MONASH University

Three Dimensional Structure Refinement using Geometry and Machine Learning based Techniques for Robotics

by

Yakupitiyage Don Thanuja Samodhye Dharmasiri

Supervisor: Prof. Tom Drummond

A thesis submitted for the degree of Doctor of Philosophy at
Monash University, September 2018.

Copyright Notice

Abstract

3D reconstruction is one of the important research problems in computer vision and robotics. When combined with camera motion tracking, reconstruction allows a robot to navigate the world and perform meaningful actions. This thesis focuses on improving the quality of reconstruction using techniques from geometry as well as machine learning.

Firstly, a novel object based SLAM system equipped with a recognizer, that can detect duplicate structures in a scene is presented. The detected objects are added as first order entities onto the SLAM map, and are used to simultaneously refine the accuracy of the recognized objects, the remaining landmarks, and the camera pose.

Secondly, depth perception is tackled using a modern approach where a convolutional neural network is used to generate a depth map from a single colour image. The close relationship shared between depths, surface normals and surface curvature is used to jointly improve the accuracy of all three tasks.

Thirdly, a probabilistic depth fusion pipeline is presented. The scale invariant formulation used within the framework allows fusion of depth maps obtained from different modalities in disparate scales.

This thesis also investigates the problem of leveraging machine learn-

ing techniques to compliment the well known optimization strategies used in geometry. Depth, optical flow and the confidence in optical flow generated from a neural network are used in a re-weighted least squares solver in order to compute the camera pose. The complete pipeline is implemented in Tensorflow and is finetuned end-to-end, creating a feedback loop where the pose information has a direct influence on optical flow and depth. The depth and motion predictions generated using this framework achieve state-of-the-art results on indoor and outdoor benchmarks.

Finally, this thesis addresses the important issue of real-time depth prediction on a mobile platform. A novel knowledge transfer strategy is used to compress the state-of-the-depth prediction pipeline into a much smaller network architecture. Although the accuracy reduces to a certain extent , the depth predictions generated from the smaller model are still able to have a noticeable impact on the output of a SLAM system.

Declaration

This thesis contains no material which has been accepted for the award for any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: 

Print Name: Thanuja Dharmasiri

Date: 17/09/2018

Acknowledgements

First and foremost, I thank my supervisor Professor Tom Drummond for his invaluable guidance and supervision. I have been fortunate enough to learn a lot from him over the course of my PhD. I also thank Doctor Vincent Lui for being a great mentor and a friend. I thank Doctor Georg Klein and Associate Professor Javier Civera for taking the time to review this thesis and providing valuable feedback.

I would like to thank my collaborators Vincent, Andrew and Saroj. Andrew in particular for being part of the numerous research problems that we investigated together. Thank you to my lab mates Andrew, Yanming, Ben Harwood, Ben Meyer, Vincent and Yan for their company.

Thank you to the Australian Government Research Training Program (RTP) and the Australian Research Council Centre of Excellence for Robotic Vision for financially supporting this research.

I thank my parents for encouraging me to pursue the path of a PhD in the first place and their continual support. Finally, I thank my partner Sewmee for her love and support, and for being a part of this journey with me.

Contents

CHAPTER 1

Introduction

1.1 3D Reconstruction using RGB images

Since the inception of artificial intelligence, one of the key research areas tackled by the computer vision and the robotics community is the problem of making a robot understand and interact with the world around it. Among our five senses sight, hearing, taste, touch and smell, it can be argued that vision provides us with the most amount of information. During the evolution process the creatures who could distinguish between the presence of light and its absence had a significant advantage and hence a higher rate of survival. This light sensor ameliorated over millions of years to develop into the complex organ which we now call the eyes.

In the case of a robot/computer the functionality of the eye is replicated using a camera. A camera or more specifically an RGB camera provides a 2D-image which encapsulates both the appearance information (colour and lighting) and the geometric information (distance from a point to the camera) about a scene. The process of recreating the 3D structure of the world by using one or more of these 2D RGB images is called 3D Reconstruction.

3D reconstruction is inherently coupled with another problem commonly referred to as localization or camera tracking, the technique of determining the location of the robot within the operating environment that the robot is trying to reconstruct. Solving these two problems jointly is known as Structure from Motion (SfM) and incrementally performing SfM in a real-time fashion is called Simultaneous Localization and Mapping (SLAM). While humans often refer to things in the world as objects, a robot sees it as a point cloud or a depth map.

The reconstruction process is generally performed using geometric techniques where correspondences are established across different viewpoints allowing the robot to perform triangulation. The number/density of the candidates used gives rise to three different variants: sparse ($\approx 1 - 3\%$ of all available points) , dense (uses all available points) and semi-dense (compromise between the previous two variants). Since the resurgence of Artificial Neural Networks (ANNs) and the availability of large datasets researchers have investigated the possibility of predicting structure by employing machine learning. One of the key benefits of the latter approach is the ability to predict the structure of a scene by using a single view.

In this thesis, I investigate how to apply techniques from geometry and machine learning in order to perform 3D reconstruction using a single RGB camera. After obtaining an initial estimate of the structure, the methods that could be applied to refine the three dimensional representation of the world and perform additional tasks such as object discovery and estimating camera motion by leveraging the constructed 3D depth map are also explored.

1.2 Potential Applications

Advancements in robotics have revolutionised the way we interact with the environment. An industry that was launched to replicate the dull, dirty and dangerous

Figure 1.1: Applications of this research are aimed to further enhance the developments in self-driving car technology, space exploration, environmental conservation etc. **Left :** Tesla Model S equipped with a multitude of vision and range sensors supports self-driving functionality. **Center :** NASA Curiosity space rover for unmanned space exploration on Mars. **Right :** COTSbot, A robot capable of automatically detecting Crown-Of-Thorns Starfish which destroy the coral reef and injecting it with a chemical solution.

tasks that humans do has now well and truly surpassed human level performance in an array of challenging problems including object recognition. However, one of the key technologies that enable a robot to perform meaningful actions is the knowledge about the 3D structure of a scene.

3D reconstruction when coupled with camera motion tracking act as a powerful navigation tool. Although driver-less cars come to immediate attention when we think of robotic navigation, the applications extend to search and rescue operations, space exploration, environmental conservation (terminating crown-of-thorns starfish that destroy the great barrier reef) etc. The necessity of maintaining a consistent three dimensional representation of the world (map) in all of the aforementioned scenarios is vital not only to avoid obstacles but also to revisit a previously visited location in the most time efficient manner.

Reconstruction plays a vital role in vision based robotic control or visual servoing. Typically, a robot moves to a target position based on the information it sees to perform a grasping or a manipulation task. Perceiving the distance to the object as well as its physical dimensions becomes crucial in-order to efficiently complete the task. In a similar vein, 3D reconstruction plays a key role in aiding the visually impaired community, where assistive robots help with avoiding obstacles

during navigation.

Applications also extend to augmented and mixed reality, where virtual content is fused with that of the real world. Dedicated hardware such as the Microsoft HoloLens leverage the techniques of tracking and mapping in order to create a convincing illusion that the virtual objects are indeed real.

1.3 The Challenges of Reconstruction

Given a 3D scene, the process of obtaining a 2D image is generally straightforward. However, the inverse problem which revolves around recovering the structure from one or more images is much more challenging. Whether reconstruction is performed using traditional geometric methods or using machine learning, complications arise along the pipeline.

1.3.1 Problems with geometry based approaches

The most crucial step of a reconstruction pipeline is finding correspondences. If sparse features are used to represent landmarks, keypoints are detected and their respective feature descriptors need to be computed which could then be used to find matches. However, sparse keypoint based approaches suffer a great deal from motion blur during rapid camera motion. The features detected and their respective descriptors vary considerably, if the obtained images are blurred, which makes the task of establishing correspondences much difficult. Although the problem could be alleviated to a certain degree using multi-scale image pyramids, the end result is not as accurate as that of a denser counterpart.

Given an initial estimate of the camera pose between two images, it is possible

to directly work on the images and minimize a photometric cost between the images. As more images that share the same viewpoint are taken into account when optimizing the cost function, an accurate representation of the world can be realised. The main limitation of this approach is the amount of time consumed, in order to achieve real-time performances dedicated hardware in the form of a GPU is needed.

It is common practise to assume the brightness of a particular point/object remains constant across multiple views when performing reconstruction. In order for this assumption to be valid, the operating environment needs to be static with no dynamic lighting and the surfaces need to be Lambertion (the apparent brightness remains the same regardless of the observation angle). However, in a dynamic environment the appearance of the world changes (for e.g due to change of day and night), objects that were visible at a certain point in time could become occluded due to dynamic objects and makes the reconstruction problem much more challenging.

1.3.2 Problems with machine learning approaches

Data-driven machine learning approaches aim to learn a non-linear mapping from RGB imagery to a representation of the structure of a scene. Supervised learning techniques rely on datasets of labelled data. A typical training example for depth-map prediction is an RGB image and its corresponding depth image which could be obtained from a sensor such as a Microsoft Kinect or a LIDAR. Although large datasets such as NYUv2 [140], KITTI [63] exist, there is a lack of labelled data for challenging environments such as underwater and underground mines. A direct consequence of having insufficient training data is over-fitting, where the machine learns well to model the training data but does not generalise well to test cases.

Convolutional Neural Network (CNN) models that achieve state-of-the-art performances consist of millions of parameters to tune (for e.g DenseNet [90] contains $\approx$ 25 million parameters). Large models consume a lot of memory as the activations and the gradients required for back-propagation need to be held in memory at a single point in time during training. Training a model is also a time consuming process and could range from a few hours to a few days. Reconstruction process is generally also constrained by run time; however, achieving real-time performance while maintaining a high accuracy is still a challenge.

A large proportion of neural architectures (VGG [174], ResNet [90], DenseNet [90]) were initially built to solve image classification problems. Although these models have been successfully adapted to perform pixel-wise segmentation tasks, the loss of spatial information during the downsampling process (pooling operations) greatly affect the performance of geometry related problems such as depth or surface normal prediction.

Finally, the techniques/intuitions that are proven to work using traditional methods can not also be directly applied to machine learning approaches. For example, when estimating optical flow (the pattern of apparent motion between two images) it is easier to estimate smaller motions using traditional methods. However, this is generally not the case using machine learning approaches as shown in FlowNet[92].

1.4 Contributions

This thesis is largely centred around the reconstruction problem and the methods which could be applied to refine structure of a scene. We begin with a novel geometry based technique by taking a step in the direction of object based SLAM. The main goal was to exploit the duplicate structures that are present in a scene in order to reconstruct an accurate representation of the world while improving the camera pose. The duplicate objects are detected during run-time as opposed to previous work that use a pre-populated object database. Furthermore, we demonstrate that the detected objects which are added as first-order entities in to the map can be leveraged during the map optimization process to refine both the 3D position of the landmarks and the camera poses. Chapter 4 provides a detailed analysis including synthetic and real-world experiments.

In Chapter 5, we look at improving the performance of depth predictions generated from a neural network using the knowledge of related information. More concretely, we demonstrate that by predicting surface normals and surface curvature in tandem with the depths while preserving the model capacity, the accuracy of all three predictions can be improved. This work also features the first CNN to predict surface curvature from a single RGB image.

The 3D structure of a scene is often obtained from one of three different modes: a dedicated sensor (e.g Microsoft Kinect, Velodyne LIDAR), a SLAM system or as a prediction generated from a convolutional neural network. All three modes have their pros and cons, sensors are the most accurate but can contain missing data. In addition, these sensors are generally expensive compared to a monocular camera. Sparse SLAM systems despite being fast generate sparse maps which are in arbitrary scales. Although a neural network produces a dense output, the accuracy of the depth values are lower compared to the previous two modalities. In Chapter 6, we present an approach that can be used to combine these different modes of inputs probabilistically using the confidence estimate of each modality

generated from a neural network. Our framework allows fusion of depth maps in arbitrary scales (for e.g a SLAM depth map with a depth prediction from a CNN) and runs at frame rate on a workstation GPU.

As we will see throughout the course of the thesis, CNNs have been enormously successful at tasks that benefit from feature extraction (classification, semantic segmentation, depth prediction, optical flow prediction etc.). On the other hand, computer vision and robotics researchers have successfully tackled motion estimation using techniques from geometry. In an effort to harness the useful aspects of both machine learning and geometry in Chapter 7, we train a neural network that predicts depth, optical flow and the confidence of optical flow given an image pair and use this information in a weighted least squares (WLS) optimisation pipeline in order to generate the motion parameters governing the camera motion. Since, the entire pipeline is implemented in Tensorflow[1], it allows us to reap the benefits of geometrically meaningful loss signals during training.

It can be argued that the most important aspect of any robotic framework should be the ability to execute the designated task in real-time. This is predominantly due to the fact that the applications (e.g human-robot interaction, navigation) that rely on these frameworks need to react to the changes in the environment in a timely manner. Furthermore, robotics frameworks are deployed in challenging conditions such as mines and underwater where its extremely difficult to set up desktop workstations. Due to these reasons, in Chapter 8 we explore the possibility of performing depth prediction at frame rate on a resource constrained mobile platform (NVIDIA-TX2). Our approach aims to condense the knowledge of a large, slow CNN model capable of making state-of-the-art depth predictions in to a much smaller and faster CNN model while minimizing the loss in accuracy.

In both Chapters 7 and 8 we show that through the use of depth predictions made in metric scale, we can produce more accurate visual odometry with considerably smaller scale drift.

1.4.1 Collaborative Research

In addition to the research conducted by myself, a number of successful collaborative partnerships were formed during the course of the PhD. This facilitated the investigation of a range of research problems within a span of 3 years. Collaborative work was conducted with Andrew Spek (Monash University) and Saroj Weerasekera (University of Adelaide). Both universities are part of the Australian Centre of Excellence for Robotic Vision and work on similar research themes.

Collaboration with Andrew took place within the research group itself due to common interests and related research experiences. As I had previous experience in machine learning, and Andrew had previously worked with low cost depth sensors. Chapters 5, 7 and 8 are drawn from the publications authored with Andrew Spek. Since Andrew and I are equal first authors in each of these publications, the corresponding chapters include a percentage breakdown describing the contributions I have personally made for the production of the particular work, based on mutual agreement.

Collaboration with Saroj eventuated from a conversation between the respective supervisors (Prof. Tom Drummond and Prof. Ian Reid), as both research groups were interested in constructing a dense depth map from a sparse SLAM map. Chapter 6 expands upon the findings of the joint publication [200] of which I was the second author.

1.5 Publications

The main chapters of this thesis are based on the following peer-reviewed or currently under review (as indicated) articles :

- **MO-SLAM: Multi Object SLAM with Run-Time Object Discovery through Duplicates** [38]
  T.Dharmasiri, V.Lui & T.Drummond. In *International Conference on Intelligent Robots and Systems (IROS)* , September 2016

- **Joint Prediction of Depths, Normals and Surface Curvature from RGB Images using CNNs** [39]
  T.Dharmasiri*, A.Spek* & T.Drummond. In *International Conference on Intelligent Robots and Systems (IROS)* , September 2017

- **Just-in-Time Reconstruction: Inpainting Sparse Maps using Single View Depth Predictors as Priors** [200]
  C.Weerasekera, **T.Dharmasiri**, R.Garg, T.Drummond & I. Reid. In *International Conference on Robotics and Automation (ICRA)* , May 2018

- **ENG: End-to-end Neural Geometry for Robust Depth and Pose Estimation using Convolutional Neural Networks** [40]
  T.Dharmasiri* , A.Spek* & T.Drummond. Submitted to *Asian Conference on Computer Vision (ACCV)*

- **CReaM: Condensed Real-time Models for Depth Prediction using CNNs** [177]
  A.Spek*, **T.Dharmasiri*** & T.Drummond. In *International Conference on Intelligent Robots and Systems (IROS), October 2018*

* indicates equal contribution.

1.6 Thesis Layout

A review of existing works in the literature on 3D reconstruction and machine learning for computer vision tasks is given in Chapter 2. The mathematical framework used throughout the thesis as well as an introduction to the fundamentals of convolutional neural networks is provided in Chapter 3.

The primary contributions as outlined in the above introduction will be expanded in chapters 4 to 8. Chapter 4 describes MO-SLAM, a multi-object visual SLAM system. Chapter 5 presents the multi task learning framework, which learns depths, surface normals and surface curvature in parallel. Chapter 6 describes sparse depth map inpainting. A learning framework which predicts depth,optical flow and ultimately the relative camera pose is described in Chapter 7. Chapter 8 presents the CNN that estimates depth in real-time on an NVIDIA-TX2.

Chapter 9 concludes the thesis and provides a summary of the contributions as well as a discussion on future directions.

CHAPTER 2

Background

This chapter describes the previous work undertaken in the areas relevant to this thesis including reconstruction, camera tracking, predicting structural information using a neural network.

2.1 Geometry based 3D Reconstruction and Camera Localization

A three dimensional understanding of the environment is vital for robot navigation. Creating a model of the world synthetically using computer aided design software is time consuming and involves a lot of human intervention. Furthermore, a synthetic model can fail to represent the intricate details of the real world objects. Due to these reasons it is generally desirable to acquire data through a sensor and then process this data as it arrives in order to recreate a model of the world. This section aims to summarise the 3D reconstruction and camera tracking techniques in the literature that use geometry as the driving force.

Reconstruction approaches can be divided into two main categories; active and passive methods. Active techniques use sensors that can capture range information (for e.g structured light cameras, laser scanners). In contrast, passive approaches do not directly observe 3D information, instead use images created by light falling on a light sensitive surface to perform reconstruction. These approaches can also be subdivided based on the density of the reconstructed point cloud as mentioned in the introduction. Reconstruction and localization is often tackled as a joint problem, where the robot builds a map of the world while simultaneously deducing its own location within that map. The main motivation behind solving these two tasks in tandem is due to the dependency of one aspect on the other. More concretely, in order to perform reliable mapping the robot needs to accurately localise itself, however, accurate localisation requires a good map.

2.1.1 Active Sensors

Active sensors, in addition to observing light (visible or other forms) also emit light. The commonly used active sensors are structured light cameras, time-of-flight cameras and scanning LIDAR(Light Detection And Ranging or Laser Imaging, Detection And Ranging) systems.

Structured Light Cameras such as the Microsoft Kinect (v1) and the Asus Xtion have been popular among the robotic vision researchers as they are available at a relatively low price compared to a LIDAR, and are capable of providing range information at 30 frames per second. The cameras are equipped with an infrared light projector and a sensor. The IR emitter projects a known infrared light pattern sequentially which can get deformed by the structures and objects present in the scene. The IR sensor which is horizontally displaced from the location of the projector observes the light pattern and uses this information to produce a depth map of the scene. Additionally, these sensor systems typically have a colour camera, which opened up the way for RGB-D reconstruction approaches.

Henry *et al.* constructed a dense tracking and mapping system which incorporated RGB features for the map initialisation process [81]. However, the main limitation of this approach was the time taken to generate coloured surface patches (surfels). The surfel generation process took approximately 6 seconds per frame, hence the framework was incapable of targeting real-time applications. KinectFusion [143] by Newcombe *et al.* was the pioneering work to integrate multiple frames of depth data captured from a Microsoft Kinect into a global surface model. They exploited the highly parallelisable nature of the problem to create an efficient GPGPU (General Purpose Graphics Processing Units) implementation which performs real-time tracking and mapping. An incremental 3D map of the scene was constructed using the volumetric truncated singed distance function (tsdf) [85] while performing iterative closest point (ICP) [10] based camera pose estimation. Whelan *et al.* later extended this approach to work in arbitrarily large environments [201]. KinectFusion and its extensions such as [22, 145, 14] were constructed by assuming the scene was static (i.e did not contain any moving objects). This assumption was relaxed in DynamicFusion [142], a framework capable of reconstructing non-rigidly deforming scenes in real-time. However, it remains as an open problem to extend this to large scale environments [15]. In a robotics context, Huang *et al.* performed visual odometry by installing an RGB-D camera on a micro air vehicle [89].

A time-of-flight (ToF) camera is another form of an active sensor which measures the distance to landmarks in the world by measuring the flight time of a light ray from the emitter/projector to the target surface and back to the sensor. The operating environment is typically illuminated with an intensity modulated, periodic infrared light. The distances to the landmarks in the environment from the sensor causes a time-shift in the optical signal which is equivalent to a phase shift in the periodic signal that is detected. The time shift (time to travel twice the distance) can then be used to compute the distance to the landmarks [170].

Similar to structured light cameras, researchers have explored the possibility of using time-of-flight sensors for odometry and reconstruction. Despite working

poorly in direct sun light conditions a Microsoft Kinect v2 (ToF) was successfully employed in [13] to perform mapping in coastal areas and to monitor macro algae. May *et al.* presented a 3D mapping framework that used a single ToF camera [132]. Their approach consisted of a novel calibration method to reduce measurement errors, followed by a filtering approach to remove outliers. The accuracy of the map was further refined by detecting surface normals using principal component analysis and subsequently shifting related pixels towards the detected normals. A feature based 3D SLAM system was proposed in [83] where SURF features [8] were extracted from the range image and were used as visual landmarks. Although the goal of their approach was to use a single sensor in the form of ToF camera, a more robust approach would have been to extract features from a colour image and combine that with depth estimates from the ToF sensor. Castaeda *et al.* performed this exact process in [19] and obtained improved performance over using high resolution time-of-flight images alone.

A ToF camera counterpart of KinectFusion was proposed by Wasenmller *et al.* to address the problems that arise when performing dense RGB-D SLAM using ToF cameras [198]. The same authors also presented an excellent comparison between Kinect v1 and Kinect v2 in [199]. The most notable findings are stated here for the benefit of the reader. Unlike the Kinect v1, they observed a strong correlation between temperature and depth accuracy for Kinect v2 (only producing reliable estimates after $\approx$ 25 mins of usage). Furthermore, they observed a higher precision for the depths produced by a Kinect v1. However, the accuracy of Kinect v1 degraded with distance while that of v2 was constant (-18 mm). Kinect v1 also also exhibited a striped pattern which is harder to compensate for. All things considered, they recommended the more accurate v2 sensor for SLAM applications while highlighting the necessity for pre-processing to overcome the lower precision.

LIDAR Scanners also work on the time-of-flight principle. However, a LIDAR uses a fine laser beam and needs to scan (for e.g by using a rotating mirror) in order to sense multiple 3D points of a scene. In contrast, the emitter of a ToF

camera illuminates the entire scene by diverging light, allowing the camera to capture the scene in one attempt.

Combining multiple range images using the standard ICP algorithm can often be challenging due to a few reasons. Since every individual point on a single scan is not measured at the same point in time, the output could potentially be distorted when the scanner moves during scanning. Furthermore, the spacing between the scan lines tends to vary rapidly as the distance increases rendering it difficult to establish correspondences. To address these issues, a novel oct-tree based matching heuristic was proposed by Nüchter *et al.* in [149] to improve the accuracy of ICP. In order to compensate for the motion of the range scanner, Hong *et al.* proposed to estimate the velocity of the scanner over the ICP iterations [84]. Gressin *et al.* also suggested improvements to ICP through the use of geometrical low-level features which describe the local shape around a 3D LIDAR point [69].

From a reconstruction point of view, LIDAR scans were used in [11, 151, 192, 46] for mapping buildings. Grant *et al.* demonstrated real-time LIDAR point cloud registration by employing a novel plane finding algorithm in a plane based frame to frame registration regime. Visual and LIDAR odometry were combined in the work of Zhang *et al.* [207]. Incorporating the two approaches allowed them to address the limitations of both aspects as the former handles rapid motions while the latter prevents drift. A LIDAR SLAM system was used in [153] to perform autonomous navigation in a range of different environments with varying characteristics.

2.1.2 Passive Sensors

Image based methods do not directly capture range data, however the geometry is encapsulated in the pixel intensities. The two commonly used image capturing sensors are **stereo camera rigs** and **monocular cameras**. Since the research

that use each of these sensors have a lot of underlying image processing techniques in common, the related work will be discussed jointly in this section. The literature presented will be organised based on the three different variants (sparse, semi-dense and dense) corresponding to the density of the reconstructed map.

Sparse tracking and mapping frameworks use approximately 1-3% of all available data allowing these systems to perform reliable localization and reconstruction at frame rate even on resource scarce platforms such as the onboard computer of a drone. One of the important steps of a typical sparse visual SLAM or SfM pipeline is to establish correspondences between two or more views which can then be used to solve for the relative camera pose.

First, distinct image features (generally corners) are extracted using a feature detector. Harris and Stephens proposed a corner detector [76] based on the eigenvalues of the structure tensor (derived from the image derivatives for each pixel) where two large eigenvalues indicate a corner, single large eigenvalue indicates an edge and two small eigenvalues denote a flat region. The authors proposed to use the difference of the determinant of the structure tensor and the square of the trace (multiplied by a constant) as the corner detection criteria. A better detection criteria in the form of examining the minimum eigenvalue was proposed by Shi and Thomasi in [172]. The Smallest Univalue Segment Assimilating Nucleus (SUSAN) feature detector by Smith and Brady [176] proposed to compare all pixels inside a circle centred around a nucleus (centre pixel) and when the number of pixels with the intensity similar to that of the nucleus was minimum it was determined to be a corner. An efficient variant of this, called the Features from Accelerated Segment Test (FAST) was proposed by Rosten and Drummond in [164] where the centre pixel was compared against pixels on a ring of radius 3. The process was further sped up by employing a machine learning technique to determine the optimal order of pixels on the ring to be considered.

Once the features are detected, the next step of the pipeline focuses on establishing correspondences by finding matching pairs of keypoints across views. Due to

this reason, a method to distinguish a keypoint from one another was required and a class of algorithms called feature descriptors were invented. A feature descriptor encodes the appearance information of the image patch surrounding each keypoint either as a vector of binary or floating point values. Scale Invariant Feature Transform (SIFT) by Lowe [126] proposed to employ a 128 value descriptor corresponding to a histogram of gradients in the 8 compass directions for a 4×4 block of 4×4 cells each. While SIFT features led to superior matching performance, the limitation was the computational cost. To address this, Bay *et al.* proposed Speeded up Robust Features (SURF) which followed a similar descriptor generation regime to SIFT [8], but the descriptor was based on the sum of Haar-wavelet [73] responses. The authors claimed to match the performance of SIFT while being more computationally efficient as Haar-wavelets can be computed using integral images.

Researchers started moving towards binary feature descriptors to further reduce the computational cost associated with computing the descriptors. Furthermore, matching binary descriptors using Hamming distance is a relatively inexpensive operation compared to matching floating point descriptors. Calonder *et al.* introduced Binary Robust Independent Elementary Features (BRIEF) [16], where the appearance information was encoded as a 256 bit string of intensity comparisons. The matching performance using BRIEF descriptors was superior to that of SURF on most datasets while giving an approximately 30 fold increase in feature computation. However, BRIEF descriptors were not rotational invariant and required the two images to be oriented in the same direction. Rublee *et al.* relaxed this assumption in Oriented fast and Rotated Brief (ORB) descriptors [165] where FAST keypoints were detected followed by computing a dominant orientation for each keypoint. The ORB descriptors were then aligned with the computed orientation.

Once the correspondences are established the initial relative pose between the two frames is commonly computed using the eight point algorithm [77], or a variant of the five point algorithm [147, 180, 128]. The inlier matches between the two

frames can then be used to perform triangulation, resulting in a 3D reconstruction of the world. The aforementioned steps lay a foundation for simultaneously estimating the camera motion while mapping the world as new images of the scene become available. Furthermore, the constraints provided by the novel views are then used to refine the reconstructed map and the camera pose in a process known as bundle adjustment [189].

Davison proposed the first real-time monocular camera driven SLAM system [32] by employing a Kalman filter. This work was later extended by Davison *et al.* in MonoSLAM [33]. Concurrently, a particle filter based alternative was proposed by Eade and Drummond [43]. All of the above works advocated for an active search approach to look for new landmarks. Klein and Murray [105] tackled the SLAM problem by using two separate threads running in Parallel for Tracking and Mapping (PTAM). This allowed their framework to track the camera in real-time, while performing expensive tasks such as global bundle adjustment in the background. Their system was very robust and was catered to indoor workspace environments while targeting the application domain of augmented reality. Improvements for SLAM systems were suggested in the form of a novel optimisation framework by Strasdat *et al.* [181] in Double Window Optimisation (DWO). Forster *et al.* proposed a pixel intensity based VO pipeline dubbed SVO which targeted micro aerial vehicles [56] where feature points were used during tracking, mapping on the other hand was done by directly working on the image intensities. Direct Sparse Odometry (DSO) by Engel *et al.* also used image intensities in a sparse framework [47]. However, DSO was fully direct (both tracking and mapping) and incorporated geometric and photometric calibration to enhance the accuracy of the pose estimate. ORB-SLAM by Mur-Artal *et al.* proposed to use a consistent representation of ORB features throughout multiple threads (Tracking, Mapping and Loop Closing) [138]. They later extended the framework to be compatible with stereo cameras and active sensors in [139], achieving state-of-the-art performance on indoor [182] and outdoor [63] odometry datasets.

Semi-dense and Dense Approaches

Although the sparse map generated from the previous approaches is sufficient to perform reliable tracking and mapping in most scenarios, having a denser representation allows the robot to be resilient to fast motions and perform meaningful tasks such as manipulation. Denser approaches have gained popularity due to these reasons and increase in computational power (Moore's Law) has enabled researchers to increase the density of the reconstructed map. The denser approaches work directly on pixel intensities as opposed to extracting features.

Large-Scale Direct Monocular SLAM (LSD-SLAM) by Engel *et al.* proposed a semi-dense SLAM system where the camera pose was estimated by performing image alignment in a coarse to fine manner [48]. From a mapping point of view, all points in a region with a significant intensity gradient were incorporated into the depth map. Every incoming frame was either chosen as a key-frame or was used to update the depth map of the current key-frame. The map was optimised using the optimisation package g2o [110]. The authors later proposed a version of LSD-SLAM catered towards stereo cameras [49].

Fully dense approaches use every pixel available in the image for both tracking and reconstruction. While this process is computationally expensive Newcombe *et al.* in Dense Tracking and Mapping (DTAM) [144] used a GPU to perform the two tasks simultaneously in real-time. Inverse depth maps [26] from a multitude of images were combined into a cost volume. Individual depths of a particular key-frame were then computed by finding the depth that resulted in the smallest photometric error when that point was projected on to the neighbouring frames. Although it is challenging to perform dense tracking and mapping on a CPU, DPPTAM by Concha and Civera [29] achieved this through the use of a novel 3D superpixel estimation approach.

2.2 Computer Vision and Machine Learning

Availability of large datasets [167, 63, 140] and relatively cheap high performance computational resources in the form of General Purpose GPUs have allowed researchers to solve computer vision problems using machine learning. Two of the most commonly tackled problems are classification and segmentation. While the former revolves around predicting a single meaningful label given an RGB image, the latter extends this task to predicting a label at every pixel. Although the field of computer vision using machine learning was dominated by these two problems initially, within a span of few years, the techniques have been adapted to solve a range of tasks.

It is worth mentioning that even though the terms machine learning and deep learning are used interchangeably in the literature, truly they have two distinct definitions. Machine learning is the process of enabling a robot/computer to self-learn a task without being explicitly programmed to follow a set of instructions. Deep Learning on the other hand is performing machine learning using deep neural network architectures. Deep in this case refers to the number of layers in the network. It has been shown that networks with more number of layers (deep) generally outperform networks with fewer layers (shallow) hence shallow learning is a rarely used term.

Convolutional Neural Networks (CNNs/ConvNets) have been very effectively applied to a range of robotic vision tasks including grasp pose detection [159, 71], image classification [107, 79], semantic segmentation [125], depth estimation [44, 122, 171, 114], surface normal estimation [194, 5, 44]. While the main emphasis of this thesis is on predicting structural information and refining these estimates, for completion, a short account on approaches that leverage machine learning techniques to solve other vision problems is provided next.

2.2.1 Classification and Segmentation

Recognising objects of importance purely using visual data is one of the greatest assets of any living creature with a visual system. Although humans use vision for a range of intelligent tasks such as reading, driving and medical analysis, even in the case of animals object recognition can become the difference between hunting and being hunted. While using vision for survival is not a concern for machines, replicating the useful aspects of human vision using robotic vision can help humanity in a multitude of ways. Vision based recognition in particular has applications in navigation (recognising pedestrians, other vehicles), biomedical imaging (anomaly detection) and surveillance (face detection).

Classification is the problem of assigning a semantically meaningful label to an image. Segmentation is an extension of this where a label is assigned for every pixel of the input. Data-driven approaches have been far more superior at this task compared to heuristic based approaches. Since real-life data that belong to a particular class is highly diverse, constructing a model that encapsulates all the properties using traditional approaches can be extremely difficult. For example, the examples of the class 'car' can vary vastly based on the colour, the make and model etc. Machine learning approaches aim to learn a generalised representation based on the training examples provided.

The learning approaches can be split into three main subtypes based on the use of training data and the amount of supervision provided during training. Data-driven supervised training approaches [174, 79, 90] have access to labelled ground truth data enabling the system to compute an error between the prediction and the label. Unsupervised training approaches [115, 211, 117] on the other hand, use unlabelled data during training and focus on learning a set of attributes that distinctly differentiates one set of examples from another. The third technique is semi-supervised training [72, 146, 213] which aims to combine the benefits of both supervised and unsupervised training regimes.

Prior to the resurgence of the convolutional neural networks, object recognition was tackled using a range of techniques including SVMs [41, 17], decision trees [64, 173], conditional random fields [156, 109] and boosting [193]. Since the machine learnt models presented in this thesis have been implemented using convolutional neural networks, the remainder of this subsection focuses on the CNN based classification and segmentation methods.

Artificial neural network research can be dated back to the early 1940s [134]. From a computer vision point of view, a majority of the earlier works focused on hand written character recognition [34, 116]. In the latter of these works Le Cun *et al.* demonstrated the capability of the back propagation algorithm to train neural networks with real images. Despite not being a convolutional neural network, the trained model demonstrated impressive results on unseen data.

Until recently, the main limitation of the CNNs had been the large computational cost associated with processing high resolution images. Krizhevsky *et al.* [107] proposed to address this issue by using an efficient implementation of 2D convolution on a GPU. Not only, it scaled down the training time from weeks to a few days, but their framework also managed to outperform previous approaches on the Imagenet Large Scale Visual Recognition Competition (ILSVRC) [167] by approximately 10 per cent. Concurrently, Szegedy *et al.* [186] and Simonyan *et al.* [174] made similar architectural improvements by constructing deeper neural networks with smaller convolutional filters. The former of these two works, GoogLeNet [186] leveraged inception modules. Each inception module consisted of filters with different receptive field sizes (typically 1×1, 3×3, 5×5) that were applied on the same input layer. The activations produced by each filter were then combined and served as the input to the subsequent layer. VGGNet [174] on the other hand constitutes only of 3×3 filters and fully-connected layers. Their network contained more than 130 million parameters and took 2-3 weeks to train.

Next generation of classification architectures focused on reducing the number of trainable parameters while maintaining the depth of network. ResNet [79] by

He *et al.* took a big leap in this direction by incorporating skip connections and batch normalization into the network. In contrast to VGG [140], ResNet [79] had fewer parameters despite being eight times as deep. ResNet [79] achieved a 3.57% error rate on the ImageNet test set surpassing human level performance. The skip connections of ResNet [79] were additive where the activations of one layer were added on to another. An alternative approach was proposed by Huang *et al.* [90] in DenseNets where the activations from different layers were concatenated. DenseNets [90] contained even fewer parameters and performed competitively with ResNet [79] on the ImageNet [167] dataset.

As mentioned previously, segmentation is the task of performing classification on a per-pixel basis. A segmentation framework should not only recognise the correct class label but must also determine the location for each label. This is specially important for a range of tasks including navigation and medical anomaly detection. For example, in addition to detecting a pedestrian it is important for a self-driving vehicle to know where the pedestrian is. Failing to perform either of these tasks correctly could have catastrophic ramifications. Computer vision researchers have investigated the possibility of porting over some of the techniques used in classification architectures to segmentation.

Fully Convolutional Networks (FCN) by Long *et al.* [125] introduced the first segmentation architecture which takes an arbitrary sized input image and produce a labelled image of the same resolution by modifying the classification architectures [174, 107, 186] into fully convolutional variants. They also used skip connections to re-introduce the fine appearance information to the deeper layers. Chen *et al.* [23] used a three pronged approach to tackle the segmentation problem. They incorporated atrous convolutions where the values of a kernel were spaced out based on a "dilation factor". This feature allows to arbitrarily increase the field of view while preserving the model parameters and the amount of computations. Secondly, they performed spatial pyramidal pooling to explicitly enforce the network to use convolutional features from different spatial scales. Lastly, through the use of a fully-connected Conditional Random Field (CRF) the model was

forced to allocate similar labels for pixels with similar colour and position. More recently, Lin *et al.* proposed to predict high resolution segmentation outputs using long-range residual connections in RefineNet [121]. In a similar vein, Zhao *et al.* generated high quality predictions through the use of global pyramid pooling [210]. They achieved state-of-the-art results on a multitude of datasets including PASCAL VOC 2012 [50] and Cityscapes [30].

2.2.2 Predicting Depths from RGB images

Humans are capable of understanding the structure of a scene by merely looking at the environment. This allows us to perform complex tasks such as playing sports. For example, in a game of cricket or baseball it is important to understand when the ball is travelling towards you, and the direction the bat needs to angled in order to hit the ball away. All of these tasks are done by observing the visual cues. This presents an important research problem, can a robot be trained to infer the surface quantities (depth, surface normals, surface curvature) purely from a colour image?

Depth information can be obtained from colour images using structure from motion or SLAM algorithms as discussed previously. Monocular SLAM in particular has two limitations. Firstly, the reconstruction is only accurate up to scale. Secondly, in order to initialise the map a second image which captures the scene from a slightly different viewpoint is required. Both of these issues can be addressed using machine learning approaches. Furthermore, in a robotic context going from data to information as efficiently as possible is vital, and predicting quantities from a single image is a step in that direction.

One of the earliest works to predict depth from a single image [136] was the one dimensional depth prediction framework of Michels *et al.* They targeted the problem of obstacle avoidance for a remote controlled car driven at high speeds.

A supervised learning approach was taken to predict relative depths of obstacles using columns of a particular image as the input. An extension to this approach was later proposed by the co-authors to learn the depth map for the entire image [171]. They presented a Markov Random Fields (MRFs) based technique to use image cues at multiple spatial scales to model the depths as well as the interaction between these depths at multiple scales. Hane *et al.* proposed to employ a data driven surface normal classifier as a regularisation term of the energy formulation, in order to extract a depth map from unary potentials for different labels [75]. Ladicky *et al.* used semantic labels to improve the accuracy of depth prediction [112]. However, the features used in their approach were hand-crafted. This work also introduced a threshold based metric to evaluate the performance of single image depth prediction which reports the percentage of predicted depth values that are within a range (δ) of the ground truth values. Since then, it has been commonly used to benchmark the accuracy of depth prediction platforms.

Eigen, Puhrsch and Fergus in [45] introduced the first single image depth prediction framework that used a convolutional neural network. This system employed two stacks of networks (coarse and fine). The coarse-scale network was largely based on the AlexNet [107] architecture which included two fully connected layers enabling the network to gain a global understanding. The coarse level prediction was then refined by the fine-scale network. Eigen and Fergus later extended this platform to predict depths, surface normals and semantic labels simultaneously in [44]. Predicting all three tasks in a joint framework improved the accuracy of each individual task showcasing the benefits of sharing different feature representations of the same input.

Liu et al. [122] proposed to combine graphical models in the form of a Conditional Random Field (CRF) with a CNN to improve the accuracy of monocular depth estimation. More recently, Laina et al. [114] proposed to use a fully convolutional residual architecture [79] combined with up-project blocks to achieve superior performance over previous methods. They also found that using a reverse Huber loss improved the accuracy of the predicted depths compared to using an L2

loss to train the network. While most of these methods demonstrated impressive results, explicit notion of geometry was not used during any stage of the pipeline. More concretely, the depth prediction problem was tackled purely from a machine learning perspective, largely treating it as a segmentation problem. This opened up the way for geometry based depth prediction approaches.

In one of the earliest works to predict depth using the knowledge of geometry in an unsupervised fashion, Garg *et al.* used the photometric difference between a stereo image pair, where the target image was synthesized using the predicted disparity and the known baseline [62]. Their work was largely motivated by the resource (human) intensive nature of capturing ground truth depth data. Despite using a pure unsupervised loss, they managed to achieve comparable performance to the state-of-the-are supervised methods at the time. A limitation of this approach was the not fully differentiable image formation process, in order to overcome this problem they used a Taylor series approximation to linearise the loss criterion. Left-right consistency was explicitly enforced in the unsupervised framework of Goddard *et al.* [67] which used a Siamese style network architecture.

While unsupervised training regimes have the advantage of being able to provide a training signal even under the absence of the ground-truth data, if the ground truth labels are available it is beneficial to incorporate this information during training. To this end, Kuznietsov *et al.* [111] proposed a semi-supervised depth prediction pipeline. This platform proved to be far superior compared to approaches that purely used a supervised training loss or an unsupervised loss in the form of an image reconstruction cost function. Since the ground truth data obtained from a LIDAR is highly sparse, the image reconstruction loss plays an important role by providing a training signal for the points that have a missing label. On the other hand, through the use of a supervised loss, it is possible to establish a notion of metric depths. Thus, the two approaches compliment each other well, and when combined together provide a stronger training signal.

2.2.3 Uncertainty Prediction

Neural networks are employed to predict a range of tasks expanding beyond the classification and segmentation tasks that have been discussed so far. Although most of these models are deployed in a constrained environment, the end goal is to use the trained models in the real world interacting with humans or possibly other machines. As the stakes get higher, it becomes important to have a fail safe plan. If the neural network model can inform when its uncertain about a prediction, this information can then be used to take an appropriate action depending on the scenario. The two types of uncertainties that mainly concern us are: epistemic and aleatoric uncertainties [36]. The former explains the uncertainty in the model parameters while the latter explains the uncertainties that are agnostic to changes in input and the uncertainties that varies with the input. A few notable research papers under this topic are described below. I would also like to refer the reader to the PhD thesis of Yarin Gal for a more thorough treatment of 'Uncertainty in Deep Learning' [58].

Denker and LeCun [35] were one of the earliest to explore the idea of a Bayesian Neural Network (a network that is capable of modelling uncertainty), in which they demonstrated that it's possible to learn a probability distribution over the output values. More concretely, they showed that a confidence interval could be assigned to both the weights and the outputs of the network. Recently, [59] showed that dropout can be used to model the uncertainty of neural network. Kendall and Gal [101] explored the problem of understanding the uncertainties needed for computer vision applications. They demonstrated that uncertainty associated with model parameters and inherent noisy measurements can be both learned using a Bayesian neural network approach. However, using the learnt uncertainties to further refine the depth estimates is relatively less explored problem. Finally, Kendall et al. [102] used the knowledge of uncertainty in a multi-task learning platform where the contribution of different losses were weighted using task dependant uncertainty.

2.2.4 Real-time Predictions and Mobile Platforms

The target platform for a vast majority of the neural network models trained to perform segmentation is a conventional desktop computer with a graphics card. Despite achieving state of the art performances, these models can often consume tens of seconds to produce an output. Consequently, on a mobile framework with much less compute, the inference phase can be as long as half an hour. However, real-time frameworks are of paramount importance in a robotics context, where the ultimate goal is to create an environment where robots and humans can interact seamlessly. All things considered, a compromise has to be made between accuracy and runtime and the research summarised below strive towards achieving that goal.

Redmon *et al.* proposed a real time object detection framework called YOLO to simultaneously predict a bounding box and the associated class probability [160]. MobileNets by Howard *et al.* specifically targetted mobile platforms [88]. They used depth-wise separable convolutions as well as two hyperparameters to choose the width and the resolution of each layer. MobileNets achieved similar performance to that of VGG16 [174] despite having the number of parameters reduced by a factor of 32. ICNet [209] tackled the problem of real-time semantic segmentation by using a cascaded input of varying image resolutions. A learnable fusion layer integrates the feature stacks learnt at different resolutions. Their system runs at 30fps on a TitanX GPU, while slightly outperforming deeper segmentation frameworks such as FCN [125].

CHAPTER 3

Preliminaries

With the primary goal of making the thesis as self-contained as possible, this chapter introduces the mathematical concepts used throughout the document as well as a brief introduction to Convolutional Neural Networks (CNNs). Firstly, the notation used to represent points as well camera pose is introduced. Secondly, a short account on rigid-body transformations and camera projection is included. Subsequently, a summary of various layers of a convolution neural network is presented. Finally, the chapter is concluded with an outline of the software packages used to conduct the experiments described in the thesis.

3.1 Mathematical Notation

3.1.1 Points and Vectors

A vector is represented as a lower case boldface character : $\mathbf{v}$. An n-dimensional point in Euclidean space is represented as a vector in $\mathbb{R}^n$. For example a vector

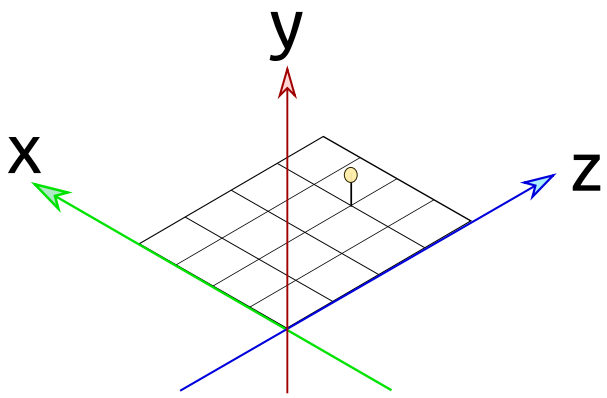

Figure 3.1: Representing a point in 3D. The point marked with the yellow dot has the coordinates (2,1,3)

in $\mathbb{R}^3$ can be represented as $\mathbf{x} = \begin{pmatrix} x & y & z \end{pmatrix}^T$

A 3D point can also be represented using homogeneous coordinates $\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} & \tilde{y} & \tilde{z} & \tilde{w} \end{pmatrix}^T$, where vectors that differ by a scale factor are considered to be equivalent.

A homogeneous vector $\tilde{\mathbf{x}}$ can be converted back into its' inhomogeneous representation $\mathbf{x}$ by dividing all elements in the vector by the last element.

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} & \tilde{y} & \tilde{z} & \tilde{w} \end{pmatrix}^T = \tilde{w} \begin{pmatrix} x & y & z & 1 \end{pmatrix}^T \tag{3.1}$$

The vector $\begin{pmatrix} x & y & z & 1 \end{pmatrix}^T$ is called the augmented vector. Homogeneous points with a last element of zero represent points at infinity and are called ideal points. Points are defined with respect to a coordinate system as shown in Figure 3.1. Having defined a representation for 3D points, we can now examine how a point or a collection of points can be transformed.

3.2 Transformations

A 3D point defined with respect to a coordinate frame can be transformed to another coordinate frame using a 4×4 motion matrix (T) as follows :

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = T \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \qquad \text{where} \quad T = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0_{1\times 3}} & 1 \end{pmatrix} \tag{3.2}$$

In this thesis, we are primarily concerned with rigid body transformations that preserve angles and distances. Rigid body transformations are also known as Euclidean transformations and can be represented using a combination of a 3×3 rotation matrix (R) and a 3×1 translation vector ($\mathbf{t}$). The rotation matrix is orthonormal where the $|R| = 1$ and the columns are orthogonal to each other. As a consequence of these two conditions we can observe that $R^T R = I$ where I is the 3×3 identity matrix.

Throughout the remainder of this thesis transformation matrices as well as its constituent matrices are expressed to reflect the direction of the transformation. More concretely, a transformation matrix is expressed as $T_{\overleftarrow{ba}}$, its rotational component as $R_{\overleftarrow{ba}}$ and the translational vector as $\mathbf{t}_{\overleftarrow{ba}}$. Where a and b are two coordinate frames and the direction of the motion is given by the direction of the arrow $\leftarrow$. Hence, a transformation from frame a to c via frame b can be expressed as follows :

$$T_{\overleftarrow{ca}} = T_{\overleftarrow{cb}} T_{\overleftarrow{ba}} \tag{3.3}$$

3.3 Lie Groups

A Lie group is a smooth differential manifold which simultaneously has a group structure consistent with its manifold structure. Different Lie groups are used throughout this thesis to represent transformations in 3D. In particular, this subsection focuses on the two Lie groups SO(3) and SE(3) which represent rotations in 3D and rigid transformations in 3D respectively.

We introduce Lie groups by first introducing the concept of a *group*.

A group is a set G with an operation $\circ : G \times G \to G$ which exhibits the following properties:

- Closure : $g_1 \circ g_2 \in G \quad \forall\ g_1,g_2 \in G$
- Associativity : $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3) \in G \quad \forall\ g_1,g_2,g_3 \in G$
- Identity : $\exists I \in G : I \circ g = g \circ I = g \quad \forall g \in G$
- Inverse : $\exists g^{-1} \in G : g \circ g^{-1} = g^{-1} \circ g = I \quad \forall g \in G$

A matrix representation of a group G can be defined as follows, if there exists an injective function:

$$\rho : G \to GL(n)$$

such that $\rho(g_1 \circ g_2) = \rho(g_1)\rho(g_2)$, $\forall\ g_1,g_2 \in G$. Studying the matrix representation of a group provides a lot of insight into the group structure as the group can be analysed closely by examining the properties of the matrix group.

3.3.1 Rotations in 3D space, SO(3)

Rotations in 3D space form a group known as Special Orthogonal group SO(3). Special in this case refers to the set of matrices with a determinant 1, orthogonal corresponds to real matrices whose inverse is the same as the transpose and finally 3 denotes $\mathbb{R}^3$. These properties are summarised below:

$$R \in \mathrm{SO}(3) \tag{3.4}$$

$$\det(R) = 1 \tag{3.5}$$

$$R^{-1} = R^T \tag{3.6}$$

When studying Lie groups, for a majority of the tasks, it is sufficient to consider an infinitesimal transformation around the identity compared to considering the group as a whole. Finite transformations can then be realised through the repeated application of the infinitesimal transformations. The tangent space around the identity is also known as the corresponding Lie algebra of the Lie Group. A basis of the Lie algebra $\mathfrak{so}(3)$ can be derived as follows:

The rotations about the x,y,z by an angle ψ_1,ψ_2,ψ_3 respectively is given by:

$$R(\psi_1) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\psi_1) & -sin(\psi_1) \\ 0 & sin(\psi_1) & cos(\psi_1) \end{pmatrix}$$

$$R(\psi_2) = \begin{pmatrix} cos(\psi_2) & 0 & sin(\psi_2) \\ 0 & 1 & 0 \\ -sin(\psi_2) & 0 & cos(\psi_2) \end{pmatrix},$$

$$R(\psi_3) = \begin{pmatrix} cos(\psi_3) & -sin(\psi_3) & 0 \\ sin(\psi_3) & cos(\psi_3) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The generators can then be obtained by applying $G_k = \frac{dR(\psi_k)}{d\psi_k}\Big|_{\psi_k=0}$ $k \in$ 1,2,3

$$G_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \quad G_2 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} \quad G_3 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.7}$$

3.3.2 Rigid Transformations in 3D space, SE(3)

A rigid transformation (T) can be separated into a translation vector and rotation matrix as shown below:

$$R \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3 \tag{3.8}$$

$$T = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0_{1\times 3}} & 1 \end{pmatrix} \in \mathrm{SE}(3) \tag{3.9}$$

The generators of the lie algebra $\mathfrak{se}(3)$ can be obtained by taking the derivative

of each dimension evaluated at the identity.

$$G_1 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad G_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad G_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$G_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad G_5 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad G_6 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.10}$$

3.3.3 The Exponential and Logarithmic Map

Given a Lie group G with the Lie algebra $\mathfrak{g}$, the exponential map takes the elements in the algebra to that in the group.

$$\exp : \mathfrak{g} \to G \tag{3.11}$$

The inverse operation which takes the members in the group to elements in the lie algebra is given by the logarithmic map.

$$\ln : G \to \mathfrak{g} \tag{3.12}$$

3.4 Camera Projection

Projecting a landmark in the world on to the image plane is an important step in a SLAM pipeline. The pinhole camera model as shown in Figure 3.2 can be considered as one of the simplest projection models. This lensless model creates an image by allowing light to pass through a pinhole sized aperture in an otherwise light-proof box.

Assuming the camera pose is known, a landmark in the world can be projected on to an image as follows:

$$\begin{pmatrix} \lambda u \\ \lambda v \\ \lambda \end{pmatrix} = KT \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.13}$$

$$\begin{pmatrix} \lambda u \\ \lambda v \\ \lambda \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{00} & R_{01} & R_{02} & t_x \\ R_{10} & R_{11} & R_{22} & t_y \\ R_{20} & R_{21} & R_{22} & t_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.14}$$

where K and T denote the camera intrinsic matrix and the camera pose respectively.

3.4.1 Optical Distortion

Optical distortion causes straight lines in a scene to appear as curved after projection (Figure 3.3). The three most common distortion patterns are radially symmetric due to the symmetry of the photographic lens. Barrel distortion causes

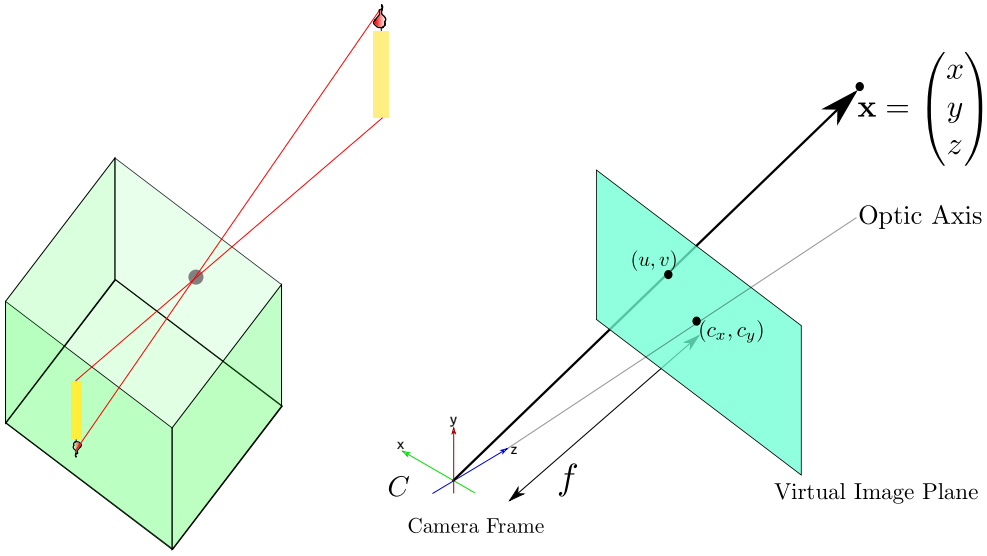

Figure 3.2: Pinhole camera model. Left : An illustration of a 3D landmark being projected on to an image using the pinhole camera model. Note that the projected image is inverted. Right : An upright image can be obtained by having a virtual image plane in front of the camera. The principal point is denoted by (c_x, c_y). The focal length (f) is the distance between the optical center of the camera and the principal point.

straight lines to curve inwards in a shape of a barrel, pincushion distortion exhibits the opposite where straight lines are curved outwards from the center and finally, moustache distortion exhibits a combination of the former two.

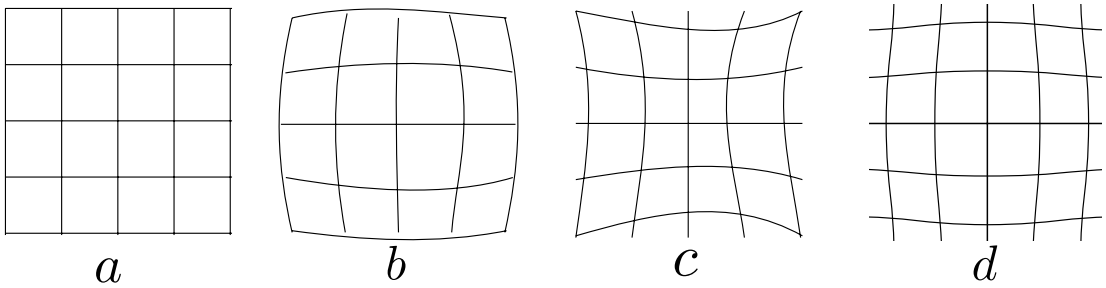

Figure 3.3: Forms of optical distortion a: undistorted image, b: barrel distortion, c: pincushion distortion, d: moustache distortion.

The authors of the KITTI[63], NYUv2 [140] and TUM [182] datasets use the lens distortion model given in the equation system 3.15 to rectify the datasets. The rectified images are used as inputs to the CNNs described in this thesis. For MO-SLAM we only use the first radial distortion coefficient as our cameras do not exhibit tangential distortion.

$$
\begin{aligned}
\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} &= \begin{pmatrix} R_{00} & R_{01} & R_{02} & t_x \\ R_{10} & R_{11} & R_{22} & t_y \\ R_{20} & R_{21} & R_{22} & t_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \\
x'' &= \frac{x'}{z'} \\
y'' &= \frac{y'}{z'} \\
u &= f_x(x''(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x'' y'' + p_2(r^2 + 2x'^2) + c_x \\
v &= f_y(y''(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y''^2) + 2p_2 x'' y'') + c_y
\end{aligned}
\tag{3.15}
$$

where $r^2 = x''^2 + y''^2$ is the squared radial distance of the normalised camera coordinates. k_1, k_2, k_3 are the radial distortion coefficients. p_1 and p_2 are tangential distortion coefficients. u, v are the rectified image coordinates. Matlab and OpenCV provide toolboxes to obtain the camera calibration parameters.

3.5 Convolutional Neural Networks (CNNs)

CNNs were originally employed to perform image classification and segmentation tasks as seen in the previous chapter and have since been extended to predict a range of quantities including depth, surface normals and curvature. CNNs are highly desired for these problems due to their capability of extracting image fea-

tures which are invariant to rotation, scale, lighting conditions etc. While the end goal is to find a non-linear mapping from the colour image to the expected output (class label, per-pixel depth), predicting surface quantities is more challenging as the variants of the same input could lead to multiple outputs as shown in Figure 3.4. This section aims to introduce the typical layers that can be found in a CNN, as well as the hyperparameters used during training.

Figure 3.4: A typical image classification network would predict "car" as the label for all three images, however a depth prediction network needs to predict radically different values for the pixels that belong to the dynamic car object as it is moving forward. The images are taken from the KITTI dataset.

3.5.1 Layers of a Convolutional Neural Network

Layers of a typical CNN can be loosely categorised into 5 types : feature extraction, activation, normalisation, sub-sampling and up-sampling. These layers are extensively used in the neural network architectures constructed in chapters 5,7 and 8.

Feature Extraction Layers

The primary goal of a feature extraction layer is to extract meaningful information from a high dimensional input such as an image, a video etc. Feature extraction is generally performed using a series of learnable convolutional filters. The height

and width of each filter and the number of convolutional filters are architecture specific hyperparameters.

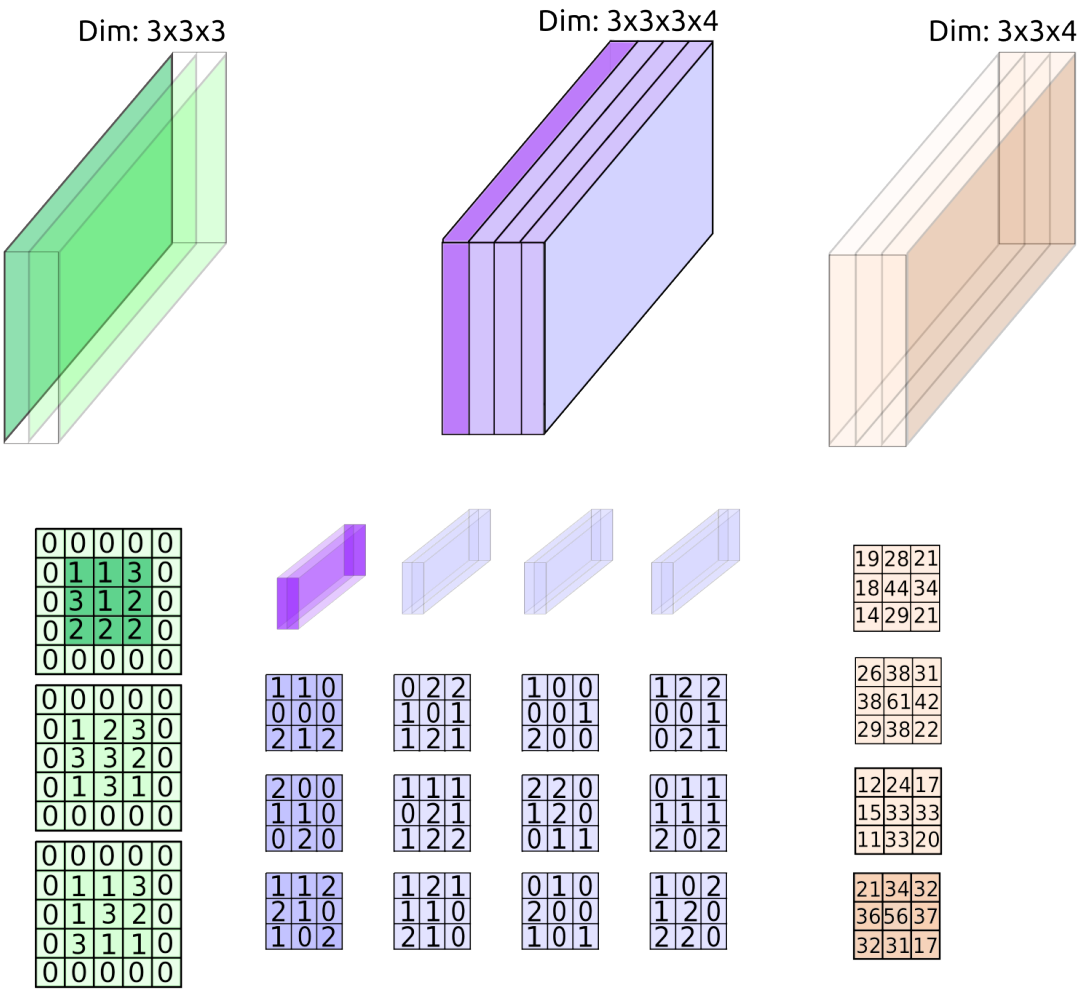

Figure 3.5: A convolutional layer (purple) is applied on an input (green) producing an output tensor (orange). Darker colours indicate slices of respective tensors. Dimensions of each tensor are indicated on top.

A **convolutional layer** is a stack of convolutional filters each with the same height and width and act as the the building block of a CNN. Applying a convolutional layer on an input image yields an output tensor as shown in Figure 3.5. This output tensor serves as the input to the next set of convolutional filters and this process is repeated to extract a hierarchy of features. Figure 3.6 shows

a numerical example of applying a convolutional filter to an input.

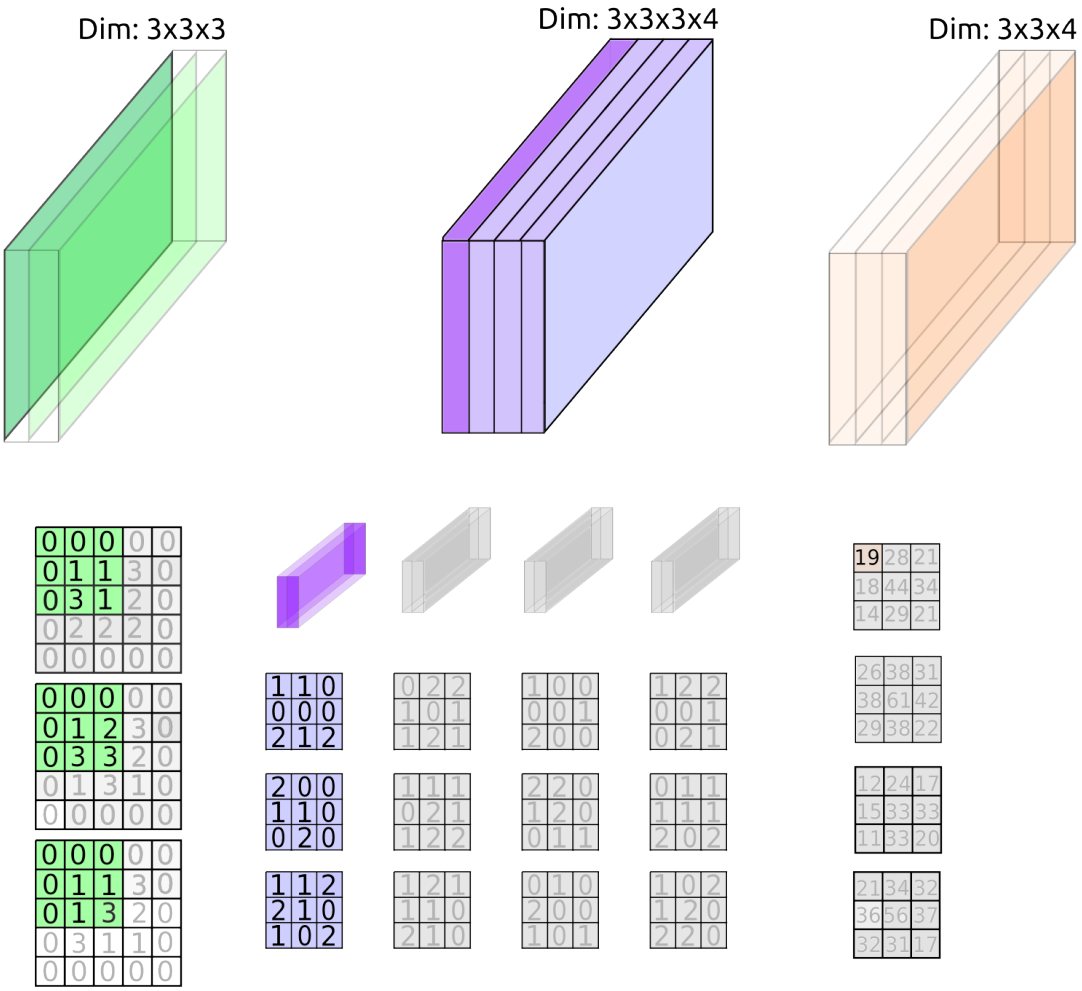

Figure 3.6: This example shows how to compute a single entry of output. This is achieved by computing the dot product of the green tiles and the purple tiles and adding the intermediate results $\{3 \times 1 + 2 \times 1\} + \{1 \times 1 + 3 \times 2\} + \{1 \times 1 + 3 \times 2\} = 19$

Fully Connected Layers are predominantly used in classification networks to perform feature extraction due to their unique characteristic of being able to see the entire image in its field view. More concretely, each weight of a fully connected layer is connected to every activation of the previous layer. This allows the network to obtain a global understanding of the scene. However, fully connected layers are expensive in terms of memory consumption and are generally applied

on a low resolution (spatial) input.

Activation Layers

Activation layers or non-linear layers are generally applied after a convolutional layer. Primary goal of an activation layer is to introduce non linearity into the CNN architecture. Although convolutional layers perform remarkably well at extracting features, convolution itself is a linear operation. Activation layers play a crucial role in achieving the end-goal of finding a non-linear mapping from the input images to the desired outputs. The most commonly used activation layers are summarised below.

- Rectified Linear Unit (ReLU)

$$y = \max(0, x) \tag{3.16}$$

- Sigmoid

$$y = \frac{e^x}{e^x + 1} \tag{3.17}$$

- Hyperbolic tangent

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.18}$$

Normalisation Layers

Feature normalisation is applied at various parts of a neural network in order to improve the performance and stability of the network. The input image itself is generally normalised by subtracting the dataset mean. In addition to this, the input can be also divided by the dataset standard deviation. The **Batch**

Normalisation [93] layer was invented to perform this process at every layer. Ioffe and Szegedy in [93] showed batch normalisation allows a network to be trained faster and more accurately. Additionally, it provides regularisation to the network and facilitates the use of larger learning rates.

Sub-sampling Layers

As performing repeated computations purely on features that are of the initial input resolution becomes expensive and memory intensive, it is a common practice to reduce the spatial dimensions progressively. This is generally achieved through the use of pooling layers (max and average) as shown in Figure 3.7 or strided convolutional layers.

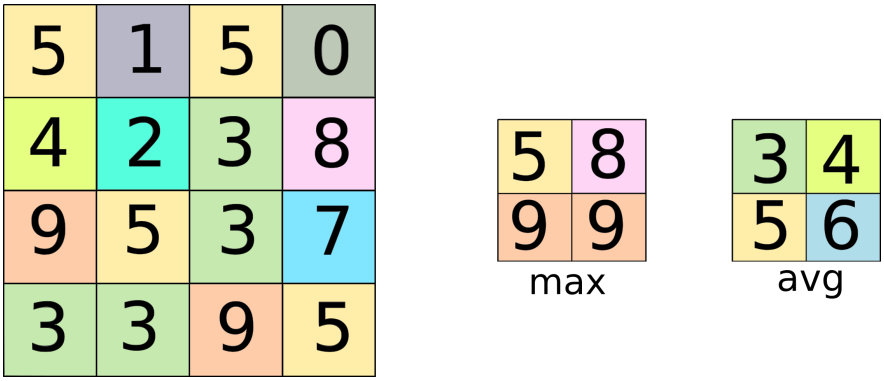

Figure 3.7: Left : 4×4 input image, Center : output after applying a 2×2 max-pooling operation with a stride of 2, Right : output after applying a 2×2 average pooling operation with a stride of 2.

Pooling layers do not contain any learnable weights and are applied on every feature map along the depth dimension, this preserves the number of feature maps along this channel, while reducing the spatial dimensions based on the pooling factor. Given a pooling window, a max-pool layer extracts the largest value (the most dominant feature), while an average-pool layer provides the mean

of the values (smaller values are not completely neglected).

Alternatively, a set of feature maps can be downsampled using a strided convolutional layer. The downsampling factor is determined by the stride and this approach has the added benefit of using learnable weights to summarise the information. However, strided convolutional operations are slightly expensive compared to pooling operations. In practice, both approaches are used at different stages of a neural network depending on the architectural requirements.

Up-sampling Layers

Segmentation networks in contrast to classification networks require an up-sampling stream inside the CNN, to recover the spatial information lost during the down sampling process as the final output is typically predicted for each pixel.

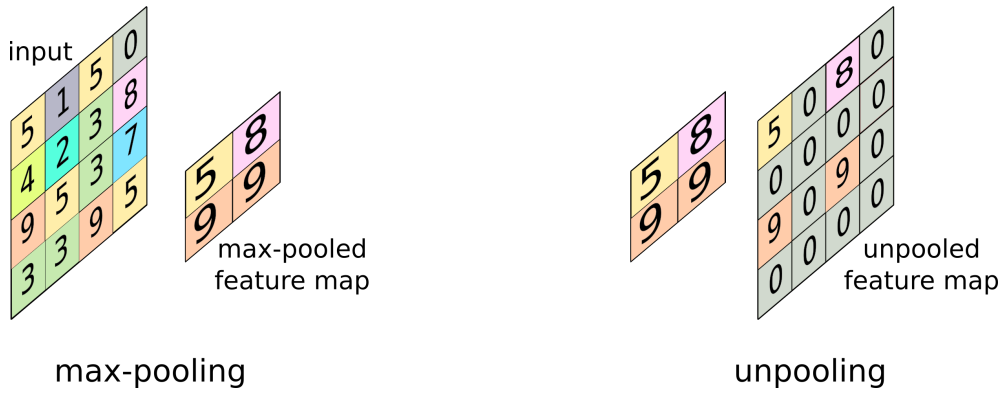

Figure 3.8: An example of 2×2 unpooling which increases the spatial resolution along the height and width dimension by a factor of 2. In this case, the input values are copied on to the top right position and the intermediate values are filled with zeros. Alternatively, these intermediate values can be filled using a 'tiling' approach by assigning the input value to all 4 positions in the 2×2 window.

The most commonly used upsampling techniques are unpooling, deconvolution and interpolation (bilinear, nearest-neighbour). Similar to the pooling layers,

unpooling layers also do not contain any trainable weights. Based on the upsampling factor the input value is copied across to the output feature map as shown in Figure 3.8.

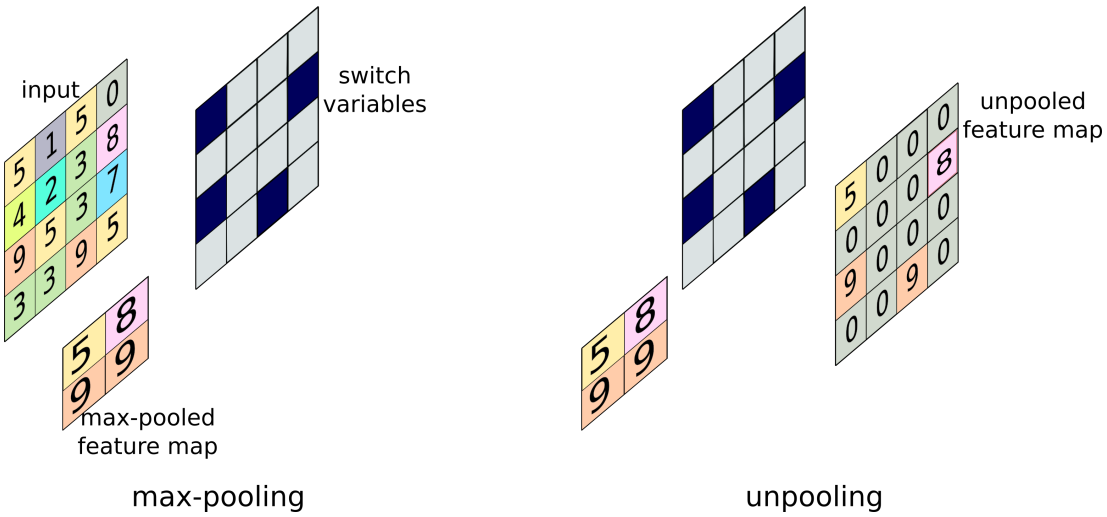

Figure 3.9: During the pooling stage in addition to the pooled output an additional map corresponding the locations of the maximum (switch variables) is created. These outputs can then be used during the unpooling stage to place the unpooled values to the corresponding locations.

Zeiler *et al.* in [206] proposed a modified variant where the locations of the maximum values ("switches") are stored during pooling and these switches are passed on to the unpooling layer as an additional input as shown in Figure 3.9.

Another popular upsampling technique is deconvolution. Despite the name, a deconvolutional layer truely performs transposed convolution. While the input stride of a convolutional layer corresponds to the downsampling factor, the output stride of a deconvolutional layer gives the upsampling factor. This can be explained best using a numerical example as shown in Figure 3.10.

Although information lost during pooling (max or average) is not fully recoverable, upsampling layers aim to reconstruct an approximate representation. Some of the finer details lost during the down-sampling process is generally introduced

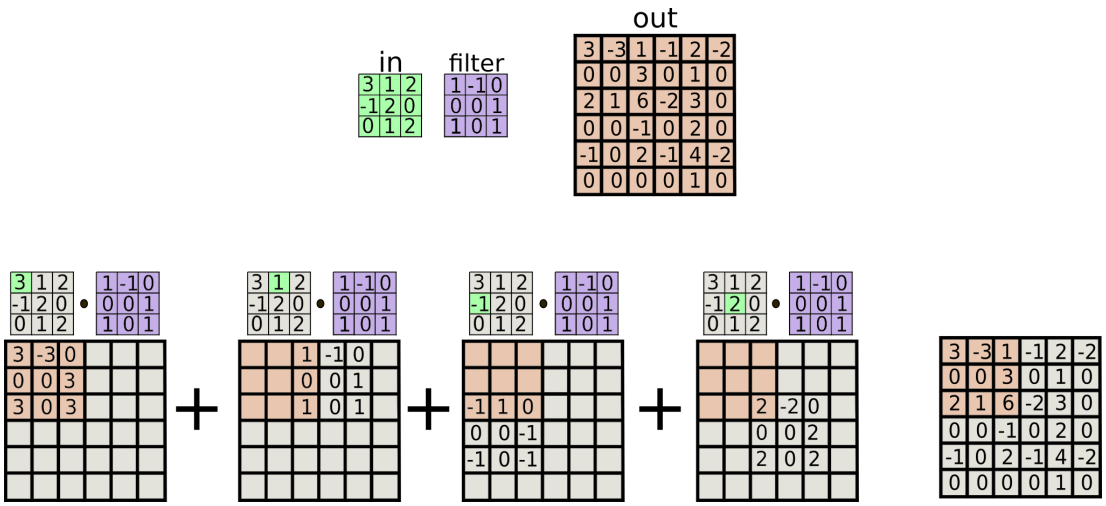

Figure 3.10: **Top**: The output of a deconvolution operation when a 3×3 filter is applied on a 3×3 input. The output stride is 2 along each dimension. **Bottom**: Demonstrates a step by step breakdown of the deconvolution operation. The filter is multiplied by the value highlighted in green and the intermediate results are summed to obtain the final output

back in to the upsampling stream using one or more skip connections. As suggested by its name a skip connection provides a direction connection from a particular layer to another layer by skipping over the intermediate connections as shown in Figure 3.11 . For example, activations of a layer in the encoder stage (downsampling stream) can be passed on as an additional input to a layer in the upsampling stream (decoder stage).

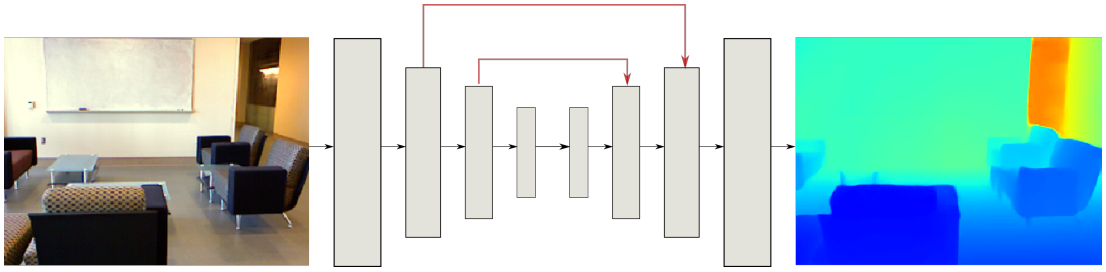

Figure 3.11: This diagram shows a high level overview of a CNN (used for depth prediction). Red arrows indicate skip connections which allow for the activations of the layers in the encoder stage to be passed on as additional inputs to layers in the decoder stage.

3.6 Training Convolutional Neural Networks

The training phase of a convolutional neural network focuses on learning a set of weights and biases, that when combined together gives a non-linear mapping from a set of inputs to the desired output.

3.6.1 Initialisation

The initial state of the neural network or the initial values of the weights and biases can be set randomly (random initialisation) or to that of an already trained network (pre-trained weights). Commonly used random initialisation techniques are Gaussian, Glorot[66] and He[78]. Random initialisation allows each learnable parameter to have a different starting point. The primary reason behind this is to ensure that all the weights contribute slightly differently to achieve the end-goal which allows the weights to be updated differently in order to extract unique features.

Alternatively, a neural network can be initialised using pre-trained weights. However, this requires the network that needs to be trained to have the same architecture as that of the already trained network. Partial initialisation is also possible if the two networks are not identical in terms of the architecture but have some layers in common. In this case, these common layers can be initialised using pre-trained weights, while the remainder of the network is initialised using random weights.

3.6.2 Loss criterion and Back-propagation

The parameters (weights,biases etc.) of a neural network are optimized using one or more loss functions. For example, under a supervised training scheme

the cost is computed by comparing how well the predicted quantities match the ground truth labels as shown in Equation 3.19 and training the neural network is largely centered around minimising this loss function. Variants of the gradient descent algorithm are used to perform this optimization process. The crux of the algorithm is centered around taking small steps proportional to the negative of the gradient.

$$\mathscr{L}(\theta) = ||\text{Prediction} - \text{Ground Truth}||^2, \tag{3.19}$$

where θ represents the learnable parameters of the neural network.

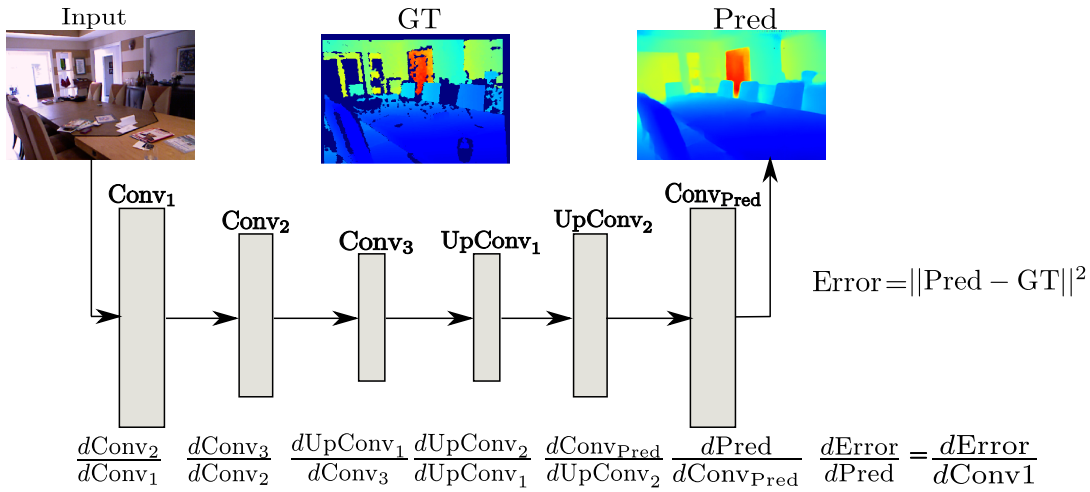

Figure 3.12: During the forward propagation step an RGB image is given as the input to the neural network, activations of each layer are passed as inputs to the following layer ultimately resulting in a prediction at the end. The error/loss can then be obtained by finding the difference between the neural network prediction and the ground truth labels. During the back propagation step, the gradients can be computed with respect to the error by applying the chain rule as shown.

Once the loss is computed for each neuron of the output layer, the required change in weights in order to minimize this loss can be computed using the back-propagation algorithm which uses the chain rule to propagate the gradients as shown in Figure 3.12. The weights are updated by taking a small step proportional to the negative of the gradient as shown in Equation 3.20, hence this step

is commonly known as gradient descent.

$$\theta(t+1) = \theta(t) - \lambda \cdot \frac{d\mathscr{L}}{d\theta}, \tag{3.20}$$

where $\theta(t)$ represents the current estimate of weight values, λ is the learning rate, $\frac{d\mathscr{L}}{d\theta}$ is the partial derivate of the weights with respect to the loss and $\theta(t+1)$ is the updated set of weights.

The length of the step taken in the direction of the negative gradient is governed by a hyperparameter called the learning rate. A larger learning rate can result in overshooting during gradient descent and can often cause the network to diverge. On the other hand, a smaller learning rate causes slower convergence. In practice, the optimal learning rate is found by trial and error.

During training, it is possible for the network to get stuck in a local minima exhibiting an oscillatory behaviour. One way to overcome this issue is through the use of the momentum method [155] which slightly changes the update equation as follows :

$$\theta(t+1) = \theta(t) - \lambda \cdot \frac{d\mathscr{L}}{d\theta} + \mu(\theta(t) - \theta(t-1)), \quad 0 \leq \mu < 1 \tag{3.21}$$

The momentum value (μ) allows the network to take larger steps towards the direction of the global minimum ensuring faster convergence.

3.6.3 Regularisation of CNNs

The primary goal behind regularisation is to prevent over-fitting which in turn allows the network to learn a more generalised non linear mapping from the input

data to the output. Over-fitting can be observed during training if the training loss continues to decrease, while the validation loss increases. The harmful effects of over-fitting can be mitigated through the use of regularisation techniques such as dropout[179] and weight decay[108].

Dropout layers function asymmetrically during the training and inference phases. At training time, the activations of the layer preceding the dropout layer are randomly set to zero with a probability of p. During test time p is set zero i.e all of the activations are passed on to the next layer unscathed. During training the neurons of a particular layer can no longer rely on the full set of activations from the previous layer and must learn to perform something useful on their own or in collaboration with a small subset of activations. This forces the network to learn more generalised representations compared to learning features that are specific for the training set.

Building deeper and wider models which in turn increases the number of learnable parameters (weights and biases) generally leads to increase in accuracy. However, this also causes the network to over-fit more easily due to the large number of free parameters present. Weight decay can be used to constrain such a network. This prevents the weights from growing too large too quickly. More concretely, for each weight (w) of the network an additional term $\frac{\gamma w^2}{2}$ is added to the cost function, where γ is the magnitude of the damping factor. This is also known as L2 regularisation. Alternatively, the weight penalty can be L1 ($\gamma|w|$), however, this generally provides inferior performance compared to L2.

3.7 Software

The platforms and the software libraries used throughout this thesis are listed below.

- TooN (Tom's Object-Oriented Numerics) is a templated library for matrix

and vector operations such as matrix and vector multiplications. TooN also provides wrappers to LAPACK (linear algebra package) library for performing matrix decompositions. Further, TooN provides matrices that are parametrized as Lie groups. Available at: https://github.com/edrosten/TooN.

- libCVD (Cambridge Vision Dynamics) is a high performance C++ based library for computer vision and image processing. Available at: https://github.com/edrosten/libcvd.

- OpenCV (Open Computer Vision) an application programming interface (API) for real-time computer vision. Written in C++ with python wrappers. Available at: https://github.com/Itseez/opencv.

- Blender (Blender Foundation) an open-source 3D computer graphics framework which was used to render 3D models of synthetic scenes. Available at: https://www.blender.org/

- Caffe (University of California, Berkely) q C++ based deep learning framework. Contains a vast array of layers used in Convolutional Neural Networks. Available at: http://caffe.berkeleyvision.org/

- Tensorflow (Google Brain) a python based deep learning framework. Allows rapid prototyping and supports automatic differentiation. Tensorflow is also bundled with Tensorboard which is a great visualisation tool to monitor the training process in real-time. Available at: https://www.tensorflow.org

- Matlab (Mathworks) is a numerical computing platform. This software platform was heavily used throughout the thesis for a range of tasks including rapid prototyping of ideas, pre-processing of datasets and to visualise outputs. Available at: https://au.mathworks.com/products/matlab.html

CHAPTER 4

Structure Refinement Using Multiple Objects

4.1 Introduction

Recovering the scene structure and the camera pose from two or more views of the same scene, and jointly improving the reconstructed scene as well as the parameters governing the camera motion is a well studied problem in robotics. This chapter takes this problem one step further by introducing objects, where the accuracy of all three aspects (structure, camera pose and objects) are refined in the context of a SLAM system. Due to the use of multiple objects in a SLAM framework, this approach is called MO-SLAM (Multi-Object SLAM).

In this chapter, the reconstruction problem and the subsequent structure refinement component is tackled using a pure geometry based approach, where a SLAM system is used to construct a map of the environment and to track the camera position. Visual SLAM systems are an attractive choice for real-time robotics due to the ubiquitous nature of standard commodity cameras as well as due to the existence of faster sparse SLAM variants which can track and map at frame rate. A SLAM system provides a robot with two key pieces of information, a

map of the environment surrounding the robot and the location of the robot with respect to that environment in real-time. However, in order to interact with the entities that are present in the environment, it becomes necessary for the robot to gain an understanding of the higher order semantics. In a nutshell, visual SLAM grants a robot the ability to perceive and navigate while semantics allow it to interact with the environment.

The word semantics has been used broadly in the field, specially with the recent advancements in machine learning which look at segmenting the scene based on a class label. In this chapter, the semantic aspect is looked at from a slightly different angle where the map points that belong to an object are given a unique label. More concretely, different objects are labelled in the form of Object_1, Object_2 etc. rather than saying"Chair, Telephone or Book". Due to the presence of an appearance profile associated with each object, a CNN based image recognition approach can later be utilised to give a meaningful class label (For e.g Object_1 $\rightarrow$ chair, Object_2 $\rightarrow$ book). The primary motivation behind this approach is to avoid the use of a pre-populated object database and to advocate the use of unsupervised object discovery. It should be noted that this framework is not intended to replace existing approaches that use an object database, we envision the techniques introduced in this chapter to complement such methods, as MO-SLAM can learn new objects during run-time.

4.1.1 Contributions

In this chapter we make the following contributions:

- A novel and efficient method to cluster subsets of the point cloud from a 3D map, and to convert these clusters into first-order entities in the 3D map.
- A tight feedback loop where additional constraints generated from converting points into first-order entities are used to improve the accuracy of the

visual SLAM system and to combat scale drift.

4.1.2 Related Work

Online Object Discovery Unsupervised object discovery has been a popular research topic in the recent past [175, 3, 68], with many of these systems attempting to discover the existence of objects from a set of training images. Karpathy *et al.* used 3D meshes to generate object hypothesis [99]. After segmenting the scene into mesh segments, they compute an objectness score based on shape parameters. Cho *et al.* suggested a threefold approach to tackle the problem of detecting duplicates in an online detection pipeline [24]. Starting off with feature matching, they used a multi-layer growing algorithm to increase the matches to an object correspondence network. Similarly, Liu and Liu used feature based visual word matching followed by a greedy randomized adaptive search procedure to detect common object patterns from a single view [124]. In [196] Wang *et al.* proposed to discover duplicate images within 2 billion images using a framework which combines global and local features. Global features are used to detect exact and global duplicates while local image descriptors are needed to handle variations such as transformations and crops. However, as all of these systems lack any 3D information they cannot distinguish between multiple instances of an object but merely different views. Therefore, to achieve true duplicate discovery, a platform that feeds in structural information is required.

Structure From Motion with Duplicates Reconstructing scenes containing symmetrical and/or duplicate structures have been first explored by the SfM community. Roberts *et al.* proposed to use an expectation-maximization algorithm in tandem with a sampling technique that incorporates geometric, photometric and time-based cues to find correspondences between duplicate objects in a large, unordered set of images [162]. They eliminated erroneous matches by looking for missing correspondences across triplet of images. Jiang *et al.* proposed a similar approach in [98] where they searched for missing correspondences in the whole

scene by reprojecting 3D points into images instead of local regions. In [28] Cohen *et al.* showed that feature matches between symmetrical structures in the scene can be used to impose additional constraints during the reconstruction process. Similarly, we use duplicate objects to impose additional constraints on the map. Finally, Wilson and Snavely proposed to disambiguate the scene by computing a bipartite local clustering coefficient for point tracks(set of matches across multiple images) [202].

Combining SLAM and Semantics Associating object labels with features in the map was successfully demonstrated by Castle *et al.* [20] and Civera *et al.* [27]. Although the former was restricted to planar objects, both of these systems used a set of object models constructed using feature descriptors during an offline stage which were recognised and registered to the map during run time. Bao *et al.* [6] proposed a semantic SfM system that was able to jointly estimate the camera poses, 3D landmarks and object labels by using both geometric and semantic properties. However, a processing time of 20 minutes per image pair rendered their system infeasible for real-time operation. More recently, Pillai *et al.* [154] developed an object recognition system which was built on top of ORB-SLAM [138], a state-of-the-art SLAM system and was able to recognise objects from multiple view points robustly. Despite using ORB-SLAM to improve the performance of their object detection module, they did not incorporate these objects as additional constraints to optimize the map.

Further, there are systems which use RGB-D data[169, 168, 130, 25, 133] to incorporate semantics into SLAM. The SLAM++ [169] system replaces the 3D model of an object built during run-time with an offline-generated 3D model of the object and shows how it improves the accuracy of the map, whereas the dense planar SLAM system [168] models the world as a set of dense 3D planes. The work of Ma and Sibley [130] discovers objects by first initializing the system with a view of the scene without objects, and then comparing a virtual scene created by this initial view with the current view to look for inconsistencies introduced by an object. In contrast, our work does not require an initialization without objects

present in the scene. Choudhary et al. [25] proposed to discover objects using a segmentation strategy on the depth map, and then using the discovered objects to improve the map. However, they assume that the scene does not contain duplicate objects whereas our system discovers objects by detecting the existence of duplicate objects in the scene.

4.2 System Overview

MO-SLAM consists of three threads running in parallel (see Figure 4.1). Similar to conventional visual SLAM systems, there is a tracking thread which estimates the pose of the camera for every incoming video frame, and a mapping thread which optimizes the map as the camera explores the environment. The difference between MO-SLAM and conventional visual SLAM systems is the addition of a *recognition* thread that is responsible for detecting duplicate objects in the environment.

The tracking thread estimates the pose of the camera for every incoming frame. The pose is initialised by performing sparse model-based image alignment [55] between the current frame and the previous frame, where the model is generated from landmarks that have been observed by the previous frame. The pose is then refined by minimizing the re-projection error of nearby landmarks into the current frame in a coarse-to-fine manner [105].

The recognition thread detects landmarks that belong to duplicate objects in the map. For every new keyframe, the recognition thread determines a set of existing keyframes that are similar in appearance to the new keyframe. Each of these existing keyframes is processed to determine if the existing and new keyframe are observing duplicate instances of the same object type. These duplicate instances are then added to the map as first order entities (see Section 4.3.2).

Finally, the mapping thread optimises the map by minimising the re-projection error of landmarks into cameras in a bundle adjustment framework. In contrast to conventional visual SLAM systems, our bundle adjustment framework treats landmarks belonging to objects as first-order entities, thus allowing us to impose additional constraints in our bundle adjustment formulation (see Section 4.3.5).

It is worth mentioning throughout the remainder of the chapter, we use the following camera projection function which models radial distortion [37]:

$$\hat{\mathbf{z}}(T_i, \mathbf{p}) = \begin{bmatrix} c_x \\ c_y \end{bmatrix} + \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \begin{bmatrix} \frac{y_1}{y_3} \\ \frac{y_2}{y_3} \end{bmatrix} (1 + k_1 r^2) \tag{4.1}$$

where $\mathbf{y} = T_i \cdot \mathbf{p}$. Further, k_1 is the distortion parameter, $r = ||\mathbf{y}||$, f_x, f_y are the focal lengths in the x and y directions, whereas c_x, c_y are the principal offsets. Landmarks $\mathbf{p}$ are represented using a global coordinate system where the first keyframe of the SLAM system is chosen as the global reference.

4.2.1 Duplicate Objects

A duplicate object O is defined as a set of landmarks that can be clustered into different instances $I_1, I_2, \cdots I_n$ where $n \geq 2$. There is a rigid body transformation ($\mathbb{SE}(3)$) from the landmarks of the first instance I_1 to the landmarks of every other instance I_j of the object. In order to distinguish a transformation between duplicate landmarks and a regular transformation, the notation $E^m_{\overleftarrow{j1}}$ is used. This denotes the transformation between the duplicate landmarks of instance I_1 and I_j of the object O_m. This results in the following constraint between duplicate landmarks:

$$\mathbf{p}^m_{I_j} = E^m_{\overleftarrow{j1}} \cdot \mathbf{p}^m_{I_1} \quad \forall j \in 2, \cdots n, \tag{4.2}$$

where $\mathbf{p}^m_{I_j}$ represents the landmarks belonging to instance I_j and $\mathbf{p}^m_{I_1}$ represents the landmarks belonging to instance I_1 of object O_m. We assign a reference keyframe to every instance of the duplicate object. This reference keyframe is

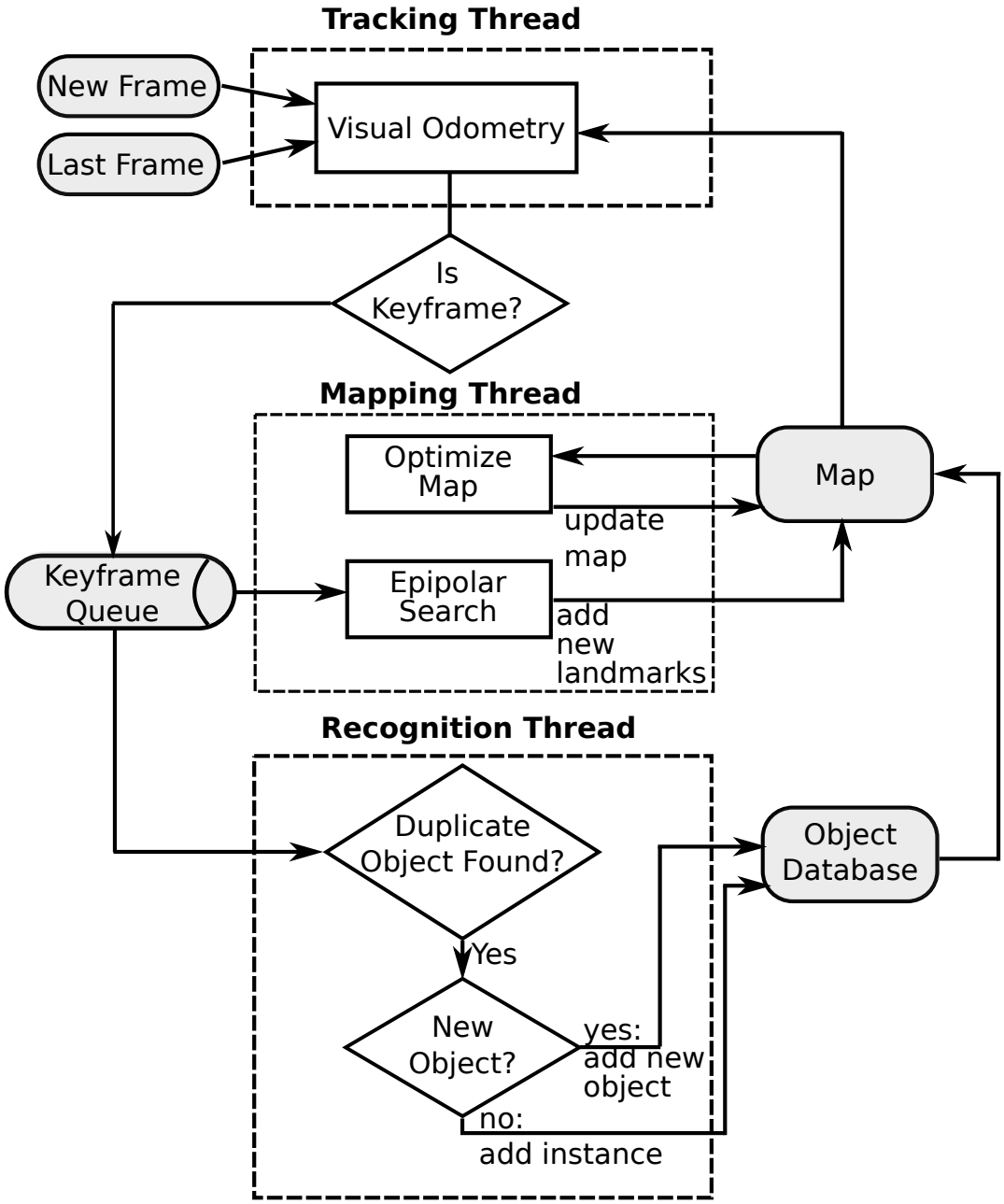

Figure 4.1: A flowchart of MO-SLAM. Note how information from the object database is integrated as part of the map, and then used to optimize the map.

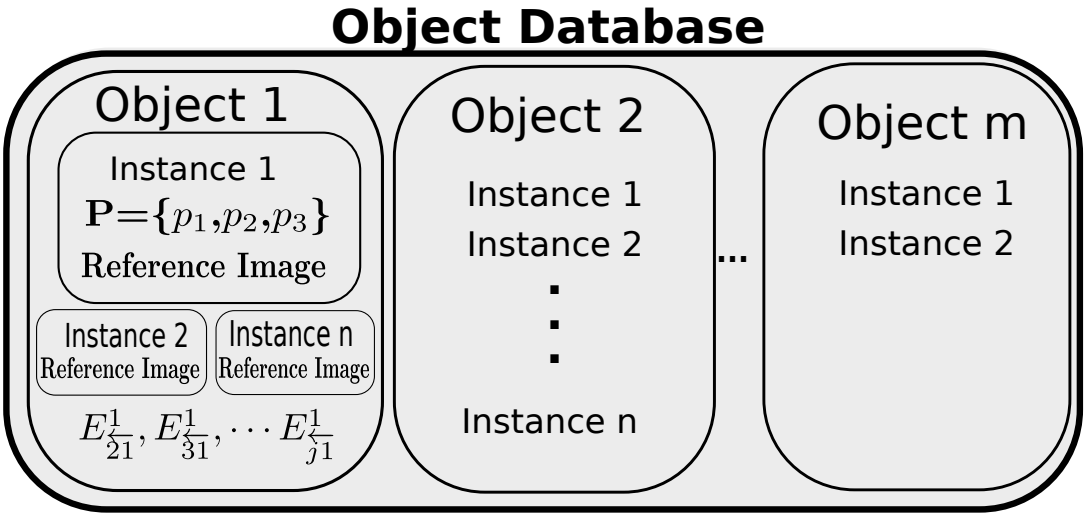

Figure 4.2: The representation of the object database. The database consists of different objects, where there are at least two instances for each object in the database.

chosen as the keyframe which contains the most number of landmarks belonging to that instance. Further, duplicate objects and their instances are stored in an object database as illustrated in Figure 4.2.

4.3 Multi-Object SLAM

4.3.1 Visual SLAM Details

The goal of this subsection is to provide an outline of the sparse SLAM system used in this chapter which was constructed by burrowing elements from the frameworks of [55, 105, 127]. Similar to conventional visual SLAM systems, the map is initialised from two views that are selected by the user through an iterative 5-point algorithm with RANSAC [127].

Subsequently, for every incoming video frame, an image pyramid is constructed

containing 5 levels with a scale factor of $\sqrt{2}$ to provide the system with some degree of robustness towards motion blur and variation in scale. FAST keypoints [164] are detected for each level. In addition, a quad tree is used [135] to ensure a uniform distribution of measurements across the image.

The camera's pose is initialised by performing sparse model-based image alignment between the previous frame and the current frame using a fast variant of the inverse compositional method [129]. The pose is then refined by minimizing the re-projection error of nearby landmarks into the current frame in a coarse-to-fine manner, where the coarse stage uses keypoints from the top two levels of the image pyramid, and the fine stage uses all the available keypoints.

If a new keyframe is added to the map, new landmarks are added to the map by performing an epipolar line search on an existing keyframe that is closest to the new keyframe. If the best point found using epipolar line search has an error smaller than a certain threshold, we triangulate the point from the two views and add the new landmark to the map. Finally, for every measurement in the new keyframe, ORB descriptors [165] are computed and added to a vocabulary tree [60]. Further, for every new landmark and every existing landmark that has been updated with an additional measurement, the landmark is assigned an ORB descriptor from one of its observations. In order to achieve this, Hamming distance is computed between the descriptors corresponding to different measurements of the same landmark, these distances are then sorted and the observation with the ORB descriptor that has the minimum median distance to the other ORB descriptors of the landmark's observations is chosen. Finally, the map is optimized using both local and global bundle adjustment. In all threads, the Tukey re-weighting function [91] is used to ensure that the system is robust to outliers.

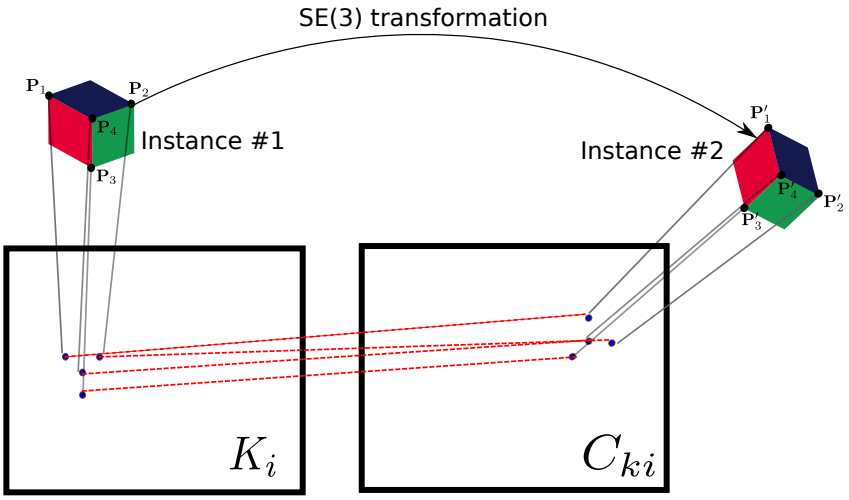

Figure 4.3: Registration of Instances. Every 3D point has a corresponding 2D point with an associated feature descriptor.This ensures a matching pair of keypoints also has a matching pair of 3D points in the world. Two instances of an object are registered if there exists a valid 3D to 3D rigid body transformation from the map points of one instance to that of the other.

4.3.2 Duplicate Object Detection

For every new keyframe C_n, the recognition thread attempts to find a set of existing keyframes $\{C|C \in C_1 \cdots C_{n-1}\}$ that is similar in appearance. For this purpose, a vocabulary tree of ORB descriptors [60] is used as a pre-filter. The list of keyframes returned by the vocabulary tree is pruned into a smaller set by discarding keyframes that have a non-trivial overlap on the scene with the current keyframe.

Feature matching is then performed between the current keyframe and the remaining candidate keyframes. The keyframe that has the highest number of matches with the current keyframe is chosen as the best candidate. The 3D landmarks corresponding to the feature matches (between the current and the best candidate keyframe) are then used to form putative 3D-3D correspondences. The Hamming distance between the median ORB descriptors of every putative

3D-3D correspondence is computed to filter out outliers. If there are insufficient correspondences, the recognition thread stops processing the current keyframe and awaits the next new keyframe. Otherwise, the correspondences form a set of landmarks between two candidate instances I_j and I_k, i.e. $\left\{M_{I_j} \leftrightarrow M_{I_k}\right\}$.

The closed-form solution of Horn *et al.* [87] is employed in a hypothesize-and-test RANSAC [54] framework to initialize the pose $E_{\overleftarrow{kj}}^{m}$ from the landmarks observed by the current keyframe C_n to the candidate keyframe C_i. Using the constraint from duplicate landmarks in Equation 4.2, we refine the pose $E_{\overleftarrow{kj}}^{m}$ by minimizing the re-projection error of landmarks $\mathbf{p}_{I_j}$ belonging to instance I_j (current) into the keyframe in which the landmarks in instance I_k (candidate) are observed:

$$\sum_{\mathbf{p}_{I_j} \in M_{I_j}} (\mathbf{z}_{ki} - \hat{\mathbf{z}}_{ki}(T_k, E_{\overleftarrow{kj}}^{m}, \mathbf{p}_{I_j})) \tag{4.3}$$

4.3.3 Active Search

Using the refined pose $E_{\overleftarrow{kj}}^{m}$, we attempt to obtain more duplicate landmarks between the two instances I_j and I_k. Landmarks observed by keyframes near the keyframe observing the landmarks in instance I_j are projected into the keyframes observing the landmarks in instance I_k to generate the predictions $\hat{\mathbf{z}}_{ki}$. We form a window around each prediction $\hat{\mathbf{z}}_{ki}$. For every existing measurement in the keyframe C_k that lies in the window, a ZMSSD (zero-mean sum squared difference) comparison is performed between the patch centered around the source measurement $\mathbf{z}_{ji}$ and the patch centered around the existing measurement $\mathbf{z}_{ki}$ in keyframe C_k. Similar to [105],the change in viewpoint is modelled by performing an affine warp A on the patch for the measurement $\mathbf{z}_{ji}$. If the best match $\mathbf{z}_{ji}^{\text{best}}$ found has a ZMSSD score lower than a certain threshold, sub-pixel refinement is performed on the best match. Finally, the landmark $\mathbf{p}_{I_k}$ is added to instance I_k and the landmark $\mathbf{p}_{I_j}$ corresponding to the best match $\mathbf{z}_{ji}^{\text{best}}$ is added to instance I_j.

4.3.4 Registration of Duplicates to Database

The pipeline is now ready to convert the duplicate landmarks into first-order entities in the map. If there are no existing objects in the database, the two sets of landmarks are registered in the database as instances I_1 and I_2 of the duplicate object O_1 (see Figure 4.3). For every subsequent successful search of duplicate instances I_j and I_k, the object database is updated as follows:

- Firstly, we need to determine if **both set of landmarks M_j and M_k belong to existing instances of a duplicate object**. This is done by re-projecting the landmarks belonging to the set M_j to the landmarks in the set M_k using the transformations that have already been registered in the object database. If the re-projection error of the best transformation is lower than a certain threshold, the landmarks in M_j and M_k are allocated to the two instances corresponding to this transformation.

- If the first case is not satisfied, the next criterion is to determine if **any single set of landmarks (M_j or M_k) belongs to an existing instance of a duplicate object**. In order to do this, for each set M_j and M_k, the number of landmarks that have already been registered to an existing instance in the database is measured. A new instance is only created if one set has no points belonging to an existing instance and the other set has a high percentage of points belonging to an existing instance.

- If none of the above two conditions are satisfied, **the landmarks do not belong to any existing instances in the database**. Thus, a new duplicate object O is initialised and its two instances from the two sets of landmarks.

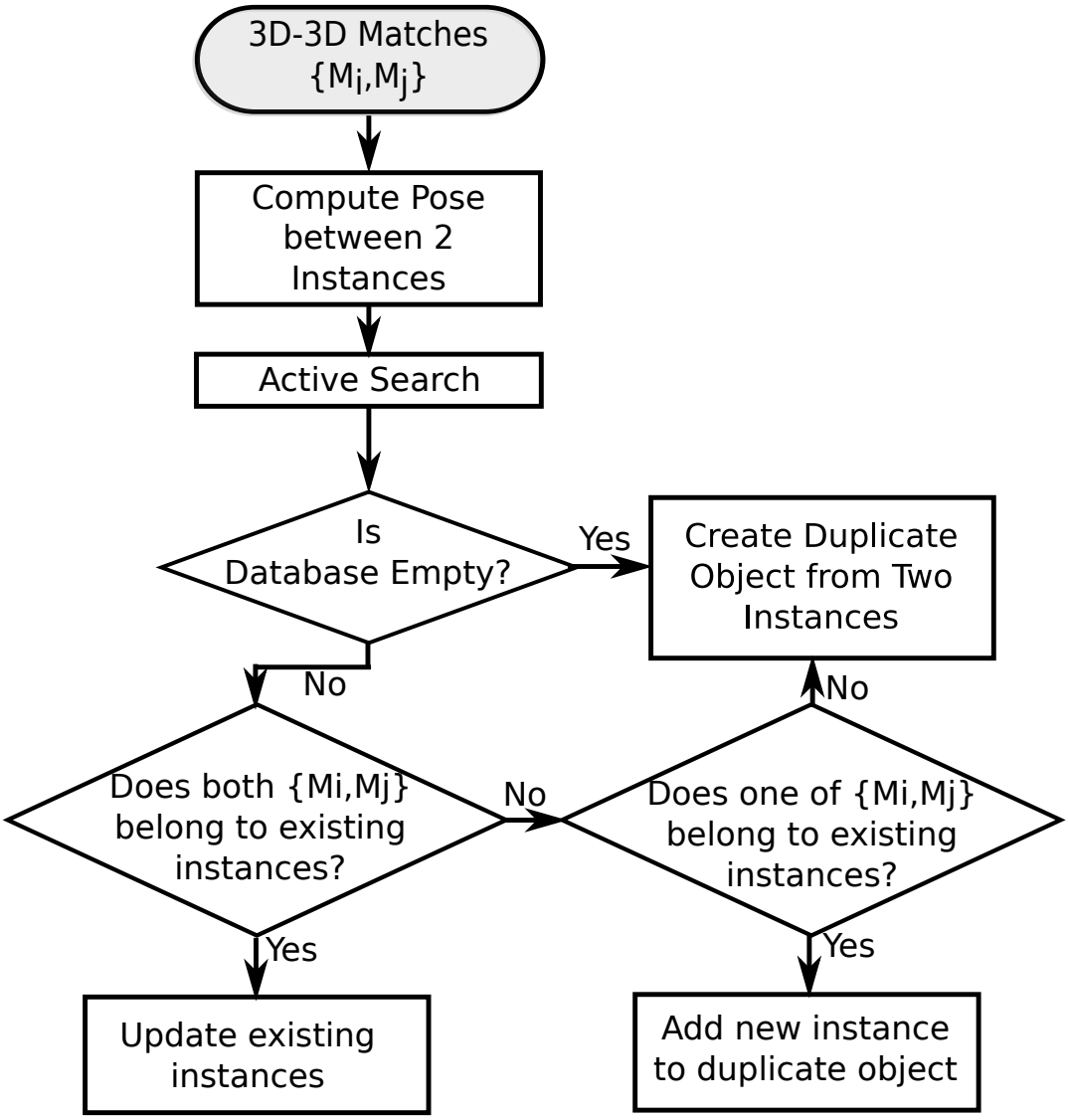

Figure 4.4: Flowchart of the recognition thread. Putative 3D-3D correspondences are used to compute a rigid body transformation, which are then used to either initialize or update the object database.

4.3.5 Optimization of Map with Objects

The map optimization thread is responsible for optimizing the camera poses and landmark positions in order to reduce error accumulation while the live camera explores the operating environment. The optimization is performed in a bundle adjustment framework where the re-projection errors of landmarks into cameras is minimized. In contrast to standard bundle adjustment frameworks, in MO-SLAM the constraints provided by duplicate objects are also incorporated into the bundle adjustment process. This establishes a tighter feedback loop where the knowledge of objects is used to refine structure and vice versa. More concretely, the following error function is minimised

$$\chi^2 = \sum_{\mathbf{z}_{i,k}} (\mathbf{z}_{i,k} - \hat{\mathbf{z}}(T_i, E^m_{\overleftarrow{j1}}, \mathbf{p}_k)) \tag{4.4}$$

over all measurements $\mathbf{z}_{i,k}$. In addition to the camera pose T_i and the landmark $\mathbf{p}_k$, the prediction $\hat{\mathbf{z}}$ of the landmark re-projection into camera C_i is now also a function of the rigid body transformations between duplicate landmarks $E^m_{\overleftarrow{j1}}\ \forall j$. Using the constraint in Equation 4.2, the re-projected landmark $\mathbf{y}$ in Equation 4.1 can now be formulated as

$$\mathbf{y}(T_i, E^m_{\overleftarrow{j1}}, \mathbf{p}_k) = \begin{cases} T_i \cdot E^m_{\overleftarrow{j1}} \cdot \mathbf{p}_k & \text{if } \mathbf{p}_k \equiv \text{ duplicate} \\ T_i \cdot \mathbf{p}_k & \text{otherwise} \end{cases} \tag{4.5}$$

Equation 4.5 demonstrates how to enforce duplicate landmarks as rigid first-order entities in the map through the transformation $E^m_{\overleftarrow{j1}}$. From here on, bundle adjustment proceeds normally. At every iteration, the update to the parameter vector $\mathbf{X} = [T_1, \cdots, T_n, E^1_{\overleftarrow{j1}}, \cdots, E^m_{\overleftarrow{j1}}, \mathbf{p}_1, \cdots, \mathbf{p}_k]^T$ is computed using the Levenberg-Marquardt algorithm:

$$\Delta\mathbf{X} = (J^T J + \Lambda I)^{-1} J^T \mathbf{e}, \tag{4.6}$$

where J is a Jacobian matrix with respect to the parameter vector $\mathbf{X}$, $\mathbf{e}$ is the error vector, and Λ is the regularization term. The Jacobian corresponding to

a measurement is computed by deriving the error vector with respect to the parameter vector $\mathbf{X}$ and can be obtained as a closed-form solution. Further, as the measurements are assumed to be independent, the standard Schur's complement trick [189] can be applied, where the landmark parameters $\mathbf{p}_1, \cdots, \mathbf{p}_k$ are marginalized so that the updates to the camera parameters can be computed first, followed by back-substitution to compute updates to the landmark parameters.

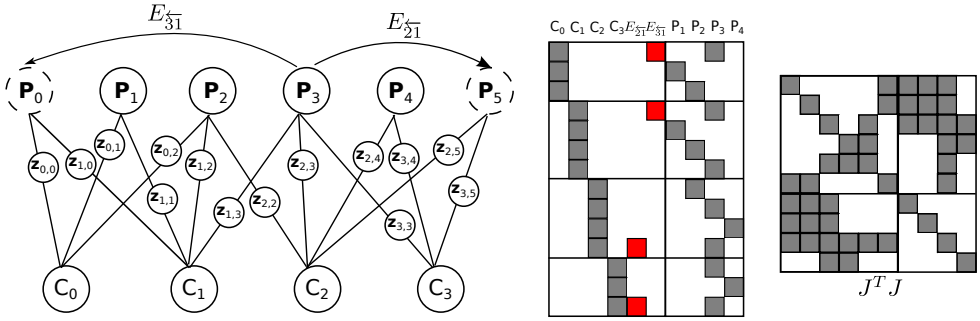

Figure 4.5: Map Optimization. **Left**: Graphical example of a simple map. **Center**: The Jacobian matrix J. **Right**: The Hessian matrix $J^T J$. Best viewed in colour.

Figure 4.5 provides an illustration of the Jacobian matrix J and the matrix $J^T J$ using the bundle adjustment formulation of MO-SLAM for a simple map with 4 cameras $C_0 - C_4$, 6 landmarks $\mathbf{p}_0 - \mathbf{p}_5$, and 14 measurements. There is one object with three instances, with each instance consisting of landmarks $\mathbf{p}_0$, $\mathbf{p}_3$ and $\mathbf{p}_5$. Further, landmark $\mathbf{p}_3$ belongs to the primary instance. The center of Figure 4.5 shows the structure of the Jacobian matrix, where shaded boxes show non-zero values. The boxes shaded in red are the additional constraints provided by duplicate landmarks. This results in the matrix $J^T J$ on the right of Figure 4.5, where the top left block matrix now has off-diagonal values as well.

4.4 Experimental Results

In this section, the performance of MO-SLAM is evaluated with experiments on synthetic and real data. For all the experiments, the same set of parameters are

used and the results are obtained from an Intel i7, 3.4GHz machine. Firstly, we compare the performance of our bundle adjustment formulation which incorporates constraints from duplicate landmarks with conventional bundle adjustment (see Section 4.4.1). Next, we evaluate the performance of MO-SLAM with and without duplicate object detection on a real image sequence in Section 4.4.2. Finally, we provide some qualitative results of our system in Section 4.4.3.

4.4.1 Synthetic Experiments

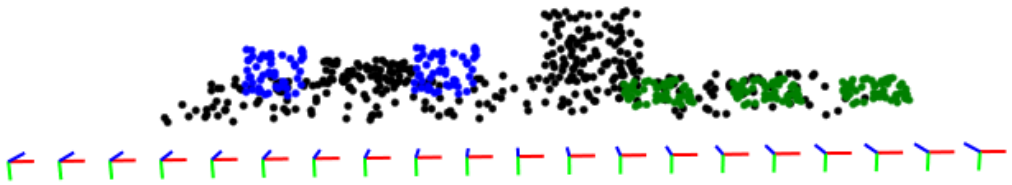

Figure 4.6: Front view of synthetic point cloud dataset. The two instances of object O_1 are shown in blue, and the three instances of object O_2 are shown in green. Best viewed in colour.

In this experiment, we generate a synthetic point cloud (see Figure 4.6) which consists of 20 keyframes and 610 landmarks. There are two duplicate object types in the dataset. Object O_1 (blue points) has two instances, with each instance consisting of 50 landmarks. Object O_2 (green points) has three instances, with each instance also consisting of 50 landmarks. We assume an image resolution of 640×480 pixels, focal length of $f = 600$ pixels, and principal point offsets of $u_0 = 320$ and $v_0 = 240$ pixels. We also assume Gaussian noise of $\delta \sim N(0, \sigma)$ is added to the image measurements.

We compare the performance between standard bundle adjustment (BA) and the bundle adjustment framework of MO-SLAM (with additional constraints from duplicate landmarks) against increasing Gaussian noise. More concretely, the root-mean-squared-errors (RMSE) were computed for translation (t_{rmse}), rotation

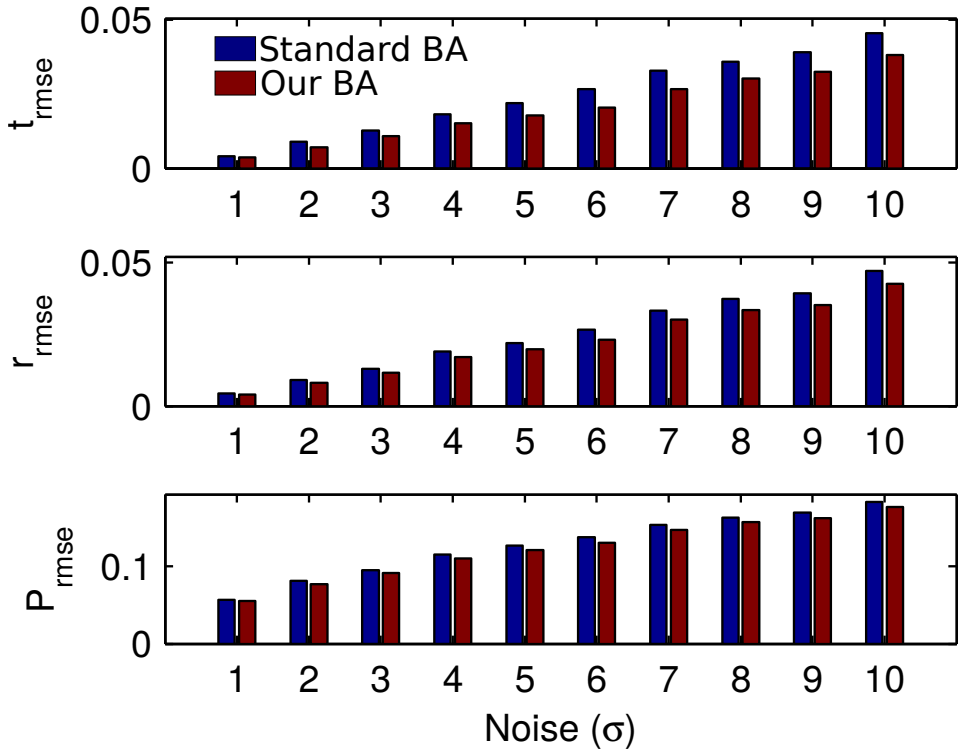

Figure 4.7: Results for synthetic point cloud experiment.

(r_{rmse}), and landmark positions (P_{rmse}) for both versions of BA. The result of this experiment is shown in Figure 4.7. Here, it can be seen that the performance of our BA degrades at a lower rate with increasing noise compared to standard BA. Hence, the additional constraints from duplicate landmarks help to improve the robustness of BA. In terms of timing, our BA is only slightly more expensive, taking an average of 125ms compared to the 118ms taken by standard BA. The increase in timing is due to the fact that there are 3 extra camera poses in our BA incorporating the rigid body transformations from the primary instance of each object to their duplicate instances.

4.4.2 Real Image Experiment

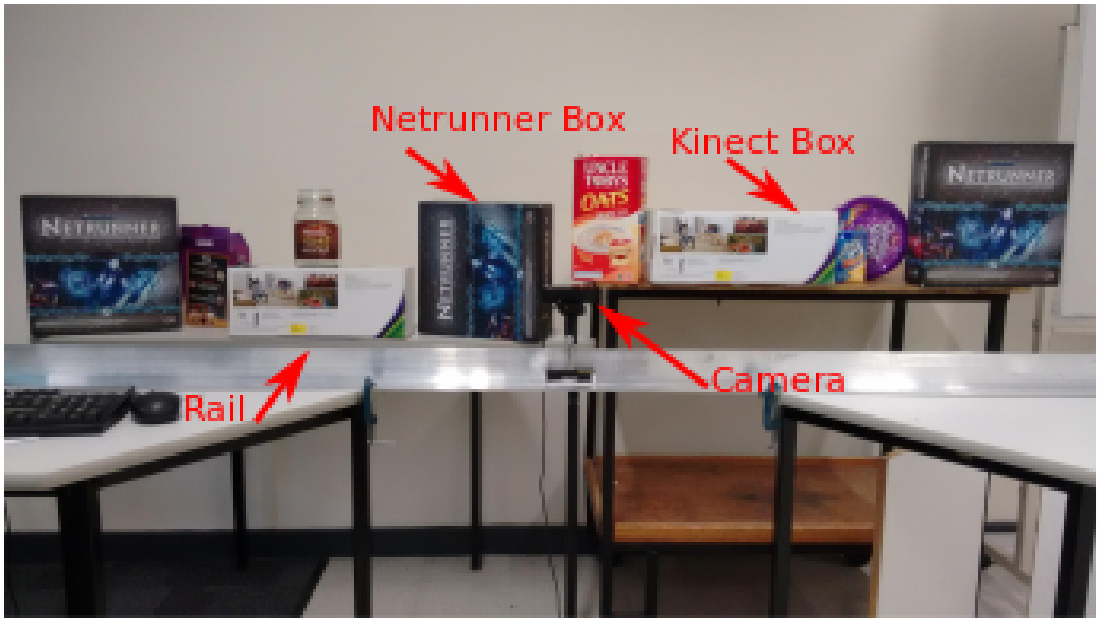

Figure 4.8: Setup for real data experiment. The camera is mounted on a rail to generate an image sequence with a trajectory along the x-axis.

The performance of MO-SLAM was further evaluated using a real image sequence. The experimental setup is shown in Figure 4.8. A Logitech C920 webcam was mounted onto a 2m steel rail that is rigidly clamped on a workbench. An image sequence with 1032 images is then captured by sliding the camera along the rail. The image sequence consists of two objects with duplicates: a Netrunner box with three instances and a Kinect box with two instances.

MO-SLAM was executed on the image sequence twice, once using standard BA at the back-end and once using our proposed BA method. For both runs, the system was initialized from the same two views and the recognition thread was enabled. Firstly, we compare the absolute poses of the keyframes obtained in both runs with the ground truth trajectory as shown on the left of Figure 4.10. The top figure shows that both bundle adjustment methods produce trajectories similar to the ground truth. However, if we zoom in closely as shown in the

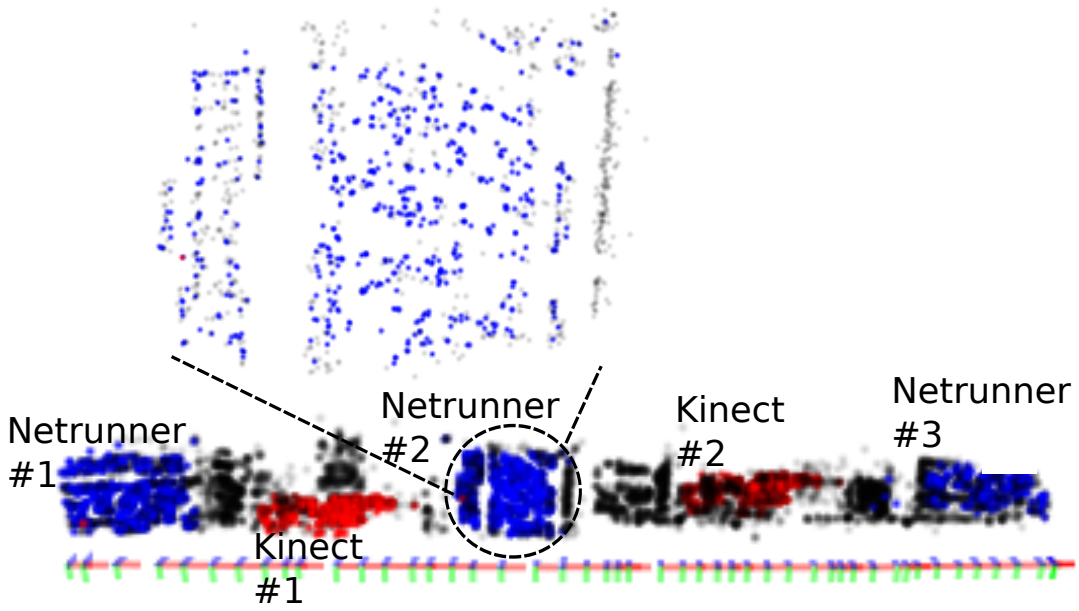

Figure 4.9: Map for real image experiment. A close-up view of the duplicate landmarks for the second instance of the Netrunner box is shown.

bottom figure we can see that our proposed BA method drifts at a lower rate compared to standard BA.

Next, we compare the scale drift of using MO-SLAM with standard BA and our proposed BA method. For this, we measured the pose of the camera at locations of 0.5, 1, 1.5, and 1.8m along the rail. We then compute the ratio of the translation norm at 1, 1.5 and 1.8m relative to the translation norm at 0.5m. This is then compared against the expected translation norm ratios to provide an estimate of scale drift, as shown on the right of Figure 4.10. Similar to the trajectory plot, we see that both standard BA and our proposed BA method exhibit an increasing drift with the length of the trajectory. However, scale drift increases at a lower rate for our BA method.

We also compare the accuracy of the landmarks found as duplicates using stan-

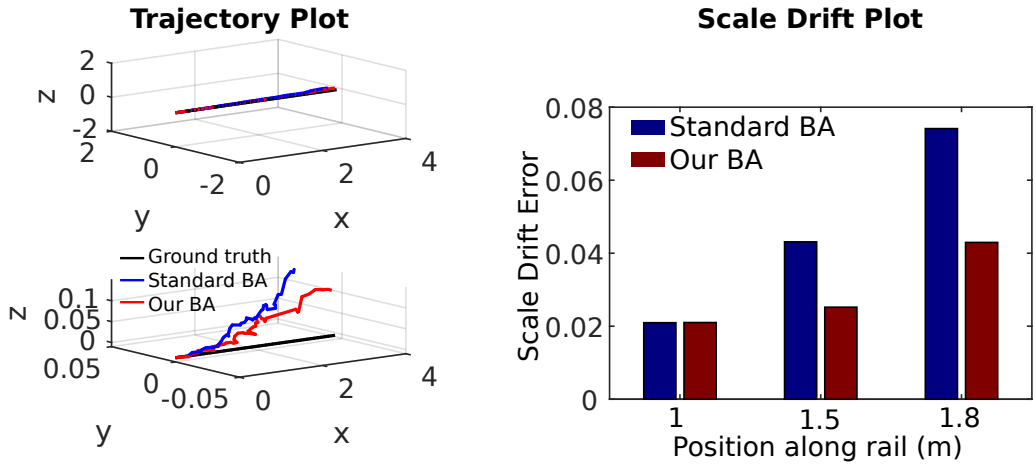

Figure 4.10: Comparison between standard BA and our BA for real image experiment. **Left**: Comparing camera trajectories with ground truth. A zoomed in version (in the y and z axis) is shown at the bottom. **Right**: Comparing the scale drift.

dard BA and our BA method. As only the front faces of the objects are seen in the image sequence, a plane was fitted on to the landmarks found as duplicates for every instance of every object found using RANSAC. In order to ensure fairness, the same error threshold is used for both methods to classify if a point is an inlier or an outlier. The point-to-plane distance error for every duplicate landmark for that instance is then computed. Standard BA produces a point-to-plane RMSE of 0.0964 and 0.0141 for the Netrunner box and the Kinect box respectively, whereas our BA produces RMSE values of 0.0230 and 0.0111. The error distribution for both methods are shown in Figure 4.11. Here, it can be seen that both methods produce landmarks that have reasonably small values. However, by zooming in on the error distributions it can be observed that the error distributions for the landmarks of our proposed BA method (bottom row of Figure 4.11) have a smaller tail. This is because our proposed BA method enforces these landmarks as first-order entities in the map. Hence, it refines duplicate landmarks with additional constraints, and removes any landmarks that do not obey the data association (the rigid-body transformation between the two duplicate instances) formed by this additional constraint.

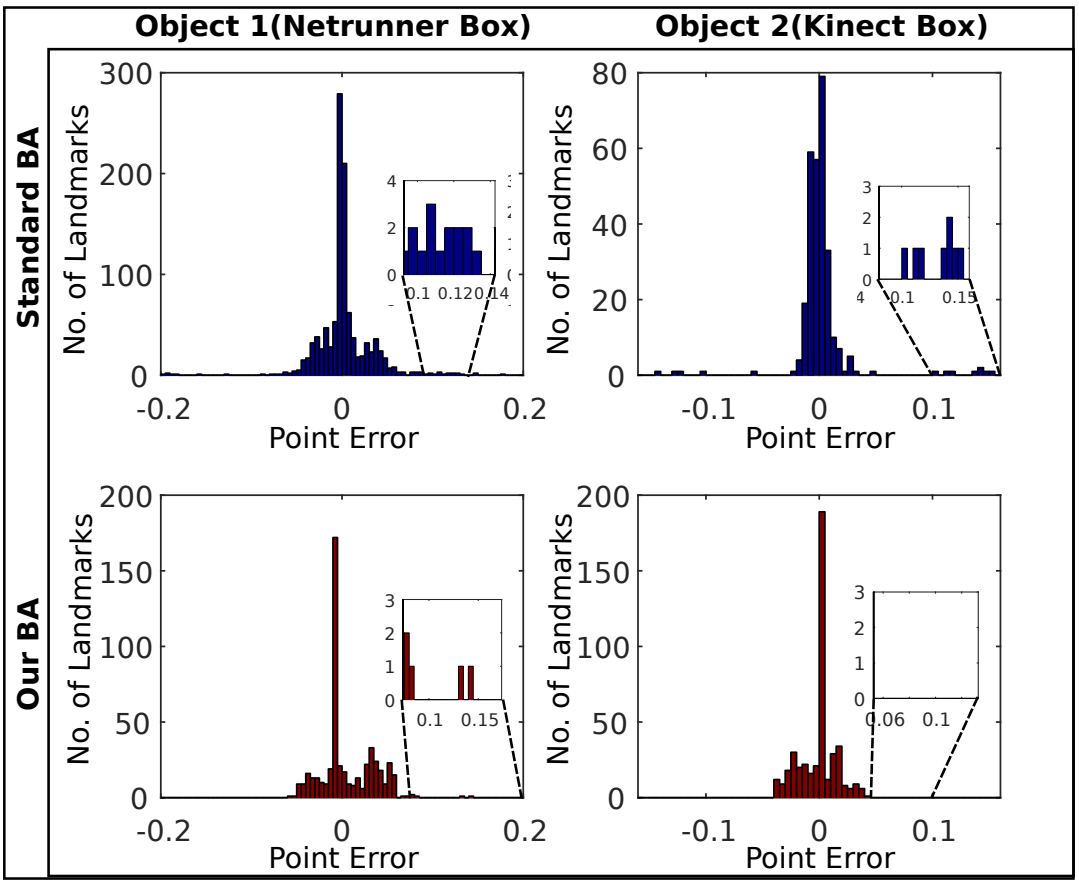

Figure 4.11: Point-to-plane error distribution. **Top**: Standard BA results. **Bottom**: Our BA results. **Left**: Error distribution for object 1 in the scene. **Right**: Error distribution for object 2 in the scene. Notice how our proposed BA has a smaller tail compared to Standard BA.

Figure 4.12 provides an illustration of the computational cost of the recognition thread. Here, it can be seen that most of the time the recognition thread incurs a trivial computational cost. However, if duplicate instances of an object are found, the recognition thread requires approximately 200-300ms to process the keyframe. Since the recognition thread runs in parallel to the other threads while also maintaining an queue for new key-frames, objects are generally added seamlessly on to the map. Finally, the map of the scene for this experiment is shown in Figure 4.9, where the coordinate frames denote the keyframes (red=x-axis, green=y-axis, and blue=z-axis). Normal landmarks are shown as black

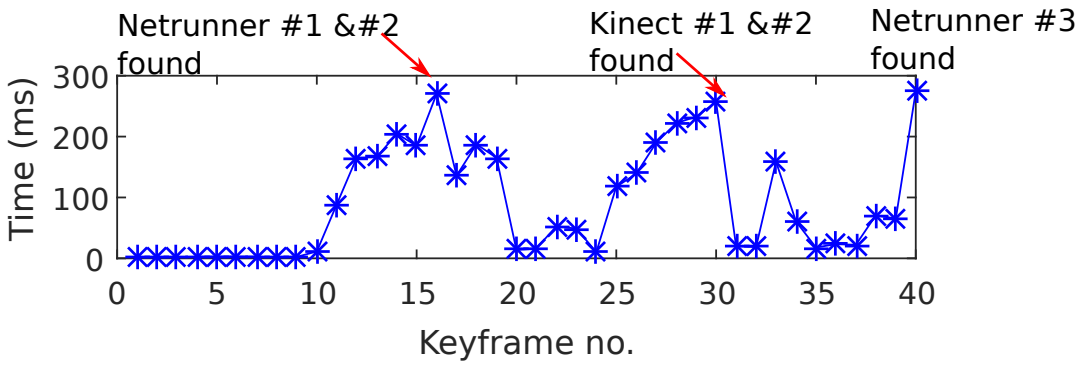

Figure 4.12: Timing results from recognition thread. Increase in computational cost occurs when duplicate instances of an object are found.

points whereas duplicate landmarks are coloured.

4.4.3 Qualitative Results and Limitations

Some qualitative results of MO-SLAM working in a typical lab environment are shown in Figure 4.13. A video that demonstrates the real-time performance(∼30Hz) of MO-SLAM can be found under https://youtu.be/UDrr2xGReLM.

The current limitations of our system are:

- As MO-SLAM requires duplicate instances of an object for recognition, if part of an object is observed in one instance and occluded in the other, those points will not be added to the object.
- In addition to this if there are not any duplicate objects in the scene, MO-SLAM would behave similar to a standard SLAM system and would not have any notion of objects. In the future work subsection I propose two alternative strategies that could be explored to address this problem.
- If two man-made objects always appear together in such way that in every

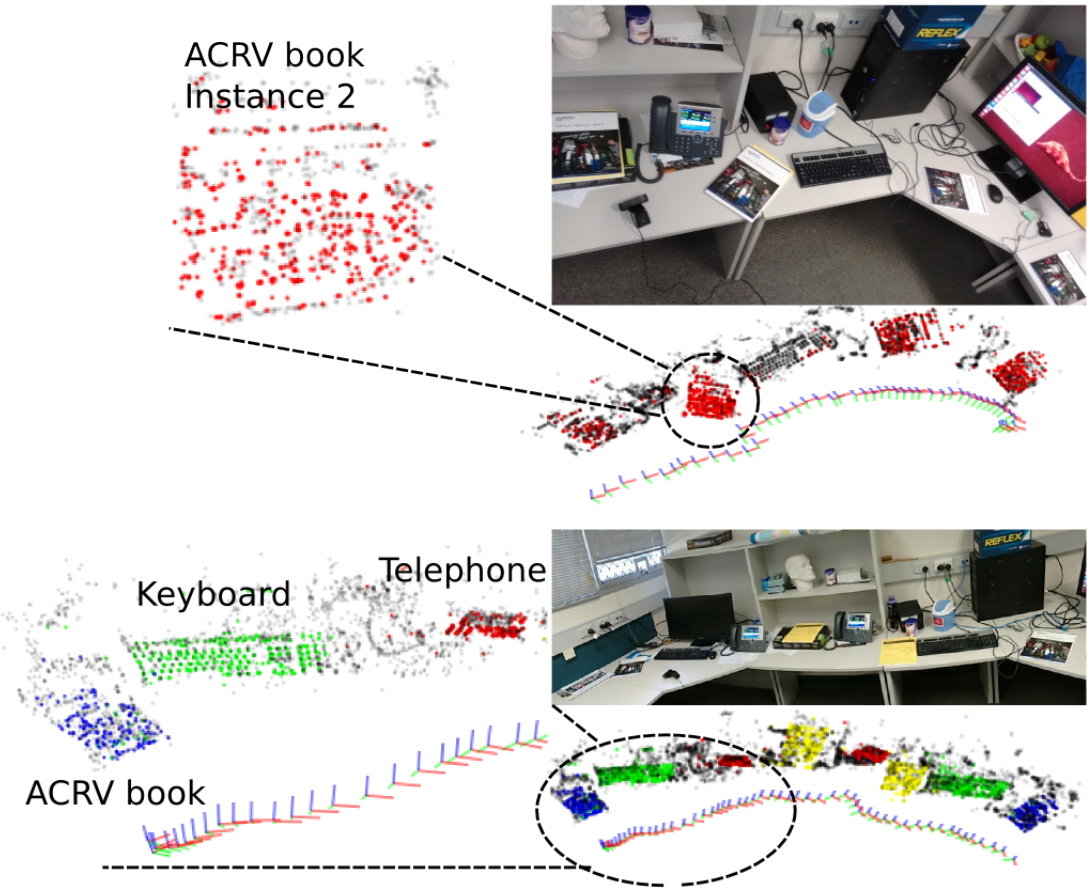

Figure 4.13: Qualitative results The top part of figure shows the reconstruction and recognition results from MO-SLAM for a scene with four instances of a single object, in this case an "ACRV book" (highlighted as red points). The bottom figure the results of MO-SLAM for a scene with four objects (ACRV book, keyboard, telephone, lab manual), where there are two instances of every object

instance these objects are observed they exhibit the same relative positioning and orientation, these two separate objects will be recognised as one object. This can be explained best using an example; If every instance of a photocopier in an office environment has a fire extinguisher adjacent to it where the relative position and orientation are identical across all instances, MO-SLAM would recognise the photocopier and the fire extinguisher as one single object. A large database of objects with associated labels or a CNN based object classifier can play an important role in this scenario and aid

MO-SLAM to distinguish between the two objects.

4.5 Future Directions

Figure 4.14: Key-point Classification using a Convolutional Neural Network All the key-points belonging to the same object are given a one colour. Background key-points are shown in white.

I envision two straightforward extensions of this work. Firstly, a CNN can be incorporated into the recognition thread to perform key-point classification as shown in Figure 4.14. This particular CNN was trained on images patches centered around FAST [164] key points and was constructed as a proof of concept to demonstrate a neural network model is able to classify key points. This allows the framework to perform supervised object recognition in addition to unsupervised object discovery using duplicate objects. Currently the bundle adjustment framework of MO-SLAM only uses multi-view constraints and the additional constraints from duplicate objects therefore if a CNN was to be connected, this needs to be extended to leverage the information obtained from the CNN.

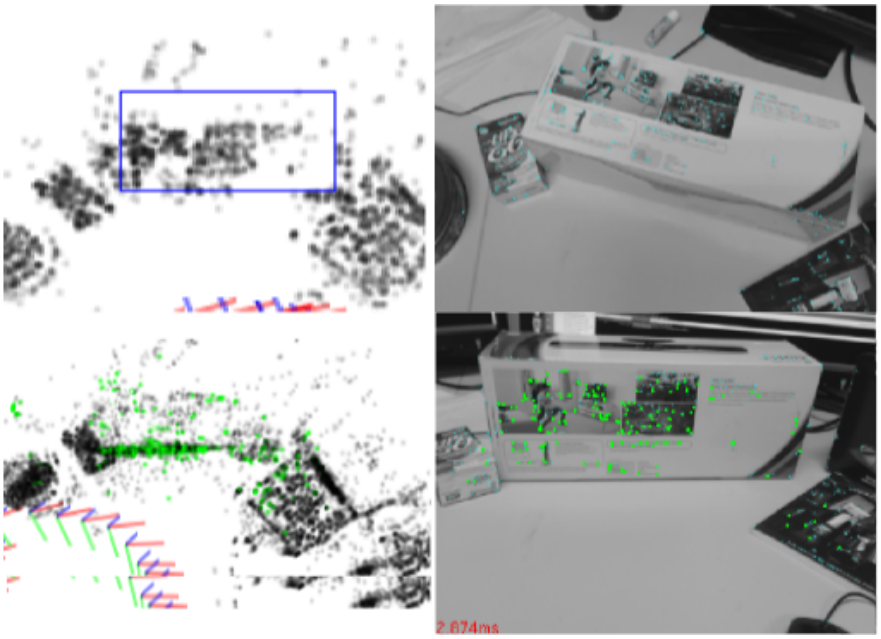

Figure 4.15: Learning objects through change detection in a dynamic environments The top figure shows the state of the map when the Kinect Box is first observed. When the same area is observed at a later time, we can see that the Kinect Box has undergone a rotation around the x-axis, The previous map points have now been erased from the map as they are no longer part of the scene.

The current semantic SLAM framework learns object through duplicate detection. This framework can also be extended to learn objects if they have undergone a change since their last seen state. The same core principles of online object discovery through duplicates that were discussed earlier can be applied to this problem as well. If we observe that an object has moved, it is possible to treat it as a pseudo duplicate instance of the object until that object is no longer observed in the location where it was first seen. Qualitatively I have made progress on this area as shown in Figure 4.15 where a set of key points that was observed at a particular point in time was observed later at a different location. This allows the framework to create a duplicate object since the two set of landmarks are unique. However, unlike in the previous scenarios the system is able to correctly identify the landmarks that belong to the first instance of the object no longer re-project on to the live view of the camera. This in turn allows the framework to remove those landmarks from the map.

4.6 Summary

In this chapter I have presented MO-SLAM, a real-time, semantic visual SLAM system which does not require a pre-generated database of objects. Instead, the objects are discovered during run-time using duplicate instances of objects that are present in the scene. Furthermore, these duplicate objects provide additional constraints for optimizing the map, which not only improves the accuracy of the landmarks and the camera poses, but also improves the output of the recognizer by removing erroneous map points that are wrongly classified as belonging to an object. MO-SLAM was evaluated extensively using both synthetic and real data experiments, in order to verify its effectiveness and real-time capabilities.

The final section of the chapter suggested two potential extensions of this work which focused on additional ways of finding objects. Firstly, through the use of a CNN or alternatively in a dynamic environment, using the knowledge of moved landmarks to create new objects.

MO-SLAM plays an important role in thesis and demonstrates that the classical structure and motion pipelines can be improved by purely using geometry based techniques in a day and age that is heavily dominated by deep learning.

CHAPTER 5

Joint Prediction of Structural Information from RGB Images using CNNs

5.1 Introduction

In the previous chapter, the scene reconstruction and the subsequent refinement problem was tackled purely from a geometric point of view using a SLAM system. In this chapter, a slightly different approach is taken where a single neural network is used to predict depths, surface normals and surface curvature from a colour image. Since neural network based depth prediction features heavily throughout the remainder of the thesis, it was important to develop an intuition about CNNs that predict structural quantities and this work takes a step in that direction. Furthermore, one of the luxuries of training a network to predict structural quantities as opposed to semantic labels is having the advantage of applying the concepts learnt from conventional geometry into the training regime, and one of the aims of this chapter is to exploit the close relationship that depths share with surface normals and curvature.

While predicting depth directly from a colour image using learning based ap-

proaches have gained popularity recently due to the resurgence of CNNs, a relatively less explored problem is predicting extremely related structural quantities in the form depths, surface normals and surface curvature. The closest work to this line of research is the framework by Eigen *et al.* [44] in which he explored the possibility of predicting depths, surface normals and semantic labels. The main focus of the work presented in this chapter is to showcase that predicting closely related structural quantities as shown in Figure 5.1 while maintaining a fixed model capacity leads to superior performance. In particular, this work demonstrates predicting surface curvature in contrast to semantic labels along side surface normals and depths has a greater influence on the latter two tasks.

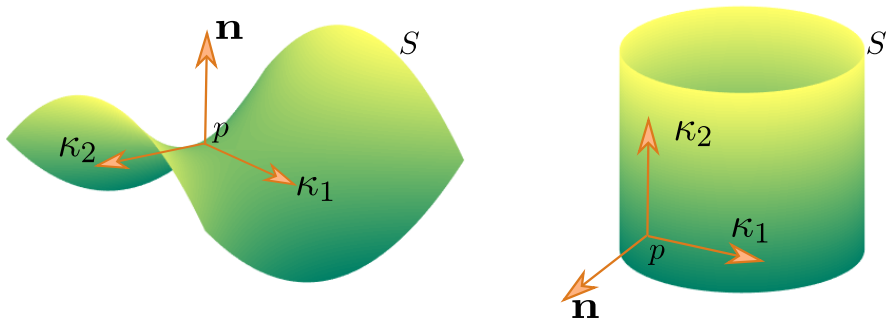

Figure 5.1: Given a surface S, $\mathbf{n}$ is a unit normal vector to the surface at point p, and κ_1 and κ_2 denote the two principal curvatures. The work presented in this chapter aims to predict the depth (which can then be used to reconstruct the surface S given the camera parameters), the surface normal $\mathbf{n}$ and the two principal curvatures κ_1 and κ_2 for each pixel given a colour image.

Surface Curvature is an important geometric surface feature, that indicates the rate at which the direction of the normals change on the surface at any particular point. It has been shown to be particularly useful for the task of segmentation on range image and 3D data [9, 42, 4, 118]. A key challenge in accurately estimating surface curvature is its sensitivity to noise in the input data, as it is a second order surface derivative, it is affected quadratically by noise. Previous works have shown that neural networks can be used to provide accurate geometric estimates

from just a single RGB image [44, 122, 171, 114], including estimating depth and normals. In this work, we extend our network to estimate principal surface curvatures as well as depth and normals and demonstrate that we can accurately perform this task.

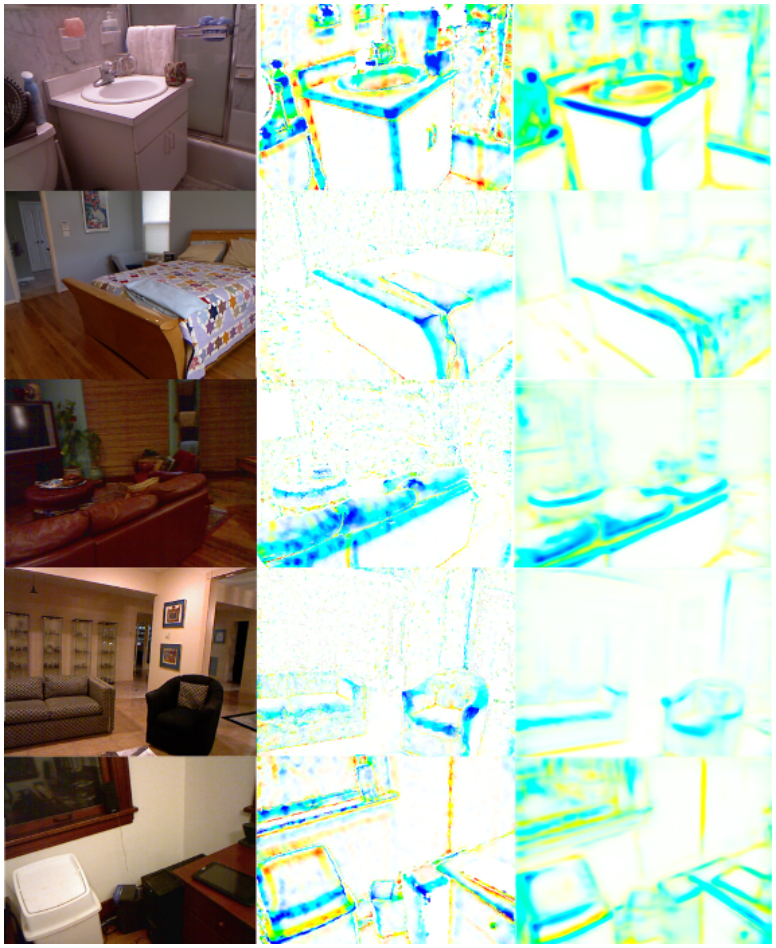

Figure 5.2: A selection of curvature predictions made by our system. The left column shows the corresponding RGB image from the NYUv2 test dataset which was used as the only source of information to estimate curvature. The middle column shows the ground truth curvature computed using the depth data and the right image shows the prediction of our network. The Positive curvatures are shown in blue, Negatives in red, Saddles in green and Planes in white.

Contrary to the popular belief that hand-engineered features are inferior compared to learnt features, we argue that well designed features combined with ma-

chine learnt representations provide improved performance. It should be stressed that the features designed are not hand calculated, but rather predicted by the network itself as part of the inference pipeline. More concretely, the network is *informed* that in order to accurately estimate a single quantity such as depth, normals or curvature the network should learn an internal representation of the other two quantities. This is demonstrated by estimating surface curvature, surface normals and depth in a multi task learning framework which gives us superior results compared to training them as individual tasks. A two-stage learning process is employed where coarse level predictions of all three quantities are used as feature maps for the finer layers.

Our work is similar to [44] in that sense, as Eigen *et al.* also estimated three quantities (depth, surface normals and semantic labels) using a single network. However, there are two main differences between our approach and theirs. Firstly, the three quantities we estimate are more tightly coupled at a primitive level whereas the semantic labels, although clearly related, should be considered a higher order quantity. Secondly, the knowledge of semantic labels are not explicitly made available to the depths and surface normal prediction layers in [44], whereas in our approach all three (depths,normals,curvature) coarse level predictions are passed on as additional feature maps to the second stage of the prediction pipeline. As it can be seen in the results section (5.6), passing the coarse level estimates of a viewpoint invariant quantity in the form of surface curvature has a positive influence on the other two tasks. The predictions generated from this framework can be applied to robotic applications that revolve around segmentation tasks such as the Amazon Picking Challenge.

5.1.1 Contributions

The main contributions of this work are as follows:

- A novel technique to estimate surface curvature of objects using purely RGB images in a joint neural network framework which predicts depth, surface normals and curvature.
- Demonstrate that joint training can improve the accuracy of all three tasks while keeping the model capacity fixed.

Since the publication corresponding to this chapter [39] has two authors with equal contribution (myself and Andrew Spek), the relative contribution I have personally made based on mutual agreement is shown below:

- Conception of Idea (50%)
- Network architecture design (90%)
- Loss/Objective function design (60%)
- Coding the network (99%)
- Training data creation (30%)
- Testing and evaluation (50%)

The main body of the chapter is largely based on the contents of the publication [39] and is expected to appear in a similar form in Andrew's thesis.

5.2 Related Work

Since prior work on depth prediction was discussed extensively in chapter 2 under section 2.2.2, this section summarises the work related to the other two tasks (surface normal and curvature).

Surface Normal Prediction

A surface normal is a vector that is perpendicular to a point on a surface. In computer vision and robotics, surface normals are largely used for segmentation and plane-fitting. Generally, surface normals are computed (e.g using least square approaches [80, 131]) from a range image. Since a representation of the surface itself can be successfully predicted as a 2D depth image, a natural extension would be to examine the viability of inferring surface normals from colour images.

Data driven single image surface normal estimation was first tackled by Fouhey et al. in [57]. They used a SVM based detector followed by an iterative optimization scheme to extract geometrically informative primitives. Ladicky *et al.* proposed to use image cues of pixel-wise and segment based methods to generate a feature representation that can estimate surface normals in a boosting framework [113]. A ConvNet approach to estimating surface normals in global and local scales while incorporating numerous constraints such as room layout and edge labels was taken by Wang *et al.* [194]. Recently, Bansal *et al.* [5] showed that by combining hierarchy of features from different levels of activations in a skip-network architecture that you could generate much finer predictions for surface normals achieving state of the art results.

Surface Curvature Estimation

Surface curvature estimation is a well explored topic in robotics and computer vision. It has been shown to be useful for object segmentation [9, 42, 118, 4] in depth scans and RGB-D imagery. There are several popular approaches to estimating the surface curvature. One technique is to simply twice-differentiate the surface [9, 4], but this can lead to a high sensitivity to noise in the data and generally requires removal or rejection of surface outliers. Another technique is to estimate the surface curvature from a locally connected surface mesh based on the change in adjacent facet normal angles [118, 166, 70]. This method is predominantly used for computer graphics and low-noise data as it operates on a

small neighbourhood of facets. Yet another technique is to use locally fit surface quadrics and directly extract the principal curvatures from their parametrisation [42, 137], which has been shown to be robust to noisy data and fast enough to be computed in real-time [178]. In this work, we use the approach proposed in [178], to compute surface curvature and surface normals from the training data sourced from the NYUv2 dataset [140] as they have shown it performs well on range image data of the type present in the dataset.

Learning Multiple Tasks

In one of the earliest works in this area Caruana *et al.* showed in [18] that by learning related tasks in parallel, the performance of all tasks could be improved, which is consistent with our findings. Multiple tasks were learned in the form of material classification and defect detection in railway fasteners in [65] where they used Deep CNN based multi task learning for railway track inspection. They were able to show the adaptability of the multi task learning platform by using different training batch sizes (due to availability of data). In our case, all three tasks were trained with the same batch size as training data for the derived quantities (normals and curvature) were computed from depth. Multi task learning algorithms were also used to perform head pose estimation [203], web search ranking [21], face verification [195] etc. Li *et al.* in their work *Learning Without Forgetting* [120] demonstrated that in the presence of a model trained on one task, it can be fine-tuned to perform better on a new task while not hindering the performance of the previous task by only using training data of the new task. However, as we have access to training data for all three tasks we train the prediction stacks jointly in order to achieve superior performance compared to fine-tuning.

5.3 Model Architecture

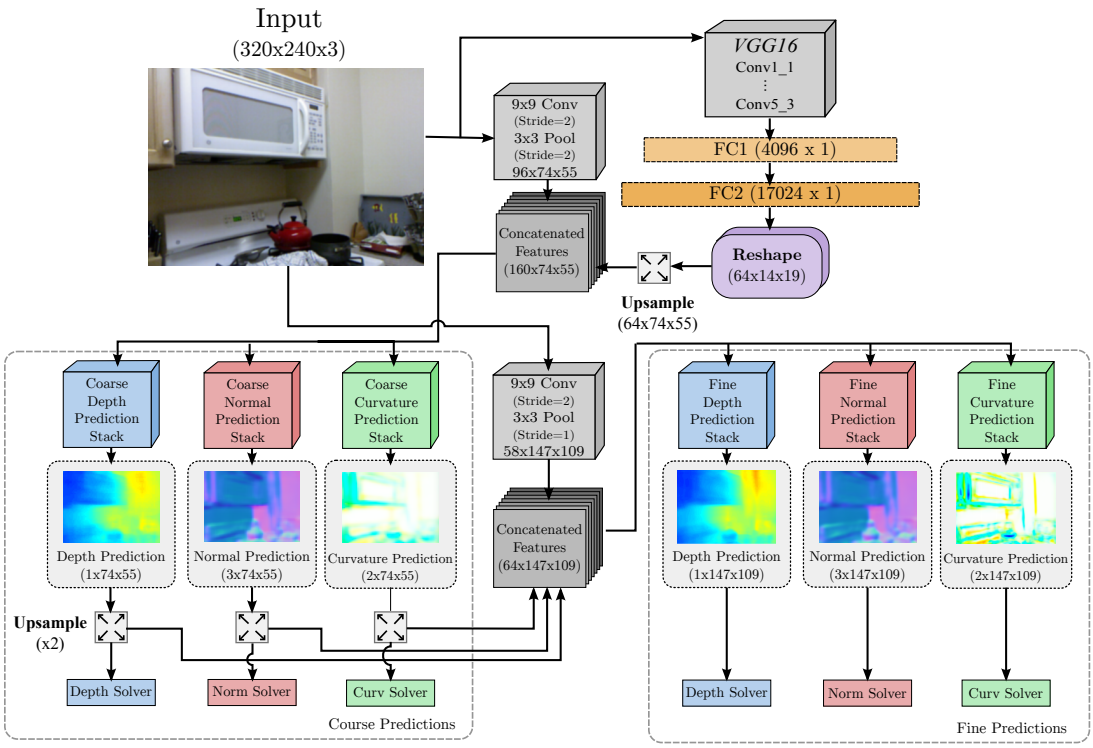

Figure 5.3: Visual Representation of Model Architecture

The functionalities of the different components of the model can be divided into 3 main categories: feature extraction, the incorporation of global context, and prediction . The first set of convolutional filters labelled VGG16 [174] in Figure 5.3 facilitate generic *image feature extraction* and are pre-trained on the ImageNet [167] dataset. This is followed by 2 fully connected layers (FC) where each activation is connected to one another allowing the network to have the entire image in its field of view. The 1-dimensional output of the second fully connected layer is then reshaped into a stack of 2D activations. These activations are concatenated with raw image features and passed on to three separate stacks of convolutional filters corresponding to the three tasks. Each stack is followed by a dedicated solver responsible for computing the error between the coarse level predictions (depths,normals and curvature) and the corresponding ground truth.

The coarse predictions are once again combined with raw image features, however these features are extracted by performing pooling with a smaller strider on the activations of the convolution layers. This allows the network to regain some of the finer details lost during the sub sampling operations. The concatenated features are then passed on to the fine level prediction stacks culminating with an additional set of 3 solvers.

It is worth mentioning that all the convolutional layers in the coarse and fine level prediction stacks perform 5x5 convolutions with a stride of 1 and a pad of 2. Therefore, the input resolution at the beginning of the feature stack is preserved at the end of the feature stack. There is an explicit up-sampling layer which up samples the coarse level prediction from 74x55 resolution to 147x109 and this is maintained throughout the final convolutional stack. Although the overall architecture is as explained above, in order to make sure the model capacity is kept constant and the contribution of each new task is indeed improving the performance of the previous tasks, several changes are made during training as explained in section 5.5.3 .

5.4 Tasks

5.4.1 Terminology

Throughout the remainder of the chapter, the following notation is used to refer to the predicted and ground truth quantities for depths, surface normals and surface curvature.

- Depths : Ground truth depth map is represented as D^* (each pixel contains a depth value in meters, missing depth values are represented with a zero). Predicted depth map is given by D.

- Normals : Ground truth surface normal map (N^*) contains a unit normal vector ($\mathbf{n}^*$) at each of the pixels with a valid depth value. The corresponding predicted surface normal map (N) contains a unit normal vector ($\mathbf{n}$) at all pixel locations.
- Curvatures: Ground truth surface curvature map (C^*) contains values of principal curvatures κ_1^* and κ_2^* for each pixel in m^{-1} denoting the inverse of the radius of curvature. Predicted surface curvature map is given by (C)

5.4.2 Depth

Similar to the previous approaches [44, 45] depth is estimated at two scales. While experiments were conducted to predict depth at more scales, the increase in performance was minimal compared to the increase in model capacity. The network is trained to predict the natural logarithm of depth using a fully supervised learning regime. Supervision is provided by employing an Euclidean loss term where the coarse and fine predictions of the neural network are compared against the logarithm of the ground truth depth. Two additional loss functions are employed similar to [44], in order to enforce the depth errors in a local region to follow a consistent structure. The full loss criterion is shown below :

$$L(D, D^*) = \frac{1}{n}\sum_{i=1}^{n} d_i^2 - \frac{1}{2n^2}\left(\sum_{i=1}^{n} d_i\right)^2 + \frac{1}{n}\sum_{i=1}^{n}\left((\nabla_x d_i)^2 + (\nabla_y d_i)^2\right) \tag{5.1}$$

where d_i is the ith pixel difference between the predicted log depth $ln(D)$ and ground truth log depth $ln(D^*)$ for the valid pixels n (pixels that contain non-zero depth values in the raw depth data). The second term computes a scale invariant error which measures the relationship between the ground truth and the predicted depth map irrespective of the absolute global scale [45]. The third loss term enforces a constraint on the local structure of the predicted depth map.

$\nabla_x d_i$ is the horizontal image gradient of the difference and $\nabla_y d_i$ is it's vertical counterpart.

5.4.3 Surface Normals

The ground truth normals are computed using different techniques in the literature. In this work, the normals are computed by first fitting a quadric patch to a set of nearby points in the point cloud. This gives a more accurate representation of the surface compared to merely fitting planar regions, while not adding an extra time complexity as the normals are computed as part of the curvature computation pipeline [178].

In order to train the the surface normal prediction layers, a pixel wise Euclidean loss is used along the three channels corresponding to the three unit vectors i, j, k. This loss term is then coupled with the difference in angle between the predicted normal and the ground truth.

$$L(N, N^*) = -\frac{1}{n}\sum_{i=1}^{n} \mathbf{n_i} \cdot \mathbf{n_i}^* + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{3}(nc_j - nc_j^*)^2, \tag{5.2}$$

where $\mathbf{n}$ and $\mathbf{n}^*$ are the predicted and ground truth unit normal maps respectively, $nc_j \in \mathbf{n}$ and $nc_j^* \in \mathbf{n}^*$ are the three components of each of the normals. n denote the valid pixels.

While the two terms effectively aim to achieve the same goal, using a combination not only provided faster convergence compared to using a single loss term but also improved the accuracy of the predictions.

5.4.4 Surface Curvature

The method described in [178] is used to compute an estimate of the principal surface curvatures, which is computed from a locally fit parabolic quadric. We use a sparsely sampled circular patch of radius 18 pixels, to fit a quadric at each point and extract the local principal curvature values. The principal curvatures κ_1, κ_2 are limited to the range $\{-100, 100\}$ in order to avoid the estimation of implausible curvatures, effectively limiting the minimum detectable radius of curvature to be 1cm aligning with the precision of the system [103] at the distances present in the training data. This provides a dense estimate of curvature for every point (640x480), which we then bicubicly downsample to 120x160 to generate the LMDBs that can be used in the training of our network. We attempt to estimate principal curvatures directly as opposed to Gaussian or mean curvature, as we found principal curvatures to provide improved performance during training.

We employed a Euclidean loss criterion with depth based weighting to predict surface curvature. Due to the inherent sensor noise, the computed principal curvatures which are used as the ground truth tend to have a large uncertainty beyond a certain distance threshold. To prevent the network from learning these rather uncertain values the following loss function is used.

$$L(C, C*) = \frac{1}{n} \sum_{i=1}^{n} \frac{(\kappa_{1i} - \kappa_{1i}^*)^2 + (\kappa_{2i} - \kappa_{2i}^*)^2}{(1 + D_i^*)^2} \tag{5.3}$$

where κ_{1i} and κ_{2i} are the predicted principal curvatures and κ_{1i}^* and κ_{2i}^* are their corresponding ground truth values while D_i^* represents the depth in meters for the i^{th} pixel.

5.5 Training

The raw depth data distribution given by the NYUv2 dataset [140] is used for training based on the official train and test scene split (that is 249 training scenes and 219 test scenes).

5.5.1 Data Generation

Training dataset is augmented by performing flips, translations, rotations and variations on the colour channels. The same transformation is applied to the RGB input, ground truth depth, surface curvature and surface normals in order to obtain consistent training data. Unlike some notable previous approaches [44], we use the raw depths directly without any post processing to fill holes or smooth surfaces. Raw depths are also used to calculate the surface normals and surface curvature providing a stronger link between the three ground truth sources.

As described in Section 5.4.4 we use the method described in [178] to produce training data for surface normals and surface curvature. Their approach was specifically targeted for noisy data such as that from a Microsoft Kinect and produces good estimates for both surface normals and principal curvatures. The network was conditioned not to predict values of vastly different magnitudes by scaling the ground truth curvature values by a factor of 0.1 producing a similar range of values to that of the depths. This conditioning resulted in qualitatively and quantitatively superior predictions. The scaling was reversed when the final predictions were made for both principal curvatures κ_1 and κ_2 by multiplying each value by 10.

5.5.2 Hyperparameters and Weight Initialisation

We use Nesterovs accelerated gradient [141] as the optimizer with a base learning rate of 0.1, a momentum of 0.95 and train for 50 epochs using an NVIDIA GeForce GTX 1080. Weights of the convolutional layers corresponding to feature extraction were initialized using the VGG model [174] pretrained on ILSVRC [167] image data. We also experimented with initializing the feature extraction layers with the VGG weights of [44] and found no change in performance. All the convolutional layers corresponding to depth, normals, curvature estimation and the fully connected layers were randomly initialized using the MSRA weight initialization scheme [78] which converged much faster compared to initializing the filters from a Gaussian distribution with zero mean and 0.01 standard deviation. Whenever the training loss plateaued (approximately every 10 epochs) we halved the learning rate and continued training. Caffe [97] was used as the learning framework and all the experiments were carried out using a mini batch size of 16.

5.5.3 Training Separate Models With Equal Model Capacity

We train several models with equivalent model capacity to estimate quantities both separately and jointly. We do this to demonstrate that the improved estimates for normals and depths are not the result of increased model capacity, but more likely due to the inclusion of derived features as tasks to the network. Explicitly we train 5 models, depth only **(D)**, normals only **(N)**, curvature only **(C)**, depth+normals **(D+N)**, depth+normals+curvature **(D+N+C)**, all while maintaining a constant model capacity for each task. In a jointly trained model, the output corresponding to a particular task is indicated in subscript form. For example Ours$\mathbf{(D+N+C)}_{depths}$ denote the depth performance of the network that was jointly trained on all three tasks and Ours$\mathbf{(D+N)}_{normals}$ indicate the performance of surface normals produced by the network that was jointly trained to

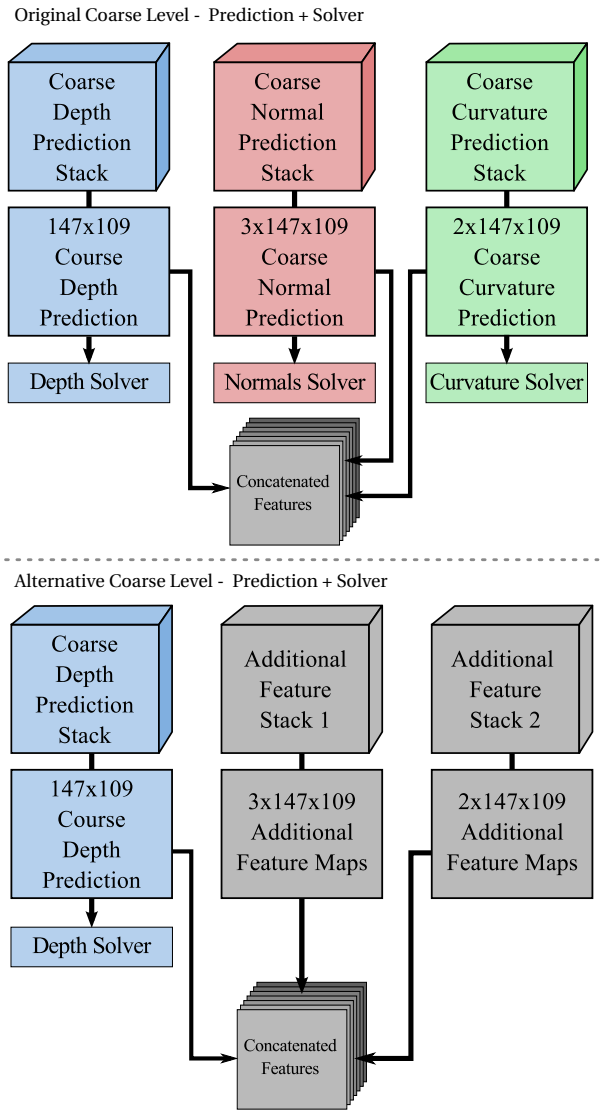

Figure 5.4: A closer look at the coarse level of the architecture for different tasks. **Top:** When all three tasks are trained jointly, there is a solver at the end of the coarse level feature stacks for all three tasks and the coarse feature maps are passed on to final level after being concatenated together. **Bottom:** When only a single task is trained(in this case depth) there is a single solver at the end of coarse level and the other two stacks now provide additional feature maps which can trained by the final level solver(not shown in the figure).

estimate depths and surface normals.

When a single quantity (depths only or normals only) is trained the coarse level convolutional layers corresponding to the other tasks are left in place. Since a dedicated solver is not attached to these stacks, they function as generic convolutional layers which are trained by the solvers of the fine level feature stacks. Figure 5.4 illustrates this idea diagrammatically, in which we are looking at the coarse prediction section of the model. When all three tasks are trained jointly, there is a solver attached at the end of each prediction stack. Simultaneously, we pass the coarse level predictions to the next stage (fine level) to be refined further.

In a scenario, where there is only one training task, the solver corresponding to the training task is kept intact while the other solvers are removed. However, the feature maps of the other stacks are still present and now act as additional weights which are trained using the fine scale solver. This allows the model capacity to be preserved as the number of feature maps are kept constant regardless of the task/tasks that is been trained while greatly influencing what is being learnt by the feature maps through the use of additional tasks.

5.6 Experiments and Results

In this section the performance of the framework is evaluated across the three tasks. For each task the predictions are evaluated against the corresponding ground truths, qualitative comparisons are also made to show the improvement of the proposed approach over the previous state-of-the art neural network based depths and surface normal prediction frameworks. Finally, a segmentation example is presented to showcase how this work could be applied in a real life scenario.

5.6.1 Depth

The depth predictions are evaluated using the proposed metrics of [44] and [112]. These metrics are listed below and will be used throughout the remainder of the thesis to evaluate depths.

$$RMS_{lin} : \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|D_i - D_i^*\|^2} \tag{5.4}$$

$$Rel_{abs} : \frac{1}{n}\sum_{i=1}^{n} \frac{|D_i - D_i^*|}{D_i^*} \tag{5.5}$$

$$RMS_{log} : \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|ln(D_i) - ln(D_i^*)\|^2} \tag{5.6}$$

$$\% \text{ of points with in } \delta : \sum_{i=1}^{n} max\left(\frac{D_i}{D_i^*}, \frac{D_i^*}{D_i}\right) < \delta, \quad \delta = 1.25 \tag{5.7}$$

where D_i is the predicted depth of the ith pixel and D_i^* is the corresponding ground truth depth.

The first three metrics correspond to errors therefore smaller numbers indicate better performance. The final metric is a measurement of the accuracy of the predictions or the number of inliers, and larger numbers indicate better performance.

The predicted depth maps are upsampled by a factor of 4 to match the image resolution of 640x480 and are evaluated against the official ground truth depth maps including the filled in areas but limited to the region where there is a valid depth map projection. In Table 5.1 the results are organised based on the metrics mentioned earlier. These results are further sub divided based on the type of neural network employed to generate the predictions. More concretely,

Depth Prediction

Type	Method	$\text{RMS}_{lin}(m)$	$\text{RMS}_{log}ln(m)$	Rel_{abs}	δ	δ^2	δ^3
single	Liu [122]	0.824	-	0.230	61.4%	88.3%	97.2%
	Eigen [45]	0.877	0.283	0.214	61.4%	88.8%	97.2%
	Ours(D)	0.646	0.216	0.156	76.5%	94.9%	98.7%
	Laina [114]	**0.573**	**0.195**	**0.127**	**81.1**%	**95.3**%	**98.8**%
joint	Eigen(Alex) [44]	0.753	0.255	0.198	69.7%	91.2%	97.7%
	Ours(D+N)$_{depths}$	0.642	0.215	**0.156**	76.6%	94.9%	98.8%
	Eigen(VGG) [44]	0.641	0.214	0.158	76.9%	95.0%	98.8%
	Ours(D+N+C)$_{depths}$	**0.624**	**0.212**	**0.156**	**77.6**%	**95.3**%	**98.9**%

Table 5.1: Depth prediction Metrics: the middle three columns indicate errors (lower better) from ground truth, the final three columns indicate the percentage of points within δ^n (higher better) of the ground truth ($\delta = 1.25$). The bold values indicate the best performing method of each type (single,joint).

the network variants that predict a single quantity are shown under the *single* tab and the variants that predict multiple quantities are shown under the *joint* tab.

Overall, the results validate the initial hypothesis of learning extremely related structural quantities improves the performance of one another. In order to dissect these results further, we can begin by comparing the performance of Ours(D) and Ours(D+N)$_{depths}$. The only difference between these two variants is the former estimates purely depth and the latter predicts depths and normals jointly. Although not overly large, the improvement in performance across all metrics suggest that learning normals in tandem with depths indeed improves the performance of depths. Next let us divert the attention to Ours(D+N)$_{depths}$ and Eigen(VGG) [44] models, where Eigen(VGG) was trained to predict three tasks (depths,normals and semantic labels). The change in performance between Ours(D+N)$_{depths}$ and Eigen(VGG) [44], albeit being very minimal shows learning three tasks is better compared to learning depths and surface normals. However changing the third task from semantic labels to curvature (Eigen(VGG) $\rightarrow$ Ours(D+N+C)$_{depths}$) results in a significant improvement where the RMSE was reduced by 0.02 as opposed to a reduction by 0.004. This demonstrates the importance of surface curvature for depth estimation. Both semantic labels and surface curvature are

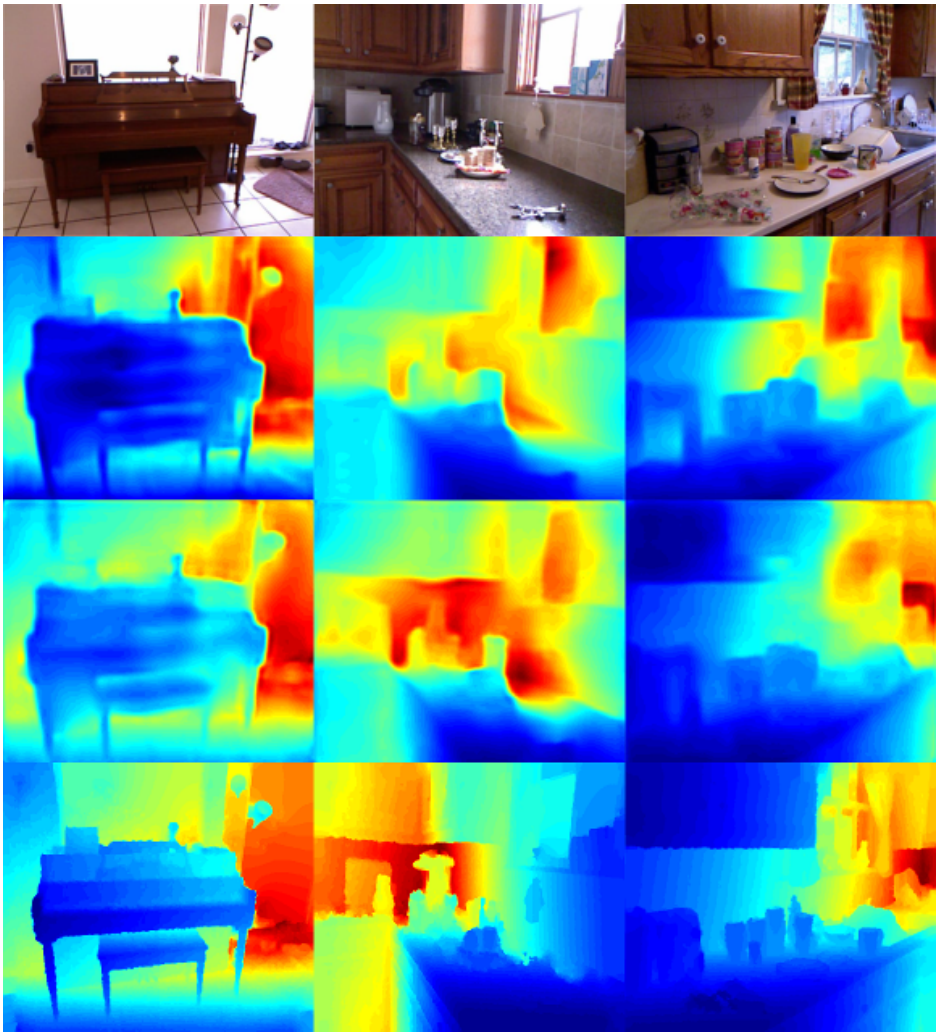

Figure 5.5: Demonstrates the qualitative improvement of our approach for depth estimation. **Top:** RGB image **1st row:** Eigen's Prediction **2nd row:** Our Prediction **Bottom:** Ground Truth

similar in certain ways in that they are both view point invariant quantities. However, surface curvature is more tightly coupled with the other structural quantities compared to semantic labels. The approaches that predict depth alone as a single task are included for completeness. Although we outperform all the methods except [11] which uses the much more powerful ResNet[20] architecture, it is important to stress the contribution of the work presented in this chapter is to demonstrate that given the same architecture, learning related quantities is

more beneficial over learning a single quantity.

5.6.2 Surface Normals

The metrics used to evaluate the predicted surface normals were introduced in [57] by Fouhey *et al.*.

$$\text{Mean Angular Error} = \frac{1}{n}\sum_{i=1}^{n}\cos^{-1}(\mathbf{n}_i \cdot \mathbf{n}_i^*) \tag{5.8}$$

$$\text{Median Angular Error} = \text{median}(\cos^{-1}(\mathbf{n}_i \cdot \mathbf{n}_i^*)) \tag{5.9}$$

$$\%\text{ of points with in t}^\circ : \sum_{i=1}^{n}\cos^{-1}(\mathbf{n}_i \cdot \mathbf{n}_i^*) < t^\circ, \quad t \in 11.25^\circ, 22.5^\circ, 30^\circ \tag{5.10}$$

where N_i is the predicted surface normal of the ith pixel and N_i^* is the corresponding ground truth depth.

Similar to [44, 5], the predicted surface normals are evaluated against the ground truth surface normals provided by Ladicky *et al.* [113]. During evaluation the regions corresponding to the missing depth values are masked out since the ground truth normals can not be accurately computed on those areas. These results are summarised in Table 5.2.

Similar to depths, predicted normals also gained an increase in accuracy when the network was trained in a multi task platform. Although, having merely depths in parallel did not make a noticeable change, extending the network to learn all three tasks resulted in a significant improvement. This is primarily due to passing the knowledge of the coarse curvatures as an additional feature map for fine level surface normal prediction. When compared with frameworks that predict surface normals as a stand-alone task, our approach outperforms all

Surface Normal Prediction

Type	Method	Angular Error Mean	Angular Error Median	Within t° ≤ 11.25°	Within t° ≤ 22.5°	Within t° ≤ 30°
single	Ladicky [113]	35.3	31.2	16.4 %	36.6%	48.2%
	Wang [194]	26.9	14.8	42.0%	61.2%	68.2%
	Ours (Normals)	21.1	13.5	43.6%	66.6%	75.4%
	Bansal et al [5]	**19.8**	**12.0**	**47.9 %**	**70.0 %**	**77.8 %**
joint	Eigen(Alex) [44]	23.7	15.5	39.2 %	62.0 %	71.1%
	Ours(D+N)$_{normals}$	21.1	13.6	43.6%	66.5%	75.4%
	Eigen(VGG) [44]	20.9	13.2	44.4%	67.2%	75.9%
	Ours(D+N+C)$_{normals}$	**20.6**	**13.0**	**44.9%**	**67.7%**	**76.3%**

Table 5.2: The mean, median angular error and the percentage of points with an angular error less than a threshold (t°) for several normal estimation approaches evaluated against the ground surface normals provided by Ladicky *et al.* [113]. Lower numbers are better for the first two columns and higher numbers are better for the last three columns.

previous approaches barring that of Bansal *et al.* Quantitatively we approach the performance of Bansal *et al.* [5] who used a skip architecture with a larger model capacity compared to ours, although arguably qualitatively both [44] and our approach outperform their predictions as shown in Figure 5.6.

5.6.3 Surface Curvature

Since our approach is the first piece of work to demonstrate surface curvature prediction from a neural network, a set of metrics had to defined first as shown below prior to evaluating the performance of the curvature predictions.

$$RMS_{\kappa_1} : \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|\kappa_1 - \kappa_1^*\|^2} \quad RMS_{\kappa_2} : \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|\kappa_2 - \kappa_2^*\|^2} \tag{5.11}$$

Figure 5.6: Demonstrates the qualitative improvement of our approach for normal estimation. **Top:** RGB image **1st row:** Bansal [5], **2nd row:** Eigen [44], **3rd row:** Our Prediction **Bottom:** Ground Truth [178]. The missing areas in the ground truth normals coincide with those in the raw depth images.

$$\text{let} \quad \text{Pred}_{mean} = \frac{\kappa_1 + \kappa_2}{2}, \text{GT}_{mean} = \frac{\kappa_1^* + \kappa_2^*}{2}$$
$$\text{Median error} = \text{median}(||\text{Pred}_{mean} - \text{GT}_{mean})||_2) \tag{5.12}$$

$$\% \text{ of points within } \sigma(m^{-1}) : \sum_{i=1}^{n} ||\text{Pred}_{mean} - \text{GT}_{mean})||_2 < \sigma,\ \sigma \in 0.25, 0.5, 1(m^{-1}) \tag{5.13}$$

Principal Curvature Predictions

Method [178]	RMS (m^{-1})		Median (m^{-1})		Within σ_t		
	κ_1	κ_2	planar	non-planar	σ_1	σ_2	σ_3
Eigen(computed from depths) [44]	5.56	7.50	3.86	1.44	25.7%	33.9%	43.5%
Ours (computed from depths)	6.03	6.50	4.23	1.38	26.9%	34.9%	44.2%
Ours (C)	3.41	5.17	1.984	0.184	52.6%	63.2%	73.2%
Ours (D+N+C)	**2.81**	**4.47**	**1.634**	**0.085**	**63.1%**	**72.7%**	**80.3%**

Table 5.3: The table shows the RMS error of estimating the principal surface curvatures (κ_1, κ_2), the median error for planar and non-planar regions and the percentage of curvatures values that are within a threshold $\sigma_1 = 0.25m^{-1}$, $\sigma_2 = 0.5m^{-1}$, $\sigma_3 = 1m^{-1}$. The first two approaches do not explicitly predict curvature and are computed from the predicted depths.

Due to the lack of true ground truth surface curvatures, the curvature maps generated using the method of [178] are adopted as ad hoc ground truth data. We evaluate the predictions made by our network against the curvature values computed from the predicted depths produced by our own network and the network of [44]. The RMS error for each of the principal curvatures (κ_1, κ_2) is computed along with the median error of the mean curvature($\frac{\kappa_1+\kappa_2}{2}$) across two categories, planar and non-planar regions. The planar surfaces are defined to be surfaces with a radius of curvature greater than 1 meter. As expected predicted curvatures clearly outperform the computed curvatures from depths. Furthermore, the predicted curvatures using the joint model which learned surface normals and depths in parallel provide better performance.

Figure 5.7 is included as a reference to show how the metrics in Table 5.3 translate into visual appearance.

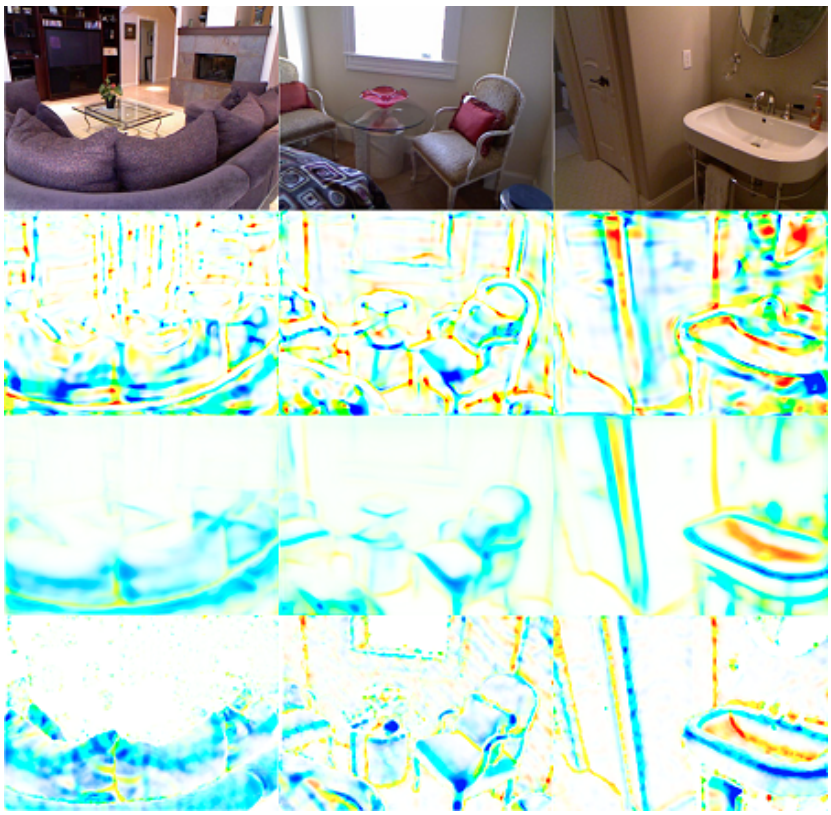

Figure 5.7: Demonstrates the qualitative improvement of our approach for surface curvature estimation. **Top:** RGB image **1st row:** Computed surface curvature based on Eigen's [44] depth prediction **2nd row:** Prediction of our system **Bottom:** Ground Truth computed from raw depth data

5.6.4 Applications of This Work

As a purely qualitative demonstration of our approach, a simple scene segmentation example was designed by combining information from the colour, depth and curvature of selected scenes. The segmentation is generated by combining the gradients of colour and depth, and curvature values. This border function $b(u, v)$ can be expressed as

$$b(u, v) = w_I \cdot \nabla I(u, v) + w_d \cdot \nabla D(u, v) + w_c \cdot C(u, v), \tag{5.14}$$

where $\nabla I(u, v)$ is the the magnitude of the image intensity gradient, $\nabla D(u, v)$ is the magnitude of the depth gradient and $C(u, v)$ is the curvature value at the point u, v. That is

$$\nabla I(u,v) = \sqrt{\frac{\partial I(u,v)}{\partial u}^2 + \frac{\partial I(u,v)}{\partial v}^2}, \tag{5.15}$$

and

$$\nabla D(u,v) = \sqrt{\frac{\partial D(u,v)}{\partial u}^2 + \frac{\partial D(u,v)}{\partial v}^2}. \tag{5.16}$$

The final segmentation is then generated by applying a simple threshold on this border function. That is a pixel is considered a border ($\mathbf{B}(u, v)$) if it satisfies the condition

$$\mathbf{B}(u,v) = \begin{cases} 1 & \text{if } b(u,v) \geq \delta_{thresh} \\ 0 & \text{otherwise} \end{cases} \tag{5.17}$$

We compare the performance of this segmentation method using the ground truth quantities and the predictions (depths and curvature) generated by our network as shown in Figure 5.8. The results are not intended to be treated as state of the art segmentations, but are included to demonstrate a possible future extension of this work and also to illustrate that the information from the network can be used to perform similar tasks.

5.7 Generalisation to newer architectures

VGG [174] was one of the state-of-the-art neural network architectures at the time of publishing the conference paper corresponding to this chapter, however, superior architectures have emerged since then. Therefore, an experiment was

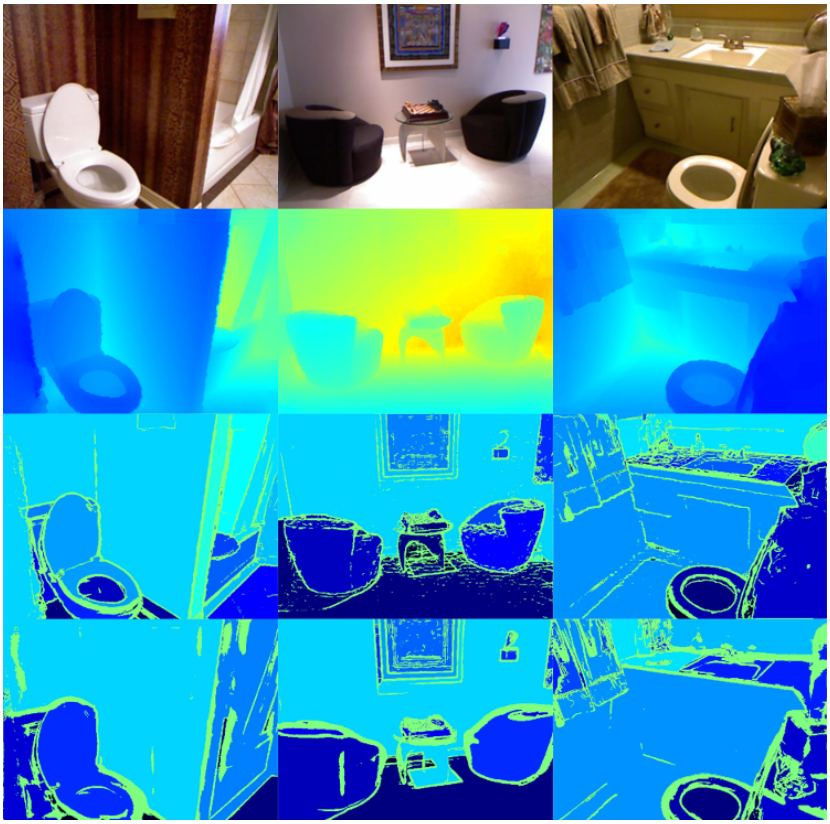

Figure 5.8: Demonstrates a basic segmentation algorithm, that uses colour, depth and curvature to generate a border function. The rows of the figure are, top to bottom: Input colour image, Input ground truth depth, Segmentation From GT data, Segmentation from Predicted Data. The key contribution of the depth and curvature to the segmentations, are on the depth boundaries and wall edges that are difficult to differentiate from colour alone.

designed to verify the generality of the approach on newer architectures by replicating the algorithm using DenseNet [90]. The results of this experiment are tabulated in Table 5.4. While there is a clear increase in performance due to the architectural improvements, the most notable result is the performance gain achieved by training depths, surface normals and surface curvatures jointly as opposed to training depths alone and thus validating our approach on multiple architectures.

Depth Prediction using DenseNet [90]

Method	RMS_{lin}	RMS_{log}	Rel_{abs}	δ	δ^2	δ^3
Depth (single)	0.555	0.176	0.119	85.5%	97.0%	99.2%
Depth (joint with normals+curvatures)	0.513	0.170	0.118	86.3%	97.2%	99.2%

Table 5.4: A comparison of depth prediction performance when trained jointly and as a single task using the DenseNet [90] architecture.

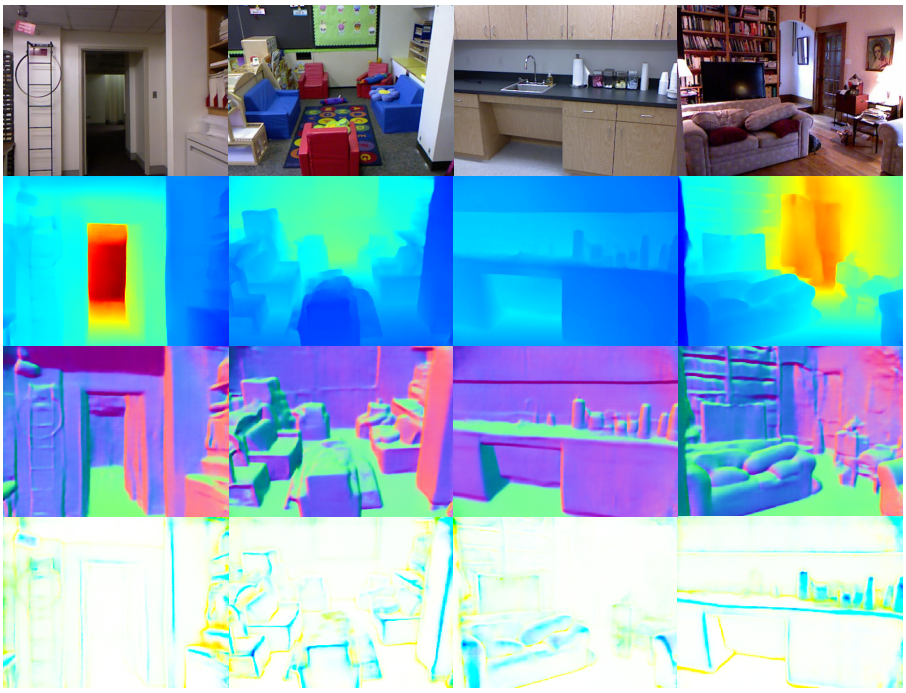

Figure 5.9: Predictions made by the joint framework using the DenseNet architecture. In row order the images show the colour image, predicted depth maps, predicted surface normals and finally the surface curvatures.

5.8 Summary

In this chapter we have presented a multi task learning platform capable of predicting depths, surface normals and surface curvatures purely from an RGB image. We show that carefully chosen handcrafted feature representations can outperform the machine learnt features, provided they are closely related to the prediction task. This shows that network guidance is a useful aspect and should

not be completely ignored when training neural networks. Extensive experiments were conducted by keeping the model capacity of the architecture fixed while gradually increasing the number of prediction tasks to verify the effectiveness of our hypothesis.

Furthermore, the approach was tested on two different neural network architectures in order to demonstrate the concepts presented are invariant to architectural changes. Finally, as a practical robotic application, the predicted quantities were combined to perform scene segmentation.

CHAPTER 6

Sparse Depth Map Inpainting using CNNs

6.1 Introduction

Perceiving depth and understanding the underlying geometry of the world is crucial in order to develop fully autonomous robots which can navigate and interact with the environment. This is largely facilitated by three common approaches. Creating a map of the world using a Simultaneous, Localization and Mapping System (SLAM) as seen in Chapter 4, using the approach discussed in Chapter 5 where a Convolutional Neural Network (CNN) was employed to predict the depth given an RGB image and finally by using a sensor which is capable of capturing range information (e.g Microsoft Kinect, LIDAR).

Each modality offers a unique set of advantages over the others. Sparse SLAM systems are highly desirable for mapping large scale environments or for relocalising the camera after failing to track, as it is much more efficient to manage sparse point clouds. Image driven depth prediction CNNs are capable of predicting depths from a single colour image even when the scene is texture less. Finally, active sensors are capable of providing metric depths and are the most accurate

modality. However, there are few drawbacks. Sparse SLAM systems and LIDARs both produce sparse depth maps as shown in Figure 6.1, the depth maps produced from a Kinect can contain missing areas and the depth predictions produced from a neural network albeit being dense are generally less accurate compared to all other modalities. Active sensors are generally more expensive and less ubiquitous compared to image driven approaches (SLAM or CNN-based) and might be less desirable for some scenarios.

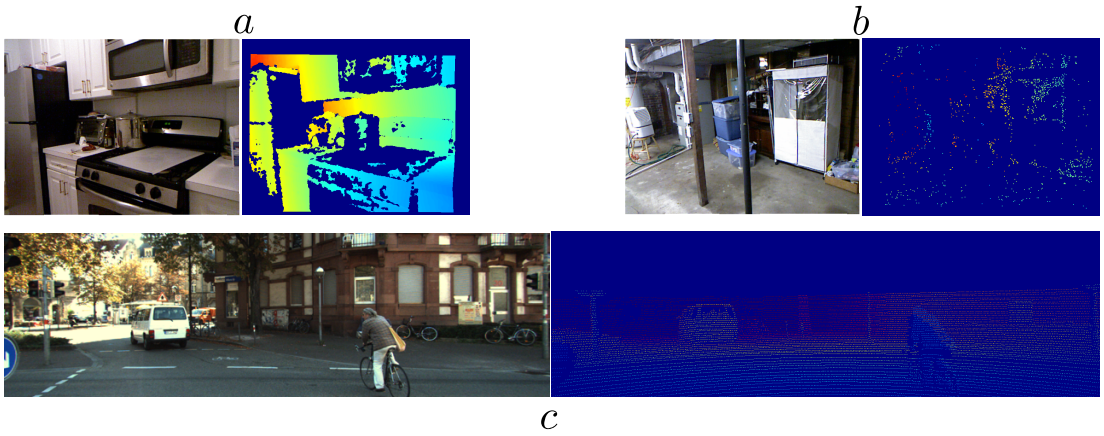

Figure 6.1: Depth maps from different modalities each with different amounts of sparsity. Image pair *a* shows the depth map produced from a Microsoft Kinect and the corresponding rgb image, *b* shows the output of a SLAM map and the corresponding rgb image, finally *c* shows the output of a LIDAR and the associated rgb image.

Due to these inherent limitations of all of the above approaches, it is becoming increasingly common to use a combination of the methods to create a better 3D representation of the world. However, this fusion process is non-trivial as the SLAM system is generally in an arbitrary scale, while the predictions of the CNN or the range data from the sensors are in metric scale. Furthermore, even the performance of learning approaches suffer if the images fed at test time have radically different focal lengths and it is a common practice to generate a separate depth estimation model for indoors and outdoors. Having a representation of confidence for each modality can be of great benefit during the fusion process, as the confidences can then be used to fuse the depth maps probabilistically.

6.1.1 Contributions

The publication [200] corresponding to this chapter was produced in collaboration with Saroj Weerasekera who is part of the computer vision and machine learning group of University of Adelaide led by Prof. Ian Reid.

I was the second author of this publication and the relative contributions I have made is as follows:

- Conception of Idea(50%)
- Network architecture design (50%)
- Loss/Objective function design (30%)
- Coding the network (40%)
- Training data creation (50%)
- Testing and evaluation (50%)

In the context of the thesis, this chapter aims to bridge some of the concepts presented in Chapters 4 and 5 by combining sparse SLAM maps with depth predictions using CNNs.

The main contributions of this work are:

- A real-time framework that is capable of generating a dense reconstruction of a scene given a sparse depth map and a neural network that generates an estimate of the scene depth.
- A novel CRF formulation which allows the fusion of depth maps in different scales.

6.2 Related Work

The closest work to the approach proposed in this chapter is that of Fácil *et al.* [51], in which they proposed to fuse multi-view depth obtained using LSD-SLAM [48] with the predicted depth maps of Eigen *et al.* [44]. Their approach was largely motivated by the fact that multi-view depth estimation methods exhibit small errors compared to CNN based methods for areas with large image gradients, while this relationship is reversed for areas with low image gradients. The experiments conducted in Chapters 4 (SLAM depths) and 5 (CNN depths) also support these findings. The main difference between their algorithm and ours is they fuse the depth maps based on a product of four pre-defined weighting factors as opposed to ours where the two depth maps are fused probabilistically based on learned confidences. Fertsl *et al.* were able to fuse depths captured from two different sensors [52]. A time of flight tensor which operates at a large frame rate albeit with a low lateral resolution, was combined with the output of a stereo sensor which has a relatively large lateral resolution. They employed a primal-dual optimisation scheme to perform the fusion process. On a similar note [123] Liu *et al.* demonstrated fusion of depth map obtained from an IR based depth sensor and a stereo rig. Contrary to all these works, we propose a scale-invariant approach allowing us to fuse depth maps in radically different scales. I would like mention since publishing the conference paper corresponding to this chapter, several works have emerged in this field [208, 94, 96], further highlighting the importance of this research problem.

Non-learning based depth map completion methods (as demonstrated in the NYUv2 dataset[140]) draw from image processing approaches [119, 188]. The former technique by Levin *et al.* was initially applied to colourise monochromatic images given a greyscale image with colour scribbles. This was achieved by imposing constraints such as neighbouring pixels should have similar colours if the corresponding pixels have similar intensities. Sparse or semi-dense depth map densification can also be performed using an approach akin to this provided

that there's a corresponding colour image. Missing depth values can then be approximated based on the neighbouring depths by using the colour information as a guide. In a similar vein, Tomasi and Manduchi proposed *bilateral filtering* [188] an approach that combined domain filtering which enforces geometric closeness with range filtering which focuses on photometric similarity. When applied to depth densification, this forces nearby pixel values as well as pixels with similar colours to have similar depth values. However, both of these approaches suffer when there are large areas of missing data in the depth image as shown in Figure 6.2.

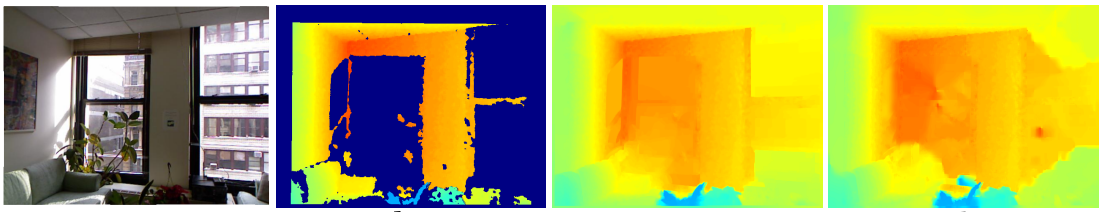

Figure 6.2: In column order, a: RGB image, b : ground truth depth map (with missing values), c: densified depth map using [119], d: densified depth map using [188]

Krähenbühl *et al.* [106] proposed to employ a fully connected CRF model to fuse class labels with image data to obtain a refined set of class labels. However, these CRFs ignore the arbitrary scale, varied irregular sparsity and/or uncertainty at which visual SLAM maps are created. Moreover, they are also very restrictive, relying on handcrafted priors that link image colours and textures to depths.

6.3 Method

6.3.1 Problem Definition and Terminology

The proposed framework aims to infer over a fully connected learnable CRF in order to densify/inpaint a sparse map with P points that has been aligned

with a single RGB image I with N pixels and represented as a *partial* log-depth map $\mathbf{y^s} = [y_1^s, .., y_N^s] = ln(d^s) = [ln(d_1^s), .., ln(d_N^s)]$, where d_i^s is the depth of pixel i, with valid depths only at the projected pixel locations. Assuming that $\mathbf{c^s} = [c_1^s, .., c_N^s], 0 \leq c_i^s \leq 1, \forall i$ is the confidence associated with the map estimated by a probabilistic SLAM approach or learned using a neural network trained to predict map confidence from input data, the end goal is to infer the dense log-depth map $\mathbf{y} = [y_1, .., y_N] = ln(d) = [ln(d_1), .., ln(d_N)]$

We also assume that for this inpainting task we are given a dense single view depth prediction network and a data-driven depth confidence prediction network (will be discussed in Section 6.3.4). We denote the log-depths regressed by the depth prediction network to be $\mathbf{y^p} = [y_1^p, .., y_N^p] = ln(d^p) = [ln(d_1^p), .., ln(d_N^p)]$, and respective confidence maps to be $\mathbf{c^p} = [c_1^p, .., c_N^p], 0 \leq c_i^p \leq 1$.

The inpainted dense depth map produced by our framework using a sparse SLAM depth map is by default in the arbitrary scale of the SLAM system. Alternatively, the dense depth map can also be produced in metric scale as explained in Section 6.4.

6.3.2 Sparse Depth Map Inpainting

In order to inpaint the sparse depth map as described above we propose to minimize the following CRF energy with respect to $\mathbf{y}$:

$$E(\mathbf{y}) = \alpha E_u(\mathbf{y}, \mathbf{y^s}, \mathbf{c^s}, \mathbf{b}) + \beta E_{fc}(\mathbf{y}, \mathbf{y^p}, \mathbf{c^p}) + \gamma E_{lc}(\mathbf{y}, \mathbf{y^p}, \mathbf{c^p}) \tag{6.1}$$

where $\mathbf{b}$ is binary mask denoting pixels with valid sparse depths.

E_u is the unary term generating the log depths for image I to be consistent

with the sparse map, E_{fc} is a fully-connected pairwise term and E_{lc} is a locally connected pairwise term, both penalizing incorrect pairwise depth relationships (depth ratios as scale invariant measures) of the inferred dense log depth map using the single view depth predictions as the learned priors. In a general form, multiple sparse and dense depth maps obtained at arbitrary scale from various sources can be used in our framework, replacing the CNN-based single view depth prediction, and simply minimizing the sum of the pairwise and fully connected terms described above. The tunable parameters $(\alpha, \beta, \gamma) > 0$ signify the relative importance of each term. A detailed description of each CRF term, the motivation behind using them and relations of these to existing frameworks are described in the following subsections.

Nodes of CRF

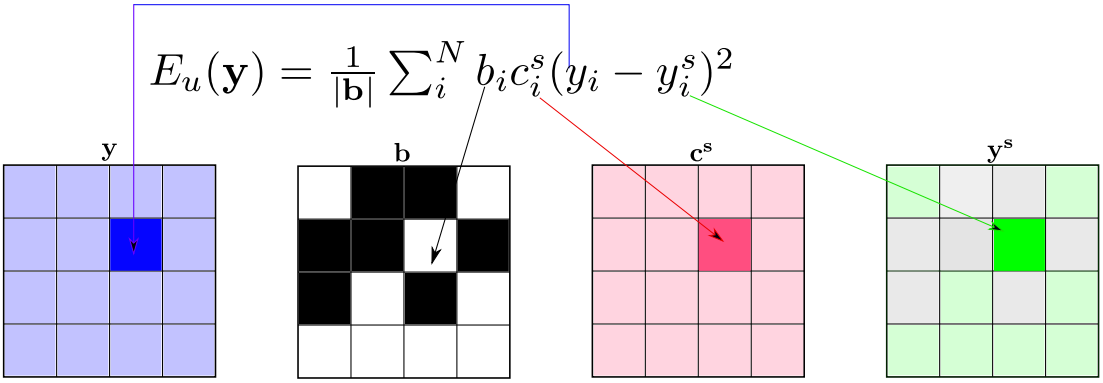

Figure 6.3: Given a sparse depth map $\mathbf{y^s}$ where gray pixels indicate missing values, the unary term aims to pull the dense prediction $\mathbf{y}$ to be consistent with the sparse depth value $\mathbf{y^s}$. The term is only computed on the pixels with a valid sparse value given by the binary mask $\mathbf{b}$ where black indicates missing values. The term is weighted by the learned confidence map $\mathbf{c^s}$.

The unary term in the CRF pulls the inferred depth map to be consistent with the sparse map obtained via SLAM or a sensor. Inspired by [112, 45, 44], we use squared natural log-depth differences for every point on the sparse map as the unary potentials of our CRF:

$$E_u(\mathbf{y}) = \frac{1}{|\mathbf{b}|} \sum_{i}^{N} b_i c_i^s (y_i - y_i^s)^2 \tag{6.2}$$

Each term in E_u is weighted by the learned confidence of map accuracy c_i^s.

Edges of fully connected CRF model

As the end goal is to inpaint sparse maps of arbitrary scale we aimed to design a learnable prior which is insensitive to the scale of the scene. For this purpose, we propose the pairwise potentials of our fully connected CRF to be:

$$E_{fc}(\mathbf{y}) = \frac{1}{2N} \sum_{i,j} c_{ij}^p \big((y_j - y_i) - (y_j^p - y_i^p)\big)^2 \tag{6.3}$$

where c_{ij}^p are the learnable parameters of the CRF which aims to approximate the confidence of the log of depth-ratio $ln(d_j^p/d_i^p) = y_j^p - y_i^p$ of the single view depth predictions for any two points i and j learned in a data driven fashion. This is expressed graphically in Figure 6.4

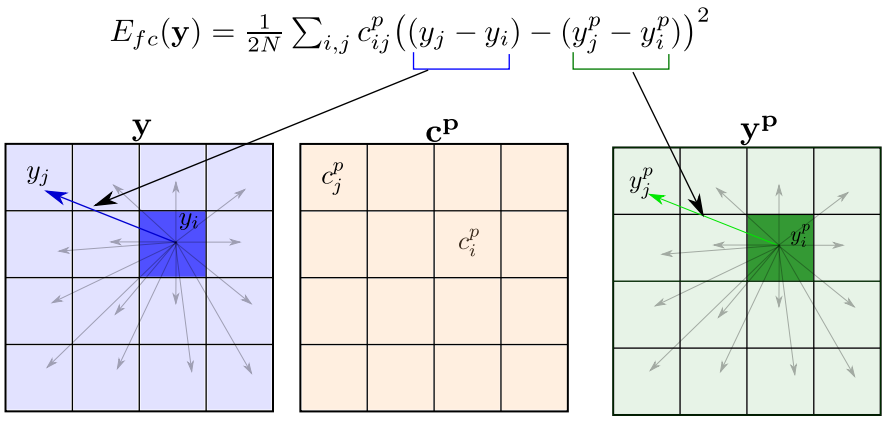

Figure 6.4: The fully connected term aims to enforce the ratio of the inferred depths of two points to be same as that of the single view depth prediction network. c_{ij}^p is approximated by the product $c_i^p c_j^p$.

Pairwise terms of our fully connected CRF can be best interpreted as the terms

which enforce scale-invariant ordinal relationships of the inferred depths of two points to be same as that of the single view depth prediction network. The intuition behind using this fully connected CRF is that depth ratios of two points in a scene are invariant to the scale of the scene. Any other scale invariant function $f(d_i, d_j)$ may be used without loss of generality in our framework in place of $ln(d_j/d_i)$.

The fully connected pairwise CRF defined in Equation 6.3 is however intractable in its most generic form, as the number of learnable parameters $c^p_{ij}, \forall(i,j)$ grows quadratically with the number of pixels in the image. Approximations are generally used to model c^p_{ij} in parametric form for reducing the number of independent learnable parameters and for efficient inference. The most common practice is to model c^p_{ij} as the sum of Gaussian Radial Basis Function (RBF) kernels each having two learnable parameters, which are mean and variance. For example, [7] and [106] define c^p_{ij} in the form of Gaussian RBF kernels that are a function of the distance between pixel i and j and the colour difference between those pixels. These CRF models allow for fast inference but are very restrictive.

In this work, we propose a different relaxation where $c^p_{ij} = c^p_i c^p_j$ which allows for efficient inference, while having many more learnable parameters for expressiveness. The intuition is that the accuracy of the pairwise term is limited by the least confident depth value forming the ratio, and thus the overall confidence can be approximately expressed as a product of individual ones. This simple approximation significantly reduces the number of parameters to learn, and also allows for tractable inference as the fully connected term in Equation 6.3 (and thereby its gradient) can now be re-written in an alternative form that allows for linear time computation:

$$E_{fc}(y) = \frac{1}{N}\sum_{j}^{N} c^p_j \sum_{i}^{N} c^p_i (y_i - y^p_i)^2 - \frac{1}{N}\Big(\sum_{i}^{N} c^p_i (y_i - y^p_i)\Big)^2 \tag{6.4}$$

Local Grid Connected Edges of CRF

Additionally, we define a grid connected term of the CRF to give more importance to the local structures learned by the single view depth predictor:

$$E_{lc}(\mathbf{y}) = \sum_{i,k} c_i^p c_k^p \big((y_k - y_i) - (y_k^p - y_i^p)\big)^2 \tag{6.5}$$

where $k \in \{+u(i), +v(i)\}$ denotes pixel locations to the right of and below the pixel i in the image plane as shown in Figure 6.5. This enforces the solution to trust pairwise relations in $\mathbf{y^p}$ around a local neighbourhood for each pixel i, and is thus helpful for providing local support for the unary term where depth information is absent. The end-effect of E_{lc} is similar to a data-driven local smoothing as shown in Figure 6.6, where information obtained from the local pairwise depth relationships of the depth predictions ($\mathbf{y^p}$) are used to "smooth over" the areas where the sparse map's points are fused in the solution, while still anchoring the solution onto the sparse map. We only consider a 4-connected graph for the locally connected pairwise term (E_{lc}) as the fully connected term (E_{fc}) already encompasses the full pairwise connectivity graph.

We denote the set of pixels in the neighbourhood of i as $\mathcal{N}(i)$. Additional model expressibility can be added to our locally connected pairwise terms by increasing the size of $\mathcal{N}(i)$. One way of achieving this is through the introduction of a multiplicative pairwise pixel-distance based Gaussian RBF kernel with tunable variance to the terms in Equation 6.5 such that nearby pairwise depth ratio inconsistencies are penalized more strongly than those further apart.

6.3.3 Inference Method

The inference objective is to find $min_y E(\mathbf{y})$. For ease of expression we can re-write (6.1) in the following form:

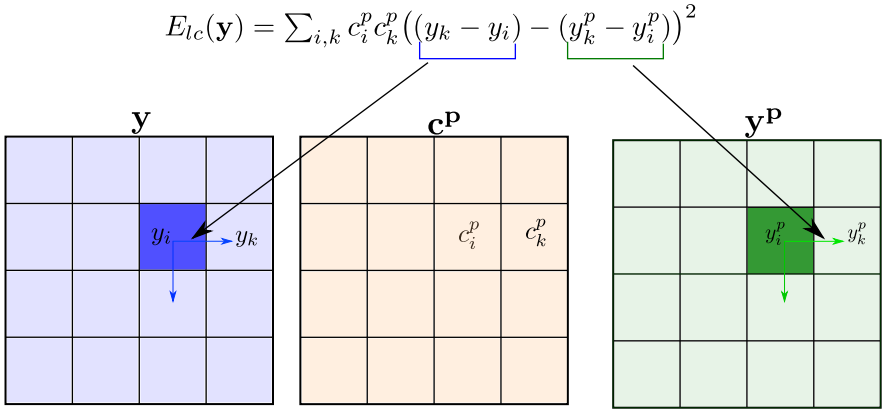

Figure 6.5: The locally connected term similar to the fully connected term enforces the ratio of the inferred depths of two points to be same as that of the single view depth prediction network. However, the main goal of the E_{lc} term is to highlight the importance of the local structures. Therefore, for each pixel the influence of E_{lc} limited to a neighboured given by $k \in \{+u(i), +v(i)\}$ denoting the pixels to the right and directly below.

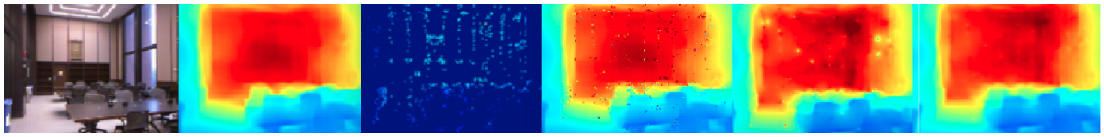

Figure 6.6: Qualitative ablation of the impact of data-driven smoothing as defined by our local CRF pairwise terms E_{lc}, and the impact of incorporating confidences into the energy. In column order 1^{st}: RGB Image, 2^{nd}: Dense depth prediction of Eigen [44] (metric scale), 3^{rd}: Sparse depth map of [138] (arbitrary scale), 4^{th}: Our reconstruction with learned confidences but without E_{lc}, 5^{th}: Our reconstruction with E_{lc} but without learned confidences, 6^{th}: Our final reconstruction with both E_{lc} and learned confidences.

$$E(\mathbf{y}) = \mathbf{y}^T A \mathbf{y} - 2(\mathbf{y}^T A^s \mathbf{y^s} + \mathbf{y}^T A^p \mathbf{y^p}) + \mathbf{y^s}^T A^s \mathbf{y^s} + \mathbf{y^p}^T A^p \mathbf{y^p} \tag{6.6}$$

where $A = (A^s + A^p)$ is a $N \times N$ symmetric positive (semi-) definite matrix. A^s is a diagonal matrix with entries $A^s_{ii} = \alpha c^s_i, \forall i$, while A^p is a dense $N \times N$

symmetric positive (semi-) definite matrix with entries as follows:

$$\begin{aligned} A^p_{ii} &= c^p_i\Big(\frac{\beta}{N}\sum\nolimits_{j,j\neq i}^{N} c^p_j + \gamma\sum\nolimits_{j\in\mathcal{N}(i)} c^p_j\Big) && \forall i \\ A^p_{ij} &= -c^p_i\Big(\frac{\beta}{N}c^p_j + \gamma c^p_j\Big) && \forall i,j \in \mathcal{N}(i) \\ A^p_{ij} &= -c^p_i\Big(\frac{\beta}{N}c^p_j\Big) && \forall i, j\neq i, j\notin\mathcal{N}(i) \end{aligned} \tag{6.7}$$

Differentiating Equation 6.6 with respect to y and then setting the resulting expression to 0 we obtain:

$$A\mathbf{y} = A^s\mathbf{y^s} + A^p\mathbf{y^p} \tag{6.8}$$

Algorithm 6.1 Conjugate gradient method for solving for the optimal log depth map for an image I

Initialize:
$\mathbf{y}_0 = 0$
$\mathbf{r}_0 = A^s\mathbf{y^s} + A^p\mathbf{y^p} - A\mathbf{y}_0$
$\mathbf{p}_0 = \mathbf{r}_0$
k = 0
while $||\mathbf{r}_k|| < \epsilon$ **do**
  $a_k = \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}A\mathbf{p}_k}$
  $\mathbf{y}_{k+1} = \mathbf{y}_k + a_k\mathbf{p}_k$
  $\mathbf{r}_{k+1} = \mathbf{r}_k - a_kA\mathbf{p}_k$
  $\beta_k = \frac{\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}$
  $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$
  k = k+1
$\mathbf{y} = \mathbf{y}_{k+1}$

To solve for y in Equation 6.8 we use the iterative conjugate gradient method as shown in Algorithm 6.1. For the algorithm it is not necessary to explicitly construct the matrices A^s and A^p, instead we can simply evaluate the gradients $A^s\mathbf{y^s}$ and $A^p\mathbf{y^p}$ at the start, and $A\mathbf{p}_k$ at each iteration-step. Note that computing $A^p\mathbf{y^p}$ and $A\mathbf{p}_k$ require $O(N^2)$ operations, however based on the simplified form of Equation 6.4, a linear time expression for gradient computation can be derived.

To further accelerate the process we implement the algorithm to run on the GPU, where per pixel operations are parallelized. In practice, the solution converges rapidly to within a desired threshold in $n \ll N$ iterations.

Note that for the system of linear equations to have a unique solution, i.e. non-zero determinant for A, α should be non-zero. Intuitively, it means that at least one input depth map must contribute to the absolute scale of the fused depth map, else infinite solutions exist where the fused depth map is correct up-to-scale. Also, to prevent a potential condition number of infinity due to diagonal entries in A being equal to 0, we add a small epsilon to $\mathbf{c^d}$.

6.3.4 Learning to Predict Confidence Weights

The goal here is to learn separate CNN models that can model the conditional error distributions of the sparse depth map $\mathbf{y^s}$ and the depth prediction generated from a CNN $\mathbf{y^p}$, and predict $\mathbf{c^s}$ and $\mathbf{c^p}$ respectively, given the respective depth maps and the image I as input. Since the training setup and network architecture is almost identical for the two cases, for brevity we focus on the training procedure for predicting $\mathbf{c^s}$, and mention the differences.

The inputs to the CNN model are $\mathbf{y^s}$ and I. The RGB image is first passed through a (9x9 kernel size) convolutional layer with 127 output feature maps. The output feature maps are then concatenated with the input log-depth map and passed through 6 more (5x5 kernel size) convolution layers with each having 128 output feature maps, except for the last layer which regresses the confidence map. All layers are followed by ReLU activation functions, except the output layer which we leave as linear. At test time the predicted values are clipped between 0 and 1 inclusive.

The irregular sparsity structure in $\mathbf{y^s}$ poses a difficulty to the learning process,

as the network has an additional task of learning which points in the input depth map are valid. This demands higher model capacity. One way to facilitate the learning process is to explicitly model the network to be invariant to the sparsity of the data, for instance by performing masked convolutions at each layer [190] which require significant computation overhead.

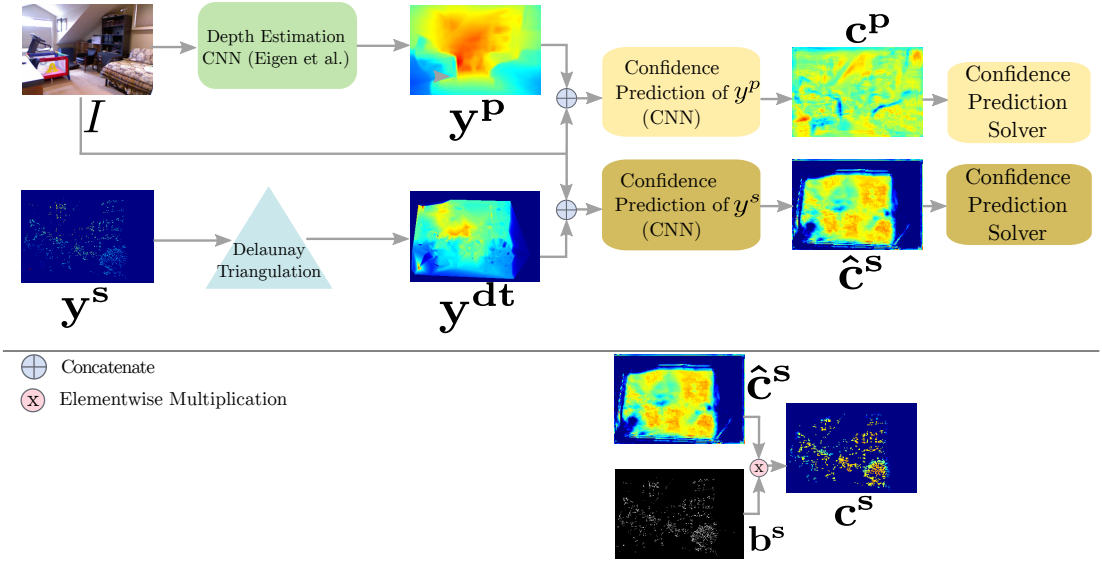

Figure 6.7: The top figure shows the training pipeline. In the bottom figure we show that by performing element-wise multiplication between $\hat{c}_i^s$ (the confidence map corresponding to the Delaunay triangulated depth map) and b^s (the binary mask of y^s) we can obtain c^s (the confidence map of the sparse depth map).

We believe that a more efficient way to facilitate learning with a small network is to densify the data itself before feeding it into the convolutional layers based on some assumption about the data. Here we opted to perform Delaunay triangulation on the 2D image coordinates corresponding to valid points on the depth map, followed by barycentric-coordinate-based linear interpolation (in inverse depth space) to fill the triangles with log depth values. The latter can be efficiently carried out on the GPU. This densification method is motivated by the fact that most regions in a depth map are typically piecewise-planar. Doing so also enhances errors in the sparse log depth map that otherwise would have been difficult for the network to pick up.

The training loss L^s for predicting c^s is defined as follows:

$$L^s = \frac{1}{|\mathbf{b^s}||\mathbf{b^{gt}}|} \sum_{i}^{N} b_i^s b_i^{gt} (\hat{c}_i^s - \hat{c}_i^{s*})^2 \tag{6.9}$$

where $\hat{\mathbf{c}}^\mathbf{s}$ is the predicted confidence map for Delaunay triangulated depth map, $\mathbf{b^s}$ and $\mathbf{b^{gt}}$ are binary vectors indicating the values for which the sparse depth map and the ground truth depth map are non-zero respectively. Since confidence of a depth value is inversely proportional to its error (and in the more general case scale-invariant error) we define $\mathbf{c_i^{s*}}$ as follows:

$$\hat{c}_i^{s*} = b_i^s b_i^{gt} e^{-\lambda^s |E_i^s|} \tag{6.10}$$

where $\lambda^s > 0$ is a tunable parameter controlling the contrast of c^s, and E_i^s is the scale-invariant error for a depth value in pixel i defined as:

$$\begin{aligned} E_i^s = (\alpha^s + \beta^s) b_i^s b_i^{gt} (y_i^s - y_i^{gt}) - \frac{\beta^s}{|\mathbf{b^s}||\mathbf{b^{gt}}|} \sum_{j}^{N} b_j^s b_j^{gt} (y_j^s - y_j^{gt}) \\ + \gamma^s \sum_{j \in \mathcal{N}(i)} b_i^s b_j^s b_j^s b_j^{gt} \big((y_j^s - y_i^s) - (y_j^{gt} - y_i^{gt}) \big) \end{aligned} \tag{6.11}$$

where $\mathbf{y^{gt}}$ represents the groundtruth log-depth map. The parameters $(\alpha^s, \beta^s) > 0$ can be set based on the amount of scale-invariance we require E_i^s to be ($\alpha^s = 0$ for full-scale-invariance in the case of y^s, as it is in a random scale), and $\gamma^s > 0$ determines whether the network should emphasize more on learning confidences in local pairwise connectivity.

In order to get $\mathbf{c^s}$ from the network output $\hat{\mathbf{c}}^\mathbf{s}$ (which is now the predicted confidence map corresponding to the triangulated dense log depth map), we simply perform an element-wise multiplication of the network output with $\mathbf{b^s}$. The random variability in map scale of $\mathbf{y^s}$ also poses a difficulty to the learning process, and as a solution to this we scale each triangulated dense log depth map so that its mean is equal to the mean of the ground-truth log depths in the entire train set, before passing it into the network.

For training, we use the Caffe [97] framework. Training was performed with a batch size of 16, a learning rate of $1e^{-2}$, momentum of 0.9 using SGD as the optimizer on a NVIDIA GTX 1080 Ti GPU.

6.4 Results

This section provides a summary of the quantitative and qualitative results of the proposed framework for different experimental settings on NYU Depth v2 [140] and KITTI [63] datasets. For all indoor experiments the network proposed in Eigen *et al.* [44] is used as the virtual depth sensor and for the outdoor experiments the neural network of Garg *et al.*[62] is used to generate depth predictions. The baselines chosen for comparison are predominantly image inpainting methods as depth map completion approaches were not openly available at the time of writing conference paper corresponding to this chapter.

We first demonstrate the performance of sparse depth map densification for ORB-SLAM depth maps. For this experiment, we use the train/test split specified in [44], however our train and test sets are a fraction of the original dataset as the sequences that ORB-SLAM [138] failed to track on were removed. Since the original ORB-SLAM depth maps are in arbitrary scales for different sequences, the scale invariant error employed by Ummenhofer *et al.* [191] and Eigen *et al.*[45] as shown in Equation 6.12 is used as the metric to evaluate the depths maps.

$$\text{Scale-Invariant Error}(D, D^*) = \sqrt{\frac{1}{n}\sum_i d_i^2 - \frac{1}{n^2}\left(\sum_i d_i\right)^2}, \tag{6.12}$$

where d_i is the difference in predicted log depth (D) and ground truth log depth (D^*) for the valid pixels n (pixels that contain non-zero depth values in the raw depth data)

Method	Scale Invariant Error
Sparse Map ([138])	0.492
Cross-bilateral Filter[188]	0.491
Colorization [119]	0.372
Depth Map Prediction ([44])	0.159
Ours	**0.144**

Table 6.1: Quantitative results of inpainting ORB-SLAM maps on the NYU[140] dataset.

Quantitative results of the experiment are summarized in Table 6.1. The error reported for the ORB-SLAM[138] sparse depth maps is computed only on the points visible in the map and does not precisely correspond to that of denser error measures. However, the large error provides a strong indication that these ORB-SLAM maps contain gross outliers. While the raw image based inpainting techniques of [188] and [119] demonstrate improvement over the sparse depth maps alone, the neural network depth prediction of Eigen *et al.* significantly reduces the scale invariant error by decoding the structural information efficiently from raw image data. We demonstrate that the scale invariant error of the sparse depth map can be further improved by fusing the sparse depth map with that of the neural network using the learned confidences.

In Table 6.2, we provide an ablation study justifying importance of confidence estimation. Incorporating confidences of both the sparse depth map and the single view depth predictor is important for obtaining more accurate reconstructions. Excluding the predicted confidences of either the sparse map or the dense depth predictor (or both) degrades the accuracy of the final depth map.

Figure 6.8 shows the sensitivity of the reconstruction as we vary the strength of the fully connected and the local grid connected terms of the CRF. It is evident that both the terms contribute to the performance in this case.

Qualitative results of ORB-SLAM densification are shown in Figure 6.9. Dense

Learned Confidence		Scale Invariant Error
Sparse Map	CNN Prediction	
x	x	0.150
x	✓	0.149
✓	x	0.145
✓	✓	**0.144**

Table 6.2: An ablation of the effects of incorporating learned pointwise confidences of the sparse map and predicted depth map.

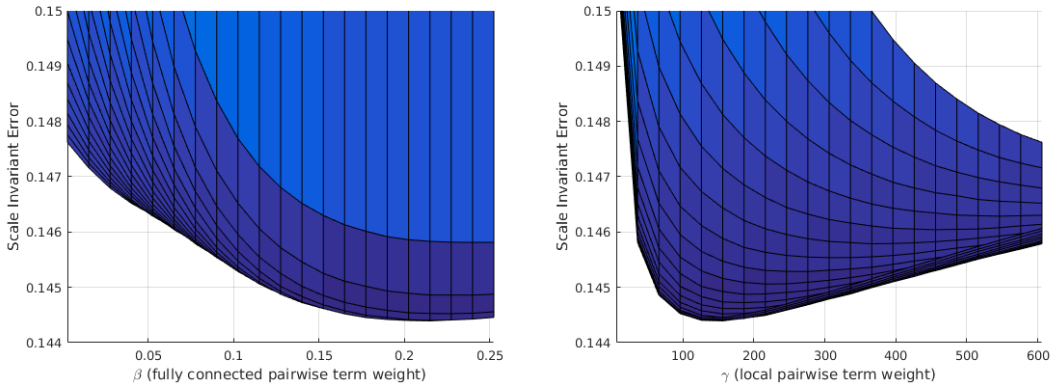

Figure 6.8: plots showing the scale invariant error (on vertical axis) of our reconstructions on NYU dataset as we change the hyperparameters (on horizontal axis): β (left) and γ of our CRF, with $\alpha = 10$. Black curve at the bottom in each plot represents best reconstruction error against varied fully connected terms strength and locally connected terms strength respectively

predictions generated from the CNN of Eigen *et al.* [44] are often inaccurate at the edges as well as at regions which are further away from the camera. Our confidence prediction networks help to reduce most of the depth errors and gross outliers from making into the fused result by predicting the confidences of the input depth data (even for the sparse ORB-SLAM depth maps which are at arbitrary scale). An added advantage of our approach is the ability to generate the fused result to be closer to metric scale if needed. This can be achieved by simply using y^p in the unary terms, and y^s in the fully-connected term of our energy formulation (instead of the other way around), and re-tune the hyper-parameters α, β, and γ.

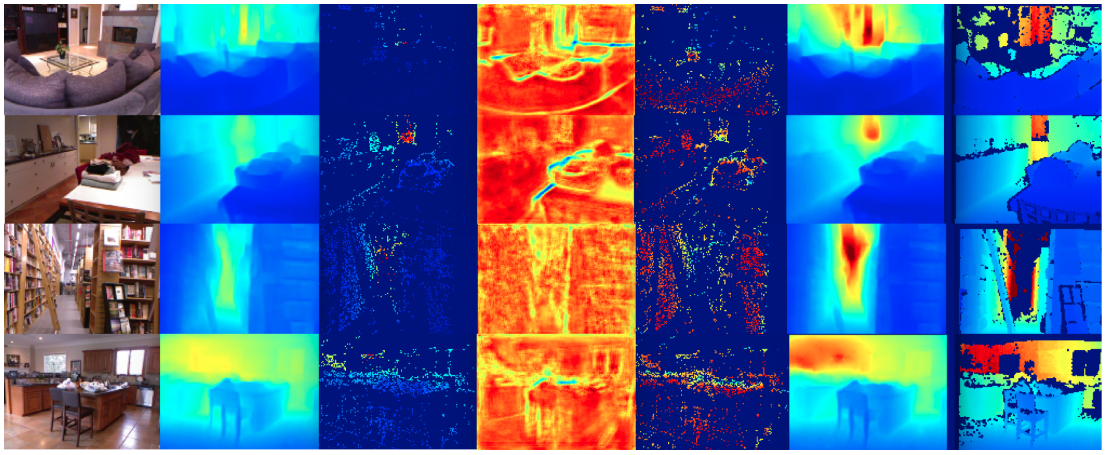

Figure 6.9: ORB-SLAM inpainting results for some of the images in the NYU test set. In column order: (1) RGB image, (2) Depth map prediction ([44]), (3) ORB-SLAM depth map, (4) Predicted confidences using our method for depth predictions (red implies higher confidence), (5) Predicted confidence using our method for the sparse map, (6) Ours, (7) Ground truth.

6.4.1 Inpainting Sparse Depth Maps from Depth Sensor

We also evaluate our method for inpainting sparse sensory data captured in indoor and outdoor environments. We found that for this task using the unary and local pairwise terms were sufficient to generate an accurate fused result. This is because unlike the ORB-SLAM maps, the sensory data is largely accurate with little to no outliers, and thus does not require a fully-connected pairwise neighbourhood of learned depth relationships to rectify the existing sensor data. Hence, we can fill-in the missing depth values based on the local depth relationships of the single view depth predictions (only the local pairwise weight γ need to be tuned on a validation set for a fixed α).

We first evaluate our method for inpainting Kinect depth maps on our NYU test subset, again using the network in [45] as the virtual depth sensor. To quantitatively evaluate the results, while respecting the structured sparsity pattern, we introduce a synthetic version of the NYU test set where a random rectangular

Table 6.3: Inpainting results on the NYU dataset for Kinect depth maps which are further sparsified by removing random crops. The results are evaluated against the original (downsampled) Kinect depth maps, on a *subset* of [44]'s test split (the same subset used for the ORB-SLAM inpainting experiments). Note that it is a common practice in the literature to evaluate against the original depth resolution of 640x480. However we performed inpainting at the resolution of 147x109 to increase efficiency and for real-time performance, and therefore did our evaluation against 147x109 resolution Kinect depth maps, downsampled using the nearest neighbour method. The random crops which are of size 50x50 were removed from the downsampled Kinect depth maps.

Method	*lower better*			*higher better*		
	RMS_{lin}	RMS_{ln}	Rel_{abs}	δ	δ^2	δ^3
Eigen_{vgg} [44]	0.679	0.217	0.155	74.0%	95.2%	99.0%
Cross-bilateral filter [188]	0.249	0.077	0.023	97.5%	99.3%	99.8%
Colorization [119]	0.200	0.059	0.019	98.4%	99.7%	99.9%
Ours	**0.169**	**0.047**	**0.018**	**99.1**%	**99.9**%	**100.0**%

region is cropped from the Kinect depth map to be labelled missing and then inpainted using the rest of the visible depth map. These results are tabulated in Table 6.3. The neural network prediction of Eigen *et al.* is chosen as the baseline for this scenario. Both the colorization[119] and the cross-bilateral filtering approach [188] use the neural network prediction as well as the Kinect depth (after the removal of the random crop) as the inputs and improve upon the stand-alone neural network prediction. As we use the knowledge of the confidence of the neural network prediction in to account in our approach, we are able to generate more accurate reconstructions.

Figure 6.10 shows the qualitative comparison of the results of our learnable confidence based inpainting with that of the baselines. It is clear that the proposed method produce more realistic depth maps as opposed to the mostly piecewise constant and inaccurate depth maps of the competing approaches.

Next we evaluate our method for inpainting sparse LIDAR maps in the KITTI dataset. To facilitate quantitative evaluation we remove $2/3^{rd}$ of the map points (respecting the sparsity structure of the LIDAR data) and evaluate the inpainted results against the removed points. Quantitative results on the test set are shown

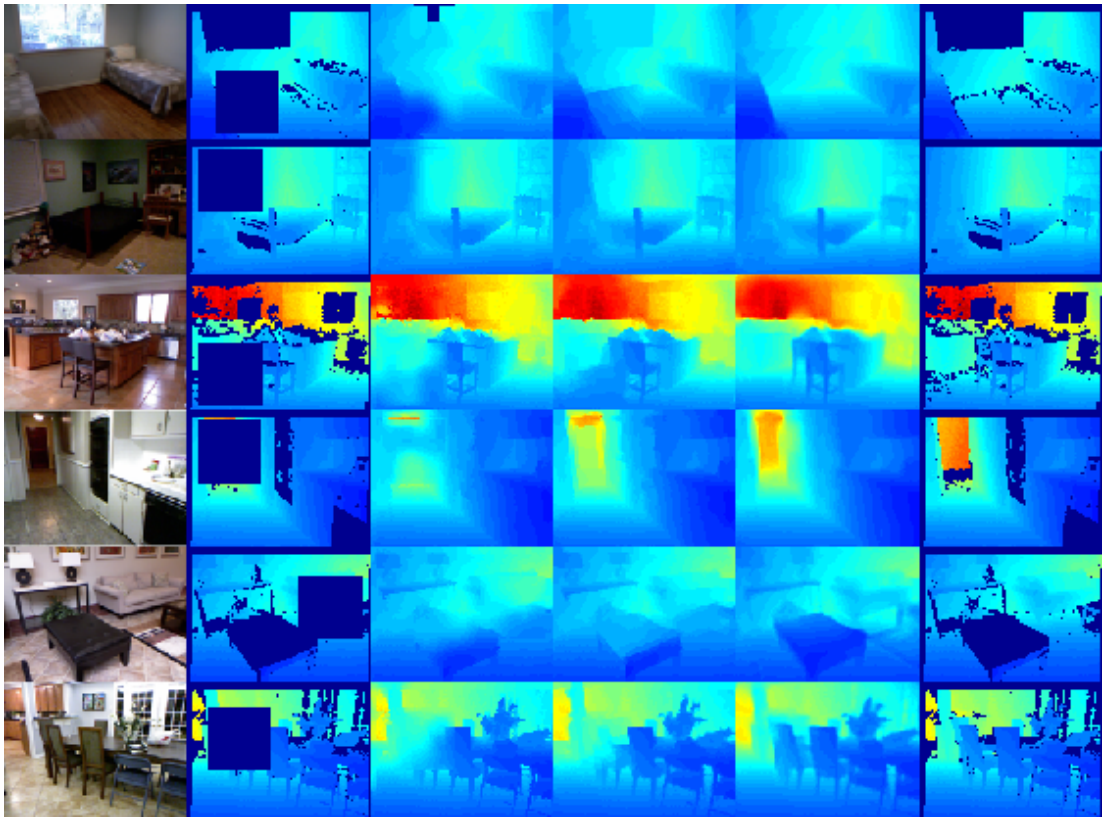

Figure 6.10: Comparison of methods for inpainting Kinect depth maps from the NYU test set with randomly removed data. In column order, 1^{st}: RGB image, 2^{nd}: Kinect depth map with randomly removed data, 3^{rd}: Inpainted depth map using cross-bilateral filtering, 4^{th}: Inpainted depth map using Colorization, 5^{th}: Our reconstruction result, 6^{th}: Raw Kinect depth map.

in Table 6.4 where our inpainting method improves greatly over the baseline CNN predictions of [62].

Qualitative examples of LIDAR inpainting are shown in Figure 6.11 where we can observe sharper object boundaries in our inpainted result, in comparison to the blurred object boundaries in the predicted depth map due to the loss in resolution that very deep feed forward CNNs suffer from. Most other errors in scene structure in the depth predictions have also been corrected in our fused result.

Table 6.4: Inpainting results on the KITTI dataset for LIDAR depth maps which are further sparsified by randomly removing a 2/3rd of its points. The results are evaluated against the original (downsampled) sparse LIDAR depth maps, in a subset of [45]'s test split. Note that it is a common practice in the literature to evaluate against the original depth resolution of 1242x375 within a masked pixel region of [44:1196, 153:370] (with 0-based indexing). However we performed inpainting at the resolution of 608x160 to increase efficiency and for real-time performance, and therefore did our evaluation against 608x160 resolution LIDAR depth maps, downsampled using the nearest neighbour method, within a masked pixel region of [21:585, 65:157]

Method	*lower better*			*higher better*		
	RMS_{lin}	RMS_{ln}	Rel_{abs}	δ	δ^2	δ^3
Garg *et al.* [62]	5.555	0.285	0.180	69.2%	90.3%	96.5%
Cross-bilateral filter [188]	3.854	0.195	0.113	85.3%	95.5%	98.3%
Colorization [119]	1.956	0.104	0.048	96.5%	98.9%	99.5%
Ours	**1.838**	**0.092**	**0.032**	**96.9%**	**99.0%**	**99.6%**

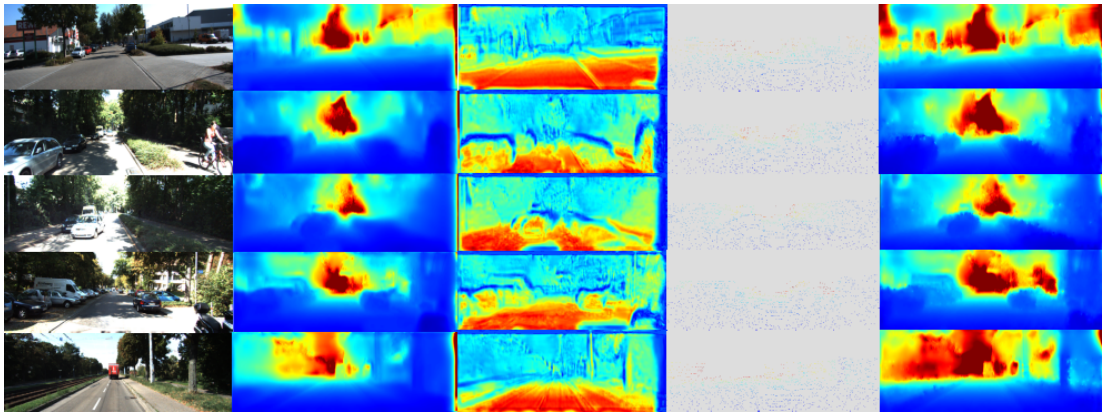

Figure 6.11: Inpainting LIDAR depth maps. In column order, 1^{st}: RGB image, 2^{nd}: Depth predictions [62], 3^{rd}: Predicted confidences using our method for depth predictions (red denotes higher confidence), 4^{th}: LIDAR depth map 5^{th}: Our output

6.4.2 Latency

All experiments were conducted on a machine with an NVIDIA GTX 980 GPU and Intel i7 4790 CPU. ORB-SLAM inpainting as well as Kinect depth map inpainting on the NYU dataset can be performed at approximately $30ms$ at 147×109 image resolution, which is the same resolution as the predicted depth map by [45]. However, if these tasks are performed at the full image resolution

of 640x480 by upsampling the depth predictions, the inference time increases to $\approx 200ms$. Inference time for LIDAR depth map inpainting on the KITTI dataset is $\approx 100ms$, if the reconstruction is performed at the same resolution as that of predicted depth map by Garg *et al.*[62] of 608×160. Total overhead time for neural network depth and confidence predictions is $\approx 50ms$ for both datasets.

6.4.3 Additional Qualitative results

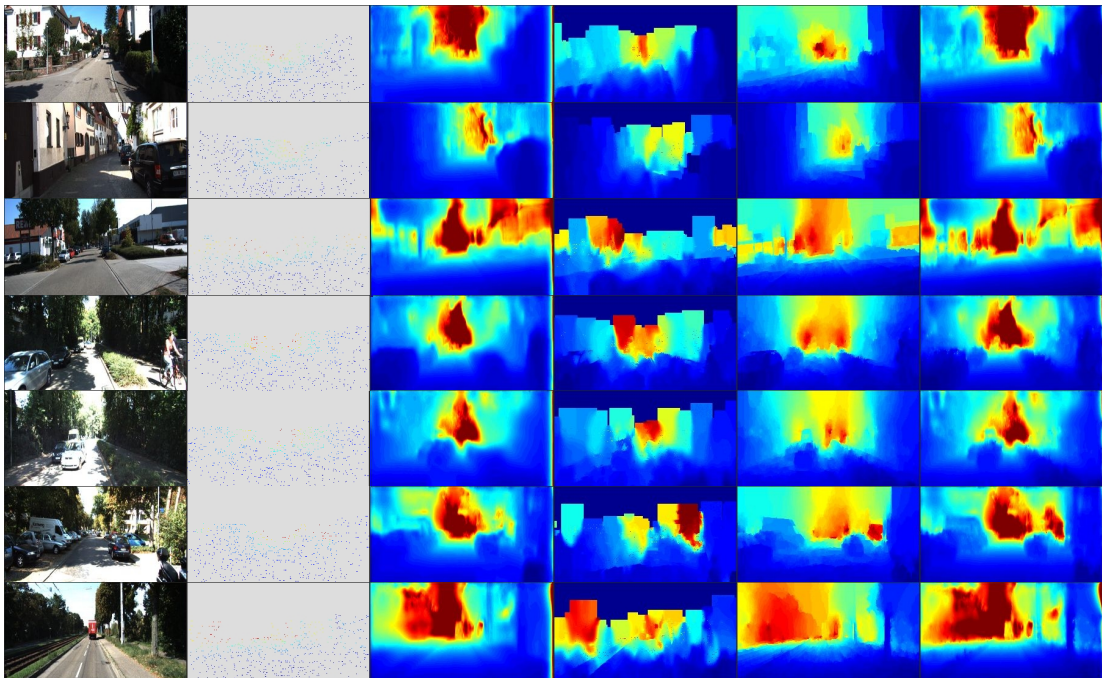

Figure 6.12: Comparison of our method against baselines for inpainting LIDAR depth maps in the KITTI dataset. In column order: (1) RGB image, (2) LIDAR depth map (3) Depth map prediction ([62]), (4) Cross-bilateral filter, (5) Colorization, (6) Our reconstruction result.

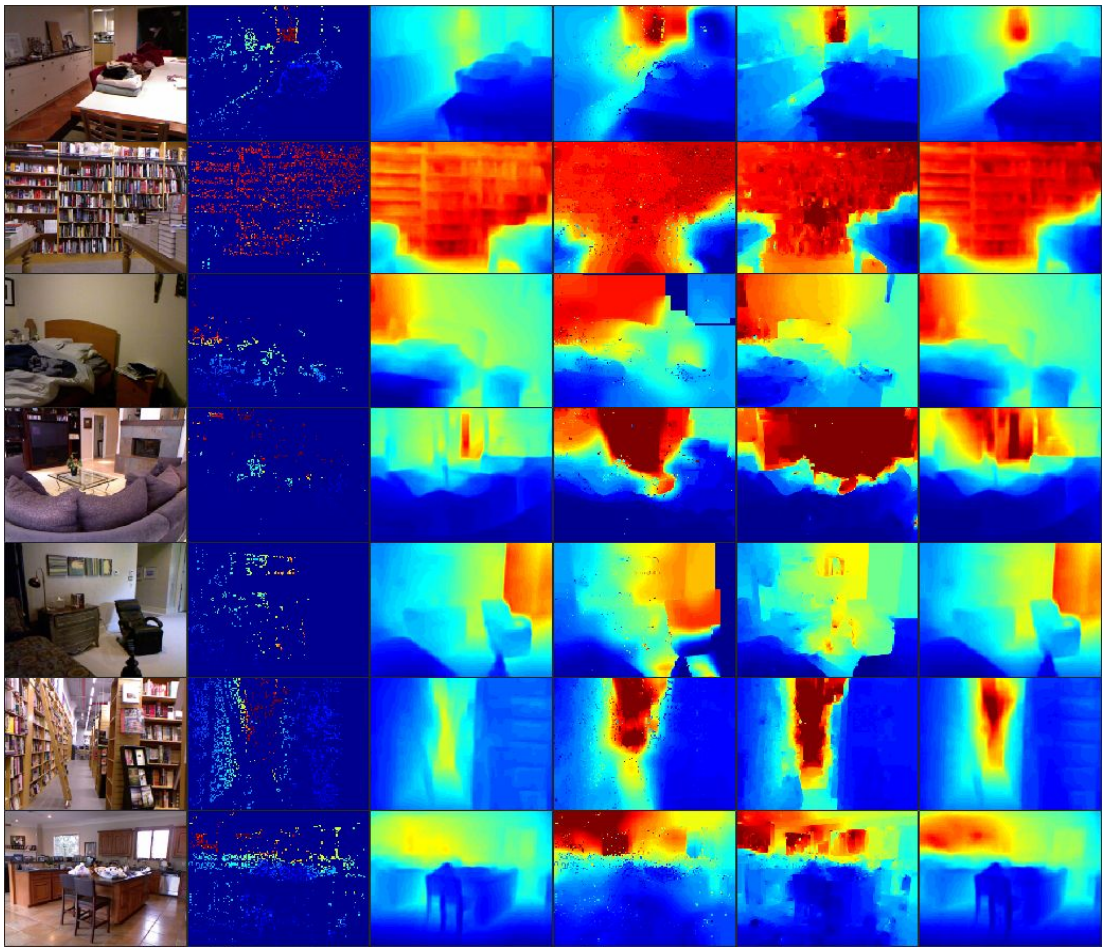

Figure 6.13: Comparison of our method against baselines for inpainting ORB-SLAM depth maps in the NYU dataset. In column order: (1) RGB image, (2) ORB-SLAM depth map, (3) Depth map prediction ([44]), (4) Cross-bilateral filter, (5) Colorization, (6) Our reconstruction result.

6.5 Summary

In this chapter, we presented a framework that can be used to generate a dense reconstruction of a scene on demand given a sparse depth map as well as a CNN capable of predicting a depth map from a single image. This approach is particularly useful during large-scale mapping as well as during relocalization where it's inefficient to store detailed information about the map and all the corresponding images.

We modelled the task as inference over a novel fully connected CRF model, with nodes anchoring the solution to the sparse map and the scale-invariant edges enforcing pairwise depth relationship information based on depth predictions of a deep neural network, given the live RGB image. The CRF model was also parametrised by the point-wise confidences of both the sparse map and the dense depth prediction. These confidences were predicted using CNNs, given the depth maps and the live RGB image as the input. This form of probabilistic data fusion allowed the solution to be less sensitive to the presence of erroneous depths, and a simple relaxation of the pairwise confidence weight enabled efficient inference of the solution. We applied our method to perform real-time image-guided inpainting of sparse maps obtained from three different sources: a sparse monocular SLAM framework, Kinect and LIDAR.

CHAPTER 7

Depth and Relative-Pose Estimation using CNNs

7.1 Introduction

The previous two chapters were largely centred around manipulating the depth prediction from a neural network in order to perform a range of tasks including predicting correlated information and denisifying sparse depth maps. In this chapter, the tracking and mapping problem is revisited from a different perspective where a CNN is once again used as the workhorse of the framework. However, instead of replacing every component of the SLAM pipeline with CNNs, this work demonstrates it's better to use CNNs for tasks that greatly benefit from feature extraction (depth and optical flow estimation), while using geometry for tasks its proven to work well (motion estimation given the depths and flow)

In this work, we draw from both machine learning approaches as well as SfM techniques to create a unified framework which is capable of predicting the depth of a scene and the motion parameters governing the camera motion between an image pair. We construct our framework incrementally where the network is first

trained to predict depths given a single colour image. Then a colour image pair as well as their associated depth predictions are provided to a flow estimation network which produces an optical flow map along with an estimated measure of confidence in x and y motion. Finally, the pose estimation block utilises the outputs of the previous networks to estimate a motion vector corresponding to the logarithm of the Special Euclidean Transformation $\mathbb{SE}(3)$ in $\mathbb{R}^3$, which describes the relative camera motion from the first image to the second.

7.2 Contributions

The work presented in this chapter was done in collaboration with Andrew Spek and the relative contributions were equal in all areas. The main body of the chapter is largely based on the contents of the submitted work [40] and is expected to appear in a similar form in Andrew's thesis.

The main contributions are summarised below:

- We present a unified framework which predicts depth, optical flow, flow confidences and relative camera pose.
- We present the first approach to use a neural network to predict the full information matrix which represents the confidence of the optical flow estimate.
- We achieve state-of-the-art results for single image depth prediction on both NYUv2 (indoor) and KITTI (outdoor) datasets.
- We outperform previous camera motion prediction frameworks on both TUM and KITTI datasets.
- We actively combat scale drift using the knowledge of metric depths and subsequently produce more accurate visual odometry estimates.

7.3 Related Work

The existing works in the literature that are related to the ideas presented in this chapter, specially in the areas of optical flow prediction and pose estimation using neural networks are summarised below. Previous depth prediction approaches were discussed in the background chapter and can be found in Section 2.2.2

7.3.1 Optical Flow Prediction

Optical flow estimation is the process of computing the apparent motion field of objects in a scene caused by the relative motion between the camera and the scene itself. Popular computational approaches are based on a variational formulation and an associated energy minimisation problem led by the pioneering work of Horn and Schunck [86]. A range of modifications have been proposed since then in the works of [185, 183]. An alternative approach is to predict the motion field using a CNN given an image pair.

An early work in optical flow prediction using CNNs was FlowNet [53]. This was later extended by Ilg *et al.* to FlowNet 2.0 [92] which included stacked FlowNets [53] as well as warping layers. Ranjan and Black proposed a spatial pyramid based optical flow prediction network [158]. More recently, Sun *et al.* proposed a framework which uses the principles from geometry based flow estimation techniques such as image pyramid, warping and cost volumes in [184]. As our end goal revolves around predicting camera pose, it becomes necessary to isolate the flow that was caused purely from camera motion. In order to achieve this we extend upon these previous works to predict both the optical flow and the associated information matrix of the flow. Although not in a CNN context [197] showed the usefulness of estimating flow and uncertainty.

7.3.2 Pose Estimation

CNNs have been successfully used to estimate various components of a Structure from Motion pipeline. Earlier works focused on learning discriminative image based features suitable for ego-motion estimation [2, 95]. Yi *et al.*[204] showed a full feature detection framework can be implemented using deep neural networks. Rad and Lepetit in BB8[157] showed the pose of objects can be predicted even under partial occlusion and highlighted the increased difficulty of predicting 3D quantities over 2D quantities. Kendall and Cipolla demonstrated that camera pose prediction from a single image catered for relocalization scenarios [100].

However, all of the above works lack a representation of structure as they do not explicitly predict depths. The work presented in this chapter is more closely related to that of Zhou *et al.* [212] and Ummenhofer *et al.* [191] and their frameworks SfM-Learner and DeMoN. Both of these approaches also predict a single confidence map in contrast to ours which estimates the confidence in x and y directions separately. Since our framework predicts metric depths in comparison to theirs, we are able to produce far more accurate visual odometry and combat against scale drift. CNN SLAM by Tateno *et al.* [187] incorporated depth predictions of [114] into a SLAM framework. Our method performs competitively with CNN-SLAM as well as ORB-SLAM[139] and LSD-SLAM[48] despite solely computing sequential frame-to-frame alignments, whereas the SLAM approaches have the added advantage of performing loop closures and local/global bundle adjustments.

7.4 Method

7.4.1 Network Architecture

The overall architecture consists of 3 subnetworks in the form of a depth, flow and camera pose network. A large percentage of the model capacity is invested into the depth prediction component for two reasons. Firstly, the output of the depth network also serves as an additional input to the other subsystems. Secondly, we wanted to achieve superior depths for indoor and outdoor environments using a common architecture. In order to provide an overall understanding of the data flow, a high level diagram of the network is shown in Figure 7.1. Expanded architectures for each of the subnetworks is included in the respective subsections.

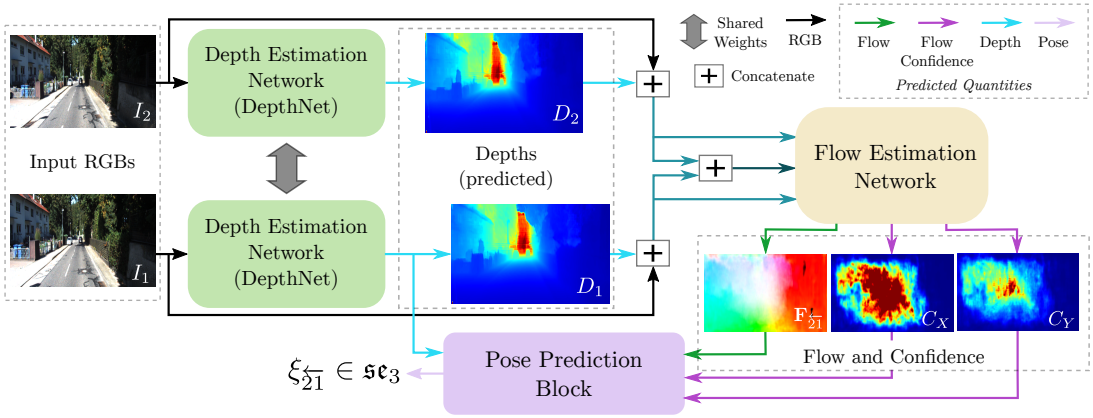

Figure 7.1: Overview of our system full pipeline. Please note that we use the notation $\overleftarrow{ij}$ to indicate from j to i

7.4.2 Depth Prediction

The depth prediction network consists of a feature encoder module followed by a decoder. The encoder network is largely based on the DenseNet161 architecture

described in [90]. In particular we use the variant pre-trained on ImageNet [167] and slightly increase the receptive field of the pooling layers. As the original input is down-sampled 4 times by the encoder, during the decoding stage the feature maps are up-sampled back 4 times to make the model fully convolutional. We employ skip connections in order to re-introduce the finer details lost during pooling. Since the first down-sampling operation is done at a very early stage of the pipeline and closely resemble the image features, these activations are not reused inside the decoder. Up-project blocks are used to perform up-sampling in our network, which provide better depth maps compared to de-convolutional layers as shown in [114].

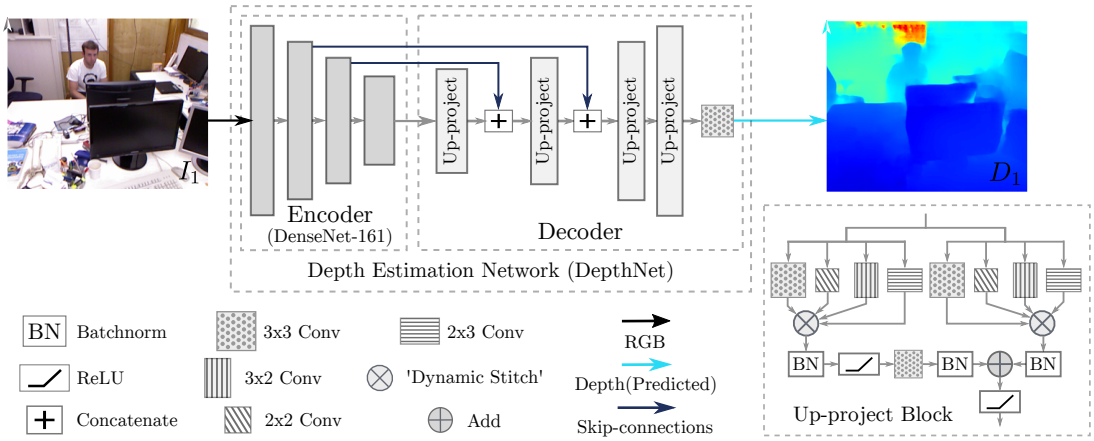

Figure 7.2: The depth estimation uses an architectural combination of the powerful classification network 'DenseNet-161' described in [90] and the 'Up-project' style blocks proposed in [114].

Due to the availability of dense ground truth data for indoor datasets (e.g NYUv2 [140], RGB-D[182]) this network can be directly utilised to perform supervised learning. Unfortunately, the ground truth data for the outdoor datasets (KITTI) are much sparser and meant we had to incorporate a semi-supervised learning approach in order to provide a strong training signal. Therefore, during training on KITTI, we use a Siamese version of the depth network with complete weight-sharing, and enforce photometric consistency between the left-right image pairs through an additional loss function. This is similar to the previous approaches [62, 111] and is only required during the training stage, during inference only a

single input image is required to perform depth estimation using our network.

7.4.3 Flow Prediction

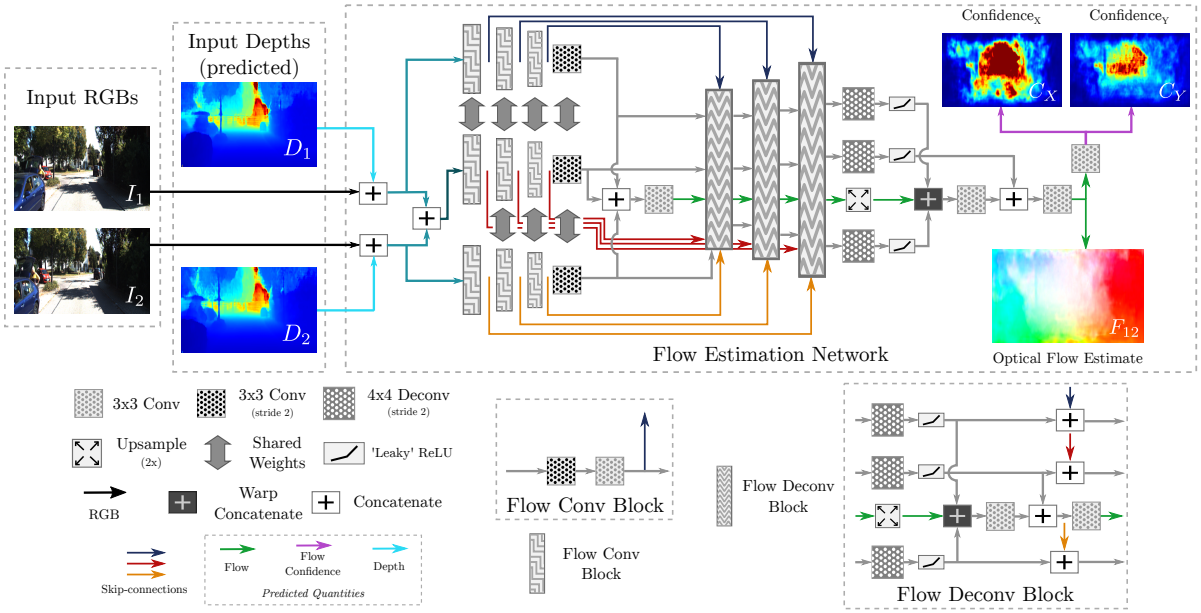

Figure 7.3: The flow network has 3 streams where all the layers except the first layer have shared weights.

The flow network has three streams. The first stream takes the left image and its' predicted depth map as the input, the second stream receives the right image and the corresponding predicted depth map and finally, the third stream receives both the left and right images and their associated depth predictions. Barring the first layer, all other layers of each stream share their weights. During the decoder stage the predicted flow is used to perform warp concatenations, where the right images activations are warped and concatenated with that of the left image. Since we are estimating optical flow in a coarse to fine manner, where the latter layers compute a residual to be added to the initial flow estimate, warp-concatenations help to capture the small displacements more effectively.

The output of the flow subnetwork is an estimation of the optical flow along with the associated confidences given an image pair. These outputs combined with predicted depths allow us to predict the camera pose. As part of the ablation studies we integrated the flow predictions of [92] with our depths, however, the main limitation of their approach was the lack of a mechanism to filter out the dynamic objects which are abundant in outdoor environments. This was solved by estimating confidence, specifically the information matrix in addition to the optical flow. More concretely, for each pixel our flow network predicts 5 quantities, the optical flow $\mathbf{F} = [\Delta u, \Delta v]^T$ in the x and y direction, and the quantities $\hat{\alpha}$, $\hat{\gamma}$ and $\hat{\beta}$, which are required to compute the information matrix ($\boldsymbol{\mathcal{I}}$) or the inverse of the covariance matrix as shown below.

$$\mathcal{I} = \begin{bmatrix} C_x & C_{xy} \\ C_{xy} & C_y \end{bmatrix}, \qquad C_x = e^{\hat{\alpha}}, C_y = e^{\hat{\gamma}}, C_{xy} = e^{\frac{\hat{\gamma}+\hat{\alpha}}{2}} \tanh(\hat{\beta}). \tag{7.1}$$

This parametrisation guarantees $\mathcal{I}$ is positive-definite and can be used to parametrise any 2×2 information matrix. We found that the gradients are much more stable compared to predicting the information matrix directly as the determinant of the matrix is always greater than zero since $\tanh(\hat{\beta}) = \pm 1$ only when $\hat{\beta} \to \pm\infty$.

7.4.4 Pose Estimation

We take two approaches to pose estimation, as shown in Figure 7.4, an iterative and a fully-connected(FC). This contrasts the ability of a neural network to estimate using the available information, and the simplicity of a standard computer vision approach using the available predicted quantities. We use FC layers to provide the network with as wide a receptive field as possible, to compare more equivalently against using the inferred quantities in the iterative approach.

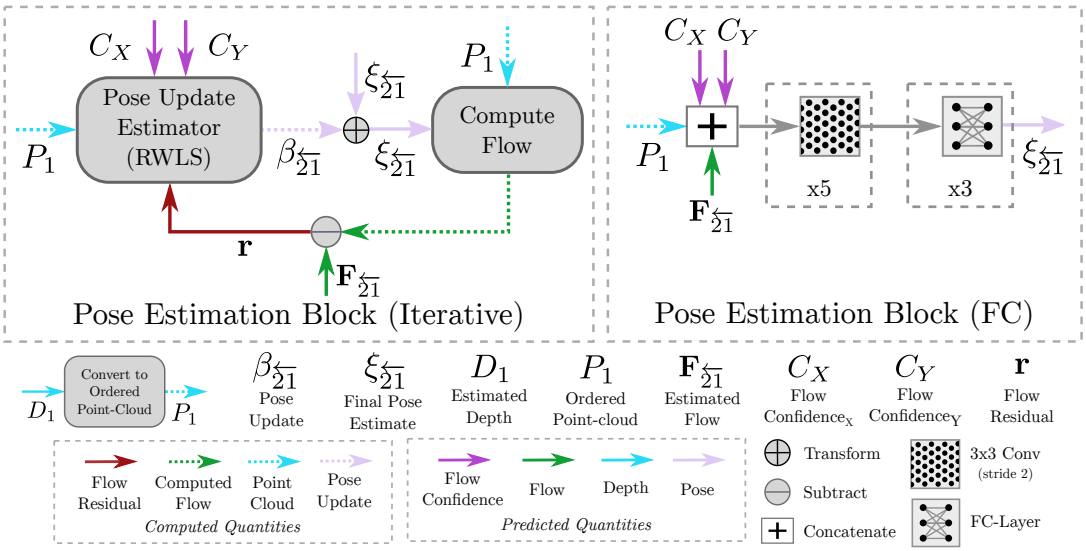

Figure 7.4: We detail the two approaches we took to estimating the relative pose alignment between adjacent frames (best viewed in colour). *Top-Left* shows the iterative approach we took, that incorporates a re-weighted least-squares solver (RWLS) into a pose estimation loop. Details in Section 7.4.4. *Top-Right* shows our fully-connected (FC) approach, which incorporates a succession of 3×3 strided convolutions, followed by several FC layers. Detailed in Section 7.4.4. *Bottom* we include a summary of the information in the figure

Iterative

This approach uses a more conventional method for computing relative pose estimates where the predicted depth, optical flow and the confidences are used in an iterative formulation. We describe this process in a step by step manner.

The predicted flow $\mathbf{F}_{\mathbf{pred}\overleftarrow{21}}$ generated by the neural network in pixel coordinates can be used to obtain the predicted flow $\mathbf{F}_{\overleftarrow{21}}$ in normalised camera coordinates as follows:

$$\mathbf{F}_{\overleftarrow{21}} = \begin{pmatrix} \frac{1}{f_x} \\ \frac{1}{f_y} \end{pmatrix} \odot \mathbf{F}_{\mathbf{pred}\overleftarrow{21}} \tag{7.2}$$

Given an initial estimate of the relative transformation between the two frames and the depth map of the first frame, an estimate of the optical flow $\mathbf{F}^{+}_{\overleftarrow{21}}(\mathbf{u}_i)$ can also be computed in normalised camera coordinates :

$$\mathbf{F}^{+}_{\overleftarrow{21}}(\mathbf{u}_i) = \begin{pmatrix} \frac{1}{f_x} \\ \frac{1}{f_y} \end{pmatrix} \odot (\pi(\mathrm{KT}_{\overleftarrow{21}}\mathbf{x}_i) - \pi(\mathrm{KT}_{\mathrm{init}}\mathbf{x}_i)) \tag{7.3}$$

where $\mathbf{u}_i = \begin{bmatrix} x & y \end{bmatrix}^T_i$ is the i^{th} pixel coordinate, $\mathbf{x} \in P_1$ is the i^{th} inverse depth coordinate $\mathbf{x} = \begin{bmatrix} u & v & 1 & q \end{bmatrix}^T$ of an ordered point cloud (P_1). $\pi(x) = ((x_0/x_2),(x_1/x_2))^T$ is a normalisation function where $x \in \mathbb{R}^3$, K is the camera intrinsic matrix. $\mathrm{T}_{\mathrm{init}} \in \mathbb{SE}(3)$ is a constant denoting the identity, and $\mathrm{T}_{\overleftarrow{21}} \in \mathbb{SE}(3)$ represents the current estimate of the transformation from frame 1 to 2. Note that at the the start of the first iteration, $\mathrm{T}_{\overleftarrow{21}} = T_{\mathrm{init}}$. With the predicted and estimated optical flow defined in normalised camera coordinates as above, the loss criterion we attempt to minimize iteratively can now be expressed as:

$$\mathbf{e} = \sum_{i=1}^{N} \mathbf{r}_i^T \mathcal{I}_i \mathbf{r}_i \quad \text{where} \qquad \mathbf{r}_i = (\mathbf{F}^{+}_{\overleftarrow{21}}(\mathbf{u}_i) - \mathbf{F}_{\overleftarrow{21}}(\mathbf{u}_i)) \tag{7.4}$$

$\mathcal{I}_i$ is the information matrix of the ith pixel. To simplify the mathematics we can represent the transformation using a matrix exponential as $\mathrm{T}_{\overleftarrow{21}} = e^{\sum_{j=0}^{6} \alpha_j \mathbf{G}_j}$, where $\alpha_j \in \xi_{\overleftarrow{21}}$ is the j^{th} component of the motion vector $\xi_{\overleftarrow{21}} \in \mathbb{R}^6$, which is a member of the Lie-algebra $\mathfrak{se}_3$, and $\mathbf{G}_j$ is the generator matrix corresponding to the relevant motion parameter. We can now differentiate the residual function with respect to the motion parameters to generate the following Jacobian

$$\mathrm{J}_i = \begin{bmatrix} q & 0 & -uq & -uv & u^2+1 & -v \\ 0 & q & -vq & -v^2-1 & uv & u \end{bmatrix}, \tag{7.5}$$

where J_i is the i^{th} Jacobian, which can be stacked to form a larger Jacobian

matrix $\mathrm{J} = \left[\mathrm{J}_1^T, \mathrm{J}_2^T, \ldots, \mathrm{J}_N^T\right]^T$, additionally the residual vectors can be stacked $\mathbf{r} = \left[\mathbf{r}_1^T, \mathbf{r}_2^T, \ldots, \mathbf{r}_N^T\right]^T$. This allows us to iteratively reduce the loss function $\mathbf{e}$ using a standard Gauss-Newton approach given by

$$\beta = \left(\mathbf{J}^T\mathbf{W}\mathbf{J}\right)^{-1}\mathbf{J}^T\mathbf{W}\mathbf{r}, \tag{7.6}$$

where β is the additive update to the motion parameters $\xi_{\overleftarrow{21}}$, and W is a block diagonal weight matrix consisting of individual $\mathrm{W}_i = \mathrm{W}(\mathbf{u_i})$ weight matrices for each pixel of the image. The elements of the W_i matrix are given below:

$$\mathrm{W}_i = \frac{m^2}{m^2 + \mathbf{r}_i^T\mathcal{I}_i\mathbf{r}_i}\begin{bmatrix} C_X(\mathbf{u_i}) & C_{XY}(\mathbf{u_i}) \\ C_{XY}(\mathbf{u_i}) & C_Y(\mathbf{u_i}) \end{bmatrix}, \tag{7.7}$$

where m is a constant that is computed from the residual $\mathbf{r}_i$ (Equation 7.4), to be the mean residual magnitude of a single image. This pipeline is implemented in Tensorflow [1] and allows us to train the network end to end.

Fully-Connected

Similar to Zhou *et al.*[212] and Ummenhofer *et al.* [191] we also constructed a fully connected layer based pose estimation network. This network utilises 3 stacked fully connected layers and uses the same inputs as our iterative method. While we outperform the pose estimation benchmarks of [212] and [191] using this network the iterative network is our recommended approach due to its close resemblance to conventional geometry based techniques.

7.4.5 Loss Functions

Depth Losses

For supervised training on indoor and outdoor datasets we use a reverse Huber loss function [114] defined by

$$\mathcal{L}_{\mathrm{B}}(D_i, D_i^*) = \begin{cases} |D_i - D_i^*| & |D_i - D_i^*| < c, \\ ((D_i - D_i^*)^2 + c^2)/2c & |D_i - D_i^*| > c, \end{cases} \tag{7.8}$$

where $c = \frac{1}{5}max(D_i - D_i^*)$, and $D_i = D(\mathbf{u}_i)$ and $D_i^* = D^*(\mathbf{u}_i)$ represent the i^{th} predicted and the ground truth depth respectively. For the KITTI dataset we employed an additional photometric loss during training as the ground truth is highly sparse. This unsupervised loss term enforces left-right consistency between stereo pairs, defined by

$$\begin{aligned} \mathcal{L}_{\mathrm{C}} = &\frac{1}{n}\sum_{i=1}^{n} |I_L(\mathbf{u}_i) - I_R(\pi(\mathrm{KT}_{\overleftarrow{RL}}\pi^{-1}(D_i^L, \mathbf{u}_i))| \\ &+\frac{1}{n}\sum_{i=1}^{n} |I_R(\mathbf{u}_i) - I_L(\pi(\mathrm{KT}_{\overleftarrow{LR}}\pi^{-1}(D_i^R, \mathbf{u}_i))|, \end{aligned} \tag{7.9}$$

where I_L and I_R are the left and right images and D_i^L and D_i^R are their corresponding depth maps, $\pi^{-1}(D, \mathbf{u}) = D\mathrm{K}^{-1}(\mathbf{u})$ is the transformation from pixel to camera coordinates, and $\mathrm{T}_{\overleftarrow{RL}} \in \mathbb{SE}(3)$ and $\mathrm{T}_{\overleftarrow{LR}} \in \mathbb{SE}(3)$ define the relative transformation matrices from left-to-right and right-to-left respectively. In this case the rotation is assumed to be the identity and the matrices purely translate in the x-direction. Additionally, we use a smoothness term defined by

$$\mathcal{L}_{\mathrm{S}} = \frac{1}{n}\sum_{i=1}^{n} \left(|\nabla_x D_i| + |\nabla_y D_i|\right), \tag{7.10}$$

where ∇_x and ∇_y are the horizontal and vertical gradients of the predicted depth. This provides qualitatively better depths as well as faster convergence. The complete loss function used to train KITTI depths is given by

$$\mathcal{L}_{ss} = \lambda_1 \mathcal{L}_{\mathrm{B}} + \lambda_2 \mathcal{L}_{\mathrm{C}} + \lambda_3 \mathcal{L}_{\mathrm{S}}, \qquad \lambda_1 = 2,\, \lambda_2 = 1\ ,\, \lambda_3 = e^{-4}\ , \tag{7.11}$$

where $\mathcal{L}_{\mathrm{B}}$ and $\mathcal{L}_{\mathrm{S}}$ are computed on both left and right images separately.

Flow Loss

The probability distribution of a multivariate Gaussian in 2D can be defined as follows.

$$p(\mathbf{x}|\mu, \boldsymbol{\mathcal{I}}) = \frac{|\boldsymbol{\mathcal{I}}|^{\frac{1}{2}}}{2\pi} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \boldsymbol{\mathcal{I}} (\mathbf{x}-\mu)}, \tag{7.12}$$

where $\boldsymbol{\mathcal{I}} = \Sigma^{-1}$ is the information matrix or inverse covariance matrix Σ^{-1}. The flow loss $\mathcal{L}_F$ criterion can now be defined by

$$\mathcal{L}_F = \frac{1}{2}((\mathbf{F}_{\overleftarrow{21}} - \mathbf{F}^*_{\overleftarrow{21}})^T \mathcal{I} (\mathbf{F}_{\overleftarrow{21}} - \mathbf{F}^*_{\overleftarrow{21}}) - \log(|\boldsymbol{\mathcal{I}}|)), \tag{7.13}$$

where $\mathbf{F}_{\overleftarrow{21}}$ is the predicted flow, and $\mathbf{F}^*_{\overleftarrow{21}}$ is the ground truth flow. This optimises by maximising the log-likelihood of the probability distribution over the residual flow error.

Pose Loss

Given two input images I_1, I_2, the predicted depth map D_1 of I_1 and the predicted relative pose $\xi_{\overleftarrow{21}} \in \mathbb{R}^6$ the unsupervised loss $\mathcal{L}_{\mathrm{U}}$ and pose loss $\mathcal{L}_{\mathrm{P}}$ can be defined

as

$$\mathcal{L}_{\mathrm{U}} = \frac{1}{n}\sum_{i=1}^{n} |I_1(\mathbf{u}_i) - I_2(\pi(\mathrm{KT}_{\xi_{21}}\pi^{-1}(D_1, \mathbf{u}_i))|, \text{and} \tag{7.14}$$

$$\mathcal{L}_{\mathrm{P}} = ||\xi_{\xi_{21}} - \log_e(\mathrm{T}^*_{\xi_{21}})||_2 = ||\xi_{\xi_{21}} - \xi^*_{\xi_{21}})||_2, \tag{7.15}$$

where $\log_e(\mathrm{T})$ maps a transformation T from the Lie-group $\mathbb{SE}(3)$ to the Lie-algebra $\mathfrak{se}_3$, such that $\log_e(\mathrm{T}) \in \mathbb{R}^6$ can be represented by its constituent motion parameters, and $\xi^*_{\xi_{21}}$ is the ground truth relative pose parameters.

7.4.6 Training Regime

We train our network end-to-end on NYUv2 [140], TUM[182] and KITTI[63] datasets. We use the standard test/train split for NYUv2 and KITTI and define our scene split for TUM. It is worth mentioning that the amount of training data we used is radically reduced compared to [212] and [191]. More concretely, for NYUv2 we use $\approx 3\%$ of the full dataset, for KITTI $\approx 25\%$. We use the Adam optimiser [104] with an initial learning rate of 1e-4 for all experiments and chose Tensorflow [1] as the learning framework and train using an NVIDIA-DGX1. The full training schedule for each of the subnets is given below.

Depth Training

All of the DenseNet-161 layers [90] of the depth nets are initialised using Imagenet[167] pretrained weights. Remainder of the layers are initialised using MSRC[78] initialisation. NYUv2[140] and TUM[182] models are trained purely using the supervised loss term. The network is regularized using a weight decay of $1e^{-4}$ through out training and the learning rate schedule is shown below :

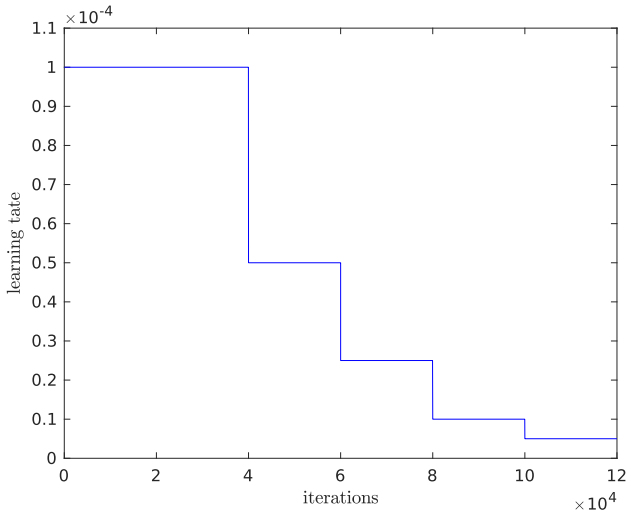

Figure 7.5: Learning rate schedule for NYUv2[140] depth training

Out of ≈ 400,0000 images in the NYU dataset, only 12,000 are used during training. We perform data augmentation 4 times (a total training set of 48000 images) using colour shifts, random crops and left-right flips. Although, data augmentation can be implemented during training we noticed a considerable speed up by performing this step in an offline stage. The training images and the corresponding ground truth are downsampled by a factor of 2. Each training batch contains 8 images and we use 4 GPUs, resulting in a overall batch size of 32. In terms of training speed we observe on average 19.3 examples/sec or 0.415 sec/batch.

For the KITTI dataset we use 10,000 training images. Out of the training images that were defined in [45] we further prune our training set to exclude any images that are part of the odometry test set. We adopt a learning rate schedule which spans for half the duration of the NYU. This is primarily to avoid over fitting as we are now working with a comparatively small training dataset.

Optical Flow Training

In order to compute the ground truth optical flow image, for the NYUv2 [140] dataset we first compute the camera pose using the Iterative Closest Point (ICP) algorithm which can then be used with the ground truth depth map to compute optical flow. This process is slightly simplified for the TUM[182] dataset as the ground truth pose is provided. The network is then trained using the optical flow loss criterion.

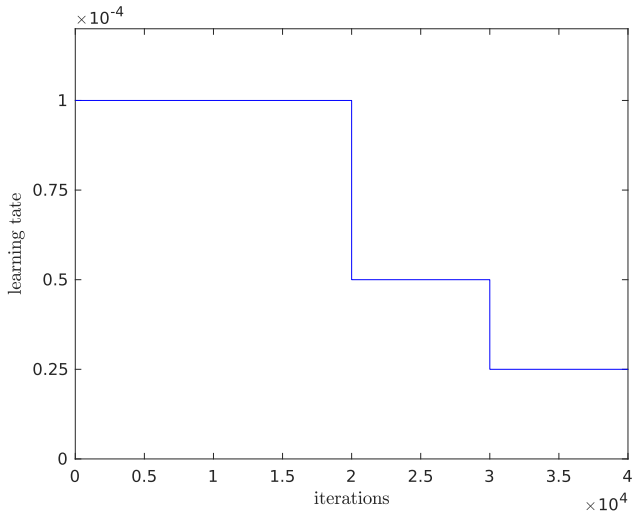

Figure 7.6: Learning rate schedule for optical flow training

All the layers of the flow network are initialised using the MSRC [78] initialisation and the learning rate schedule is shown in Figure 7.6. As it can be seen, the training duration is much smaller compared to the depth network training as the primary objective at this stage is to obtain a crude representation for both optical flow and the information matrix. Complete end-to-end fine tuning happens when the network is trained using the pose loss criterion.

Pose Training

We optimize the full network end-to-end using the pose loss and demonstrate that the state-of-the art depths can be further improved using the knowledge of pose. The network is trained for 20,000 iterations with an initial learning rate of $1e^{-5}$ which is halved at the half-way point.

7.5 Results

In this section we summarise the single-image depth prediction and relative pose estimation performance of our system on several popular machine learning and SLAM datasets. The effect of using alternative optical flow estimates from [92] and [161] in our pose estimation pipeline is also investigated as an ablation study. A video demonstration of the full pipeline can be found under `https://youtu.be/JPOEGRzo5kw`. This video was awarded the best technical demonstration at the annual Symposium of the Australian Centre of Excellence of Robotic Vision in 2018.

7.5.1 Depth Estimation

The performance of the single-image depth estimation subnet across the datasets NYUv2[140], RGB-D[182] and KITTI[63] are tabulated in Tables 7.1, 7.3 and 7.2 respectively. Once again the standard metrics defined in Section 5.6.1 are used to evaluate the predictions.

We train *Ours(baseline)* model to showcase the improvement we get by purely using the depth loss. This is then extended to use the full end-to-end training loss (depth + flow + pose losses) in the *Ours(full)* model which demonstrates a

consistent improvement across all datasets. Most notably in Tables 7.2 and 7.3 for which ground truth pose data was available for training. This validates our approach for improving single image depth estimation performance, and demonstrates a network can be improved by enforcing more geometric priors on the loss functions. We would like to mention that the improvement we gain from *Ours(baseline)* to *Ours(full)* is purely due to the novel combined loss terms as the flow and pose sub networks do not increase the model capacity of the depth subnet itself.

Table 7.1: The performance of several approaches evaluated on single-image depth estimation using the standard test set of NYUv2[140] proposed in [44]

Method	*lower better*			*higher better*		
	RMS_{lin}	RMS_{ln}	Rel_{abs}	δ	δ^2	δ^3
Eigen_{vgg} [44]	0.641	0.214	0.16	76.9%	95.0%	98.8%
Laina *et al.*[114]	0.573	0.195	0.13	81.1%	95.3%	98.8%
Kendall *et al.* [101]	0.506	-	**0.110**	81.7%	95.9%	98.9%
Ours (baseline)	0.487	0.164	0.113	86.7%	97.7%	99.4%
Ours (full)	**0.478**	**0.161**	0.111	**87.2%**	**97.8%**	**99.5%**

Table 7.2: The performance of previous state-of-the-art approaches evaluated on the standard test set of the KITTI dataset [63]

Cap	Method	*lower better*			*higher better*		
		RMS_{lin}	RMS_{ln}	Rel_{abs}	δ	δ^2	δ^3
0-80m	SFM-learner[212]	6.856	0.283	0.208	67.8%	88.5%	95.7%
	Godard *et al.*[67]	4.935	0.206	0.141	86.1%	94.9%	97.6%
	Kuznietsov *et al.* [111]	4.621	0.189	0.113	86.2%	96.0%	98.6%
	Ours (baseline)	4.394	0.178	0.095	89.4%	96.6%	98.6%
	Ours (full)	**4.301**	**0.173**	**0.096**	**89.5%**	**96.8%**	**98.7%**
0-50m	SFM-learner[212]	5.181	0.264	0.201	69.6%	90.0%	96.6%
	Garg *et al.* [62]	5.104	0.273	0.169	74.0%	90.4%	96.2%
	Godard *et al.* [67]	3.729	0.194	0.108	87.3%	95.4%	97.9%
	Kuznietsov *et al.*[111]	3.518	0.179	0.108	87.5%	96.4%	98.8%
	Ours(baseline)	3.359	0.168	0.092	90.5%	97.0%	98.8%
	Ours(full)	**3.284**	**0.164**	**0.092**	**90.6%**	**97.1%**	**98.9%**

Additionally we include qualitative results for NYUv2[140] and KITTI[63] in Figures 7.7 and 7.8 respectively. Each of which illustrates a noticeable improvement over previous methods. We also demonstrate that the improvement is beyond the numbers, as our approach generates more convincing depths even when the

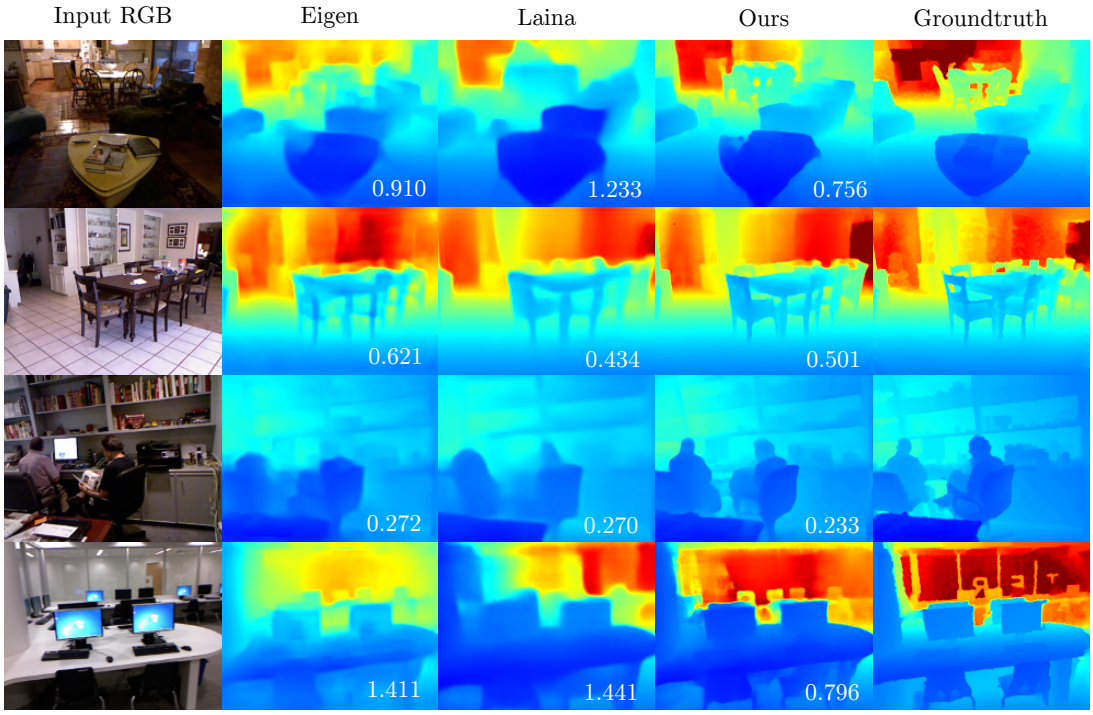

Figure 7.7: Resulting single image depth estimation for several approaches and ours against the ground truth on the dataset NYUv2[140]. The RMSE for each prediction is included.

Table 7.3: The performance of previous state-of-the-art approaches on a randomly selected subset of the frames from the RGB-D dataset [182]. We post separate entries for DeMoN(est) and DeMoN(gt), former is scaled by the estimated scale of their system, while the latter is scaled by the median ground-truth depth

Method	*lower better*			*higher better*		
	RMS_{lin}	RMS_{log}	Rel_{abs}	δ	δ^2	δ^3
Laina *et al.*[114]	1.275	0.481	0.189	75.3%	89.1%	91.8%
DeMoN(est)[191]	2.980	0.910	1.413	21.0%	36.6%	48.9%
DeMoN(gt)[191]	1.584	0.555	0.301	52.7%	70.7%	80.7%
Ours(baseline)	1.068	0.353	0.128	86.9%	92.2%	93.5%
Ours(full)	**0.996**	**0.329**	**0.108**	**90.3%**	**93.6%**	**94.5%**

RMSE may be higher, as is the case in the second row of Figure 7.7, where [114] computes a lower RMSE. More impressive still are the results in Figure 7.8, where we compare against previous approaches that are both trained on much larger

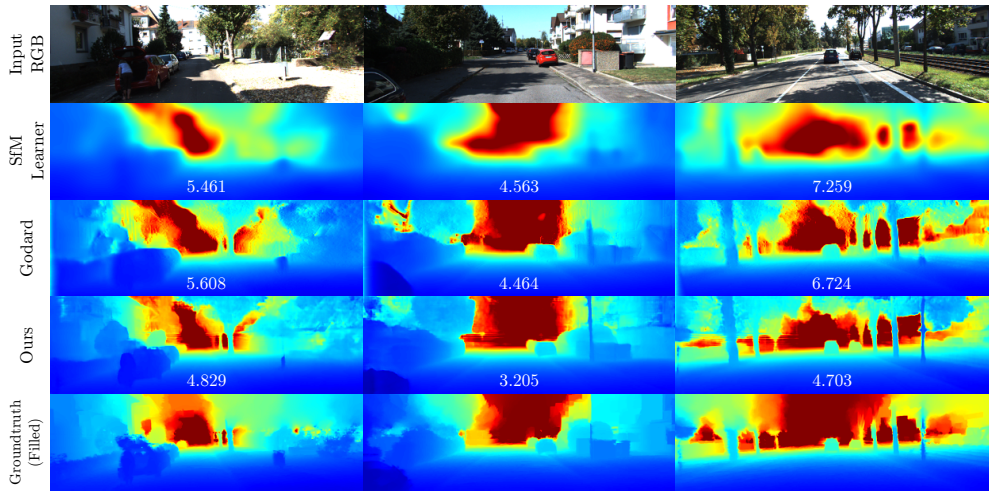

Figure 7.8: The resulting single image depth estimation for several approaches including SfM-Learner[212], Godard[67] and Ours against a ground truth filled using [119] on the test set of the KITTI dataset [63]. We include the RMSE values for each methods prediction. Filled depths are included for visualisation purposes during evaluation the predictions are evaluated against the sparse Velodyne ground truth data.

training sets than our own and still show noticeable qualitative and quantitative improvements.

7.5.2 Pose Estimation

To demonstrate the ability of the framework to perform accurate relative pose estimation, we compare our approach on several unseen sequences from the datasets for which ground-truth poses were available. To quantitatively evaluate the trajectories we use the Absolute Trajectory Error (ATE) and the Relative Pose Error (RPE) as proposed in [182]. To mitigate the effect of scale-drift on these quantities all poses are scaled to the ground-truth associated poses during evaluation. By using both metrics it provides an estimate of the consistency of each pose estimation approach. We summarise the results of this quantitative analysis for KITTI[63] in Table 7.4 and for RGB-D[182] in Table 7.5. We include comparisons of the performance against other state-of-the-art pose estimation networks

namely SFM-Learner[212] and DeMoN[191]. Additionally we include results from current state-of-the-art SLAM systems (ORB-SLAM2[139] and LSD-SLAM[48]).

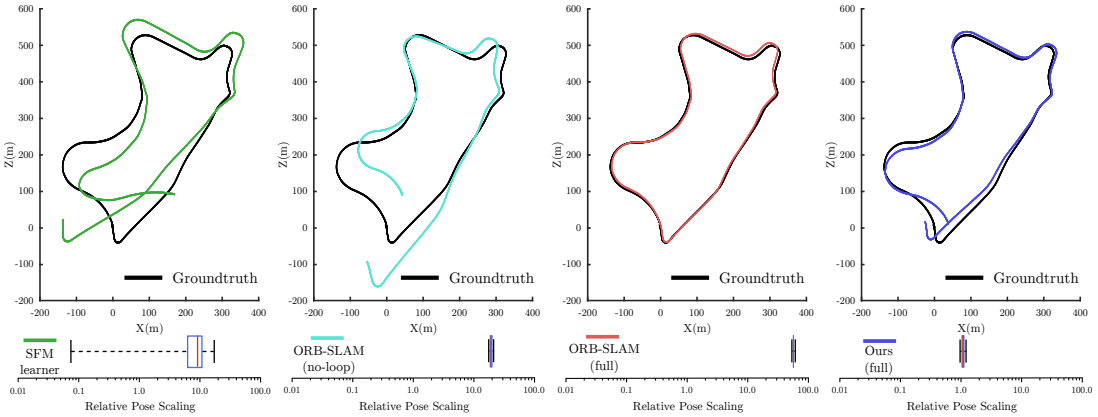

Figure 7.9: *Top* the scaled and aligned trajectories for SFM-Learner [212], ORB-SLAM2 [138] (with and without loop-closure) and Ours respectively. *Bottom* box-plots of the relative pose scaling required to bring the predicted translation to the same magnitude as the ground-truth pose

In Table 7.4 we demonstrate a noticeable improvement over SfM-Learner on both sequences in all metrics. We evaluate SfM-Learner on its frame-to-frame tracking performance for adjacent frames (*SFM-Learner(1)*) and separations of 5 frames (*SFM-Learner(5)*), as they train their approach to estimate frame gap of this length. Even with the massive reduction in accumulation error expected by taking larger frame gaps (demonstrated in reduced ATE) our system still produces more accurate pose estimates.

We show the resulting scaled trajectories of sequence 09 in Figure 7.9, as well as the relative scaling of each trajectories poses in a box-plot. The spread of scales present for SFM-Learner indicates scale is essentially ignored by their system, with scale drifts ranging across a full log scale, while ORB-SLAM and our approach are barely visible at this scale. Another thing to note is that our scale is centered around 1.0, as we estimate scale directly by estimating metric depths. This seems to provide a strong benefit in terms of reducing scale-drift and we believe makes our system more usable in practice. Another qualitative result

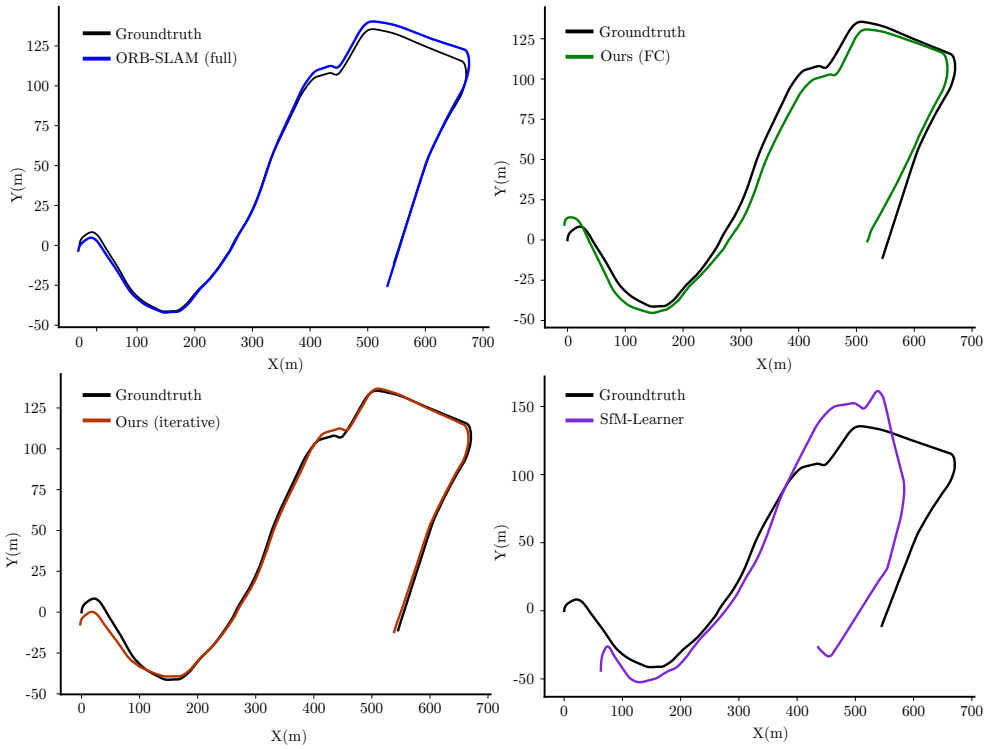

Figure 7.10: Trajectories of Our method in both configurations, as well as the resulting trajectories of ORB-SLAM(full) [138] and SfM-Learner [212]. We demonstrate comparative quality to ORB-SLAM, and significantly out perform SfM-Learner

is shown in Figure 7.10 demonstrating the performance of our approach against competing methods on sequence 10 of the KITTI dataset.

In Table 7.5, we show a significant improvement in performance against existing machine learning approaches across several sequences from the TUM RGB-D dataset[182]. The corresponding qualitative results are shown in Figure 7.11. We evaluate against DeMoN[191] in two ways, frame-to-frame (*DeMoN(1)*) and we again try to provide the same advantage to DeMoN as SfM-Learner by using wider baselines, which they claim improves their depth estimations[191], using a frame gap of 10 (*DeMoN(10)*). It can be observed that even with the massive reduction in accumulation error over our frame-to-frame approach, we still manage to significantly outperform their approach in ATE, even surpassing LSD-SLAM

Table 7.4: Performance of several approaches evaluated on two sequences of the KITTI dataset [63]. SfM-Learner(1) and SfM-Learner(5) indicates the different frame gaps used to construct the trajectories. The results are separated by SLAM and machine learning approaches. We highlight the strongest results in bold

Sequence	09			10		
Method	ATE(m)	RPE(m)	RPE(°)	ATE(m)	RPE(m)	RPE(°)
ORB-SLAM(no-loop)	57.57	**0.040**	0.103	8.090	0.033	0.105
ORB-SLAM(full)	**9.104**	0.056	**0.084**	**7.349**	**0.031**	**0.100**
SfM-learner(5)	58.31	0.077	0.803	31.75	0.069	1.242
SfM-learner(1)	81.09	0.050	0.976	75.89	**0.045**	1.599
Ours(fully connected)	41.50	0.087	0.387	29.29	0.081	0.486
Ours(iterative)	**7.89**	**0.039**	**0.123**	**9.274**	0.056	**0.190**

on the sequence *fr1-xyz*. ORB-SLAM is still the clear winner, as they massively benefit from the ability to perform local bundle-adjustments on the sequences used, which are short trajectories of small scenes. We include an example of a frame from the sequence *fr3-walk-xyz* in Figure 7.12, which shows this scene is not static, but our system has the ability to deal with this through the flow confidence estimates, discussed in Section 7.5.3.

Table 7.5: Performance of pose estimation on several sequences from the RGB-D dataset [182]. DeMoN(1) and DeMoN(10) indicates the trajectories were constructed with a frame gap of 1 and 10 respectively. Both [138] and [48] fail to track on *fr2-360-hs*. The results are separated by SLAM and machine learning approaches. We highlight the strongest results in bold

Sequence	fr1-xyz			fr2-360-hs			fr3-walk-xyz		
Method	ATE (m)	RPE (m)	RPE (°)	ATE (m)	RPE (m)	RPE (°)	ATE (m)	RPE (m)	RPE (°)
LSD-SLAM	0.090	-	-	-	-	-	0.124	-	-
ORB-SLAM	**0.009**	**0.007**	**0.645**	-	-	-	**0.012**	**0.013**	**0.694**
DeMoN(10)	0.178	**0.021**	**1.193**	0.601	0.035	2.243	0.265	0.049	1.447
DeMoN(1)	0.183	0.037	3.612	0.669	0.032	3.233	0.279	0.040	3.174
Ours(fully connected)	0.169	0.028	1.887	0.883	0.030	1.799	0.268	0.044	1.698
Ours(iterative)	**0.071**	0.024	1.237	**0.461**	**0.020**	**0.736**	**0.240**	**0.026**	**0.811**

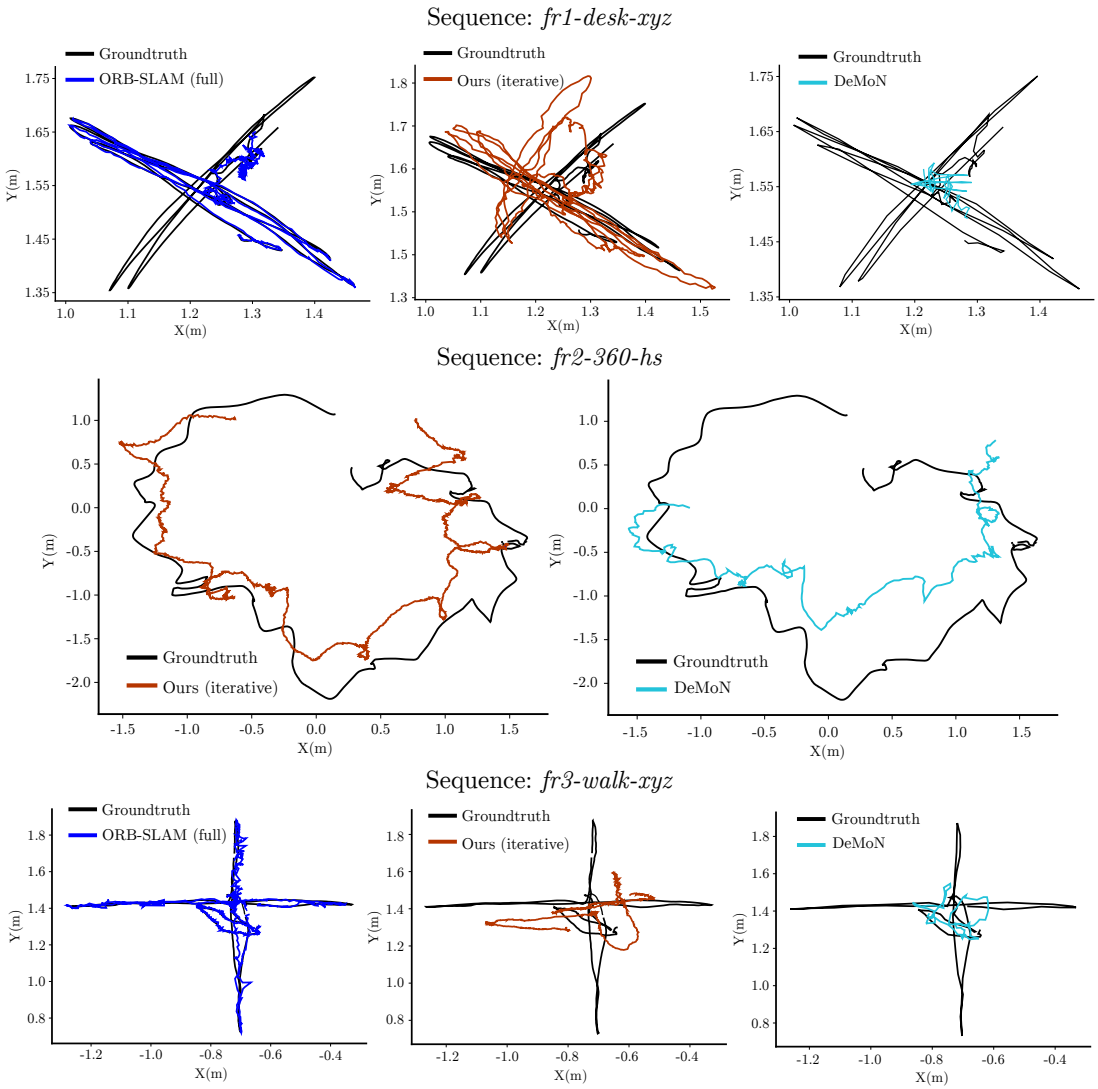

Figure 7.11: Trajectories of Our method against ORB-SLAM [138] and DeMoN(10) [191], for the evaluated sequenced from the RGB-D dataset [182]. We demonstrate a marked improvement upon DeMoN which, although being given a slight advantage in some respects by widening the baseline and reducing accumulated pose error, still performs poorly. However against ORB-SLAM, both methods come up a little short, as ORB-SLAM is able to perform local bundle-adjustments across multiple keyframes, which greatly reduces the overall error.

7.5.3 Ablation Experiments

The first set of ablation experiments were designed to verify the performance improvements we achieve (specially for depth prediction) are due to the use of novel loss functions and not merely due to using a better architecture. To this end, we replaced the architecture of our depth estimation network using that of Kuznietsov *et al.* [111]. As it can be seen below by using the full training loss we are able to improve the accuracy of the depth estimation results indicating the generality of the approach.

	Dataset	**NYU**[140]		**TUM**[182]		**KITTI** [63]	
		Baseline	Full	Baseline	Full	Baseline	Full
Metric	RMSE(m)	0.536	0.525	1.096	1.015	3.518	3.425
	$< \delta$ (=1.25)	82.5	82.8	79.9	81.1	87.5	89.5
	$< \delta^2$	96.3	96.7	90.4	91.8	96.4	96.9
	$< \delta^2$	99.2	99.3	93.8	94.6	98.8	98.8

In order to examine the contribution of using each component of our pose estimation network, we compare the pose estimates under various configurations on sequences 09 and 10 of the KITTI odometry dataset[63], summarised in Table 7.6. We examine the relative improvement of iterating on our pose estimation till convergence (*ours(iterative)*), against a single weighted-least-squares iteration (*ours(single)*), which demonstrates iterating has a significantly positive effect.

We demonstrate the improved utility of our flows by replacing our flow estimates with other state-of-the-art flow estimation methods from [92] and [161] in our pose estimation pipeline, and consistently demonstrate an improvement using our approach. We show the result of optimising with and without our estimated confidences, demonstrating quantitatively how important they are to pose estimation accuracy, with significant reductions across all metrics.

We also demonstrate qualitatively one of the ways in which estimating confidence improves our pose estimation in Figures 7.12 and 7.13. This shows that our system

Table 7.6: Results of pose estimation on KITTI[63] with various components of the network removed or replaced. We highlight the strongest results in bold

Sequence	09			10		
Method	ATE(m)	RPE(m)	RPE(°)	ATE(m)	RPE(m)	RPE(°)
Ours(noconf)	53.40	0.356	0.931	58.50	0.308	1.058
Ours(noconf,iterative)	33.18	0.248	0.421	35.87	0.280	0.803
Flownet2.0[92]	29.64	0.349	0.838	51.90	0.222	0.954
Flownet2.0(iterative)[92]	24.61	0.185	0.400	22.61	0.142	0.484
EpicFlow[161]	119.0	0.566	0.931	20.98	0.199	0.853
EpicFlow(iterative)[161]	59.79	0.379	0.459	14.80	0.154	0.581
Ours(single)	20.89	0.064	0.183	9.5903	0.060	0.194
Ours(iterative)	**7.89**	**0.039**	**0.123**	**9.274**	**0.056**	**0.190**

Figure 7.12: For a frame pair (I_i and I_j) from the sequence *fr3-walk-xyz* , $\mathbf{F}_{\overleftarrow{ji}}$ is the estimated optical flow from I_i to I_j, and C_x and C_y are the estimated flow confidences in the x and y direction respectively

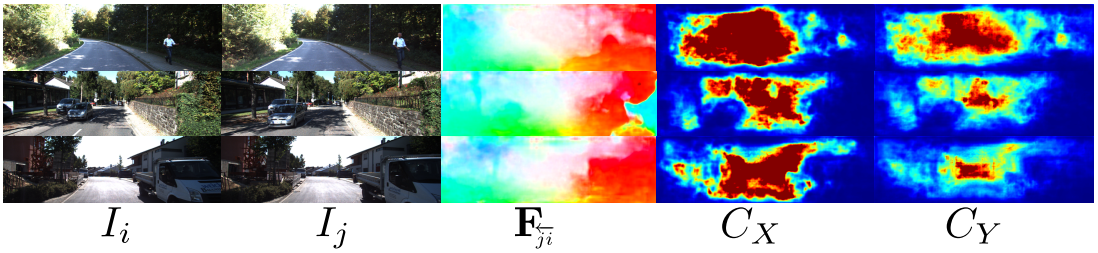

Figure 7.13: A selection of optical flow predictions made by our framework on the KITTI dataset[63]. Dynamic objects and the objects that do not appear in both frames due to large camera motion have low confidence

has learned the confidence of moving objects is lower than its surroundings and the confidences of edges are higher, helping our system focus on salient information during optimisation in an approach similar to [48].

7.6 Summary

In this chapter, we have presented the first piece of work that performs least squares based pose estimation inside a neural network using predicted depths and optical flow. Our approach is also the first to use a neural network to predict the full information matrix which represents the confidence of our optical flow estimate. The proposed formulation is fully differentiable and is trained end-to-end.

The importance of incorporating techniques from geometry into the machine learning pipeline, particularly when predicting structural quantities is highlighted in this chapter. To support this argument we generate poses using two different methods. Firstly, using a fully connected layer and next by employing a least squares solver to compute the camera pose given the predicted depths, optical flow and the confidences in x and y directions. The experimental results validate the initial hypothesis demonstrating the superiority of the computed poses (using the predicted quantities) over the predicted poses. To recapitulate, we advocate for a hybrid approach where a CNN is used for tasks that benefit from feature extraction, while conventional geometry is used for motion estimation given the predicted quantities.

The main limitation of the framework is the time taken to infer the depths taking approximately 200ms per frame and this issue is addressed in the next chapter.

CHAPTER 8

Real-time Depth Prediction on Mobile Frameworks

8.1 Introduction

All of the depth prediction pipelines presented thus far in this thesis despite being accurate and temporally consistent, pose a challenge to real-time robotics in that the prediction time ranges from 5-10 frames per second on a workstation GPU. Unfortunately, this also means these large networks are virtually inoperative on resource constrained mobile environments. However, the importance of real-time frameworks for the field of robotics cannot be overstated. A vast majority of tasks performed by a robot (autonomous vehicle navigation, visual servoing, and object detection) require it to interact with other robots or humans and respond to actions of one another. Due to this very reason, the low latency aspect which stipulates coherency becomes a prerequisite for robotic systems. While building a real-time system on a modern computer can be challenging as it stands, doing so on a mobile platform with less than one tenth of the compute is extremely difficult. However, these mobile platforms are much more appealing for real life scenarios as they consume less power and are compact in nature compared to the

desktop workstation counterparts.

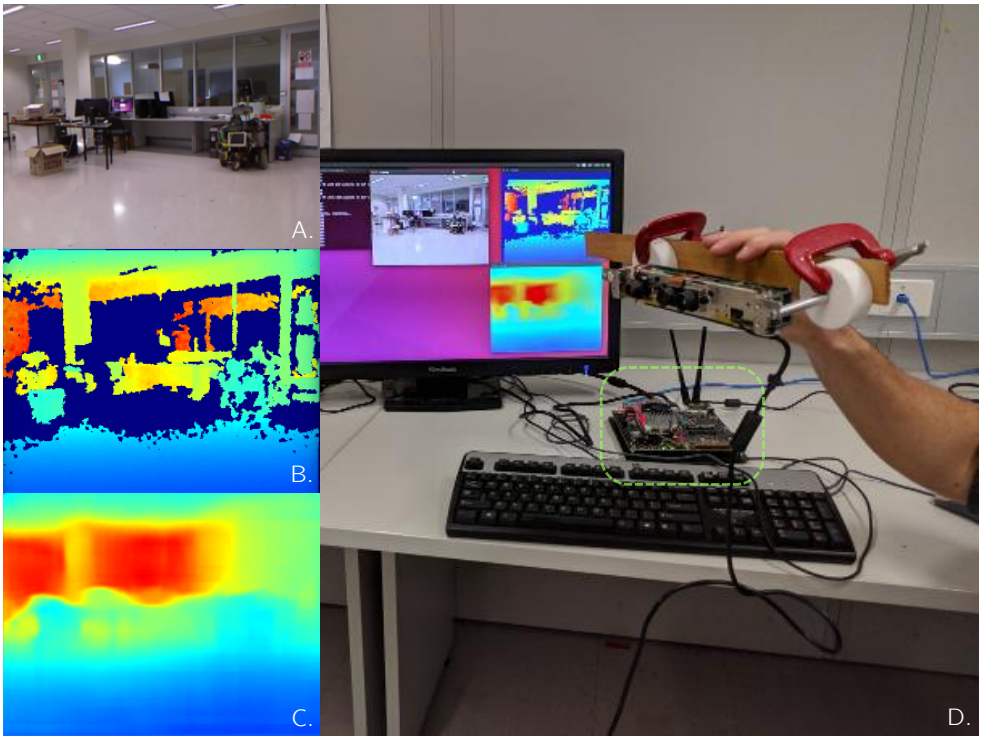

Figure 8.1: Demonstrates the model running on the NVIDIA-TX2 developatment board (shown in green square **D.**) in real-time. **A.** The colour image, **B.** The jet coloured groundtruth depth image from a the Kinect **C.** The predicted depth from the colour input image.

An area of research that has been quite popular within the robotics community is the application of machine learning to robotics problems. Neural Networks are being applied with resounding success to solve many problems and have now surpassed human level performance on tasks such as image recognition or even complex strategy games such as Go, a task once thought too challenging for a machine. We take a step towards combining real-time robotics and machine learning on a resource constrained mobile platform. More concretely, we present the first piece of work that performs single image depth prediction which runs at 30fps on a NVIDIA-TX2 or at over 300fps on an NVIDIA-GTX1080Ti. Our depth prediction framework not only runs at frame rate on an NVIDIA-TX2 but also outperforms denser architectures such as [45] despite the latter being able to

see the whole image in its field of view using stacked fully-connected layers.

Model compression is the concept of replicating the performance of a larger model or an ensemble of large models using a smaller network. Common model reduction techniques mainly focus on the architectural aspect and consist of techniques such as quantization of weights, curtailing the depth etc. In this work, the emphasis is predominantly on a training regime which tries to replicate the latent space of the deep model. This allows us to achieve superior performance over randomly initialised models of equivalent size, while training both models to convergence.

8.2 Contributions

The work presented in this chapter was done in collaboration with Andrew Spek. The contribution I have made personally based on mutual agreement is given below:

- Conception of Idea (20%)
- Network architecture design (50%)
- Loss/Objective function design (50%)
- Coding the network (50%)
- Training data creation (50%)
- Testing and evaluation (50%)

The following bullet points provide a summary of the contributions made in this chapter in-order to build dense real-time structure prediction frameworks:

- Present the first piece of work which performs depth prediction at frame-rate on a mobile platform in the form of an NVIDIA-TX2, while outperforming model architectures which have more than 30 times the number of

parameters as ours.

- We present an analysis of the system with extensive experiments to show how different loss functions play a vital role when learning the underlying latent representation, while not compromising the training time.
- Real-time depth prediction enables us to readily integrate the predicted depths with ORB-SLAM2 [139] in order to perform tracking and mapping on mobile platforms, while significantly reducing scale-drift.

The main body of the chapter is largely based on the contents of the joint publication [177] and is expected to appear in a similar form in Andrew's thesis.

8.3 Related Work

In addition to the material presented in the computer vision and machine learning section of the background chapter, related research for this chapter include the work undertaken in the field of model compression/distillation.

The machine learning community has investigated the problem of model compression or emulating the performance of a larger network. Hinton *et al.* in [82] introduced a concept called distillation which aimed to replicate the class probabilities of a larger model using a smaller model. Since we are tackling a regression problem (compared to a classification problem), training a smaller network to replicate the prediction layer of a larger model becomes strictly suboptimal compared to training directly on the ground truth since there is no notion of class probability. Inspired by this however, we introduce a tensor loss where we aim to mimic the latent space or the embedding of the penultimate layer of the larger model. Initial results presented here indicate having the supervised tensor loss gives inferior results compared to learning the penultimate layer in an unsupervised manner. Similar to [82], Bucila *et al.* showed that it is possible to replicate the performance of an ensemble of classifiers using a single model

[12]. Their method relied on generating synthetic data using an ensemble of networks and training the smaller network on this synthetic data. Finally, Han *et al.* demonstrated *model weight compression* through the use of quantization and Huffman coding in [74] for image classification.

8.4 Proposed Framework

This section aims to provide a step by step breakdown of the proposed framework. We begin by presenting the model architecture implemented, followed by the different loss terms employed. Next we introduce the datasets that were used during training. Finally, this section is concluded with an account of the training regime that was used to train various models.

8.4.1 Model Architecture

Our model design is inspired by ENet [152] and ERFNet [163], which have targeted mobile frameworks as the deployment environment, while achieving a decent trade-off between performance and speed for the task of semantic segmentation. ENet [152] demonstrated the ability to run at near real-time (≈10fps) performing a dense semantic segmentation task on an NVIDIA-TX1. However, in this work we aimed for a much larger frame rate, to allow for every frame to have a depth estimate in real-time. Our target hardware platform was the NVIDIA-TX2, which has ≈30% more compute power over the previous NVIDIA-TX1. We use the NVIDIA supported TensorRT framework [150] in order to accelerate inference of our models. However, this restricted the available layers to those supported by the framework, which at the time of implementation did not support dilated convolutions [205]. Taking these factors into consideration and after a number of attempts, we decided on the architecture defined in Table 8.1 to provide the best compromise between runtime and accuracy.

Model Architecture Breakdown

	Layer Type	Resolution(in, out)	Channels (in, out)
E	Downsample (2×2)	320×240, 160×120	3, 16
	Downsample (2×2)	160×120, 80×60	16, 64
	Non-btl 1D (3×3)	80×60, 80×60	64, 64
	Non-btl 1D (3×3)	80×60, 80×60	64, 64
	Non-btl 1D (3×3)	80×60, 80×60	64, 64
	Non-btl 1D (3×3)	80×60, 80×60	64, 64
	Downsample (2×2)	80×60, 40×30	64, 128
	Non-btl ND (3×5)	40×30, 40×30	128, 128
	Non-btl ND (3×5)	40×30, 40×30	128, 128
	Non-btl ND (3×7)	40×30, 40×30	128, 128
D	Deconv (4×4)	40×30, 80×60	128, 64
	Non-btl 1D (3×3)	80×60, 80×60	64, 64
	Deconv (4×4)	80×60, 160×120	64, 64
	Non-btl 1D (3×3)	160×120, 160×120	64, 64
	Deconv (4×4)	160×120, 320×240	64, 64
P	Conv 2D (3×3)	320×240, 320×240	64, 1

Table 8.1: A summary of the architecture implemented given an input resolution of 320×240, which is used for both [140] and [182]. The left column refers to the broad section of the network as shown in Figure 8.2, **E:** Encoder, **D:** Decoder and **P:** Predictor, where the predictor layer is the layer that can be transplanted from the supervisor network.

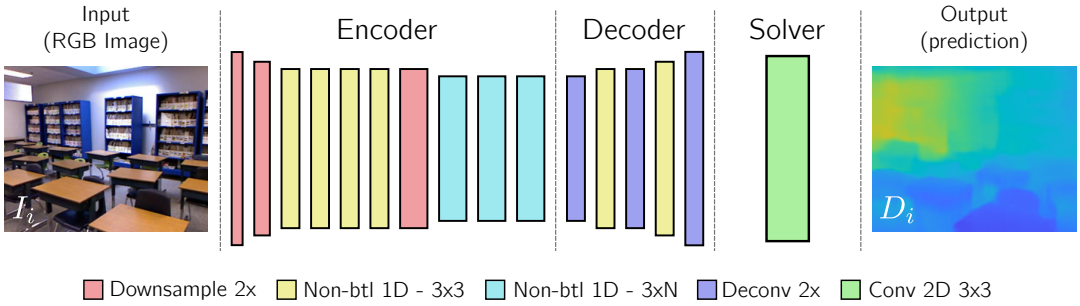

Figure 8.2: The model architecture for our real-time depth estimation network. This network is constructed from mostly Non-bottleneck blocks (Non-btl in figure), which are a series or residual type blocks shown in Figure 8.3. Downsample, Conv 2D and Deconv are all standard operations.

8.4.2 Loss Functions and the Knowledge Transfer Process

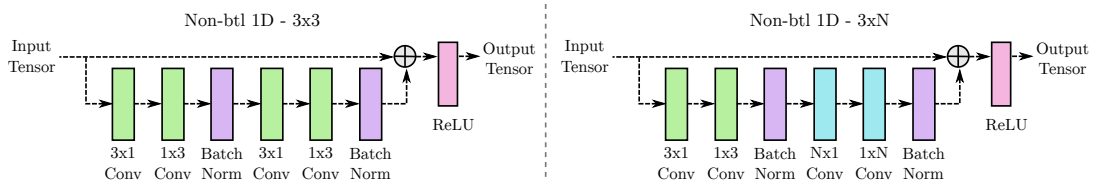

Figure 8.3: Shows the submodules of the Non-bottleneck 1D blocks. "Non-bottleneck" refers to the channel count which remains unchanged when passing through this layer. The Nx1, 1xN, 3x1 and 1x3 Conv operations are standard asymmetrical convolutions where N is chosen. In practice we used two Non-btl ND - 3x5 blocks followed by one Non-btl ND - 3x7 blocks, in an attempt to increase the receptive field as much as possible. The *plus* indicates the addition of the two sets of activations, followed by ReLU activation.

The most commonly used loss function when performing regression is the L_2 distance between the prediction and the ground truth as shown below.

$$L_d = \frac{1}{N}\sum_{i=0}^{N} ||D_i - D_i^*||_2 = \frac{1}{N}\sum_{i=0}^{N} ||d_{pi}||_2 \tag{8.1}$$

where D_i represents the predicted depth map and D_i^* represents the ground truth depth map obtained from a Kinect or a Velodyne LIDAR. Additionally, we define the distance between the ith predicted and ground-truth depth to be d_{pi}, shown in Figure 8.4, where the super script ^{c} and ^{b} are used to denote the error from the condensed and large network predictions respectively.

We choose this as a starting point to train our condensed networks defined in Table 8.1. The random models trained using this formulation are referred to as **R** models throughout the remainder of the chapter. Note that since there are no existing networks which share the same architecture as the condensed network, all the weights were initialized using MSRA initialization [78].

Upon training the randomly initialized model using the Euclidean loss defined in Equation 8.1, the next stage focused on improving the accuracy. Since making the

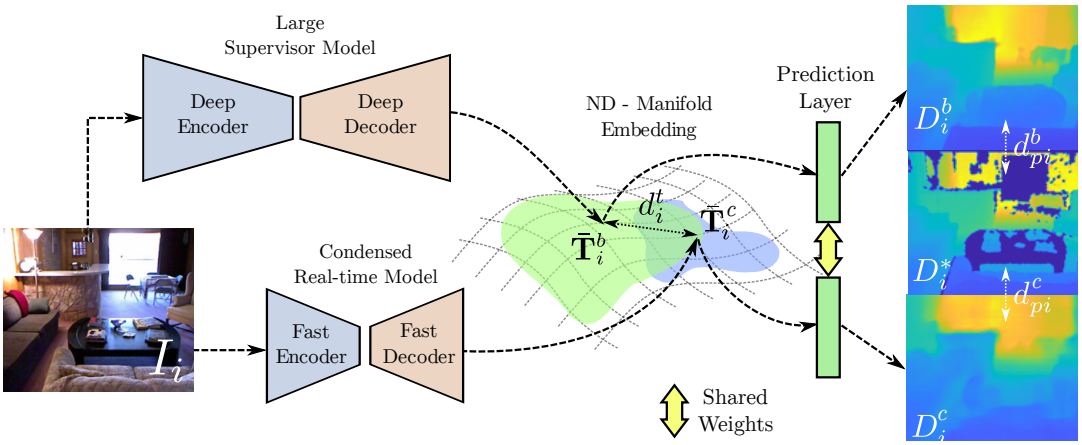

Figure 8.4: This demonstrates the concept behind our training regimes we use to perform model compression through knowledge transfer. The initial strategy was to minimise the difference (d_i^t) between the intermediate activations (also referred to as the tensor loss) produced as input to the prediction layer by the network. The other approach involves transplanting of the prediction layer from the large network onto the condensed network. We also examine a combination of both approaches. In practice the transplant alone is both more effective and much faster to train, although all knowledge transfer approaches improve the performance over random.

model deeper was not an option as this would compromise the capability of the system to predict at 30fps, we use different loss functions similar to Kuznietsov *et al.* [111] to improve the accuracy. Another important factor that was taken into consideration was the availability of larger models that achieve state-of-the-art results when performing depth prediction as demonstrated in the previous chapter. This inspired us to create architectures that could learn from a bigger model in a knowledge-transfer fashion.

The main goal was to leverage the predictive capability of a state of the art depth estimation network [40] and attempt to transfer the useful knowledge to the condensed counterpart as demonstrated by Hinton *et al* [82]. This creates a performance cap that is the performance of the larger network, but the intuition was that the performance could potentially be improved over the random variant(**R**) to be usable in robotics. We designed our condensed network around the idea that the final layer of the large depth estimator could be transplanted on to

our condensed network as shown in Figure 8.4.

The first set of models focused on replicating the activations of the penultimate layer of the larger/deeper network using the smaller/shallower network. The network was trained using the tensor loss function as given in Equation 8.2. This is a supervised loss where we attempt to enforce the tensors of large and condensed model to match.

$$L_t = \frac{1}{N}\sum_{i=0}^{N} ||T_i - T_i^*||_2 = \frac{1}{N}\sum_{i=0}^{N} ||d_i^t||_2 \tag{8.2}$$

T_i represents a tensor corresponding to the activations of the penultimate layer of the condensed network and T_i^* represents that of the deeper network.

After training till convergence using the tensor loss, the final layer is freed and the network is fine tuned using the depth loss. This model is denoted as **T** in the results section.

We also propose an alternative loss where the penultimate layers are trained in an unsupervised manner in which we transplant the final/prediction layer of the larger model on to a randomly initialized condensed model. The network is then trained using Equation 8.1 to provide useful activations for the prediction layer. The transplanted model (**TR**) updates all the layers barring the prediction layer.

Finally, we train the **T+TR** model which uses a combination of the tensor loss and the transplanted solver. Under this formulation the network is trained for approximately 20 epochs using the tensor loss followed by the transplantation of the prediction layer. This transplanted network is then further fine-tuned using the depth loss for another 5 epochs.

8.5 Results

We evaluate the output of the proposed networks on both indoor (NYUv2 [140]) and outdoor (KITTI [63]) datasets. In addition to the qualitative and quantitative comparisons, the usefulness of our approach to practical robotic applications is demonstrated through a mapping and navigation scenario.

8.5.1 Depth Evaluation

The depths are evaluated using the same set of metrics defined in Chapter 5.

NYUv2 [140]						
Method	RMS_{lin}	RMS_{log}	Rel_{abs}	δ	δ^2	δ^3
Liu [122]	0.824	-	0.230	61.4%	88.3%	97.2%
Eigen_{alex} [44]	0.753	0.255	0.198	69.7%	91.2%	97.7%
Eigen_{vgg} [44]	0.641	0.214	0.158	76.9%	95.0%	98.8%
Laina [114]	0.573	0.195	0.127	81.1%	95.3%	98.8%
Baseline [40]	**0.480**	**0.111**	**0.161**	**87.2%**	**97.8%**	**99.5%**
Real-time Networks						
Ours (R)	0.765	0.277	0.216	64.4%	89.3%	97.1%
Ours (T)	0.713	0.261	0.204	68.5%	90.9%	97.5%
Ours (T+TR)	0.715	0.262	0.205	68.3%	90.8%	97.5%
Ours (TR)	**0.687**	**0.251**	**0.190**	**70.4%**	**91.7%**	**97.7%**

Table 8.2: The metrics are explained in Subsection 8.5.1. Lower numbers are better for the first three columns as these represent errors and higher number are better for the last three columns as they represent percentage of inliers.

We tabulate the results for the NYUv2 dataset in Table 8.2 and for KITTI in 8.3. Additionally some qualitative results from the NYUv2, RGB-D and KITTI datasets are included in Figures 8.5 and 8.6. From the numerical results in both

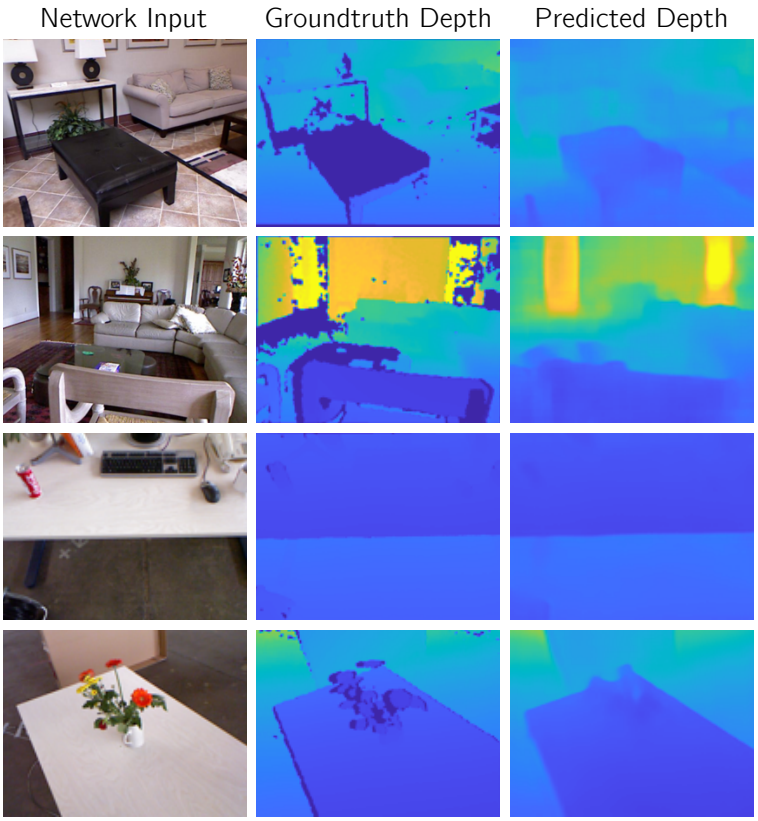

Figure 8.5: Qualitative results on the indoor datasets. All the images are from the test sets, and are not present in the training data.

tables we observed a consistent behaviour for both datasets with the following trend:

$$\text{Random} < \text{Tensorloss} < \text{Transplanted}$$

The fact that the random model is clearly inferior compared to all other variants highlights the importance of knowledge transferring process particularly when using condensed networks. Next we examine the contributions of the tensor loss model (**T**) and the transplanted model (**TR**) closely. As it can be seen in Figure 8.7 the tensor angles highly correlate with that of the supervisor network when

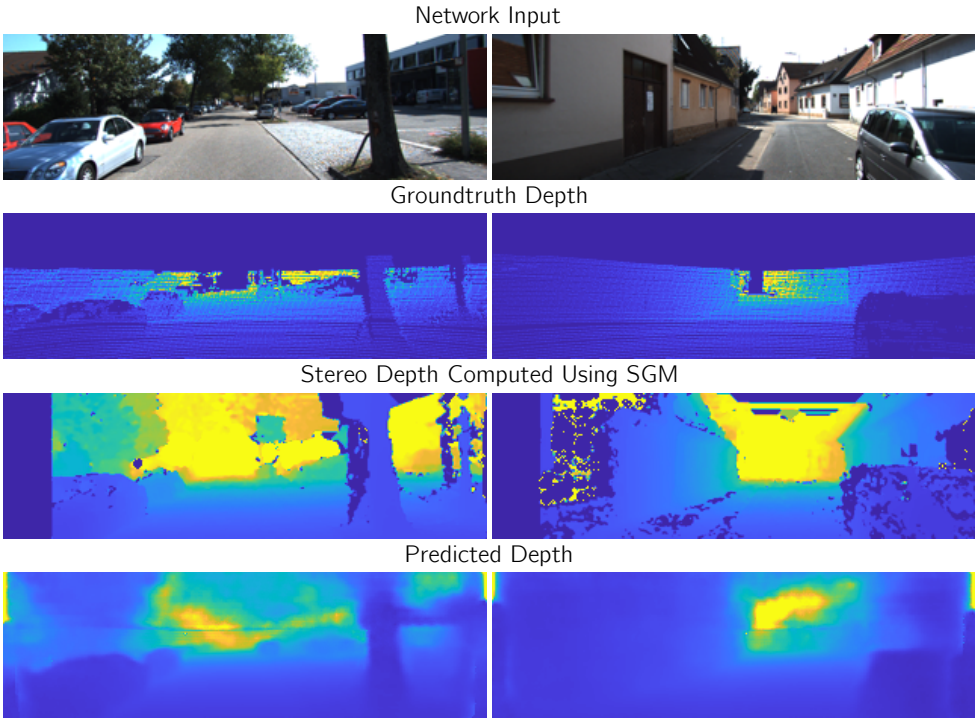

Figure 8.6: Qualitative results on the outdoor KITTI dataset

0-50m KITTI [63]						
Method	RMS_{lin}	RMS_{log}	Rel_{abs}	δ	δ^2	δ^3
Zhou [212]	5.181	0.264	0.201	69.6%	90.0%	96.6%
Garg [62]	5.104	0.273	0.169	74.0%	90.4%	96.2%
Goddard [67]	4.471	0.232	0.140	81.8%	93.1%	96.9%
Kuznietsov [111]	3.518	0.179	0.108	87.5%	96.4%	0.98.8%
Baseline [40]	**3.359**	**0.168**	**0.092**	**90.5%**	**97.0%**	**98.8%**
Real-time Networks						
Ours(R)	4.530	0.234	0.147	80.3%	93.3%	97.3%
Ours(T)	4.434	0.228	**0.139**	81.7%	93.7%	97.5%
Ours(T+TR)	4.426	0.225	0.140	81.7%	93.8%	97.6%
Ours(TR)	**4.363**	**0.224**	0.156	**81.8%**	**94.0%**	**97.7%**

Table 8.3: Results of evaluating KITTI dataset, using the same metrics as defined in Subsection 8.5.1 and Table 8.2

trained using the tensor loss. However, the magnitude of the activations correlate less strongly, which appears to negatively effect the quality of the reconstruction.

Another noteworthy observation is that the angle negatively correlates to the depth error, that is the most aligned tensors have the least error, and this is flipped between the **T** and **TR**. There seems to be some resemblance to the 'uncanny valley' concept in this case where the small network initially gets better and better at emulating the penultimate activations but ultimately falls short of perfectly reproducing the tensors and gets stuck in a suboptimal minima. On the other hand, the less restricted **TR** network is free to navigate to a minima that exploits as much of the information it can from the transplanted last layer.

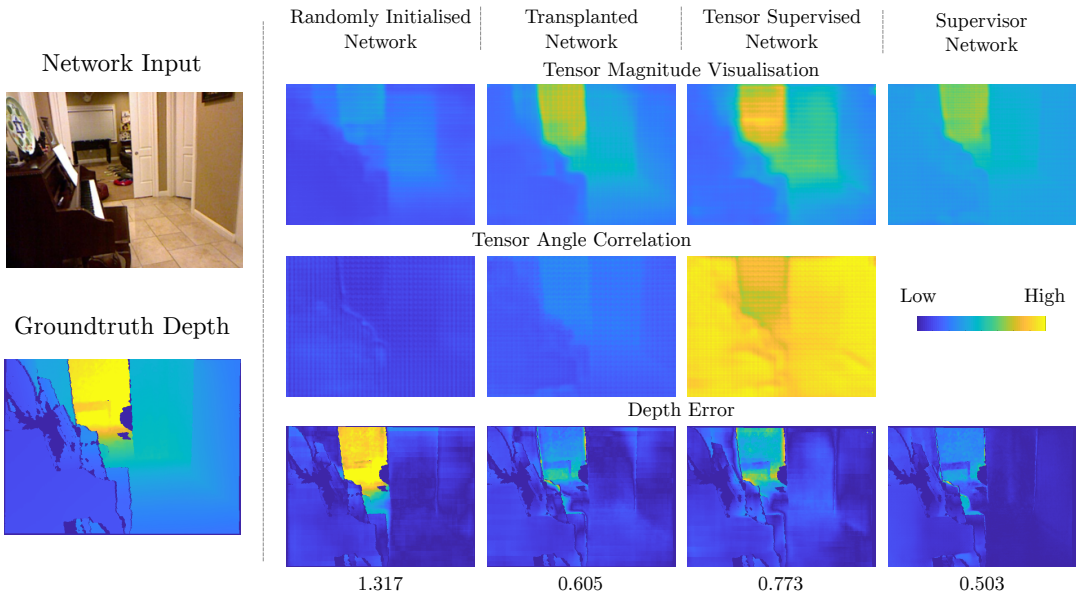

Figure 8.7: We demonstrate the relationship between the tensors produced with the three different training approaches. The first row shows the RGB input, the ground-truth depths, and the magnitude of the tensors for the last layer of the large network we use to transfer knowledge from. The second row shows the tensor magnitude images for each network, which are a visualisation of the norm of the tensor value. The third row shows the angle correlation, which is the degree to which the direction of the tensors agree. The final row shows the magnitude of the depth error between the prediction and ground truth. We include the RMS error in meters for each of the predictions below their respective columns.

8.5.2 Pose Estimation

As an application of our work, we evaluate the camera pose produced by an off-the-shelf SLAM system (ORB-SLAM2 [139]) using the depths inferred by our real-time network. We compare the resulting poses against the ground-truth pose data available on a select number of KITTI datasets. These results are summarised in Table 8.4 where the performance of the original SLAM system [139] in the mono and stereo configurations are compared against the trajectories obtained using our predicted depths as the input.

KITTI Odometry Absolute Trajectory Error (m)

Sequence	*Ours* *Predicted Depths*	*Mono* *ORB-SLAM [138]*	*Stereo* *ORB-SLAM [139]*
Seq00	4.23	6.62	1.3
Seq05	2.01	8.23	0.8
Seq07	1.15	3.36	0.5

Table 8.4: Pose estimation evaluation on KITTI sequences, measuring the ATE as defined in [182].

Additionally, in Figure 8.9 we show a qualitative comparison of trajectory accuracy using our predicted depths compared to using purely monocular data against the ground-truth trajectory. Again we compute these trajectories using the popular ORB-SLAM2 system [139]. This demonstrates that by using our predicted depths the system outperforms the monocular only approach, even when including the bundle-adjustment and loop-closure present for both approaches.

In an attempt to show a concrete example of what this system can contribute to an off-the-shelf SLAM approach, we demonstrate in Figure 8.8 the reduction in scale-drift given the same SLAM configuration, using our predicted depths as opposed to using the the colour data alone. This establishes a practical application given that our approach can also infer at over 70FPS as shown in Table 8.5.

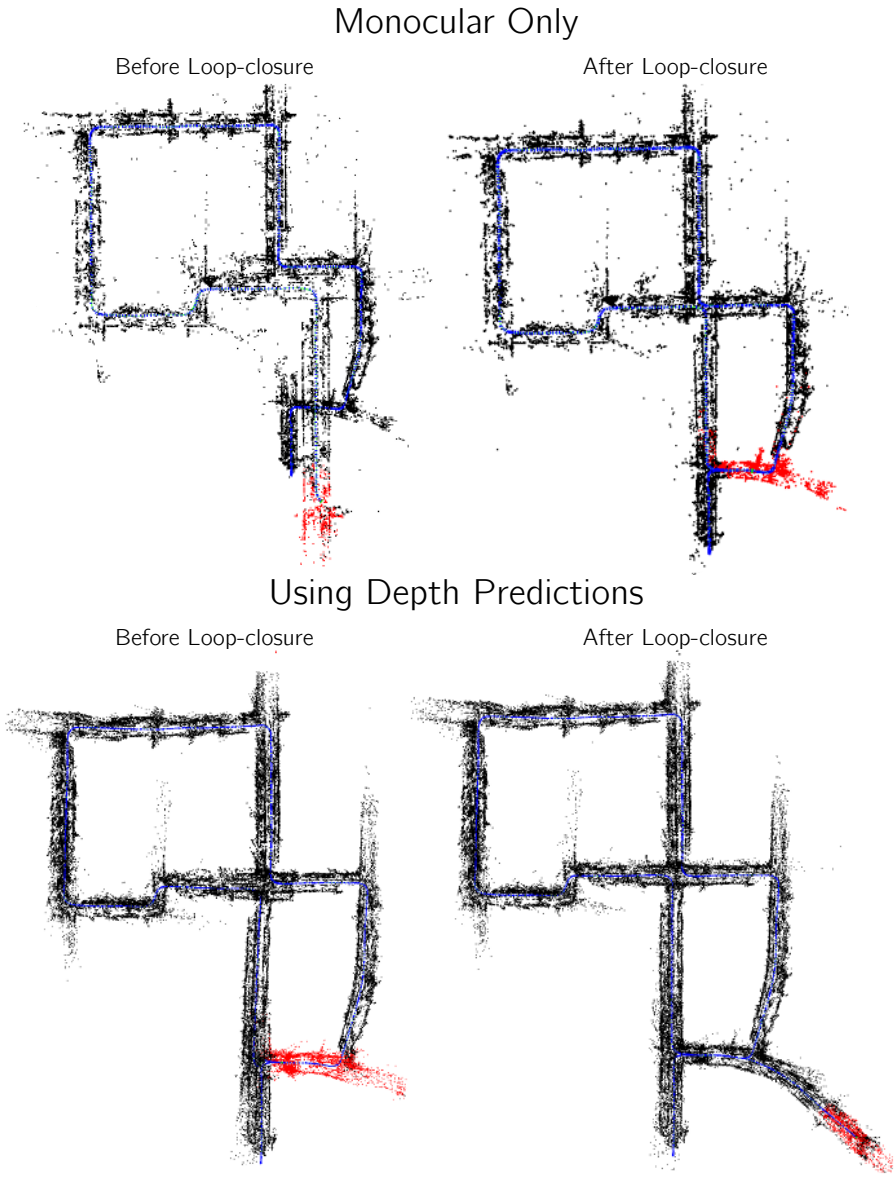

Figure 8.8: Demonstrates the amount to which the scale-drift can be reduced using our approach. The first row shows the performance of standard monocular ORB-SLAM2 [139] on sequence 00 of KITTI-odometry [63]. Before loop-closure a very pronounced level of scale-drift is present. In contrast when we provide our estimated depths, the scale drift is almost completely removed, and the difference before and after loop-closure is barely visible.

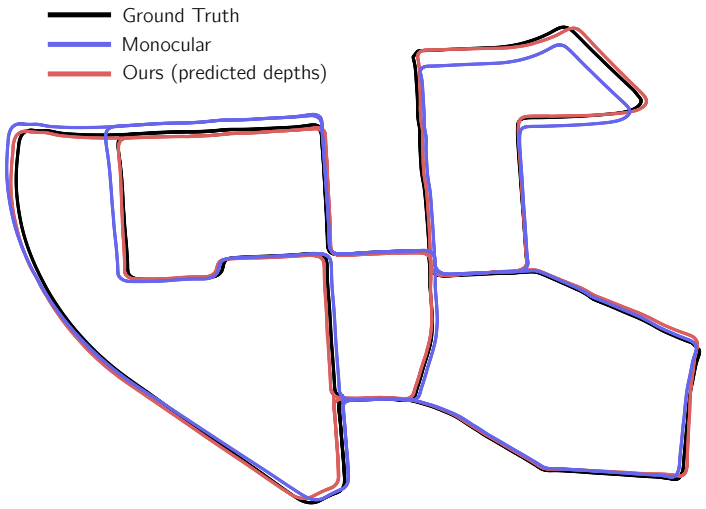

Figure 8.9: Demonstrates the improvement to the trajectory produced using our predicted depths from colour vs using colour images alone as input to the off the shelf SLAM system [139] for seq00 of KITTI-odometry [63].

Average FPS over 50 runs (Min, Max)		
Resolution	*GTX1080Ti*	*TX2*
640×480	105.96 (100.82, 107.92)	7.68 (7.50, 7.71)
320×240 †	312.29 (295.25, 320.00)	30.03 (27.76, 30.37)
640×192 ‡	214.75 (202.76, 221.58)	19.08 (18.23, 19.21)
320×96	473.09 (439.01, 498.73)	70.95 (65.54, 72.63)

Table 8.5: Depth Prediction - Real-time Performance FPS Speed comparisons for different output resolutions, on each device. The configurations marked with † and ‡ are the typical output resolutions of the state-of-the-art networks for indoor and outdoor datasets respectively.

8.5.3 Speed and Computation Performance

We include the timing information for our approach in Table 8.5. This shows that at the typical operating resolutions we can infer at real-time on the NVIDIA-TX2. Despite the predictions of KITTI being generated at a resolution of 320×96, the accuracy of these predictions are adequate enough to dramatically improve the quality of the SLAM map and reduce the scale-drift in tracking as shown

in Figure 8.8. A video demonstration of the framework can be found under https://youtu.be/Kc3Tbf3X7Cw.

8.6 Summary

The accuracy of a CNN and its model capacity often exhibit a direct correlation, where a significant reduction in model capacity typically results in vastly inferior predictions. In this chapter, we have demonstrated with examples that a system with a reduced model capacity can provide good enough depths to improve the accuracy of a standard monocular SLAM system, and disambiguate the scale without calibration. In addition, the predicted depth maps generated by the condensed network outperform those of the previous works [62, 67, 44] with much larger model capacities. This shows these compressed models can be taken well into the realms of real-time performance on low-power hardware without sacrificing much performance.

CHAPTER 9

Conclusion

In this thesis, we have tackled the problem of recovering 3D structural information from one or more colour images. Techniques from classical geometry as well as deep learning were employed to develop novel frameworks capable of predicting structure and motion parameters.

9.1 Summary of Contributions

- A real-time, multi-object visual SLAM system (MO-SLAM) capable of discovering duplicate objects present in a scene was presented in Chapter 4. The duplicate objects are then added on to the SLAM map as first order entities. The additional constraints given by the objects can then be used during bundle adjustment to improve the accuracy of the landmarks and the camera poses as well as the output of the recognizer, by removing erroneous map points that were wrongly classified as belonging to an object.

- A multi task learning platform was used to predict depths, surface normals

and surface curvatures in tandem purely from colour images. Extensive experiments were conducted by keeping the model capacity of the architecture fixed, while gradually increasing the number of prediction tasks in order to demonstrate that learning closely related structural information can enhance the performance of all three tasks (Chapter 5).

- An efficient technique to create a dense depth map in real time given a sparse SLAM map or sparse sensory data. For this task, it is assumed a CNN capable of predicting depth from a single colour image is available (which can be trained using the approaches given in Chapters 5,7,8). A smaller CNN was trained to learn the confidence of the sparse SLAM map as well as that of the CNN that predicts depth. Sparse depth map inpainting was achieved by minimizing a CRF energy term. Nodes of the CRF anchors the solution to the sparse map, while the scale-invariant edges enforce pairwise depth relationships based on the learnt depth predictions and the confidences (Chapter 6).

- A novel framework which predicts depth, optical flow, the flow confidences and the relative camera motion given an image pair. This work also features the first approach to use a neural network to predict the full information matrix which represents the confidence of the optical flow estimate. The predictions generated from the depth subnetwork achieve state-of-the-art performance on both indoor and outdoor test benchmarks. The relative camera poses produced from our framework outperform that of previous motion prediction approaches, while performing competitively with SLAM approaches (ORB-SLAM [139], LSD-SLAM[48], CNN-SLAM [187]) that perform local/global bundle adjustment and loop closures (Chapter 7).

- The first piece of work which performs depth prediction at frame-rate on an NVIDIA-TX2. Due to the lack of pretrained models for the novel smaller architecture, different loss functions were investigated with extensive experiments in order to transfer the knowledge of the state-of-the-art depth

estimation network into the smaller network. In addition to the experiments that quantify CNN based depth predictions, a practical SLAM example was designed to show the efficacy of the predicted depth maps generated from the condensed model (Chapter 8)

- Using depth map predictions to actively combat scale-drift when performing visual odometry (Chapters 7 and 8)

9.2 Future Work

The work presented in this thesis can be extended not only to improve the reconstruction problem, but also to tackle closely related areas such as place recognition.

- MO-SLAM can be coupled with a CNN-based semantic segmentation system to allow the framework to perform both supervised and unsupervised discovery. Furthermore, novel objects can be discovered in an unsupervised manner in dynamic environments by treating the set of points that moved together as a pseudo duplicate object. As I mentioned in Section 4.5, progress has been made on both fronts and I envision these additions will improve the utility of MO-SLAM.

- Similar to predicting a 2D Gaussian corresponding to the optical flow and the flow confidences as shown in Chapter 7, confidence estimation networks (depth) introduced in Chapter 6 can be improved by learning to predict a 1D Gaussian, where the mean represents the depth and the sigma represents the uncertainty of the estimate[101]. This should ultimately lead to more robust fusion and sparse depth map inpainting.

- Chapter 8 presented a model compression technique to condense the knowl-

edge of a large CNN into a smaller network. This work is currently being extended to create a joint network capable of predicting depth and semantic information in real-time on a mobile framework.

- Recognizing a place previously visited is an important feature in any navigation system. While appearance based methods [148, 31, 61] successfully tackle this problem in most cases, if the environment has drastically changed since the first time the robot has seen a particular place, most of these approaches fail to correctly identify the two locations to be the same. The same place can appear different due to changes in season, weather or time of the day (for e.g midday vs midnight). However, place recognition under challenging environmental conditions is specially important for long-term tracking and mapping systems. We believe the approach presented in Chapter 7 can be useful in this scenario. Firstly, the depth and optical flow subnetworks need to be fine-tuned in order to be able to predict depth and flow under different conditions. The predicted information can then be used to obtain a crude estimate of the relative camera pose.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. TensorFlow: Large-scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] P. Agrawal, J. Carreira, and J. Malik. Learning to See by Moving. In *International Conference on Computer Vision (ICCV)*. IEEE, 2015.

[3] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 34(11), 2012.

[4] Y. Alshawabkeh, N. Haala, and D. Fritsch. Range Image Segmentation Using the Numerical Description of Mean Curvature Values. In *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS)*, 2008.

[5] A. Bansal, B. Russell, and A. Gupta. Marr Revisited: 2D-3D Alignment via Surface Normal Prediction. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

[6] S. Y. Bao and S. Savarese. Semantic Structure from Motion. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2011.

[7] J. T. Barron and B. Poole. The Fast Bilateral Solver. In *European Conference on Computer Vision (ECCV)*. Springer, 2016.

[8] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3), 2008.

[9] P. Besl and R. Jain. Segmentation Through Variable-Order Surface Fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 10(2), 1988.

[10] P. J. Besl and N. D. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2), 1992.

[11] M. Bizjak. 3D Reconstruction of Buildings from LiDAR Data. In *The Central European Seminar on Computer Graphics*, 2015.

[12] C. Bucilu, R. Caruana, and A. Niculescu-Mizil. Model Compression. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. ACM, 2006.

[13] T. Butkiewicz. Low-Cost Coastal Mapping Using Kinect v2 Time-of-Flight Cameras. In *Oceans*. IEEE, 2014.

[14] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions. In *Robotics: Science and Systems (RSS)*, 2013.

[15] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *IEEE Transactions on Robotics*, 32(6), 2016.

[16] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary Robust Independent Elementary Features. In *European Conference on Computer Vision (ECCV)*. Springer, 2010.

[17] B. Caputo, K. Sim, F. Furesjo, and A. Smola. Appearance-based Object Recognition using SVMs: Which Kernel Should I Use. In *NIPS workshop on statistical methods for computational experiments in visual processing and computer vision*, 2002.

[18] R. Caruana. Multitask Learning. In *Learning to learn*. Springer, 1998.

[19] V. Castaneda, D. Mateus, and N. Navab. SLAM combining ToF and high-resolution cameras. In *Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2011.

[20] R. O. Castle, G. Klein, and D. W. Murray. Combining monoSLAM with object recognition for scene augmentation using a wearable camera. *Image and Vision Computing*, 28(11), 2010.

[21] O. Chapelle, P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng. Boosted Multi-Task Learning. *Machine learning*, 85(1-2), 2011.

[22] J. Chen, D. Bautembach, and S. Izadi. Scalable Real-time Volumetric Surface Reconstruction. *ACM Transactions on Graphics (ToG)*, 32(4), 2013.

[23] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *arXiv preprint arXiv:1606.00915*, 2016.

[24] M. Cho, Y. M. Shin, and K. M. Lee. Unsupervised Detection and Segmentation of Identical Objects. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2010.

[25] S. Choudhary, A. J. B. Trevor, H. I. Christensen, and F. Dellaert. SLAM with Object Discovery, Modeling and Mapping. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2014.

[26] J. Civera, A. J. Davison, and J. M. Montiel. Inverse depth parametrization for monocular SLAM. *IEEE Transactions on Robotics*, 24(5), 2008.

[27] J. Civera, D. Gálvez-López, L. Riazuelo, J. D. Tardós, and J. M. M. Montiel. Towards semantic SLAM using a monocular camera. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011.

[28] A. Cohen, C. Zach, S. N. Sinha, and M. Pollefeys. Discovering and Exploiting 3D Symmetries in Structure from Motion. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[29] A. Concha and J. Civera. DPPTAM: Dense Piecewise Planar Tracking and Mapping from a Monocular Sequence. In *International Conference on Intelligent*

Robots and Systems (IROS). IEEE, 2015.

[30] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

[31] M. Cummins and P. Newman. Appearance-only SLAM at large scale with FAB-MAP 2.0. *The International Journal of Robotics Research (IJRR)*, 30(9), 2011.

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

[33] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis & Machine Intelligence (PAMI)*, 29(6), 2007.

[34] J. S. Denker, W. Gardner, H. P. Graf, D. Henderson, R. Howard, W. Hubbard, L. D. Jackel, H. S. Baird, and I. Guyon. Neural Network Recognizer for Hand-Written Zip Code Digits. In *Advances in Neural Information Processing Systems (NIPS)*, 1989.

[35] J. S. Denker and Y. Lecun. Transforming Neural-Net Output Levels to Probability Distributions. In *Advances in Neural Information Processing Systems (NIPS)*, 1991.

[36] A. Der Kiureghian and O. Ditlevsen. Aleatory or epistemic? Does it matter? *Structural Safety*, 31(2), 2009.

[37] E. Devernay and O. Faugeras. Straight lines have to be straight: automatic calibration and removal of distortion from scenes of structured enviroments. *Machine Vision and Applications*, 13(1), 2001.

[38] T. Dharmasiri, V. Lui, and T. Drummond. MO-SLAM: Multi object SLAM with run-time object discovery through duplicates. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016.

[39] T. Dharmasiri, A. Spek, and T. Drummond. Joint Prediction of Depths, Normals and Surface Curvature from RGB Images using CNNs. In *International*

Conference on Intelligent Robots and Systems (IROS). IEEE, 2017.

[40] T. Dharmasiri, A. Spek, and T. Drummond. ENG: End-to-end Neural Geometry for Robust Depth and Pose Estimation using CNNs. *arXiv preprint arXiv:1807.05705*, 2018.

[41] E. M. dos Santos and H. M. Gomes. Appearance-based object recognition using support vector machines. In *Brazilian Symposium on Computer Graphics and Image Processing*. IEEE, 2001.

[42] I. Douros and B. Buxton. Three-Dimensional Surface Curvature Estimation using Quadric Surface Patches. *Scanning*, 44(0), 2002.

[43] E. Eade and T. Drummond. Scalable Monocular SLAM. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2006.

[44] D. Eigen and R. Fergus. Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture. In *International Conference on Computer Vision (ICCV)*. IEEE, 2015.

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

[46] A. F. Elaksher, J. S. Bethel, et al. Reconstructing 3d buildings from lidar data. *International Archives Of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3A), 2002.

[47] J. Engel, V. Koltun, and D. Cremers. Direct Sparse Odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, PP(99), 2017.

[48] J. Engel, T. Schops, and D. Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.

[49] J. Engel, J. Stueckler, and D. Cremers. Large-scale direct slam with stereo cameras. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015.

[50] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The

pascal visual object classes (voc) challenge. *International Journal of Computer Vision (IJCV)*, 88(2), 2010.

[51] J. M. Fácil, A. Concha, L. Montesano, and J. Civera. Single-view and multi-view depth fusion. *IEEE Robotics and Automation Letters*, 2(4), 2017.

[52] D. Ferstl, R. Ranftl, M. Ruther, and H. Bischof. Multi-modality depth map fusion using primal-dual optimization. *International Conference on Computational Photography (ICCP)*, 2013.

[53] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. Van der Smagt, D. Cremers, and T. Brox. Flownet: Learning Optical Flow with Convolutional Networks. In *International Conference on Computer Vision (ICCV)*. IEEE, 2015.

[54] M. A. Fischler and R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Graphics and Image Processing*, 24(6), 1981.

[55] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO : Fast Semi-Direct Monocular Visual Odometry. *International Conference on Robotics and Automation (ICRA)*, 2014.

[56] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast Semi-Direct Monocular Visual Odometry. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2014.

[57] D. F. Fouhey, A. Gupta, and M. Hebert. Unfolding an Indoor Origami World. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.

[58] Y. Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[59] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Appendix. *arXiv:1506.02157*, 2015.

[60] D. Gálvez-López and J. D. Tardós. Bags of binary words for fast place recognition in image sequences. *Transactions on Robotics*, 28(5), 2012.

[61] D. Gálvez-López and J. D. Tardos. Bags of Binary Words for Fast Place Recognition in Image Sequences. *IEEE Transactions on Robotics*, 28(5), 2012.

[62] R. Garg, V. K. BG, G. Carneiro, and I. Reid. Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue. In *European Conference on Computer Vision (ECCV)*. Springer, 2016.

[63] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets Robotics: The KITTI Dataset. *The International Journal of Robotics Research (IJRR)*, 32(11), 2013.

[64] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1), 2006.

[65] X. Gibert, V. M. Patel, and R. Chellappa. Deep Multitask Learning for Railway Track Inspection. *Transactions on Intelligent Transportation Systems*, 18(1), 2017.

[66] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. Society for Artificial Intelligence and Statistics, 2010.

[67] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised Monocular Depth Estimation with Left-Right Consistency. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[68] K. Grauman and T. Darrell. Unsupervised learning of categories from sets of partially matching image features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2006.

[69] A. Gressin, C. Mallet, J. Demantké, and N. David. Towards 3d lidar point cloud registration improvement using optimal neighborhood knowledge. *ISPRS journal of photogrammetry and remote sensing*, 79, 2013.

[70] W. Griffin, Y. Wang, D. Berrios, and M. Olano. Real-time GPU surface curvature estimation on deforming meshes and volumetric data sets. *IEEE Transactions on Visualization and Computer Graphics*, 18(10), 2012.

[71] M. Gualtieri, A. ten Pas, K. Saenko, and R. Platt. High precision grasp pose

detection in dense clutter. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016.

[72] M. Guillaumin, J. Verbeek, and C. Schmid. Multimodal semi-supervised learning for image classification. In *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2010.

[73] A. Haar. Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, 69(3), 1910.

[74] S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149*, 2015.

[75] C. Hane, L. Ladicky, and M. Pollefeys. Direction Matters: Depth Estimation with a Surface Normal Classifier. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.

[76] C. Harris and M. Stephens. A Combined Corner and Edge Detector. In *Alvey Vision Conference*, 1988.

[77] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[78] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *International Conference on Computer Vision (ICCV)*. IEEE, 2015.

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

[80] M. Hebert. Outdoor scene analysis using range data. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 1986.

[81] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *The International Journal of Robotics Research (IJRR)*, 31(5), 2012.

[82] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

[83] S. Hochdorfer and C. Schlegel. 6 DoF SLAM using a ToF camera: The challenge of a continuously growing number of landmarks. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010.

[84] S. Hong, H. Ko, and J. Kim. VICP: Velocity Updating Iterative Closest Point Algorithm. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2010.

[85] H. Hoppe. Progressive meshes. In *Conference on Computer Graphics and Interactive Techniques*. ACM, 1996.

[86] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3), 1981.

[87] B. K. P. Horn, H. M. Hilden, and S. Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5(7), 1988.

[88] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[89] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *International Symposium on Robotics Research (ISRR)*, 2011.

[90] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. Densely Connected Convolutional Networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[91] P. Huber. *Robust Statistics*. Wiley, 1981.

[92] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[93] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167*, 2015.

[94] M. Jaritz, R. De Charette, E. Wirbel, X. Perrotton, and F. Nashashibi. Sparse and Dense Data with CNNs: Depth Completion and Semantic Segmentation. In *International Conference on 3D Vision (3DV)*. IEEE, 2018.

[95] D. Jayaraman and K. Grauman. Learning Image Representations Tied to Egomotion. In *International Conference on Computer Vision (ICCV)*. IEEE, 2015.

[96] J. Jeon and S. Lee. Reconstruction-based Pairwise Depth Dataset for Depth Image Enhancement Using CNN. In *European Conference on Computer Vision (ECCV)*, 2018.

[97] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[98] N. Jiang. Seeing double without confusion: Structure-from-motion in highly ambiguous scenes. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012.

[99] A. Karpathy, S. Miller, and F. F. Li. Object Discovery in 3D Scenes via Shape Analysis. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2013.

[100] A. Kendall and R. Cipolla. Modelling Uncertainty in Deep Learning for Camera Relocalization. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2016.

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

[102] A. Kendall, Y. Gal, and R. Cipolla. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In *arXiv:1705.07115*, May 2017.

[103] K. Khoshelham and S. O. Elberink. Accuracy and resolution of Kinect depth data for indoor mapping applications. *Sensors*, 12(2), Jan 2012.

[104] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[105] G. Klein and D. Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 2007.

[106] P. Krähenbühl and V. Koltun. Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[107] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[108] A. Krogh and J. A. Hertz. A Simple Weight Decay Can Improve Generalization. In *Advances in Neural Information Processing Systems (NIPS)*, 1992.

[109] S. Kumar and M. Hebert. Discriminative random fields. *International Journal of Computer Vision (IJCV)*, 68(2), 2006.

[110] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A General Framework for Graph Optimization. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2011.

[111] Y. Kuznietsov, J. Stückler, and B. Leibe. Semi-Supervised Deep Learning for Monocular Depth Map Prediction. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[112] L. Ladicky, J. Shi, and M. Pollefeys. Pulling Things out of Perspective. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2014.

[113] L. Ladicky, B. Zeisl, and M. Pollefeys. Discriminatively Trained Dense Surface Normal Estimation. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.

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

[115] Q. V. Le. Building high-level features using large scale unsupervised learning. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013.

[116] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten Digit Recognition with a Back-Propagation network. In *Advances in Neural Information Processing Systems (NIPS)*, 1990.

[117] H. Lee, C. Ekanadham, and A. Y. Ng. Sparse deep belief net model for visual area V2. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[118] J. Lee, S. Kim, and S.-J. Kim. Mesh segmentation based on curvatures using the GPU. *Multimedia Tools and Applications*, 74(10), 2014.

[119] A. Levin, D. Lischinski, and Y. Weiss. Colorization using Optimization. In *International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM, 2004.

[120] Z. Li and D. Hoiem. Learning without Forgetting. In *European Conference on Computer Vision (ECCV)*. Springer, 2016.

[121] G. Lin, A. Milan, C. Shen, and I. Reid. RefineNet: Multi-path Refinement Networks for High-Resolution Semantic Segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

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

[123] J. Liu, C. Li, X. Fan, and Z. Wang. Reliable Fusion of Stereo Matching and Depth Sensor for High Quality Dense Depth Maps. *Sensors*, 15(8), 2015.

[124] J. Liu and Y. Liu. GRASP Recurring Patterns from a Single View. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2013.

[125] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

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

[127] V. Lui and T. Drummond. An Iterative 5-pt Algorithm for Fast and Robust Essential Matrix Estimation. In *British Machine Vision Conference (BMVC)*, 2013.

[128] V. Lui and T. Drummond. Image Based Optimisation without Global Consistency for Constant Time Monocular Visual SLAM. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.

[129] V. Lui, D. Gamage, and T. Drummond. Fast Inverse Compositional Image Alignment with Missing Data and Re-weighting. In *British Machine Vision Conference (BMVC)*, 2015.

[130] L. Ma and G. Sibley. Unsupervised Dense Object Discovery, Detection, Tracking and Reconstruction. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.

[131] Z. C. Marton, R. B. Rusu, and M. Beetz. On Fast Surface Reconstruction Methods for Large and Noisy Point Clouds. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2009.

[132] S. May, D. Dröschel, S. Fuchs, D. Holz, and A. Nüchter. Robust 3D-Mapping with Time-of-Flight Cameras. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009.

[133] J. McCormac, R. Clark, M. Bloesch, A. Davison, and S. Leutenegger. Fusion++: Volumetric Object-Level SLAM. In *International Conference on 3D Vision (3DV)*. IEEE, 2018.

[134] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), 1943.

[135] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid. RSLAM: A System

for Large-Scale Mapping in Constant-Time Using Stereo. *International Journal of Computer Vision (IJCV)*, 94(2), June 2010.

[136] J. Michels, A. Saxena, and A. Y. Ng. High Speed Obstacle Avoidance using Monocular Vision and Reinforcement Learning. In *International Conference on Machine Learning (ICML)*. ACM, 2005.

[137] N. J. Mitra, N. Gelfand, H. Pottmann, and L. Guibas. Registration of point cloud data from a geometric optimization perspective. In *SIGGRAPH symposium on Geometry processing*. ACM, 2004.

[138] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *Transactions on Robotics*, 31(5), 2015.

[139] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *Transactions on Robotics*, 33(5), 2017.

[140] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *European Conference on Computer Vision (ECCV)*. Springer, 2012.

[141] Y. Nesterov. A Method for Solving a Convex Programming Problem with Convergence Rate $O(1/K^2)$. *Soviet Mathematics Doklady*, 27(2), 1983.

[142] R. A. Newcombe, D. Fox, and S. M. Seitz. DynamicFusion: Reconstruction and Tracking of Non-rigid Scenes in Real-Time. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.

[143] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 2011.

[144] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. DTAM : Dense Tracking and Mapping in Real-Time. In *International Conference on Computer Vision (ICCV)*. IEEE, 2011.

[145] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Recon-

struction at Scale using Voxel Hashing. *ACM Transactions on Graphics (ToG)*, 32(6), 2013.

[146] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using em. *Machine learning*, 39(2-3), 2000.

[147] D. Nister. An Efficient Solution to the Five-Point Relative Pose Problem. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2003.

[148] D. Nister and H. Stewenius. Scalable Recognition with a Vocabulary Tree. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2006.

[149] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6D SLAM3D mapping outdoor environments. *Journal of Field Robotics*, 24(8-9), 2007.

[150] NVIDIA. NVIDIA TensorRT - Programmable Inference Accelerator, 2018. URL: https://developer.nvidia.com/tensorrt.

[151] P. Pahlavani, H. Amini Amirkolaee, and B. Bigdeli. 3d reconstruction of buildings from lidar data considering various types of roof structures. *International journal of remote sensing*, 38(5):1451–1482, 2017.

[152] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[153] A. Pfrunder, P. V. Borges, A. R. Romero, G. Catt, and A. Elfes. Real-time autonomous ground vehicle navigation in heterogeneous environments using a 3D LiDAR. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.

[154] S. Pillai and J. Leonard. Monocular SLAM Supported Object Recognition. In *Robotics, Science and Systems Conference (RSS)*, 2015.

[155] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5), 1964.

[156] A. Quattoni, M. Collins, and T. Darrell. Conditional Random Fields for Object Recognition. In *Advances in Neural Information Processing Systems (NIPS)*,

2005.

[157] M. Rad and V. Lepetit. BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth. In *International Conference on Computer Vision (ICCV)*. IEEE, 2017.

[158] A. Ranjan and M. J. Black. Optical Flow Estimation Using A Spatial Pyramid Network. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[159] J. Redmon and A. Angelova. Real-time grasp detection using convolutional neural networks. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.

[160] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

[161] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid. EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.

[162] R. Roberts, S. N. Sinha, R. Szeliski, and D. Steedly. Structure from motion for scenes with large duplicate structures. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2011.

[163] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo. ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1), 2018.

[164] E. Rosten and T. Drummond. Machine Learning for High-Speed Corner Detection. In *European Conference on Computer Vision (ECCV)*. Springer, 2006.

[165] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *International Conference on Computer Vision (ICCV)*. IEEE, 2011.

[166] S. Rusinkiewicz. Estimating Curvatures and Their Derivatives on Triangle Meshes. In *Symposium on 3D Data Processing, Visualization and Transmission*

(3DPVT). IEEE, 2004.

[167] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3), 2015.

[168] R. F. Salas-Moreno, B. Glocker, P. H. J. Kelly, and A. J. Davison. Dense planar SLAM. In *IEEE International Symposium on Mixed and Augmented Reality*, pages 367–368, 2014.

[169] R. F. Salas-Moreno, R. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison. SLAM++: Simultaneous localisation and mapping at the level of objects. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1352–1359, 2013.

[170] H. Sarbolandi, D. Lefloch, and A. Kolb. Kinect range sensing: Structured-light versus Time-of-Flight Kinect. *Computer vision and image understanding*, 139, 2015.

[171] A. Saxena, S. H. Chung, and A. Y. Ng. Learning depth from single monocular images. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[172] J. Shi and C. Tomasi. Good Features to Track. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 1994.

[173] J. Shotton, M. Johnson, and R. Cipolla. Semantic Texton Forests for Image Categorization and Segmentation. In *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2008.

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

[175] J. Sivic, B. C. Russell, A. Zisserman, W. T. Freeman, and A. Efros. Unsupervised Discovery of Visual Object Class Hierarchies. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2008.

[176] S. M. Smith and J. M. Brady. SUSAN - A New Approach to Low Level Image Processing . *International Journal of Computer Vision (IJCV)*, 23(1), 1997.

[177] A. Spek, T. Dharmasiri, and T. Drummond. CReaM: Condensed Real-time Models for Depth Prediction using Convolutional Neural Networks. *arXiv preprint arXiv:1807.08931*, 2018.

[178] A. Spek and T. Drummond. A Fast Method For Computing Principal Curvatures From Range Images. In *Australasian Conference on Robotics and Automation (ACRA)*. ARAA, 2015.

[179] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15(1), 2014.

[180] H. Stewenius, C. Engels, and D. Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4), 2006.

[181] H. Strasdat, A. J. Davison, J. M. Montiel, and K. Konolige. Double window optimisation for constant time visual SLAM. In *International Conference on Computer Vision (ICCV)*. IEEE, 2011.

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

[183] D. Sun, S. Roth, and M. J. Black. Secrets of Optical Flow Estimation and their Principles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2010.

[184] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz. PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume. *arXiv preprint arXiv:1709.02371*, 2017.

[185] S. Sun, F. Zhuge, J. Rosenberg, R. M. Steiner, G. D. Rubin, and S. Napel. Learning-enhanced simulated annealing: method, evaluation, and application to lung nodule registration. *Applied Intelligence*, 28(1), 2008.

[186] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Van-

houcke, A. Rabinovich, et al. Going Deeper with Convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.

[187] K. Tateno, F. Tombari, I. Laina, and N. Navab. CNN-SLAM: Real-Time Dense Monocular SLAM With Learned Depth Prediction. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[188] C. Tomasi and R. Manduchi. Bilateral Filtering for Gray and Color Images. In *International Conference on Computer Vision (ICCV)*. IEEE, 1998.

[189] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle Adjustment – A Modern Synthesis. In *International Workshop on Vision Algorithms*. Springer, 1999.

[190] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger. Sparsity invariant cnns. *arXiv preprint arXiv:1708.06500*, 2017.

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

[192] V. Verma, R. Kumar, and S. Hsu. 3D Building Detection and Modeling from Aerial LIDAR Data. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2006.

[193] P. Viola and M. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2001.

[194] X. Wang, D. Fouhey, and A. Gupta. Designing Deep Networks for Surface Normal Estimation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.

[195] X. Wang, C. Zhang, and Z. Zhang. Boosted Multi-Task Learning for Face Verification With Applications to Web Image and Video Search. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2009.

[196] X. J. Wang, L. Zhang, and C. Liu. Duplicate Discovery on 2 Billion Internet Images. In *Conference on Computer Vision and Pattern Recognition Workshops*.

IEEE, 2013.

[197] A. S. Wannenwetsch, M. Keuper, and S. Roth. Probflow: Joint Optical Flow and Uncertainty Estimation. In *International Conference on Computer Vision (ICCV)*. IEEE, 2017.

[198] O. Wasenmüller, M. D. Ansari, and D. Stricker. DNA-SLAM: Dense Noise Aware SLAM for ToF RGB-D Cameras. In *Asian Conference on Computer Vision (ACCV)*. Springer, 2016.

[199] O. Wasenmüller and D. Stricker. Comparison of Kinect V1 and V2 Depth Images in Terms of Accuracy and Precision. In *Asian Conference on Computer Vision (ACCV)*. Springer, 2016.

[200] C. Weerasekera, T. Dharmasiri, R. Garg, T. Drummond, and I. Reid. Just-in-Time Reconstruction: Inpainting Sparse Maps using Single View Depth Predictors as Priors. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.

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

[202] K. Wilson and N. Snavely. Network Principles for SfM: Disambiguating Repeated Structures with Local Context. In *International Conference on Computer Vision (ICCV)*. IEEE, 2013.

[203] Y. Yan, E. Ricci, R. Subramanian, G. Liu, O. Lanz, and N. Sebe. A Multi-Task Learning Framework for Head Pose Estimation under Target Motion Related Articl. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 38(6), 2016.

[204] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. LIFT: Learned Invariant Feature Transform. In *European Conference on Computer Vision (ECCV)*. Springer, 2016.

[205] F. Yu and V. Koltun. Multi-scale Context Aggregation by Dilated Convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[206] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.

[207] J. Zhang and S. Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.

[208] Y. Zhang and T. Funkhouser. Deep Depth Completion of a Single RGB-D Image. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.

[209] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia. ICNet for Real-Time Semantic Segmentation on High-Resolution Images. *arXiv preprint arXiv:1704.08545*, 2017.

[210] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[211] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object Detectors Emerge in Deep Scene CNNs. In *International Conference on Learning Representations (ICLR)*, 2015.

[212] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. Unsupervised Learning of Depth and Ego-Motion From Video. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[213] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, 2002.