of the curve (Fig. 4.4b).

The network is trained on the SceneNet RGB-D dataset [McCormac et al., 2017b] which is composed of photo realistic renderings of randomised indoor scenes. It provides five million colour and depth images as well as semantic labeling and poses for training, out of which we only make use of the two former ones. We make use of the ADAM optimiser [Kingma and Ba, 2015] with an initial learning rate of 10^{-4}, which is reduced by a factor of 0.1 every 3 epochs. We train the network for 6 epochs with a batch size of 16 and evaluate our loss at four levels of image resolution.

4.3 Dense Warping

Due to the latent cost of the variational auto-encoder that forces the code distribution to a zero-mean normalised Gaussian, the zero code can be used to obtain the most likely single view depth prediction for the supplied intensity image $D(I, 0)$ (see Figure 4.7). However, if overlapping views are available we can leverage stereopsis to refine the depth estimates. This can be done by computing dense correspondences between the views: Given the image I_A and the estimated code $\mathbf{c}_A$ of a view A, as well as the relative transformation $\mathbf{T}_{BA} = (\mathbf{R}_{BA}, {}_B\mathbf{t}_A) \in SO(3) \times \mathbb{R}^3$ to a view B, we compute the correspondence for every pixel $\mathbf{u}$ with:

$$w(\mathbf{u}, \mathbf{c}_A, \mathbf{T}_{BA}) = \pi(\mathbf{R}_{BA}\, \pi^{-1}(\mathbf{u}, D_A(\mathbf{u})) + {}_B\mathbf{t}_A) \ , \tag{4.6}$$

where π and π^{-1} are the projection and inverse projection operators introduced in Section 2.3 and $D_A = D(I_A, \mathbf{c}_A)$ denotes the depth image obtained from the intensity image and the corresponding code. Using this notation, we can derive the photometric error using the code depth:

$$I_A(\mathbf{u}) - I_B(w(\mathbf{u}, \mathbf{c}_A, \mathbf{T}_{BA})) \ . \tag{4.7}$$

The above expressions are differentiable w.r.t. to their inputs and we can compute the corresponding Jacobians using the chain rule:

$$\frac{\partial I_B(\mathbf{v})}{\partial _B\mathbf{t}_A} = \frac{\partial I_B(\mathbf{v})}{\partial \mathbf{v}} \frac{\partial \pi(\mathbf{x})}{\partial \mathbf{x}} \ , \tag{4.8}$$

$$\frac{\partial I_B(\mathbf{v})}{\partial \mathbf{R}_{BA}} = \frac{\partial I_B(\mathbf{v})}{\partial \mathbf{v}} \frac{\partial \pi(\mathbf{x})}{\partial \mathbf{x}} (-\mathbf{R}_{BA}\, \pi^{-1}(\mathbf{u}, d))^{\times} \ , \tag{4.9}$$

$$\frac{\partial I_B(\mathbf{v})}{\partial \mathbf{c}_a} = \frac{\partial I_B(\mathbf{v})}{\partial \mathbf{v}} \frac{\partial \pi(\mathbf{x})}{\partial \mathbf{x}} \mathbf{R}_{BA} \frac{\partial \pi^{-1}(\mathbf{u}, d)}{\partial d} \frac{\partial D_A(\mathbf{u})}{\partial \mathbf{c}_A} \ , \tag{4.10}$$

where $^{\times}$ refers to the skew symmetric matrix of a 3D vector and with the abbreviations:

$$\mathbf{v} = w(\mathbf{u}, \mathbf{c}_A, \mathbf{T}_A^B) \ , \tag{4.11}$$

$$\mathbf{x} = \mathbf{R}_{BA}\, \pi^{-1}(\mathbf{u}, D_A(\mathbf{u})) + {}_B\mathbf{T}_A \ , \tag{4.12}$$

$$d = D(I_A, \mathbf{c}_A)(\mathbf{u}). \tag{4.13}$$

Most partial derivatives involved in Equations (4.8) to (4.10) are relatively well-known from the dense tracking literature [Kerl et al., 2013] and include the image gradient ($\partial I_B(\mathbf{v})/\partial \mathbf{v}$), the differential of the projection ($\partial \pi(\mathbf{x})/\partial \mathbf{x}$), as well as transformation related derivatives (also refer to [Bloesch et al., 2016] for more details). The last factor in Equation (4.10), $\partial D_A(\mathbf{u})/\partial \mathbf{c}_A$, is the derivative of the depth w.r.t. the code. It is much more computationally costly to evaluate (up to 1 sec depending on the size of the network) compared to a standard forward pass of the network, as calculating the derivative involves multiple passes (at minimum, as many times as the number of code entries). In case of a linear decoder this term can be pre-computed which significantly accelerates the evaluation of the Jacobians.

4.4 Inference Framework

4.4.1 N-Frame Structure from Motion (Mapping)

The proposed depth parametrisation is used to construct a dense N-frame Structure from Motion (SfM) framework (see Figure 4.5). We do this by assigning an unknown code and an unknown pose to every frame. All codes and poses are initialised to zero and identity, respectively. For two frames A and B with overlapping field of view we then derive photometric and geometric residuals, E_{pho} and E_{geo}, as follows:

$$E_{\text{pho}} = L_p(I_A(\mathbf{u}) - I_B(w(\mathbf{u}, \mathbf{c}_A, \mathbf{T}_{BA}))) \ , \tag{4.14}$$

$$E_{\text{geo}} = L_g(D_A(\mathbf{u}) - D_B(w(\mathbf{u}, \mathbf{c}_A, \mathbf{T}_{BA}))) \ . \tag{4.15}$$

The loss functions L_{pho} and L_{geo} have the following masking and weighting functionality:

(i) mask invalid correspondences,

(ii) apply relative weighting to geometric and photometric errors,

(iii) apply a Huber weighting,

(iv) down-weight errors on strongly slanted surfaces,

(v) down-weight pixels which might be occluded (only L_{pho}).

In order to optimise both sets of residuals w.r.t. our motion and geometry we compute the Jacobians w.r.t. all codes and poses according to Section 4.3. As mentioned above, we investigate the applicability of linear decoding networks (see Section 4.2.1) as this allows us to compute the Jacobian of the decoder $D(I, \mathbf{c})$ w.r.t. the code $\mathbf{c}$ only once per keyframe. After computing all residuals and Jacobians we apply a damped Gauss-Newton algorithm in order to find the optimal codes and poses of all frames.

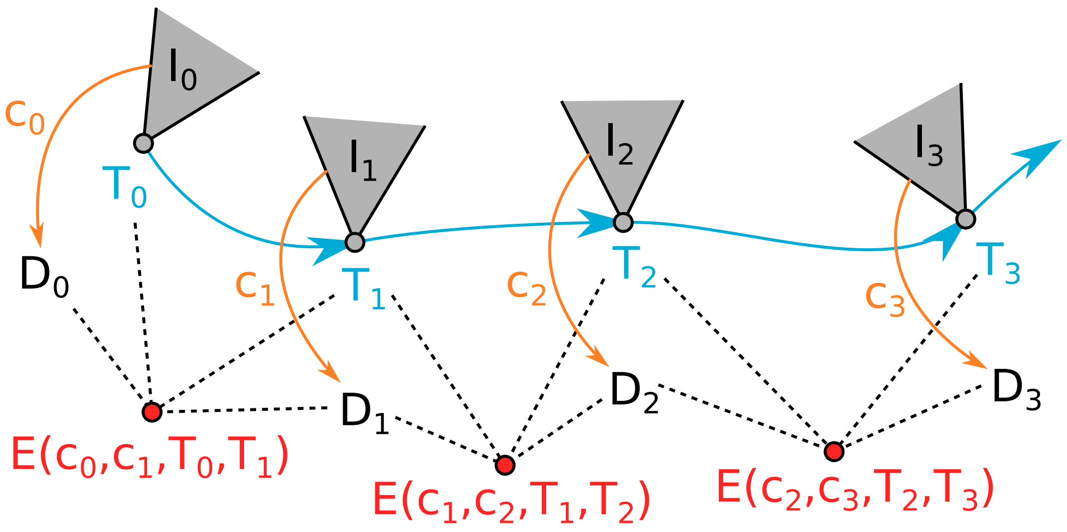


Figure 4.5: Illustration of the SfM system. The image I_i and corresponding code $\mathbf{c}_i$ in each frame are used to estimate the depth D_i. Given estimated poses $\mathbf{T}_i$, we derive relative error terms between the frames (photometric and geometric). We then jointly optimise for geometry ($\mathbf{c}_i$) and motion ($\mathbf{T}_i$) by using a standard second-order method.

4.4.2 Tracking (Localisation)

While camera localisation is performed within the SfM approach described above, it would very costly and impossible to perform at real-time speeds. We utilise a separate tracking system that estimates camera pose with respect to an existing keyframe map and run the full reconstruction system only when a new keyframe is introduced to the map, which significantly speeds up the computation.

This tracking system can be built much in the spirit of the above SfM approach. The current frame is paired with the last keyframe and the estimated relative pose results from a cost-minimisation problem. In our vision-only setup we do not have access to the current depth image (except for a rough guess), and thus in contrast to the described SfM system we do not integrate a geometric cost.

In order to increase tracking robustness we perform a coarse to fine optimisation by first doing the dense alignment at low depth image resolutions.

4.4.3 SLAM System

We implement a preliminary system for Simultaneous Localisation and Mapping inspired by PTAM [Klein and Murray, 2007] where we alternate between tracking and mapping. The initialisation procedure takes two images and jointly optimises for their relative pose and the codes of each frame. After that we can track the current camera pose w.r.t. the last keyframe. Once a certain baseline is achieved we add a keyframe to the map and perform a global optimisation, before continuing with the tracking. If the maximum number of keyframes is reached we marginalise old keyframes and thereby obtain a linear prior on the remaining keyframes. In a 4-keyframe setup, we achieve a map update rate of 5 Hz. The system currently relies on Tensorflow for image warping, and could be sped up with a more targeted warping and optimisation system which are both part of future work.

4.5 Experimental Evaluation and Discussion

Please also see our submitted video which includes demonstrations of our results and system http://www.imperial.ac.uk/dyson-robotics-lab/projects/codeslam/.

4.5.1 Image Conditioned Depth Encoding

First we present results and insights related to our key concept of encoding depth maps conditioned on intensity images.

We trained and compared multiple variations of our network. Our reference network has a code size of 128, employs greyscale image information only, and makes use of a linear decoder network in order to speed up Jacobian computation. The linear version of the decoder is obtained by replacing non-linear activation functions such as ReLU with an identity mapping. Figure 4.6 shows results on reconstruction accuracy using different code sizes as well as setups with RGB information and non-linear depth decoding. The use of colour or nonlinear decoding did not significantly affect the accuracy. With regard to code size, we observe a saturation of the accuracy at a code size of 128; there is little to be gained from making the code bigger. This

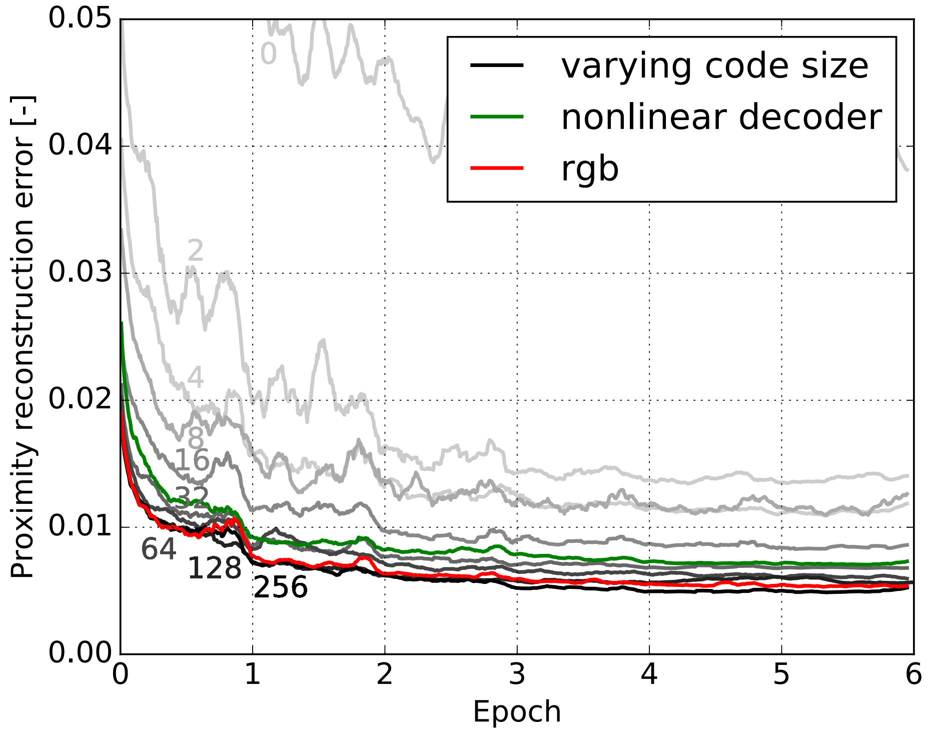


Figure 4.6: Validation loss during training on the per-pixel proximity errors. As the reference implementation, we use a network trained on greyscale images with a linear decoder. Lower losses can be achieved by increasing the code size (increasing shades of grey). Using a nonlinear decoder or colour images during training does not affect the results in a significant way.

value may be surprisingly low, but the size seems to be large enough to transmit the information that can be captured in the code by the proposed network architecture.

Figures 4.7 to 4.9 provide some insight into how our image conditioned depth encoding works. In Figure 4.7 we show how we encode a depth image into a code of size 128. Using the corresponding intensity image this can then be decoded into a reconstructed depth image, which captures all of the main scene elements well. We also show the reconstruction when passing a zero code to the decoder as well as with a code that is optimised for minimal reconstruction error. The zero code captures some of the geometrical details but fails to properly reconstruct the entire scene. The reconstruction with the optimised code is very similar to the one with

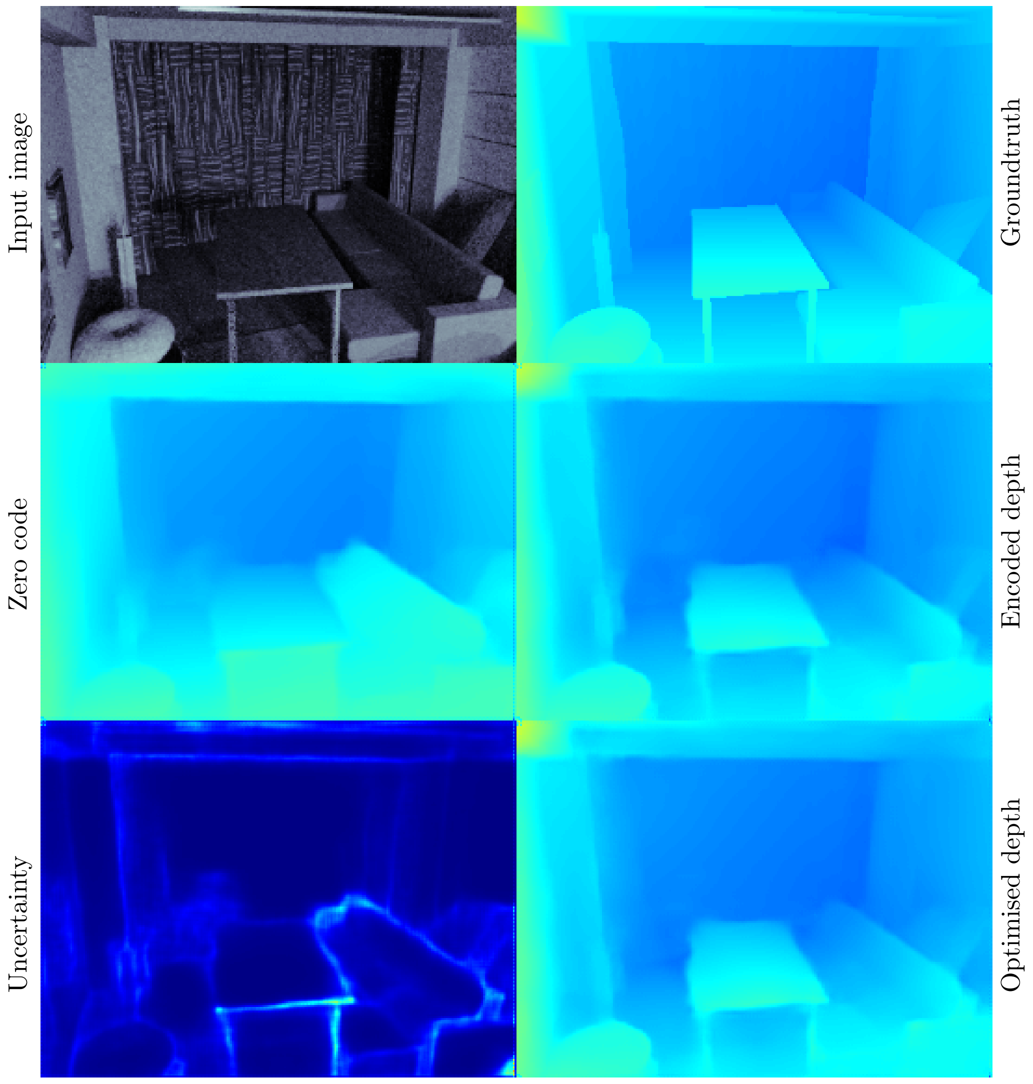


Figure 4.7: An example image passed through encoding and decoding. Top left: input image. Top right: ground truth depth. Middle left: zero code reconstruction (image only prediction). Middle right: decoded depth (code from encoder). Bottom left: estimated reconstruction uncertainty (scaled four times for visibility). Bottom right: optimised depth (code minimising reconstruction error).

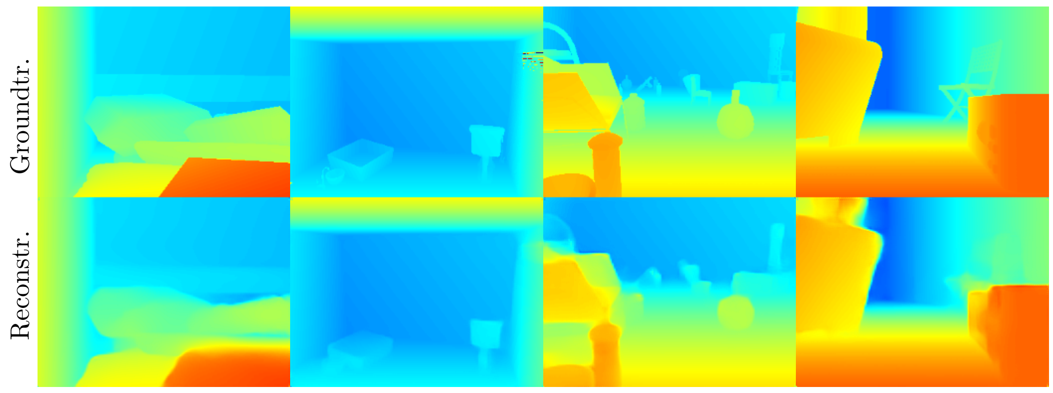


Figure 4.8: Encodings of different depth images. The encoding allows to capture even fine geometrical details.

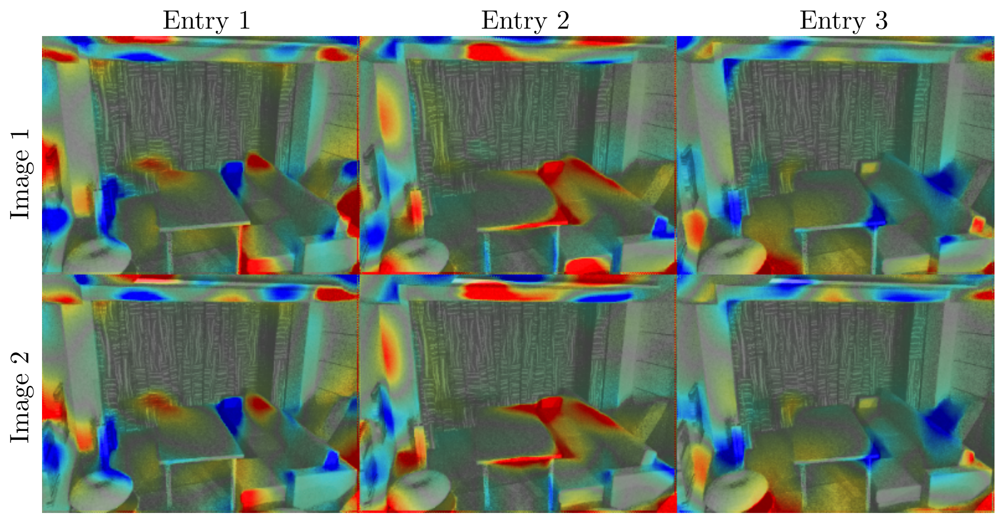


Figure 4.9: Visualisation of the influence of the code on depth reconstruction. The Jacobian of the depth w.r.t. a specific code entry is used to colourise the input image (blue and red depict negative and positive values, respectively). Columns represent code entries (1-3). Rows represent two different input images.

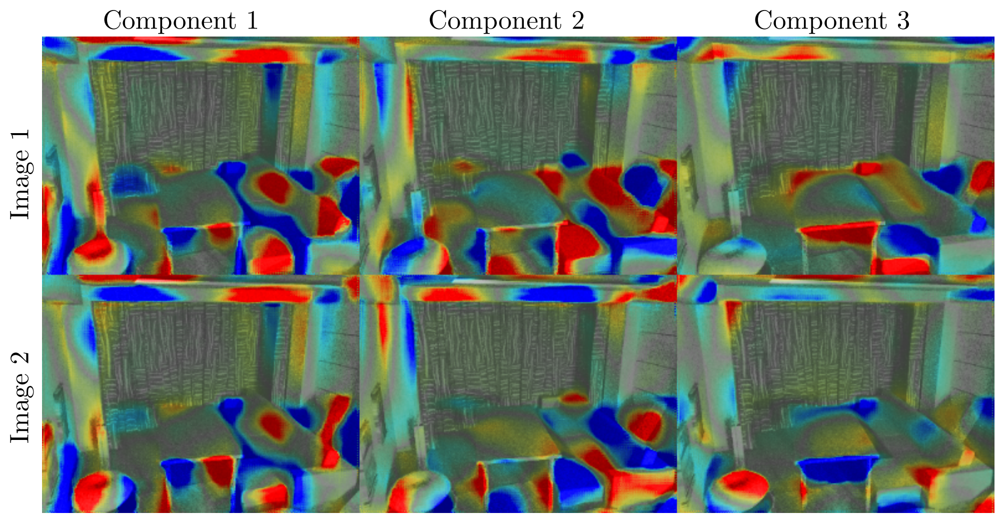


Figure 4.10: Visualisation of the first three principal components of the code Jacobian for two different but similar viewpoint (rows) using Principal Component Analysis (PCA). Blue and red regions in the image denote negative and positive values, respectively. Columns represent principal components (1-3). Rows represent different input images.

the code from the encoder which indicates that the encoder part of the network works well. The associated depth uncertainty is also visualised and exhibits higher magnitudes in the vicinity of depth discontinuities and around shiny image regions (but not necessarily around high image gradients in general). Further examples of depth encoding are shown in Figure 4.8.

In Figure 4.9 we visualise the Jacobians of the depth image w.r.t. to the code entries. An interesting observation is that the single code entries seem to correspond to specific image regions and, to some extent, respect boundaries given by the intensity image. While the regions seem to be slightly fragmented, the final reconstructions will always be a linear combination of the effect of all code entries. We also compare the regions of influence for two different but similar images and can observe a certain degree of consistency. Figure 4.10 presents the results of Principal Component Analysis of the same two views used in Figure 4.9. The first three principal components are visualised instead of raw colorised jacobian entries.

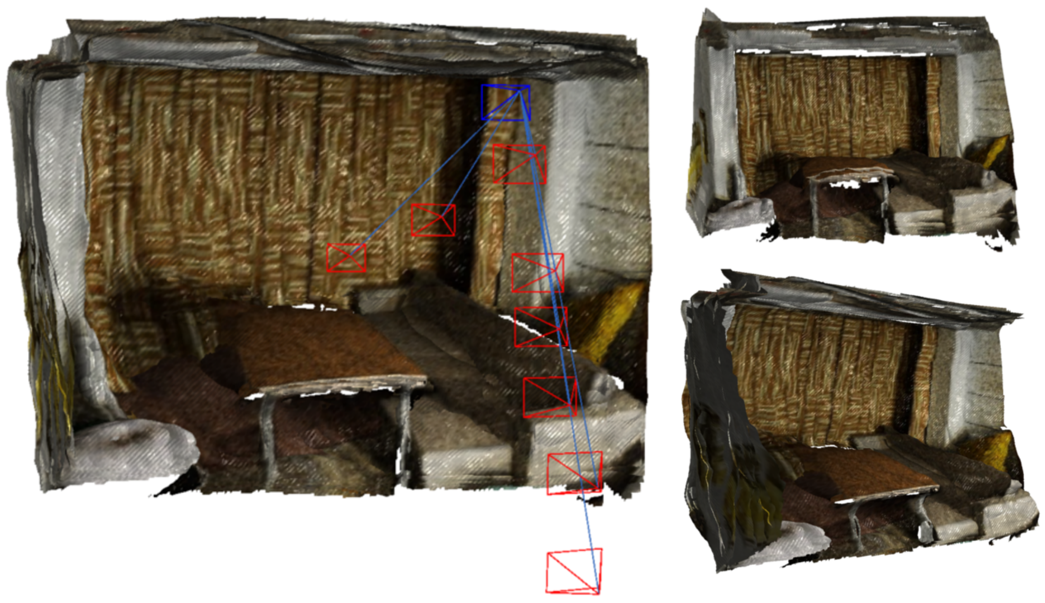


Figure 4.11: Monocular 3D reconstruction using 9 keyframes. During optimisation a selected master keyframe is paired with the other frames. The depth images of all frames are used for the 3D rendering. The employed geometric error term ensures the consistency between the depth of the different views.

# frames	1	2	3	4	5	6
RMSE [10^{-2}]	2.65	2.47	2.31	2.39	2.30	2.14

Table 4.1: RMS of pixel proximity estimation error with different amounts of master keyframe-frame pairs in the optimisation problem. The error is evaluated between the master keyframe proximity and its corresponding ground truth proximity. Frames 1-3: downward-backwards motion. Frames 4-6: left-forward motion.

4.5.2 Structure from Motion

The proposed low dimensional encoding enables continuous refinement of the depth estimates as more overlapping keyframes are integrated. In order to test this, we have implemented a SfM system which incrementally pairs one pre-selected frame with all the remaining frames (which were selected from SceneNet RGB-D). Table 4.1 shows the obtained reconstruction error w.r.t. the number of frames that are connected to the first frame. The observed reduction of the reconstruction error well illustrates

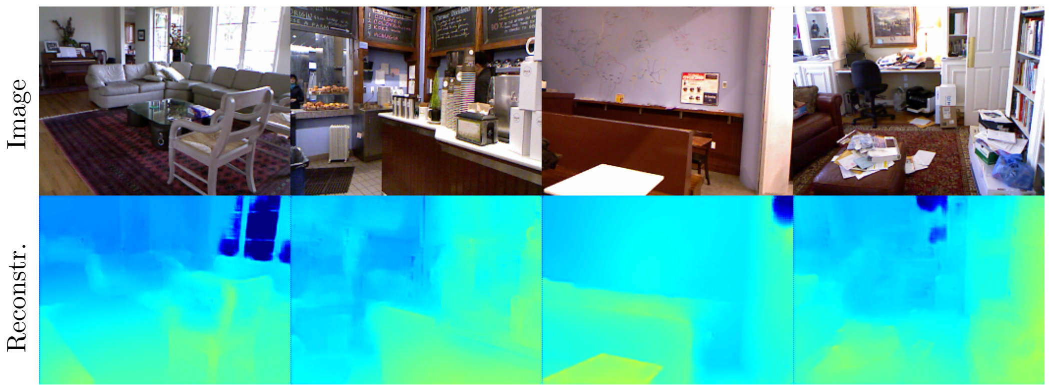


Figure 4.12: Two-frame SfM on selected pairs from the NYU V2 dataset. Top row presents one of the images used for reconstruction, while the bottom row contains respective depth estimates. The main elements of all scenes can be well perceived in the depth image. The overexposed image regions saturate to infinite depth values, which is a result of using the SceneNet RGB-D dataset for training, which contains many scenes with windows (similar to the one in the left image).

the strength of the employed probabilistic inference method, application of which is enabled by the low dimensionality of the optimisation space. The magnitude of depth refinement depends on the information content of the new frames (whether they present the scene under a new view and exhibit sufficient baseline). Figure 4.11 presents a 3D reconstruction based on 9 frames for the scene used in the above error computations. Since in this rendering all the frame depth maps are superimposed, one can observe the quality of the alignment. In a future full SLAM system, these keyframes would be fused together in order to form a single global scene. Before visualisation, high frequency elements are removed from the depth maps with bilateral filtering and highly slanted mesh elements are cropped.

Being exposed to a large variety of depth images during training, the proposed network embeds geometry priors in its weights. These learned priors seem to generalise to real scenes as well: Figure 4.1 depicts a two-frame reconstruction with images from the real image EuRoC dataset [Burri et al., 2016] taken by a drone in an industrial setting. The result corresponds to 50 optimisation steps, each taking around 100 ms to complete. Since significant exposure changes occur between the images, we perform an affine illumination correction of the frames. The validation

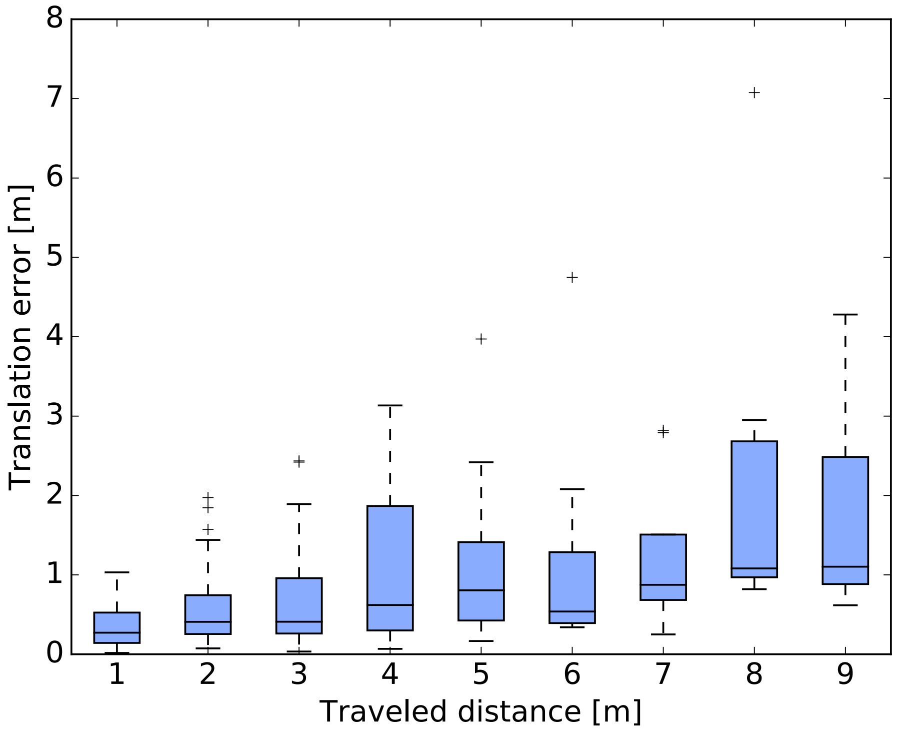


Figure 4.13: Translation error versus travelled distance on the EuRoC dataset MH02. Despite training the auto-encoder on SceneNet RGB-D, its decoder generalises to other datasets (after correcting for camera intrinsics).

of the two-frame reconstruction performance is of high importance as it is directly connected to the initialisation procedure of the full SLAM system. In order to further highlight its effectiveness we include results on a selection of pairs taken from the NYU V2 dataset [Silberman et al., 2012] (Figure 4.12).

4.5.3 SLAM System

In contrast to most dense approaches, our low dimensional geometry encoding allows joint optimisation of motion and geometry. Furthermore, due to the inherent prior contained in the encoding, the framework is able to deal with pure rotation motions. The system is tested in a sliding window visual odometry mode on the EuRoC dataset on the MH_02_easy trajectory. Even though the dataset is significantly different

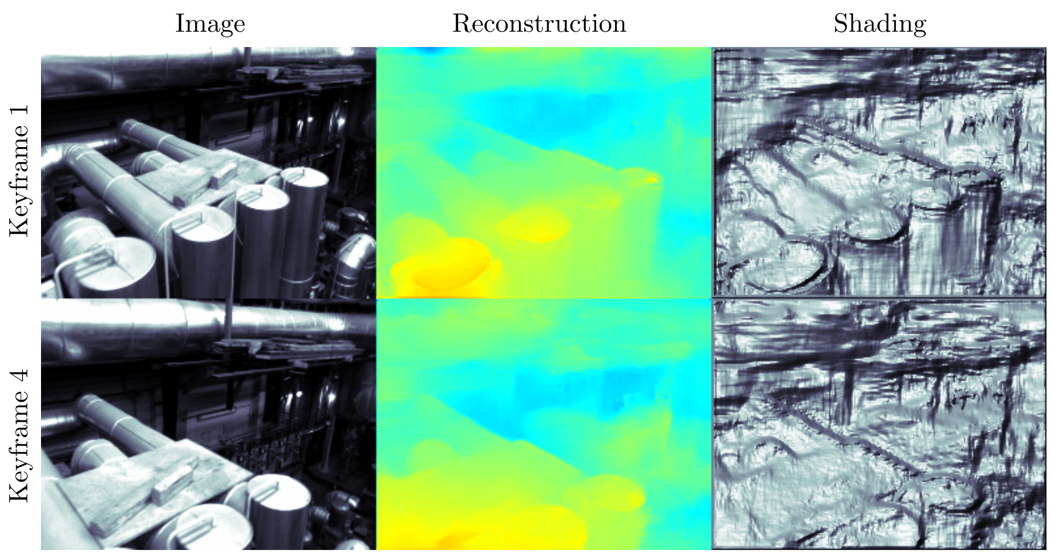


Figure 4.14: Example structure from motion results on frames from the EuRoC dataset. From the left: image, estimated proximity, shaded proximity.

from the data the network is trained on (with many metallic parts and many reflections), the proposed system is able to run through most of this dataset that is arguably very difficult for visual systems (we do not use the available IMU data).

Figure 4.13 shows the error against travelled distance. While this cannot compete with a state-of-the art visual-inertial system, it performs respectably for a vision only-system and exhibits an error of roughly 1 m for a travelled distance of 9 m. In Figure 4.14 the first and last key-frame of our 4-frame sliding window system are illustrated. This shows the intensity image of the encountered scene together with the estimated proximity image and a normal based shading. Considering that the network was trained on artificial images only which were very different in their nature, the reconstructed depth is sensible and allows for reliable camera tracking.

4.6 Conclusions and Future Work

We have shown that a learned representation for depth which is conditioned on image data provides an important advance towards future SLAM systems. By employing an auto-encoder like training setup, the proposed representation can contain generic and detailed dense scene information while allowing efficient probabilistic joint optimisation together with camera poses. This allows to tackle the correlation problem described in Section 1.2 which is prevalent in current dense SLAM methods.

In the longer term, we would like to move beyond a keyframe-based approach, where our dense geometry representations are tied to single images, and work on learned but optimisable compact representations for general 3D geometry, eventually tying our work up with 3D object recognition. Especially interesting would be learning minimal parametrisations of objects to build a hierarchical semi-semantic map for SLAM, more efficient for both inference and storage compared to the fully dense approach. Learning a code representation for larger entities such as rooms or buildings rather than depth maps is also an interesting research direction.

By replacing the input modalities (image, depth) the method presented in this chapter can also be extended to different applications. An example would be learning a manifold of human skeleton poses and optimising a photometric discrepancy between the observed image and a rendering of a view of the skeleton estimate. Other modalities such as force sensors in robotic grippers and different manifolds such as kinematic joint space models could be also used here.

In the next chapter, we will use the components demonstrated here to build a full real-time keyframe-based SLAM system and explore the impact of the optimisable code representation on traditional SLAM pipelines.

Chapter 5

Dense Probabilistic Monocular SLAM

Contents

5.1 Introduction

The previous chapter has introduced the concept of learning a compact optimisable representation and utilising it to solve the dense structure-from-motion problem. The feasibility of this idea has been proved by implementing a windowed 3D reconstruction/visual odometry (CodeSLAM). The presented system lacked features of full SLAM, was not capable running fast enough to be used with a live camera and did not generalise to real handheld camera scenes very well.

This chapter describes *DeepFactors*, a real-time SLAM system that builds and maintains a dense reconstruction but allows for probabilistic inference and combines the advantages of different SLAM paradigms. It also presents a tight integration of learning and model based methods through a learned compact dense code representation that drives significant changes to the core mapping/tracking components of the SLAM pipeline. The main contributions of this work can be summarised as follows:

- The first real-time probabilistic dense SLAM system — capable of joint optimisation of dense depth and camera poses.
- A system that integrates learned priors over geometry with classical SLAM formulations in a probabilistic factor-graph formulation.

We build on the idea of compact code optimisation and explore its impact on traditional SLAM pipelines. DeepFactors is a complete new SLAM system built from scratch with a different mapping backend, error formulations (the sparse geometric and reprojection errors) and all the various design choices within the SLAM algorithm like keyframing, map maintenance and tracking. It contains features that CodeSLAM was missing – local and global loop closure or relocalisation and optimises the full map in batch instead of a fixed window only. The optimised GPU usage,

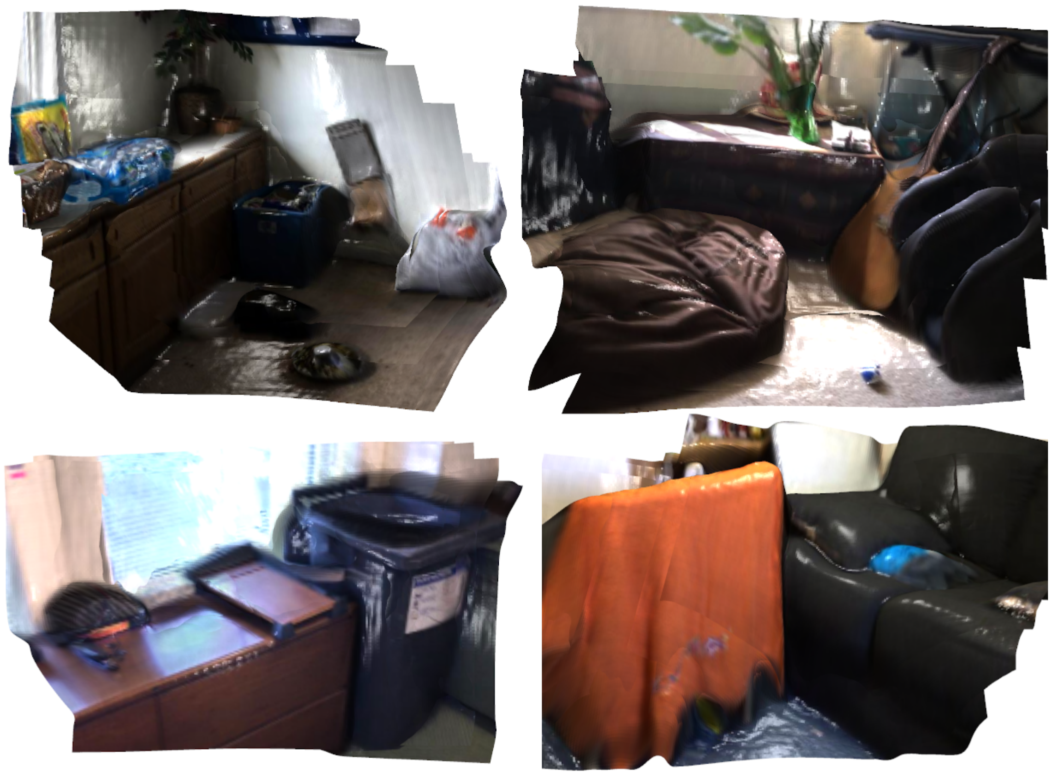


Figure 5.1: Example reconstructions of scenes from the ScanNet validation set created with our system. Reflective shading has been exaggerated in order to highlight structure.

efficient implementation and SLAM design choices enable real-time performance and the use of a standard factor graph software backend (GTSAM [Dellaert, 2012]) allows for straightforward probabilistic integration of different sensor modalities which was not possible with the previous formulations of dense SLAM.

An example of other directly comparable work is CNN-SLAM [Tateno et al., 2017]. Although with a substantially different principle of operation, the authors present a real-time full SLAM system that builds a large scale map and supports loop closures by utilising LSD-SLAM [Engel et al., 2014] and incorporating learned priors. It doesn't, however, optimise the full depth maps during mapping. We use CNN-SLAM as a baseline comparison for our method.

There exists a range of systems with learned components which are not fully-featured SLAM systems but instead focus on multi-frame dense reconstruction [Laidlow et al., 2019, Clark et al., 2018, Liu et al., 2019, Weerasekera et al., 2017]. In BA-Net [Tang and Tan, 2019] the authors use a technique similar to CodeSLAM where a set of basis depth maps is predicted from the image and optimised in a bundle adjustment problem to find a dense per-pixel reconstruction. The system has been trained end-to-end with the optimisation included which might result in a learned representation better suited for the structure from motion problem. In contrast to our work, BA-Net is not a real-time SLAM system that builds and optimises a consistent multi-keyframe map. In [Tulsiani et al., 2017], the authors introduced a differentiable ray consistency metric that allowed them to use various types of 2D observations of simple 3D objects to train a network that predicts their 3D reconstruction from a single camera view.

A different approach is to build a fully differentiable pipeline and train it on either navigation or trajectory estimation tasks. In [Gupta et al., 2017], the authors propose a network architecture that unifies mapping and navigation, estimating an egocentric 2D occupancy map which is used by a differentiable path planner to solve synthetic navigation tasks generated from 3D scene scans. In the test dataset, the 2D agent moves on a ground plane, with the rotation restricted to 90 degree turns. Moreover, the method requires perfect odometry as input, which is problematic for real use-cases.

In MapNet [Henriques and Vedaldi, 2018], the authors propose a learned mapping system that estimates an allocentric rather than egocentric 2D top-down map. The method takes in RGB-D images and extracts multi-channel features, then projects them onto the ground plane and registers the result against the previous map using convolutions. The map is then updated using the registered features. In contrast to [Gupta et al., 2017], the ground orientation of the camera is not restricted.

In [Savinov et al., 2018] the authors propose a learned topological map called semi-parametric topological memory (SPTM), which consists of a (non-parametric) graph with nodes corresponding to locations in the environment. A neural network retrieves relevant graph nodes based on observations and a planning module computes actions to reach the set goal. This approach was tested on planar mazes in a simulated environment based on the game Doom with the camera rotation restricted to ground plane rotation only.

A notable learned tracking and mapping system is DeepTAM [Zhou et al., 2018], which builds upon DTAM [Newcombe et al., 2011] by replacing both the TV-L1 optimisation and camera tracking with a deep convolutional neural network and achieves results outperforming standard model-based methods. In contrast to our work, it follows the same tracking and mapping split used in all dense methods and is not capable of real-time operation. Although the authors took special care to address the generalisation problem, their system still ultimately relies on seeing all possible variations of input data, which is hard in the case of full 6 DoF motion in real world conditions. Our approach relies less on the network generalisation as we perform optimisation that allows to correct for bad network predictions. In our system, neural networks are mainly used for obtaining an image conditioned manifold to optimise over.

In contrast to the previously mentioned methods, our work strives towards the goal of a unified real-time 6 DoF SLAM framework that works on real imagery and which incorporates both learned and model-based methods as well as dense and sparse approaches to localisation and mapping and possibly points towards a new generation of SLAM systems. In the remainder of our paper, we explain the building blocks of our system and show evaluation on real world sequences.

5.2 Code Based Optimisation

To reconstruct a dense representation of the scene geometry and estimate the camera motion we formulate a multi-view dense bundle adjustment problem. We parametrise the reconstructed geometry G as a set of depth maps at each camera frame $G = \{D_0, D_1, ..., D_n\}$. In a naïve formulation, pixels of each depth are uncorrelated and optimised independently, which makes the problem too ill-posed and costly to solve due to the large number of parameters.

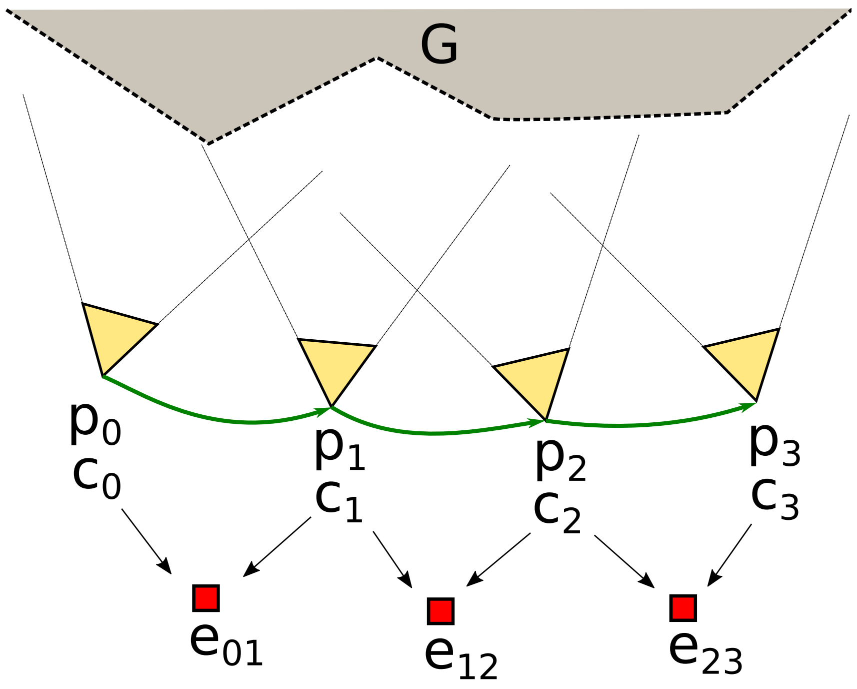


Figure 5.2: Dense multi-frame Structure From Motion problem using an optimisable compact dense code representation. Each camera frame i consists of a 6DoF world pose p_i and an associated code $\mathbf{c}_i$. We minimise various pairwise consistency losses e_{ij} in order to find the best estimate for scene geometry $G = \{D_0(c_0), ..., D_n(c_n)\}$ and camera poses $p_0...p_N$.

Following the methodology from Chapter 4 we optimise depth on a learned compact manifold (code) to mitigate both of these problems (Figure 5.2). We express the depth map D_i of a frame i as a function of code $\mathbf{c}_i$ and the associated image I_i. In order to avoid costly relinearisations during optimisation, we require this relation to be linear:

$$D_i = f(\mathbf{c}_i, I_i) = D_i^0 + \mathbf{J}(I_i)\mathbf{c}_i, \tag{5.1}$$

where $D_i^0 = f(\mathbf{0}, I_i)$ is the depth map resulting from decoding an all-zero code and $\mathbf{J}(I_i) = \frac{\partial D_i}{\partial \mathbf{c}_i}$ is the image-conditioned Jacobian.

When optimising on the code manifold, groups of depth pixels are correlated together which makes the optimisation problem more tractable. Figure 5.3 presents the pixels affected by perturbing different elements of the latent code vector.

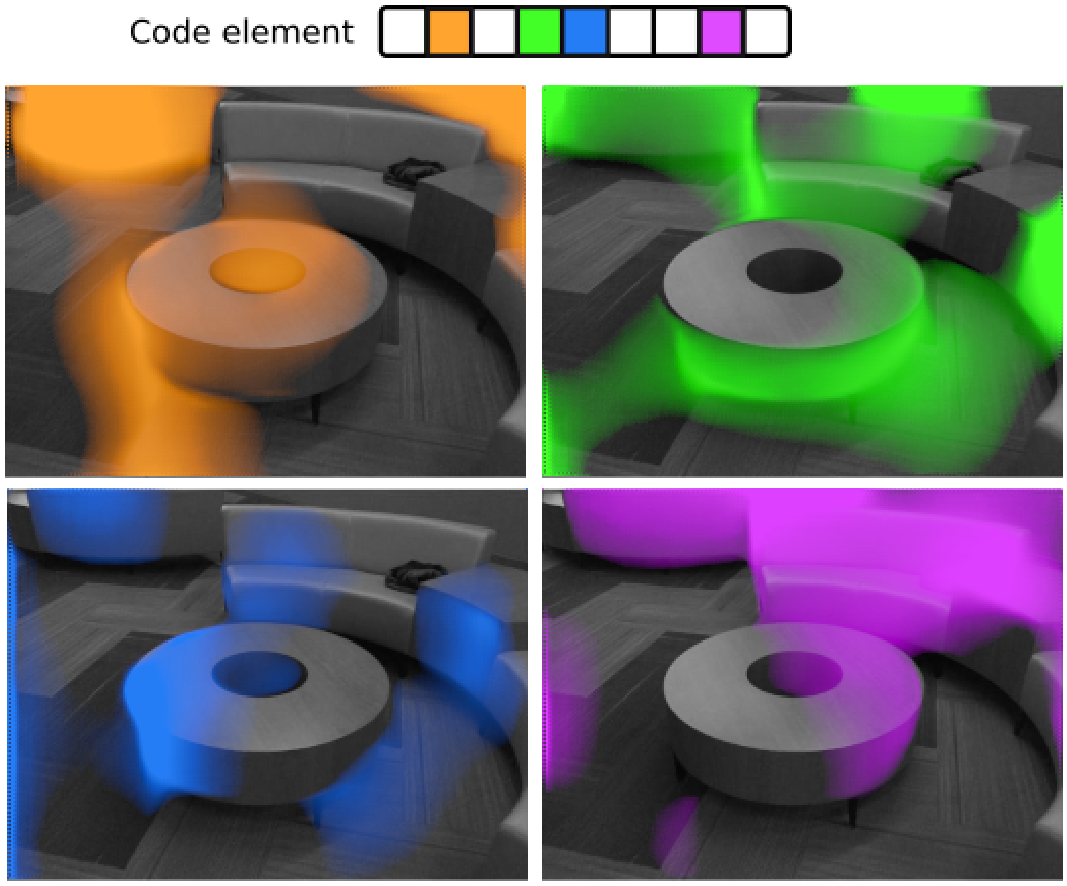


Figure 5.3: Visual representation of the code Jacobian $\frac{\partial D_i}{\partial \mathbf{c}_i}$. Perturbing different code elements affects various regions of the depth image.

Using the code representation, we reformulate three different objective functions found in the SLAM literature: photometric, reprojection and geometric error. These functions measure consistency between observations from the overlapping camera frames and allow for finding the best estimate of scene geometry and camera motion. We use them as pairwise constraints in the factor graph used by our SLAM system. The following sections describe the different factors in greater detail.

5.2.1 Photometric Factor

A consistency loss typically used in Dense SLAM is the photometric error, which is based on differences between the RGB pixel values and forms the foundation of our method. Due to its whole-image nature, it is resilient to image blurring and provides a signal in most areas of the image, which is strongest in the areas with good image gradient. The photometric factor measures the difference directly between source image intensities I_i and the target image I_j warped into frame i:

$$e^{ij}_{pho} = \sum_{\mathbf{x}\in\Omega_i} ||I_i(\mathbf{x}) - I_j(\omega_{ji}(\mathbf{x}, \mathbf{c}_i, I_i))||^2, \tag{5.2}$$

where ω_{ji} warps pixel coordinates $\mathbf{x}$ in frame i to frame j:

$$\omega_{ji}(\mathbf{x}, \mathbf{c}_i, I_i) = \pi(\mathbf{T}_{ji}(\pi^{-1}(\mathbf{x}, D_i(\mathbf{x})))), \tag{5.3}$$

where π and π^{-1} are the projection and reprojection function respectively, $D_i(\mathbf{x}) = D(\mathbf{x}, \mathbf{c}_i, I_i)$ is the depth map decoded from code $\mathbf{c}_i$ and $\mathbf{T}_{ji} \in SE(3)$ is the relative 6DoF transformation from frame i to j.

5.2.2 Reprojection Factor

We also use the indirect reprojection error widely used in classical structure from motion. This error is based on detecting sparse keypoints in the images, typically corners. The small number of features are matched using their descriptors, one of them is reprojected in the 3D space using depth obtained from the code representation and projected onto the other camera. The error is measured in the image plane — a distance between the reprojected feature and the location of its matched counterpart. Finding explicit correspondences between point features is very robust and provides a strong signal for the camera pose and the keyframe depth, but lacks information for the rest of the image in the traditional formulation of the reprojection error. Using our code representation, the reprojection error 'pins' the depth in the feature locations and the optimisation over the learned manifold adapts the rest of the depth map according to the priors learned from data.

Given a set of salient image features M_{ij} matched between frame i and j, the reprojection factor measures the differences between their observed and hypothesised locations:

$$e^{ij}_{rep} = \sum_{(\mathbf{x},\mathbf{y})\in M_{ij}} ||\omega_{ji}(\mathbf{x}, \mathbf{c}_i, I_i) - \mathbf{y}||^2. \tag{5.4}$$

To handle mismatched features, we use the Cauchy robust cost function that has a constant response to outliers. We use BRISK [Leutenegger et al., 2011] to detect and describe key points in images.

5.2.3 Sparse Geometric Factor

Another way to express consistency is to measure differences between the scene geometry in the 3D space. In our simplified, form, we compare depth map D_j with depth map D_i warped into frame j:

$$e_{geo}^{ij} = \sum_{\mathbf{x}\in\Omega_i} ||[\mathbf{T}_{ji}(\pi^{-1}(\mathbf{x}, D_i(\mathbf{x})))]_z - D_j(\hat{\mathbf{x}})||^2, \tag{5.5}$$

where $\hat{\mathbf{x}} = \omega_{ji}(\mathbf{x}, \mathbf{c}_i, I_i)$ and $[\mathbf{x}]_z$ denotes taking the z component of the vector $\mathbf{x}$. Maximising the consistency of the estimated geometry ensures that the optimised keyframes align in the 3D space. This is especially important for untextured, flat objects, which result in parts of depth not being constrained by the photometric and the reprojection error. Since there is no signal in these areas during the optimisation, the learned manifold might produce unrealistic results in order to fit to the other image regions where the signal is stronger. Therefore, it is important to constrain the two depth maps to remain similar, which introduces a prior about existence of a single observed surface and allows estimating geometrically consistent keyframe maps.

We use the Huber norm [Huber, 1964] on the error as a robust cost function. In order to save computation, we evaluate the loss only for a sparse set of uniformly sampled pixels. It is possible to sample a different set of pixels at each iteration to stochastically optimise the loss over the whole image.

5.3 Network Architecture

For learning the compact code representation we use an improved version of the network used in CodeSLAM. The full network architecture is presented in Figure 5.4. The middle part represents a U-Net [Ronneberger et al., 2015] extracting

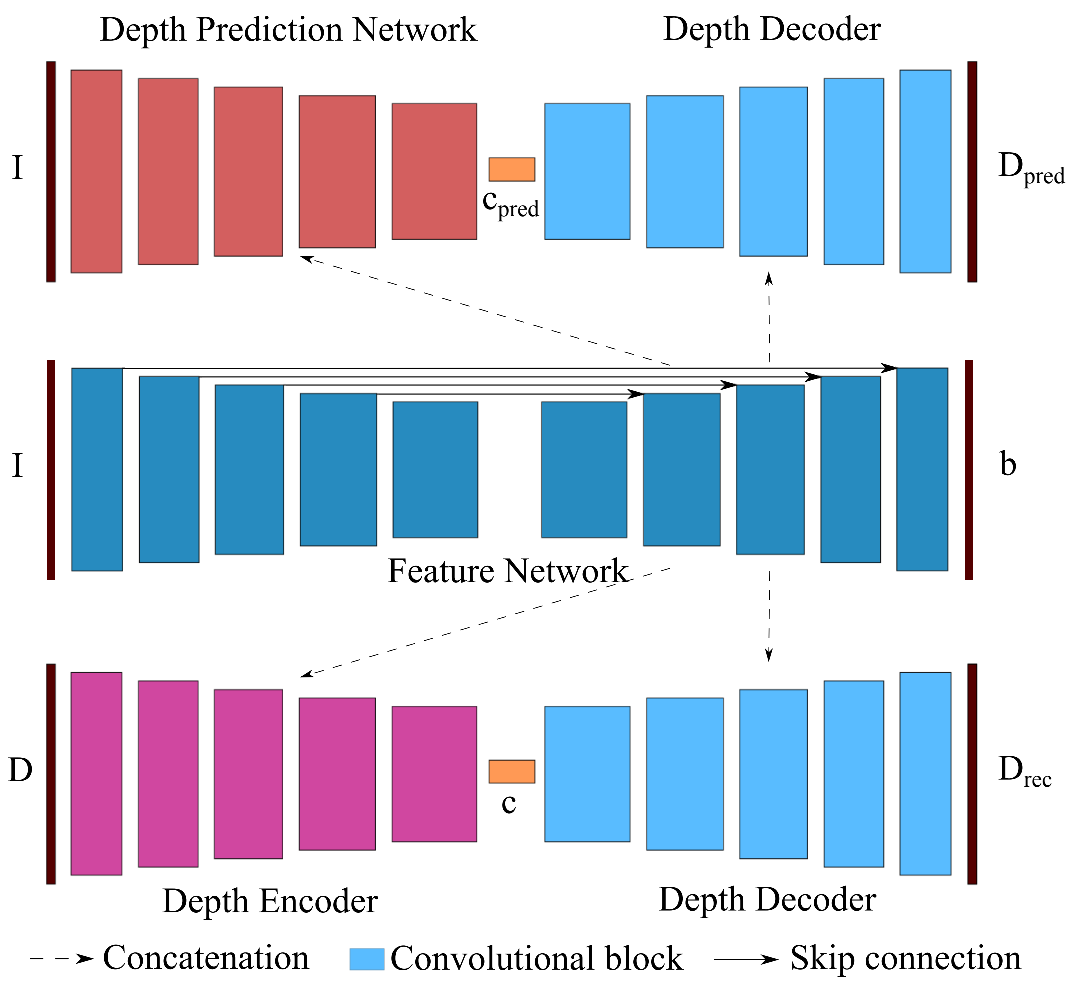


Figure 5.4: Network architecture. The bottom path is a U-Net depth auto-encoder without skip connections that learns the optimisable compact code c. Its encoder and decoder are conditioned on image features concatenated from the Feature Network. The top part of the network learns to predict optimal code c_{pred} from the input RGB image that is decoded into a mono depth prediction. The depth decoder is shared between the two networks (light blue).

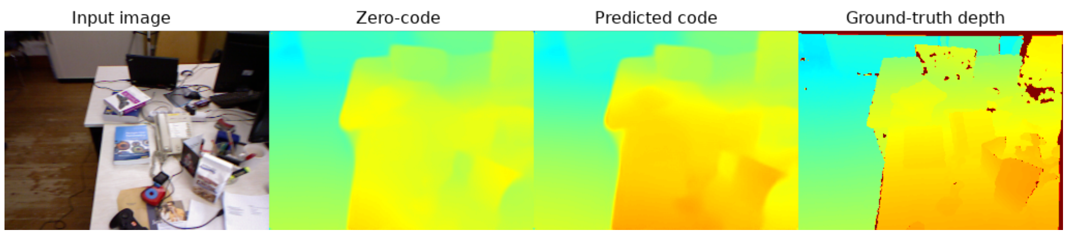


Figure 5.5: A comparison of two ways of initialising new depth maps given an input intensity image: zero-code and explicit code prediction. The inclusion of an additional network that predicts an initial code directly from the image allows for better initial depth estimates.

features from the input RGB image. The input is processed with blocks of convolutions with each block reducing the size and applying multiple convolutions on the reduced resolution. The bottom part of the figure depicts the main Variational Auto-Encoder(VAE) [Kingma and Welling, 2014] that learns the optimisable compact depth representation. The encoder and decoder are conditioned on the above-mentioned features using concatenations.

Similar to CodeSLAM, in order to keep the relation between the reconstructed depth D_{rec} and the code c linear, we do not use any non-linear activations in the depth decoder. To also ensure that the input image retains influence on that relation, we add an element-wise multiplication of each two concatenated layers in the conditioning concatenations.

Due to the KL-divergence based latent loss applied to the code of the depth VAE, an all-zero code corresponds to a most likely depth map for the input image I. In our experiments with running the system on a real camera, we have found that we can achieve better initial depth predictions by augmenting the network with a separate encoder that explicitly predicts an optimal code from the input image I. For the predicted code to lie in the same space as the learned code c, we apply the same latent loss to it. The added explicit network path is less constrained than the zero code prediction, which allows us to achieve better results, as shown in Figure 5.5.

The network also predicts an uncertainty parameter b with the Feature Network which is used in a multi-resolution negative log of Laplacian likelihood loss for the reconstructed depth D_{rec}:

$$\sum_{x \in \Omega} \frac{|D_{rec}(x) - D(x)|}{b(x)} + \log(b(x)),$$

where Ω is the set of all pixel coordinates of the input depth D. The predicted depth map is supervised with an L1 loss:

$$\sum_{x \in \Omega} |D_{pred}(x) - D(x)|. \tag{5.6}$$

5.4 System

The system builds and maintains a keyframe map. Incoming new camera images are resized and corrected to match the network focal length and tracked against the nearest keyframe (Section 5.4.1). Once sufficient baseline and other criteria are met, a new keyframe is initialised at the estimated pose with an initial code prediction and added to the graph (Section 5.4.3).

To optimise the map, we maintain a factor graph of the batch MAP problem and optimise it each time new observations are introduced. Each new keyframe is connected to the last N keyframes in the map using selected pairwise consistency factors as presented in Section 5.2. Real-time performance of solving for the batch solution is achieved by using an incremental mapping algorithm (Section 5.4.2).

To increase performance, the system alternates between tracking and joint mapping. After creating a new keyframe, the graph is optimised for a set number of iterations or until convergence. During that time the tracking and mapping optimisation steps are interleaved to ensure that the system keeps up with incoming new camera images.

To obtain a good quality photometric signal a mixture of low and high baseline image pairs is required. Since we only connect the last N keyframes with pairwise constraints, this might not always be the case. To mitigate this while keeping the computational complexity low we introduce "one-way" frames that do not have attached depth and are used to feed information to refine the latest keyframe. After optimising for a set amount of steps, active one-way frames are marginalised and removed from the graph. This allows for inexpensive integration of many views into the optimisation and high quality reconstruction of depth.

The details of each component of the system are described in the following sections.

5.4.1 Camera Tracking

Each camera frame is tracked against the closest keyframe using our GPGPU implementation of a standard direct whole-image SE3 Lucas-Kanade [Lucas and Kanade, 1981, Newcombe, 2012, Lovegrove, 2011]. In case tracking is lost, we perform relocalization by attempting to align a small resolution image against all keyframes.

5.4.2 Incremental Keyframe Mapping

We formulate and jointly optimise a batch MAP estimation problem involving all keyframes in the map. Figure 5.6 presents a factor graph representation of an example instance of a map built by our system. Each keyframe is represented by a pose p_i and a code c_i variable with variables of neighbouring keyframes involved in pairwise consistency factors (photometric, reprojection or geometric factors). Since the factors were designed to represent a single-way warping between two keyframes, they allow optimisation of the code/depth of only a single keyframe of the pair. Two factors are needed to optimise both keyframes. Because during training the code manifold is enforced to be close to a zero-mean Gaussian (variational latent loss), the code has to be kept within the appropriate region during optimisation by using zero-code prior factors that regularise it.

The mapping step performs batch optimisation of all keyframes in the map using standard factor graph software (GTSAM [Dellaert, 2012]). To make optimisation feasible in large scale scenarios we rely on an incremental mapping algorithm — iSAM2 [Kaess et al., 2012]. It stores a factorisation of the batch Jacobian in the form of a Bayes Tree and incrementally updates it when new variables are added.

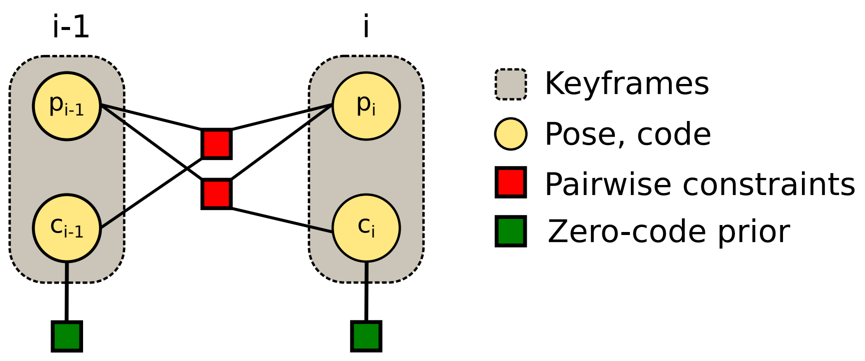


Figure 5.6: An example instance of the factor graph used in our system containing three keyframes and two one-way frames attached to the latest keyframe. Multiple different pairwise constraints can be used simultaneously (photometric, geometric or reprojection) but for simplicity, only a single type has been presented in this figure.

Only the affected factors are relinearised and re-factorised, which greatly limits the computation required for obtaining the batch solution and allows near constant time updates in an exploration-type movement.

In order to enable marginalisation of one-way frames, we enforce a specific elimination order of the graph variables to ensure that they are leaves in the clique tree built and maintained by iSAM2.

5.4.3 Initialization

We use the explicit code prediction network to obtain an initial code for each new keyframe. The depth decoder is then used to transform the codes into a depth map estimate. On start, the system can be trivially initialised in the exact same manner. The network prediction must be good enough for tracking the initial camera movement so that new views can be fed into the system to refine the depth prediction.

In cases when the single image prediction fails, a multi-frame initialisation can be used, in which a keyframe is created for each input frame and the initial graph is optimised before starting the system. We found the single frame initialisation to perform well enough for most scenes.

5.4.4 Loop Closure

We detect and close local and global loops. Within an active region of the last 10 keyframes, we assume the relative estimated poses to be correct and therefore use a pose-based criteria to detect loops and add additional pairwise constraints between keyframes. This tightens the graph and allows for more consistent local reconstructions.

At each new input frame we also test for global loop closure events. Outside of the active window we assume that too much drift has been accumulated and therefore we cannot use our estimates to close loops. Initial loop candidates are detected using a bag of words approach [Gálvez-López and Tardós, 2012] and later further eliminated by attempting to track the current frame against each of them and checking the resulting number of inliers and the estimated pose distance. Because of the photometric error typically having a smaller convergence basin we add only reprojection factors when closing a detected global loop between two keyframes.

5.5 Experimental Results

5.5.1 Training

We have trained our network on a fragment of the ScanNet dataset [Dai et al., 2017] following the official training/test split. The depth data comes from a real sensor and therefore contains missing values. Since the dataset provides PLY models of the full scene reconstructions together with ground truth poses, we have rendered depth maps from the models and combined them with the raw sensor data to obtain final merged depth maps. We used around 1.4M images in total.

The network was trained for 13 epochs with learning rate 0.0001, image size 256×192 and code size 32. This is the maximum code size that we can achieve real-time results with in our SLAM system.

5.5.2 Ablation Studies

DeepFactors combines three different error metrics/factors – photometric, geometric and reprojection to estimate the camera trajectory and the observed scene geometry. In order to determine how each factor type influences the system performance, we have evaluated various configurations in terms of the quality of the estimated trajectory and reconstruction. The photometric factor is treated as a baseline system and the other two types are included both individually and together. We perform this evaluation on selected shortened scenes from the validation set of the ScanNet dataset and use three error metrics: RMSE of the Absolute Trajectory Error (ATE-RMSE), the absolute relative depth difference (absrel) [Eigen et al., 2014] and the average percentage of pixels in which the estimated depth falls within 10% of the true value (pc110) [Tateno et al., 2017]. The results are presented in Table 5.1.

The inclusion of either of the factors reduces the trajectory error and increases the reconstruction accuracy, with the reprojection factor typically having a stronger influence on the former and the geometric factor on the latter. Explicit feature matching utilised within the reprojection error improves local minima avoidance and increases convergence rate, which adds robustness to the system. The geometric error introduces a prior about the world that only a single surface is observed and pins separate depth maps together to form a single reconstruction in the textureless areas that lack photometric information. Combining all three factors achieves the best trajectory and reconstruction results.

5.5.3 Reconstruction

We evaluate our reconstruction accuracy against CNN-SLAM as it is the only relevant system that evaluates reconstruction using the whole estimated trajectory and without using the ground-truth poses. Since the authors do not provide implementation of their system, we follow their evaluation strategy by taking the results reported in their paper and using the same sequences from the ICL-NUIM [Handa et al., 2014] and TUM [Sturm et al., 2012] datasets. For all the keyframes produced by each system, we compare their estimated depth against the ground-truth by cal-

Factor Comparison				
Sequence	**Factors Used**	**ATE RMSE↓**	**absrel↓**	**pc110↑**
scene0565_00	pho	0.128	0.108	57.56%
	pho+rep	**0.112**	0.104	59.80%
	pho+geo	0.115	0.103	59.75%
	combined	0.114	**0.102**	**60.13%**
scene0084_00	pho	0.131	0.085	69.14%
	pho+rep	0.074	0.082	71.05%
	pho+geo	0.120	0.081	71.81%
	combined	**0.061**	**0.077**	**73.66%**
scene0606_02	pho	0.089	0.214	37.22%
	pho+rep	0.071	0.201	39.39%
	pho+geo	0.067	0.168	44.45%
	combined	**0.066**	**0.162**	**46.16%**

Table 5.1: Comparison of the influence of different factor types on the estimated trajectory and reconstruction errors (*pho:* photometric. *geo:* geometric, *rep:* reprojection)

.

culating the percentage of the pixels for which the depth is within 10% of the true value. The results are presented in Table 5.2.

Since our system is monocular and does not produce up-to-scale trajectories and reconstructions, we use the optimal scale calculated with the TUM benchmark scripts to scale both the trajectory and the depth maps (as done in e.g. [Zhou et al., 2017]).

We outperform all compared methods on most of the sequences and on the average. Our system performs worse on the tum/seq2 trajectory as the camera observes a flat textured wall and due to the small code size (32) used in our system we are typically not able to represent fully flat depth easily.

We also visually present example reconstructions created by our system in Figure 5.7.

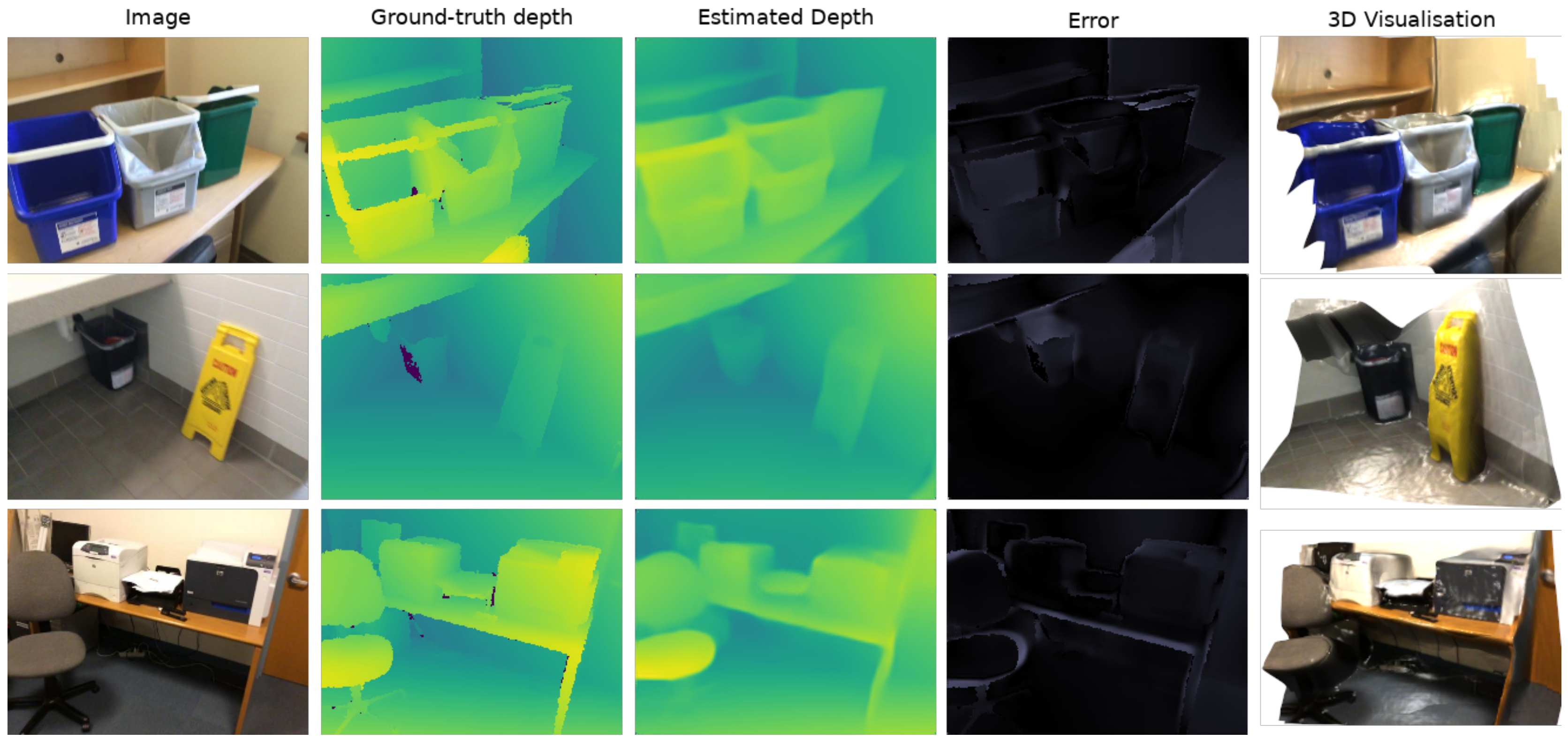


Figure 5.7: Example reconstructions on selected scenes from the validation set of the ScanNet dataset. The specular lighting in the 3D visualisation has been exaggerated to better highlight structure. Best viewed on a computer screen.

Perc. Correct Depth [%]				
Sequence	Ours	CNN-SLAM [Tateno et al., 2017]	LSD-BS [Engel et al., 2014]	Laina [Laina et al., 2016]
icl/office0	**30.17**	19.41	0.60	17.19
icl/office1	20.16	**29.15**	4.76	20.84
icl/living0	**20.44**	12.84	1.44	15.01
icl/living1	**20.86**	13.04	3.03	11.45
tum/seq1	**29.33**	12.48	3.80	12.98
tum/seq2	16.92	**24.08**	3.97	15.41
tum/seq3	**51.85**	27.40	6.45	9.45
Avg.	**27.10**	19.77	3.44	14.62

Table 5.2: Evaluation of average percentage of pixels across all estimated keyframes for which the estimated depth falls within 10% of the true value on the ICL-NUIM and TUM datasets (tum/seq1: *fr3_long_office_household*, tum/seq2: *fr3_nostructure_texture_near_withloop*, tum/seq3: *fr3_structure_texture_far*)

5.5.4 Trajectory Estimation

We evaluate the Absolute Trajectory Error (ATE) of our proposed system and compare it against CNN-SLAM and CodeSLAM. For the sake of completeness, we also include comparison with DeepTAM, despite the fact that it does not achieve interactive (real-time) performance. We have used the same version of CodeSLAM that was used to generate the results in its respective paper. The code for CNN-SLAM is not available and the authors of DeepTAM do not provide a combined tracking and mapping system and we were not able to reproduce the results reported in the paper. For this reason, we include the numbers from the DeepTAM paper and evaluate our system on the same set of trajectories. We omit the room and plant sequences as they contain significant number of dropped frames, which skew the results. In order to limit computation we have disabled the geometric factor for the evaluation.

The results are presented in Table 5.3. We outperform CodeSLAM in all of the sequences and CNN-SLAM in all but one. In most cases we also achieve comparable results to DeepTAM, while still maintaining real-time performance.

Abs. Trajectory Error [m]				
Sequence	Ours	CNN-SLAM	DeepTAM*	CodeSLAM*
fr1/360	0.142	0.500	**0.116**	0.165
fr1/desk	0.119	0.095	**0.078**	0.654
fr1/desk2	0.091	0.115	**0.055**	0.181
fr1/rpy	**0.047**	0.261	0.052	0.078
fr1/xyz	0.064	0.206	**0.054**	0.170

Table 5.3: Evaluation of our system on the validation sequences of the TUM RGB-D Benchmark [Sturm et al., 2012]. We compare the ATE RMSE[m] against CNN-SLAM and DeepTAM, results for which have been taken from [Zhou et al., 2018]. CNN-SLAM was run without pose graph optimisation and our system without loop closures. Both CNN-SLAM and DeepTAM were initialised with the network from CNN-SLAM and we used our own initialisation. We've omitted the room and plant sequences as they contain significant frame drops that might skew the results. ***Not real-time performance**

5.6 Performance and Implementation

For a visual demonstration of the speed of the system please see the associated video, which contains real-time video recordings of our system in action.

Our SLAM system has been implemented in C++ with the dense image warping, optimisation and camera tracking offloaded to GPU with CUDA, while the reprojection and the sparse geometric error factors are computed on the CPU. We run the network, CUDA kernels and visualisation on a single NVIDIA GTX 1080 GPU and use image resolution of 256×192.

The network is ran on initialisation of each keyframe in order to obtain an initial code (depth) prediction and the Jacobian $\frac{\delta D}{\delta \mathbf{c}}$ that is used later in optimisation. This requires around 340 ms, with only 16 ms of it spent on the forward pass of the network and the rest on calculating the Jacobian using *tf.gradients*. This is due to the inefficiency of its backward-mode auto-differentiation based implementation which is optimised for gradients of scalar functions commonly used in machine learning. In our case, obtaining the derivative of each output pixel with respect to the latent representation requires a significant number of passes through the network. This time can be drastically reduced with engineering effort and we predict it should be possible to reduce the overall time required to run the network to around 30 ms.

The incoming new camera images are tracked against the latest keyframe at around 250 Hz. Once the system initialises a new keyframe, we optimise the whole map representation in batch until convergence. The mapping steps are interleaved with tracking the camera in order to keep up with new images.

The overall performance of the system varies greatly depending on the amount of connectivity between neighbouring keyframes specified by the user, the types of factors enabled, the number of factors relinearised within the iSAM2 algorithm and the occurrence of loop closures. With an explorative-type motion we achieve interactive real-time speeds, where we typically limit our system to only use the photometric and reprojection error to allow it to keep up with the fast camera movement. In a local reconstruction scenario like tabletop AR or room scale reconstruction where there is less exploration we can enable the geometric error to obtain better quality reconstructions. It is possible to further speed up the system performance through engineering effort which is part of planned future work.

The implementation of our system has been released on GitHub and is available under the following link: *https://github.com/jczarnowski/DeepFactors*

5.7 Conclusions

We have presented DeepFactors, a real-time probabilistic dense SLAM system built using the concept of a learned compact depth map representation. We have demonstrated that our system achieves greater robustness and precision by combining different paradigms from classical SLAM with priors learned from data in a standard factor-graph probabilistic framework. The use of a standard framework allows it to be easily extended with different sensor modalities, which was not previously possible in the context of purely dense SLAM. An efficient C++ implementation and careful choices in the SLAM design enable real-time performance.

Combining the strengths of different consistency metrics found in the SLAM literature allowed achieving improved results compared to the state of the art real-time deep learning SLAM systems. Explicit feature matching used in the reprojection factor provided robust and strong signal for camera poses and parts of depth. The geometric factor ensured consistency in textureless, unconstrained image regions allowing to reconstruct a geometrically coherent keyframe map.

In the future, we would like to explore the idea of including the structure-from-motion optimisation within the compact depth code training. This could allow obtaining a code manifold that is specifically trained to be later used in a mapping environment. Moreover, learning the code representation in an unsupervised manner based on the intensity images only could be an interesting experiment. An inclusion of a relative-pose prediction network could also robustify the camera tracking.

We would also like to work on improving the performance of the current system, focusing on a faster method of obtaining the network Jacobian and a better GPU implementation of the geometric factor.

While our system can achieve good results when carefully tuned, its performance degrades rapidly when encountering an environment that is too dissimilar from the training data, which seems to be a general issue with neural networks. This points towards the idea that perhaps using the system in more constrained settings could help generalisation. We would like to test its performance on simpler tasks, such as using a camera that is always oriented horizontally, introducing a purely planar motion constraint or utilising information from other sensors such as IMU or a depth camera.

Chapter 6

Conclusions and Future Work

A number of contributions have been presented in this thesis that focused on the search for new representations for 3D geometry, which transformed the SLAM pipeline or increased its robustness. We proposed different ways for combining deep learning and classic geometrical methods, addressing the shortcomings of dense SLAM presented in Section 1.2. In particular, an image transformation was proposed that allows data association without relying on the brightness constancy assumption. A learned code based depth representation was developed to enable joint solving for poses of the cameras and scene structure in real-time. This code was later used as the foundation of a novel SLAM system, possibly pointing towards a new branch of pipelines for Spatial AI.

Chapter 3 presented a real-time spherical mosaicing SLAM system that utilises Semantic Texture for tracking — a hierarchy of feature maps generated by a standard CNN trained for classification. The experiments on long-term camera tracking across dynamic lighting conditions demonstrated improved robustness and a simple feature selection strategy was proposed that allows for tuning the robustness versus performance trade-off to the specific task requirements.

While this idea gave us insight into the operation of convolutional neural networks and allowed us to gain knowledge about building real-time systems involving deep learning, a question remains open as to whether learning a specific set of features for tracking would provide better performance. This approach has been recently explored in work such as BA-Net [Tang and Tan, 2019], GN-Net [Stumberg et al., 2019] or [Lv et al., 2019], which showed promising results. Our intuition is that learning features should bring performance improvements, but might introduce other practical problems such as limited generalisation across input data. We argue that using a generic set of features for representing the image can be useful in a general robotic system, where it is very likely that a CNN will be present for solving a semantic task, readily producing the Semantic Texture representation as a by-product. Moreover, autonomous operation will require other quantities such as optical flow or geometry to be estimated, resulting in a combination of learned and classical methods used simultaneously. A common set of useful features could be used in all of these methods to: a) avoid repeating computation by networks, which commonly learn the same basic image transformations; b) transform the image into a more useful and robust representation capturing the most important information.

In Chapter 4, we presented a learned representation for depth which is conditioned on image data and which can be used for solving the dense bundle adjustment problem. The efficiency of the proposed method was demonstrated by building a preliminary visual odometry system in python and conducting experiments on simulated and real scenes. The use of the compact representation allowed for the first time to solve for camera poses jointly with the scene structure and with performance unmatched by methods found in the literature, effectively mitigating the correlation problem described in Section 1.2.

Even though the system showed promising results on simulated data and, while being trained on a synthetic dataset, still generalised to real sequences, we did not succeeded in achieving good performance with live imagery from a real camera. This was addressed in our follow-up work, DeepFactors.

We believe that the learned code is an advance towards a family of general geometry representations for the next generation of SLAM systems powering advanced robot autonomy. Scenes found in the real world have an inherent semantic structure that is discovered and recognised by the human brain. Exploiting that structure enables efficient reasoning about the world and could advance robotics beyond its current limited applications. We are interested in a general framework for Spatial AI that allows efficient inference, at the heart of which lies some form of a hierarchical semantic decomposition of the world. An interesting piece of work related to that idea is [Rosinol et al., 2020], in which the authors propose 3D Dynamic Scene Graphs, a unified representation for actionable spatial perception, which consists of a connected hierarchy of entities like places, structures, rooms and their connections. Such a level of scene understanding allows, for example, to command a robot with voice instructions such as: "bring the *red stool* to the *kitchen*". We predict that the ideal hierarchical representation will likely be placed on the spectrum between raw geometric information and semantic classes — not entirely human comprehensible but useful, similar to our proposed Semantic Texture. Further research in the direction of high-level scene understanding appears to be critical to future intuitive and interactive robotic products.

A different, promising direction would be to depart from the depth code tied to a single view. Removing variations of views of scenes should reduce the complexity of the problem, possibly improving generalisation and performance. Similarly, finding representations for single entities, such as rooms or objects, rather than requiring the network to encapsulate and decompose information about all possible views could help achieve better results. This direction has recently already received some attention [Sucar et al., 2020, Park et al., 2019].

Another venue of research would be investigating the learned code manifold more thoroughly, which could provide insights on the behaviour of the learned representation. In particular, performing tests similar to the ones demonstrated in MapNet [Henriques and Vedaldi, 2018] would be interesting, where the 2D top-down map of the maze consists of a grid of multi-dimensional embeddings learned by the network, similar to code from CodeSLAM. The authors have trained a Support Vector Machine (SVM) to classify each of the embedding into one of hand-crafted intuitive classes, such as long corridors, forks and dead-ends. The accuracy achieved by the

SVM trained on embeddings has demonstrated that they encode some recognisable aspects of the environment, as a side effect of the localisation task the system was trained for. A similar method could devised and used to investigate whether the codes learned by the CodeSLAM network correlate with semantic classes of objects found in the scene.

Chapter 5 introduced DeepFactors, a real-time dense probabilistic monocular system, which built on the learned depth representation from Chapter 4 to create a new SLAM framework for unifying different classical paradigms. The point of that work was to design a novel and fully featured system that is efficient and robust from the ground up, exploring the impact of the learned code on general SLAM.

We have demonstrated that DeepFactors achieves greater robustness and precision by combining different paradigms from classical SLAM with priors learned from data in a standard factor-graph probabilistic framework. The use of a standard framework allows it to be easily extended with different sensor modalities, which was not previously possible in the context of purely dense SLAM. An efficient C++ implementation and careful choices in the SLAM design enabled real-time performance. We have shared the knowledge gained by building complex SLAM systems using deep learning by releasing a high quality code implementation online.

DeepFactors required us to assign covariances associated to the different error factors: photometric, geometric and reprojection. While we have resorted to manual tuning for that work, this points towards a fundamental issue in utilising deep learning for robotics. Deciding how much trust the network outputs compared to other modalities will be required for any estimation task. On any robotic platform many different modalities of information are present and an ideal estimation algorithm would make careful use of them through probabilistic data fusion, requiring knowledge about their associated uncertainties, not straightforwardly obtained for learned components. Most literature related to that problem focuses on predicting uncertainty from the network [Kendall and Gal, 2017, Zhou et al., 2018, Peretroukhin et al., 2019, Liu et al., 2019, Fu et al., 2018] or introspection [Gal and Ghahramani, 2016]. We believe though that as with the predicted quantities themselves, the predicted uncertainty might not generalise to unseen data — the network will not realise when it is wrong. This appears to be a fundamental issue and a crucial research direction to enable next-level robotics.

In the future, we would like to explore the idea of including the structure-from-motion optimisation within the compact depth code training, which would allow obtaining a code manifold that is specifically trained to be used in a mapping environment. Moreover, including the estimation framework in the training procedure can allow automatic tuning of the covariances of the different factors, mentioned in the paragraph above, by learning them together with the other quantities.

We would also like to work on improving the performance of the current system, focusing on a faster method of obtaining the network Jacobian and a better GPU implementation of the geometric factor.

In DeepFactors, we have also focused on achieving the best performance and robustness of the system in real-life scenarios. While it achieves good results when carefully tuned, the system tends to break when used in an environment that is too dissimilar to the images from the training set. From this and our other experiences, we have learned that we should perhaps expect less of the neural networks and utilise them only for the tasks that they are very well suited for. Problems that at their core are based on pattern recognition or matching seem to be easily dominated by learned approaches, while tasks requiring knowledge about 3D geometry prove to be more difficult. Networks trained for predicting optical flow from a pair of images are typically trained on synthetic datasets and seem to generalise much better than, for example, their depth prediction counterparts. This might be caused by the fact that Convolutional Layers have been specifically designed for pattern matching, rather than reasoning about 3D relationships. An example of a system that limits the role of the network is DeepTAM [Zhou et al., 2018], which performs dense mapping by constructing the photometric cost volume and feeding it to a regularisation network that extracts the final depth estimate. The network does not have to predict the depth, only find a minimum surface in the volume and ensure smoothness. This indicates that finding the right place to use the network in is crucial to ensure the method generalises well between different data domains.

Since very few actual robotic tasks require a completely general solution, a different promising approach is to train or fine-tune the network specifically for the environment it will be used in, e.g. a set of rooms in the house or the interior of a specific factory. This brings attention to the issue of availability of labelled data, which, if we want to enable training neural networks for specific environments, makes unsupervised learning an important research problem. Obtaining the ground-truth for training is costly for almost any deep learning tasks and usually involves a large amount of human effort. In the case of SLAM, the true scene geometry or camera poses are difficult to get, typically requiring using a costly external device for laborious scanning and/or a real-time tracking system such as Vicon. An ideal solution avoiding these problems is using an unsupervised method that learns to predict desired quantities using a stream of RGB images without the need for ground-truth labels, e.g. from a recording of the sensor stream from a robot controlled manually around the room. This would also enable adapting to changes in the environment via online learning during system operation or gathering much larger and more diverse datasets which can help the generalisation problem limiting the use of deep learning in real-world scenarios. Unsupervised learning of depth and ego-motion from RGB images has lately received attention, resulting in a number of methods [Zhou et al., 2017, Yang et al., 2020, Gordon et al., 2019]. Since many the proposed algorithms have been evaluated mostly on a driving dataset (e.g. KITTI [Geiger et al., 2012]), which poses a much simpler problem than general 6DoF motion, it seems that unsupervised learning for general 6DoF motion and unconstrained geometry remains an unsolved problem. Depth images from a camera mounted on a car have a very predictable and average structure: a flat plane in the front positioned always at the same angle. The possible movement of a car also much more limited, practically reducing the pose estimation problem to two dimensions.

Introduction of novel computing hardware has always had a considerable impact on research. With the arrival of faster CPUs well-known offline methods from photogrammetry could finally be used for estimating large scale camera trajectories and scene geometry in finite time. Similarly, the availability of modern massive parallel processing using GPGPU has enabled a new generation of SLAM systems that estimate dense geometry, which fuelled research in the field of AR and scene understanding. We expect this trend to continue in the future, and wish to anticipate the types of computer vision algorithms that will be required in the new emerging computing paradigm. A promising direction is the graph processor developed by GraphCore, which stresses the importance of distributed, independent computation and communication. The prototype that embodies this paradigm called an IPU (Intelligent Processing Unit) was designed to cater more to the sparse nature of intelligent algorithms such as computations on graphs or evaluating and training neural networks. To make efficient use of it, it is imperative to understand the computational structure of Spatial AI and discover how to express it as a distributed graph problem [Davison, 2018]. This has been explored by the early work of Ortiz et al. [Ortiz et al., 2020], who demonstrated that it is possible to achieve significant increase of performance on an IPU for solving a sparse bundle adjustment using Gaussian Belief Propagation.

Bibliography

[Agrawal et al., 2015] Agrawal, P., Carreira, J., and Malik, J. (2015). Learning to see by moving. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 70, 73

[Antonakos et al., 2015] Antonakos, E., Alabort-i Medina, J., Tzimiropoulos, G., and Zafeiriou, S. P. (2015). Feature-based Lucas-Kanade and active appearance models. *IEEE Transactions on Image Processing*, 24(9):2617–2632. 72, 83

[Baker and Matthews, 2004] Baker, S. and Matthews, I. (2004). Lucas-Kanade 20 years on: A unifying framework: Part 1. *International Journal of Computer Vision (IJCV)*, 56(3):221–255. 44

[Banino et al., 2018] Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M., Degris, T., Modayil, J., Wayne, G., Soyer, H., Viola, F., Zhang, B., Goroshin, R., Rabinowitz, N., Pascanu, R., Beattie, C., Petersen, S., and Kumaran, D. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557. 25

[Bay et al., 2008] Bay, H., Ess, A., Tuytelaars, T., and Gool, L. V. (2008). SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding (CVIU)*, 110(3):346–359. 17

[Bloesch et al., 2018] Bloesch, M., Czarnowski, J., Clark, R., Leutenegger, S., and Davison, A. J. (2018). CodeSLAM – Learning a Compact, Optimisable Representation for Dense Visual SLAM. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 29, 96

[Bloesch et al., 2016] Bloesch, M., Sommer, H., Laidlow, T., Burri, M., Nützi, G., Fankhauser, P., Bellicoso, D., Gehring, C., Leutenegger, S., Hutter, M., and Siegwart, R. (2016). A Primer on the Differential Calculus of 3D Orientations. *CoRR*, abs/1606.0. 103

[Brown, 1958] Brown, D. C. (1958). 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. 18

[Burri et al., 2016] Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M. W., and Siegwart, R. (2016). The EuRoC Micro Aerial Vehicle Datasets. *International Journal of Robotics Research (IJRR)*, 35(10):1157–1163. 112

[Calonder et al., 2010] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). BRIEF: Binary Robust Independent Elementary Features. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 17

[Chaplot et al., 2020] Chaplot, D., Gandhi, D., Gupta, S., Mulam, H., and Salakhutdinov, R. (2020). Learning to explore using active neural slam. In *ICLR2020*. 25

[Clark et al., 2018] Clark, R., Bloesch, M., Czarnowski, J., Leutenegger, S., and Davison, A. J. (2018). LS-Net: Learning to Solve Nonlinear Least Squares for Monocular Stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 25, 30, 119

[Clark et al., 2017a] Clark, R., Wang, S., Wen, H., Markham, A., and Trigoni, N. (2017a). VidLoc: A deep spatio-temporal model for 6-dof video-clip relocalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 100

[Clark et al., 2017b] Clark, R., Wang, S., Wen, H., Markham, A., and Trigoni, N. (2017b). VINet: Visual-inertial odometry as a sequence-to-sequence learning problem. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 25, 96

[Czarnowski et al., 2020] Czarnowski, J., Laidlow, T., Clark, R., and Davison, A. J. (2020). DeepFactors: Real-Time Probabilistic Dense Monocular SLAM. In *IEEE Robotics and Automation Letters*. (submitted). 29

[Czarnowski et al., 2017] Czarnowski, J., Leutenegger, S., and Davison, A. J. (2017). Semantic texture for robust dense tracking. In *Proceedings of the International Conference on Computer Vision Workshops (ICCVW)*. 29

[Dai et al., 2017] Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017). ScanNet: Richly-annotated 3d reconstructions of indoor scene. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 131

[Davison, 2003] Davison, A. J. (2003). Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 94

[Davison, 2018] Davison, A. J. (2018). Futuremapping: The computational structure of spatial ai systems. In *arXiv preprint*. 15, 145

[Davison et al., 2003] Davison, A. J., Mayol, W. W., and Murray, D. W. (2003). Real-Time Localisation and Mapping with Wearable Active Vision. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 17

[Dellaert, 2012] Dellaert, F. (2012). Factor graphs and GTSAM. Technical Report GT-RIM-CP&R-2012-002, Georgia Institute of Technology. 119, 129

[Dellaert and Kaess, 2006] Dellaert, F. and Kaess, M. (2006). Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing. *International Journal of Robotics Research (IJRR)*, 25:1181–1203. 18

[Dellaert and Kaess, 2017] Dellaert, F. and Kaess, M. (2017). Factor graphs for robot perception. In *Foundations and Trends in Robotics*, volume 6, pages 1–139. 49, 50, 51, 52

[Durrant-Whyte and Bailey, 2006] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. *IEEE Robotics and Automation Magazine*, 13(2):99–110. 94

[Eigen et al., 2014] Eigen, D., Puhrsch, C., and Fergus, R. (2014). Depth Map Prediction from a Single Image using a Multi-Scale Deep Network. In *Neural Information Processing Systems (NIPS)*. 25, 98, 132

[Engel et al., 2017] Engel, J., Koltun, V., and Cremers, D. (2017). Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*. 94

[Engel et al., 2014] Engel, J., Schoeps, T., and Cremers, D. (2014). LSD-SLAM: Large-scale direct monocular SLAM. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 21, 94, 95, 119, 135

[Fischer et al., 2015] Fischer, P., Dosovitskiy, A., Ilg, E., Häusser, P., Hazırbaş, C., Golkov, V., van der Smagt, P., Cremers, D., and Brox, T. (2015). FlowNet: Learning Optical Flow with Convolutional Networks. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 73

[Fletcher, 1971] Fletcher, R. (1971). Modified marquardt subroutine for non-linear least squares. *Journal of Mathematical Sciences*. 54

[Fu et al., 2018] Fu, H., Gong, M., Wang, C., Batmanghelich, K., and Tao, D. (2018). Deep Ordinal Regression Network for Monocular Depth Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 142

[Fukushima, 1980] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, page 193–202. 55

[Fukushima and Miyake, 1982] Fukushima, K. and Miyake, S. (1982). Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469. 23

[Gal and Ghahramani, 2016] Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*. 142

[Gálvez-López and Tardós, 2012] Gálvez-López, D. and Tardós, J. D. (2012). Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28:1188–1197. 131

[Garg et al., 2016] Garg, R., G, V. K. B., Carneiro, G., and Reid, I. (2016). Unsupervised CNN for single view depth estimation: Geometry to the rescue. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 25, 96, 98

[Geiger et al., 2012] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 144

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org. 55

[Gordon et al., 2019] Gordon, A., Li, H., Jonschkowski, R., and Angelova, A. (2019). Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras. In *ICCV*. 144

[Gupta et al., 2017] Gupta, S., Davidson, J., Levine, S., Sukthankar, R., and Malik, J. (2017). Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv:1702.03920*. 120

[Handa et al., 2014] Handa, A., Whelan, T., McDonald, J. B., and Davison, A. J. (2014). A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 132

[Harris and Pike, 1988] Harris, C. and Pike, J. (1988). 3d positional integration from image sequences. In *Image and Vision Computing*, volume 6, pages 87–90. 17

[Harris and Stephens, 1988] Harris, C. G. and Stephens, M. (1988). A Combined Corner and Edge Detector. In *Proceedings of the Alvey Vision Conference*, pages 147–151. 17

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 24

[Henriques and Vedaldi, 2018] Henriques, J. and Vedaldi, A. (2018). Mapnet: An allocentric spatial memory for mapping environments. In *CVPR*, pages 8476–8484. 120, 141

[Hermans et al., 2014] Hermans, A., Floros, G., and Leibe, B. (2014). Dense 3d semantic mapping of indoor scenes from rgb-d images. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 68

[Hirschmüller, 2007] Hirschmüller, H. (2007). Evaluation of Cost Functions for Stereo Matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 72

[Hubel and Wiesel, 1968] Hubel, D. H. and Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, 195:215–243. 55

[Huber, 1964] Huber, P. J. (1964). Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):pp. 73–101. 125

[Izadi et al., 2011] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R. A., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A. J., and Fitzgibbon, A. (2011). KinectFusion: Real-Time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proceedings of ACM Symposium on User Interface Software and Technolog (UIST)*. 20

[Jachnik et al., 2012] Jachnik, J., Newcombe, R. A., and Davison, A. J. (2012). Real-Time Surface Light-field Capture for Augmentation of Planar Specular Surfaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 68

[Kaess, 2015] Kaess, M. (2015). Simultaneous localization and mapping with infinite planes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 95

[Kaess et al., 2012] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F. (2012). iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree. *International Journal of Robotics Research (IJRR)*. To appear. 129

[Kendall and Gal, 2017] Kendall, A. and Gal, Y. (2017). What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In *Neural Information Processing Systems (NIPS)*. 100, 142

[Kendall et al., 2015] Kendall, A., Grimes, M., and Cipolla, R. (2015). Convolutional networks for real-time 6-dof camera relocalization. *CoRR*. 73

[Kerl et al., 2013] Kerl, C., Sturm, J., and Cremers, D. (2013). Robust odometry estimation for RGB-D cameras. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 103

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 102

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 58, 98, 100, 127

[Klein and Murray, 2007] Klein, G. and Murray, D. W. (2007). Parallel Tracking and Mapping for Small AR Workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 18, 94, 95, 106

[Koller and Friedman, 2009] Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press. 49

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NIPS)*. 24

[Laidlow et al., 2019] Laidlow, T., Czarnowski, J., and Leutenegger, S. (2019). DeepFusion: Real-Time Dense 3D Reconstruction for Monocular SLAM using Single-View Depth and Gradient Predictions. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 25, 30, 119

[Laidlow et al., 2020] Laidlow, T., Czarnowski, J., Nicastro, A., Clark, R., and Leutenegger, S. (2020). Towards the Probabilistic Fusion of Learned Priors into Standard Pipelines for 3D Reconstruction. In *IEEE Robotics and Automation Letters*. (submitted). 25, 30

[Laina et al., 2016] Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., and Navab, N. (2016). Deeper Depth Prediction with Fully Convolutional Residual Networks. In *Proceedings of the International Conference on 3D Vision (3DV)*. 135

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551. 23

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 23

[Leonard and Whyte, 1991] Leonard, J. J. and Whyte, D. H. (1991). Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 22

[Leutenegger et al., 2011] Leutenegger, S., Chli, M., and Siegwart, R. (2011). BRISK: Binary robust invariance scalable keypoints. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 17, 125

[Levenberg, 1944] Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathmatics*, II(2):164–168. 54

[Liu et al., 2019] Liu, C., Gu, J., Kim, K., Narasimhan, S. G., and Kautz, J. (2019). Neural RGB-D Sensing: Depth and Uncertainty From a Video Camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 119, 142

[Liu et al., 2015] Liu, F., Shen, C., and Lin, G. (2015). Deep Convolutional Neural Fields for Depth Estimation from a Single Image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 25, 98

[Lovegrove, 2011] Lovegrove, S. J. (2011). *Parametric Dense Visual SLAM*. PhD thesis, Imperial College London. 129

[Lovegrove and Davison, 2010] Lovegrove, S. J. and Davison, A. J. (2010). Real-Time Spherical Mosaicing using Whole Image Alignment. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 74

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110. 17

[Lucas and Kanade, 1981] Lucas, B. D. and Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 20, 42, 47, 129

[Luo et al., 2016] Luo, W., Schwing, A. G., and Urtasun, R. (2016). Efficient deep learning for stereo matching. In *CVPR2016*. 73

[Lv et al., 2019] Lv, Z., Dellaert, F., Rehg, J., and Geiger, A. (2019). Taking a deeper look at the inverse compositional algorithm. In *CVPR*. 73, 91, 140

[Matsugu et al., 2003] Matsugu, M., Mori, K., Mitari, Y., and Kaneda, Y. (2003). Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5):555 – 559. Advances in Neural Networks Research: IJCNN '03. 55

[McCormac et al., 2017a] McCormac, J., Handa, A., Davison, A. J., and Leutenegger, S. (2017a). SemanticFusion: Dense 3D semantic mapping with convolutional neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 68

[McCormac et al., 2017b] McCormac, J., Handa, A., Leutenegger, S., and Davison, A. J. (2017b). SceneNet RGB-D: Can 5M synthetic images beat generic ImageNet pre-training on indoor segmentation? In *Proceedings of the International Conference on Computer Vision (ICCV)*. 102

[Microsoft Corp, 2010] Microsoft Corp (2010). Microsoft Kinect. https://www.xbox.com/en-US/xbox-one/accessories/kinect. 20

[Montiel et al., 2006] Montiel, J. M. M., Civera, J., and Davison, A. J. (2006). Unified Inverse Depth Parametrization for Monocular SLAM. In *Proceedings of Robotics: Science and Systems (RSS)*. 102

[Moutarlier and Chatila, 1989] Moutarlier, P. and Chatila, R. (1989). Stochastic multisensory data fusion for mobile robot location and environement modelling. In *Proceedings of the International Symposium on Robotics Research (ISRR)*. 17

[Mur-Artal et al., 2015] Mur-Artal, R., Montiel, J. M. M., and Tardós, J. D. (2015). ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics (T-RO)*, 31(5):1147–1163. 18, 94

[Newcombe, 2012] Newcombe, R. A. (2012). *Dense Visual SLAM*. PhD thesis, Imperial College London. 19, 21, 129

[Newcombe et al., 2011] Newcombe, R. A., Lovegrove, S., and Davison, A. J. (2011). DTAM: Dense Tracking and Mapping in Real-Time. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 20, 92, 94, 121

[Ortiz et al., 2020] Ortiz, J., Pupilli, M., Leutenegger, S., and Davison, A. J. (2020). Bundle adjustment on a graph processor. In *CVPR*. 145

[Park et al., 2019] Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019). Deepsdf: Learning continuous signed distance functions for shape representation. In *ARXIV*. 141

[Peretroukhin et al., 2019] Peretroukhin, V., Wagstaff, B., and Kelly, J. (2019). Deep Probabilistic Regression of Elements of SO(3) using Quaternion Averaging and Uncertainty Injection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 142

[Perona and Malik, 1990] Perona, P. and Malik, J. (1990). Scale space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 12(7):629–639. 72

[Pizzoli et al., 2014] Pizzoli, M., Forster, C., and Scaramuzza, D. (2014). REMODE: Probabilistic, monocular dense reconstruction in real time. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 20

[Platinsky et al., 2017] Platinsky, L., Davison, A. J., and Leutenegger, S. (2017). Monocular visual odometry: Sparse joint optimisation or dense alternation? In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 94

[Pradeep et al., 2013] Pradeep, V., Rhemann, C., Izadi, S., Zach, C., Bleyer, M., and Bathiche, S. (2013). MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 83–88. 20

[Qin et al., 2018] Qin, Z., Yu, F., Liu, C., and Chen, X. (2018). How convolutional neural networks see the world — a survey of convolutional neural network visualization methods. *Mathematical Foundations of Computing*, 1:149. 56

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 56, 99, 125

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386. 23

[Rosinol et al., 2020] Rosinol, A., Gupta, A., Abate, M., Shi, J., and Carlone, L. (2020). 3d dynamic scene graphs: Actionable spatial perception with places, objects, and humans. In *arXiv Preprint*. 141

[Rosten and Drummond, 2006] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 17

[Rublee et al., 2011] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: an efficient alternative to SIFT or SURF. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2564–2571. IEEE. 17, 18

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., McClelland, J. L., and PDP Research Group, C., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, Cambridge, MA, USA. 97

[Salas-Moreno et al., 2014] Salas-Moreno, R. F., Glocker, B., Kelly, P. H. J., and Davison, A. J. (2014). Dense planar SLAM. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*. 95

[Salas-Moreno et al., 2013] Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H. J., and Davison, A. J. (2013). SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 68, 94

[Savinov et al., 2018] Savinov, N., Dosovitskiy, A., and Koltun, V. (2018). Semi-parametric topological memory for navigation. In *ICLR*. 120

[Shi and Tomasi, 1994] Shi, J. and Tomasi, C. (1994). Good Features to Track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 16, 18, 88

[Silberman et al., 2012] Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor segmentation and support inference from RGBD images. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 113

[Simonyan et al., 2013] Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*. 68

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 24, 71

[Smith et al., 1987] Smith, R., Self, M., and Cheeseman, P. (1987). A stochastic map for uncertain spatial relationships. In *Workshop on Spatial Reasoning and Multisensor Fusion*. 17

[Stumberg et al., 2019] Stumberg, L., Wenzel, P., Khan, Q., and Cremers, D. (2019). Gn-net: The gauss-newton loss for multi-weather relocalization. In *arXiv Preprint*. 73, 91, 140

[Sturm et al., 2012] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 132, 136

[Sucar et al., 2020] Sucar, E., Wada, K., and Davison, A. (2020). Neural object descriptors for multi-view shape reconstruction. In *CVPR*. 141

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 24

[Tang and Tan, 2019] Tang, C. and Tan, P. (2019). BA-Net: Dense Bundle Adjustment Network. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 73, 91, 120, 140

[Tateno et al., 2017] Tateno, K., Tombari, F., Laina, I., and Navab, N. (2017). CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 96, 119, 132, 135

[Triggs et al., 1999] Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (1999). Bundle Adjustment — A Modern Synthesis. In *Proceedings of the International Workshop on Vision Algorithms, in association with ICCV*. 18

[Tulsiani et al., 2017] Tulsiani, S., Zhou, T., Efros, A., and Malik, J. (2017). Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *CVPR*, pages 209–217. 120

[Ummenhofer et al., 2017] Ummenhofer, B., Zhou, H., Uhrig, J., Mayer, N., Ilg, E., Dosovitskiy, A., and Brox, T. (2017). DeMoN: Depth and Motion Network for Learning Monocular Stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 25, 96

[Valentin et al., 2013] Valentin, J., Sengupta, S., Warrell, J., Shahrokni, A., and Torr, P. (2013). Mesh Based Semantic Modelling for Indoor and Outdoor Scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 68

[Wang et al., 2017] Wang, S., Clark, R., Wen, H., and Trigoni, N. (2017). DeepVO: Towards end to end visual odometry with deep recurrent convolutional neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 96

[Weerasekera et al., 2017] Weerasekera, C. S., Latif, Y., Garg, R., and Reid, I. (2017). Dense Monocular Reconstruction using Surface Normals. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 25, 119

[Whelan et al., 2015] Whelan, T., Leutenegger, S., Salas-Moreno, R. F., Glocker, B., and Davison, A. J. (2015). ElasticFusion: Dense SLAM without a pose graph. In *Proceedings of Robotics: Science and Systems (RSS)*. 20

[Whelan et al., 2016] Whelan, T., Salas-Moreno, R. F., Glocker, B., Davison, A. J., and Leutenegger, S. (2016). ElasticFusion: Real-time dense SLAM and light source estimation. *International Journal of Robotics Research (IJRR)*, 35(14):1697–1716. 68

[Yang et al., 2020] Yang, N., Stumberg, L., Wang, R., and Cremers, D. (2020). D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry. In *CVPR*. 144

[Yi et al., 2016] Yi, K., Trulls, E., Lepetit, V., and Fua, P. (2016). LIFT: Learned invariant feature transform. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 25

[Yin and Shi, 2018] Yin, Z. and Shi, J. (2018). GeoNet: Unsupervised Learning of Dense Depth, Optical Flow and Camera Pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 25, 96

[Zagoruyko, S., and Komodakis, N., 2015] Zagoruyko, S., and Komodakis, N. (2015). Learning to compare image patches via convolutional neural networks. *CoRR*. 73, 90

[Zhou et al., 2018] Zhou, H., Ummenhofer, B., and Brox, T. (2018). DeepTAM: Deep Tracking and Mapping. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 25, 121, 136, 142, 143

[Zhou et al., 2017] Zhou, T., Brown, M., Snavely, N., and Lowe, D. G. (2017). Unsupervised Learning of Depth and Ego-Motion from Video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 25, 96, 98, 100, 133, 144