



MONASH University

**Low-cost Depth Sensors to Real-time Monocular Depth
Prediction for Improved Mapping**

Andrew Spek
Monash University

Supervised by Prof. Tom Drummond

A thesis submitted for the degree of Doctor of Philosophy at
Monash University in 2018
Electrical and Computer Systems Engineering

Copyright notice

©Andrew Spek 2018

I certify that I have made all reasonable efforts to secure copyright permissions for third-party content included in this thesis and have not knowingly added copyright content to my work without the owner's permission.

Abstract

This thesis focuses on improving the quality of mapping and modelling approaches using low-cost depth sensors and machine learning. We focus on optimisation and machine learning approaches, and how these can be effectively combined to maximise the quality of sensor data. The goal is to enable a robot equipped with a monocular or depth sensor to more effectively interact with the world.

Using low-cost sensors, means the data produced can be noisy and imperfect. These imperfections can lead to significantly compromised performance in a mapping scenario. To accommodate this, the initial work performed was a novel approach to camera calibration, that is based on the desirable unsupervised mapping based calibration. This allowed the sensor to be calibrated using far fewer parameters than previous approaches, while generating improved calibration performance and requiring less intervention than some previous approaches.

Dense mapping approaches have a number of challenges, particularly re-localisation, the process of recovering the relative camera location after tracking is lost. To combat this, the idea was to compute a discriminative view-point invariant surface feature, namely surface curvature. A novel real-time approach to surface curvature estimation was developed for this thesis and demonstrated improved performance and speed of computation. Previous approaches generally fail to cope with the noisy nature of the data produced by low-cost depth sensors, or have computational requirements too demanding for real-time robotics applications. This work was extended to incorporate relative pose alignments into a novel joint optimisation function that produces both improved surface curvature estimates, and much better relative pose estimates. The joint nature of this approach provides a positive feedback loop that successively improves the performance on both tasks. The curvature values produced using the initial approach were also used to generate data for a novel machine learning approach to curvature estimation using colour.

As low-cost depth sensors have been found to have several failure cases, and suffer drawbacks in terms of noise and accuracy, we attempt to create a low-cost depth sensor from a standard colour camera. We use machine learning to predict a plausible depth image for any colour image, using purely visual clues in the data. This approach is shown to be improved by using multiple related loss functions, and even extended to incorporate relative pose alignments in a joint machine learning and classical optimisation approach. These novel network implementations resulted in state-of-the-art depth estimations. As a proof of concept, a machine learning depth estimation approach was developed which returns real-time depth estimates on low-power hardware. This work focuses on improving reliability through novel network knowledge transference techniques, and a real-time implementation that could be used to immediately improve a standard mapping approach. These works map a path towards low-cost depth sensor replacement and improvement, via machine learning.

The contributions of the thesis are presented, rigorously tested, and discussed using synthetic and real-world experiments.

Declaration

This thesis contains no material which has been accepted for the award of 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:

A solid black rectangular box used to redact the signature.

Print Name: Andrew Spek

Date: 7/5/2017

Contents

1	Introduction	2
1.1	Simultaneous Localisation and Mapping (SLAM)	3
1.1.1	Sparse Feature Based	3
1.1.2	Semi-Dense Featureless Monocular	4
1.1.3	Dense Depth Based	4
1.1.4	Iterative Closest Point (ICP)	5
1.2	Low-Cost Depth Sensors	6
1.3	Surface Curvature Estimation from Low-cost Depth Sensor Data	7
1.3.1	Real-time Surface Curvature	8
1.3.2	Surface Curvature for Improved Surface Alignment	8
1.4	Machine Learning Using Convolutional Neural Networks (CNNs)	9
1.4.1	Neural Networks	10
1.4.2	Convolutional Neural Networks (CNNs)	10
1.4.3	Extracting Geometry from Colour	12
1.4.4	Choosing the Correct Objective Functions	13
1.4.5	Towards Real-time Performance	14
1.5	Publications	17
1.6	Software and Hardware Libraries	18
1.6.1	Software	18
1.6.2	Hardware	18
2	Related Work	19
2.1	Low-cost Depth Sensors	19
2.1.1	Sensor Types	19
2.1.1.1	Stereo Camera Systems	19
2.1.1.2	Time-of-Flight (ToF)	20
2.1.1.3	Structured Light	21
2.1.2	Calibration Approaches	22
2.1.2.1	Supervised and Semi-supervised	22
2.1.2.2	Mapping Based	23
2.2	Depth in Robotics	24
2.2.1	Geometric Feature Estimation	24
2.2.1.1	Normals from Depth	24
2.2.1.2	Curvature from Depth	25
2.2.1.3	Features from Geometry	26
2.2.2	Localisation and Mapping	28

CONTENTS

2.2.2.1	Feature-based Approaches to Mapping	29
2.2.2.2	Using Geometry Approaches to Mapping	30
2.2.2.3	Combining Depth and Colour for Alignment	33
2.2.3	Scene Segmentation	34
2.2.3.1	Colour Based Segmentation	34
2.2.3.2	Geometry Based Segmentations	36
2.2.3.3	Combining Depth and Colour Information	37
2.3	Machine-Learning	38
2.3.0.4	Supervised vs Unsupervised Learning	40
2.3.0.5	Semantic Representations	40
2.4	Depth without a Low-cost Depth Sensor	42
2.4.1	Semi-Dense Depth Estimation	43
2.4.2	Dense Depth Estimation	43
2.4.3	Machine-Learning Approaches	44
2.4.4	Towards Real-time	46
3	Fundamentals	48
3.1	Points, Vectors, Matrices, and Tensors	48
3.1.1	Points	48
3.1.2	Vectors	48
3.1.3	Matrices and Tensors	48
3.1.4	Matrix Transpose	49
3.1.5	Matrix Inverse	49
3.1.6	Matrix Decomposition	50
3.1.7	Convolution	51
3.2	Camera Models and Image Projection	52
3.2.1	Pinhole-Camera	52
3.2.2	Homogeneous Coordinates	53
3.2.3	Camera / Projection via Matrices	54
3.2.4	Lens Distortion	54
3.2.4.1	Radial	56
3.2.4.2	Tangential	56
3.3	Depth Camera Variants	56
3.3.1	Time-of-flight (ToF)	56
3.3.1.1	Operation	56
3.3.1.2	Sensor Issues	57
3.3.2	Stereo	58
3.3.2.1	Operation	59
3.3.2.2	Issues	60
3.3.3	Structured-Light	60
3.3.3.1	Operation	61
3.3.3.2	Use in Robotics	63
3.3.3.3	Issues	64

CONTENTS

3.4	Review of Calculus	64
3.4.1	Multi-Variate Differentiation	64
3.4.1.1	Matrices	65
3.4.2	Vector/Scalar Fields and The Jacobian Matrix (\mathbf{J})	65
3.4.3	The Hessian Matrix (\mathbf{H}) and Approximate Hessian ($\mathbf{J}^T \mathbf{J}$)	66
3.4.4	Taylor Series Approximations	67
3.4.5	Quadrics	67
3.5	Optimization Techniques	70
3.5.1	Gaussian Elimination	70
3.5.2	Back-Substitution	71
3.5.3	Least-Squares	73
3.5.3.1	Ordinary Least-Squares (OLS)	73
3.5.4	Iterative Optimisation Approaches	75
3.5.4.1	Gradient Descent	75
3.5.4.2	Newton Method	75
3.5.4.3	Gauss-Newton Method	76
3.5.4.4	Outlier rejection	78
3.5.4.5	Weighted Least-Squares (WLS)	78
3.5.4.6	Levenburg-Marquadt	79
3.5.5	Robust Optimisation	79
3.5.5.1	Maximum Likelihood Estimators (M-Estimators)	79
3.5.6	Schurr-Complement Trick	81
3.5.6.1	Block-Diagonal Matrices	82
3.5.6.2	Symmetric Matrices	82
3.5.6.3	Use in Optimisation	83
3.6	Transformation Matrices	84
3.6.1	Groups	84
3.6.2	Matrix Exponentiation	85
3.6.3	Lie Groups, Lie Algebras and The Tangent Space	85
3.6.3.1	Lie Groups and Lie Algebras	85
3.6.4	Special Orthogonal Groups ($\mathbf{SO}(N)$)	86
3.6.4.1	Rotation in Two Dimensions ($\mathbf{SO}(2)$)	86
3.6.4.2	Rotation in Three Dimensions ($\mathbf{SO}(3)$)	87
3.6.5	Special Euclidean Groups ($\mathbf{SE}(N)$)	88
3.6.5.1	General 6DOF Transformations ($\mathbf{SE}(3)$)	88
3.6.6	Differentiating Transformation Matrices	89
3.7	Iterative Closest Point (ICP)	90
3.7.1	Selection	91
3.7.1.1	RANdom SAMpling Consensus (RANSAC)	91
3.7.1.2	Dense Sampling	91
3.7.2	Correspondence Estimation	91
3.7.3	Weighting & Outlier Removal	93
3.7.4	Error Metric	93

CONTENTS

3.7.5	Error Minimisation	94
3.7.6	Wide Baseline Problem	94
3.8	Surface Curvature	96
3.8.1	Relation to the Second Fundamental Form (II)	96
3.9	General Purpose Graphics Processing Unit (GPGPU) Programming	98
3.9.1	Serial vs Parallel Programming	98
3.9.2	Streaming Multi-processors and Thread Blocks	100
3.9.3	Memory and Coherency	101
3.9.4	Application to Robotics	102
3.10	Convolutional Neural Networks (CNNs)	102
3.10.1	Activations and Weights	103
3.10.2	Convolutional, Pooling and Fully-Connected Layers	103
3.10.2.1	Convolutions and De-convolutions (Transposed)	103
3.10.2.2	Pooling and Un-pooling	105
3.10.2.3	Fully-connected	105
3.10.3	Rectifiers and Drop-out	106
3.10.4	Training Through Back-propagation	107
3.10.4.1	Momentum, Learning Rate and Weight-decay	107
3.10.4.2	Vanishing and Exploding Gradient Problems	109
3.10.4.3	Over-fitting	109
3.10.5	Batches and Normalisation	110
4	Depth Camera Calibration	112
4.1	Motivation	112
4.2	Contributions	113
4.3	Related Work	113
4.4	Correction Function	114
4.5	Frame-to-frame Alignment	115
4.6	Joint Multi-frame Alignment and Calibration	116
4.6.1	Pose Jacobians	117
4.6.2	Depth Correction Jacobians	119
4.6.3	Radial Distortion Correction Jacobian	120
4.6.4	Full Calibration Jacobian Matrix	120
4.7	System Implementation Details	121
4.7.1	Calibration Optimisation	121
4.7.2	Maximising Calibration Performance	122
4.7.3	System overview	123
4.8	Calibration Performance Evaluation	124
4.8.1	Choosing the Degree of the Polynomial	125
4.8.2	Visualising Calibration Function	125
4.8.3	Comparative Performance	126
4.8.4	Calibrated Dataset	128
4.8.5	Improvement to Modelling	129

CONTENTS

4.9	Conclusions and Future Work	130
4.10	Code	130
5	Surface Curvature from Quadrics	131
5.1	Motivation and Related Work	132
5.2	Contributions	133
5.3	Surface Curvature Using Quadrics	133
5.3.1	Ordered Point-clouds	134
5.3.2	Second Fundamental Form (II)	134
5.3.3	Quadrics	135
5.3.4	Creating a Local Coordinate Frame	135
5.3.5	Allowing For Further Degrees of Freedom	136
5.3.6	Iterative Surface Fitting	137
5.3.7	Extracting Principal Curvature Values	140
5.4	System Summary	140
5.5	System Evaluation	141
5.5.1	Ground Truth Curvature Dataset	141
5.5.2	Measuring System Accuracy	142
5.5.3	Evaluating View-Point Invariance	144
5.5.4	Evaluating the Effect of Noise	144
5.5.5	Refined Normal Estimation	145
5.5.6	What's the Right Size Patch?	147
5.5.7	Curvature-based Correspondence Estimates	149
5.6	Wide baseline Alignment Through Surface Curvature	151
5.7	Conclusions and Future Work	153
5.8	Code and Datasets	153
6	Joint Curvature Pose Optimisation	154
6.1	Motivation	154
6.2	Contributions	156
6.3	Related Work	156
6.4	Fundamentals	157
6.4.1	Iterative Closest Point (ICP)	157
6.4.2	Quadric Surface Representation	158
6.5	Joint Frame-to-Frame Optimisation	159
6.5.1	Implementation Details	162
6.6	Joint Multi-Frame Optimisation	164
6.6.1	Implementation Details	165
6.7	System Evaluation	167
6.7.1	Evaluation Datasets	170
6.7.2	Evaluation Metrics	170
6.7.3	Compared Approaches	171
6.8	Pose Estimation Accuracy	171
6.8.1	Synthetic Dataset Evaluation	172

CONTENTS

6.8.2	Real-world Dataset Evaluation	173
6.8.3	Pose Estimation Drift	174
6.9	Curvature Estimation Evaluation	175
6.9.1	Synthetic Dataset Evaluation	175
6.9.2	Real-World Dataset Evaluation	176
6.10	Conclusions and Future Work	177
6.11	Code/Datasets	177
7	Machine Learning Approaches for Geometric Quantity Estimation	178
7.1	Motivation	178
7.2	Contributions	180
7.3	Related Work	180
7.4	Model Architecture	182
7.5	System Task Specification	183
7.5.1	Depth	184
7.5.2	Surface Normal	184
7.5.3	Surface Curvature	185
7.6	Training The Network	185
7.6.1	Data Generation	185
7.6.2	Hyperparameters and Weight Initialisation	186
7.6.3	Training Separate Models With Equal Model Capacity	186
7.7	Evaluation Metrics	188
7.8	Network Performance Evaluation	188
7.8.1	Depth	190
7.8.2	Surface Normals	191
7.8.3	Surface Curvature	193
7.8.4	Segmentation Using Predicted Quantities	194
7.8.5	Architectural Considerations	196
7.9	Conclusions and Future Work	198
8	Using Pose Priors to Improve Depth Estimation	199
8.1	Motivation	199
8.2	Contributions	200
8.3	Related Work	200
8.3.1	Single Image Depth Prediction	201
8.3.2	Optical Flow Prediction	201
8.3.3	Pose Estimation	202
8.4	Network Architecture	202
8.5	Depth Prediction	202
8.6	Flow Prediction	204
8.7	Pose Estimation	205
8.7.1	Iterative	206
8.7.2	Fully-Connected	207
8.8	Loss Function Design	208

CONTENTS

8.8.1	Depth Losses	208
8.8.2	Flow Loss	209
8.8.3	Pose Loss	209
8.9	Network Training Regime	209
8.9.1	Depth Training	210
8.9.2	Optical Flow Training	211
8.9.3	Pose Training	211
8.10	Evaluation Metrics	212
8.11	Network Performance Evaluation	212
8.11.1	Depth Estimation	212
8.11.2	Pose Estimation	216
8.12	Ablation Experiments	220
8.13	Conclusion and Further Work	221
8.14	Code/Datasets	222
9	Real-time Depth Estimation via Model Compression	223
9.1	Motivation	223
9.2	Contributions	225
9.3	Related Work	226
9.4	Proposed Framework	227
9.4.1	Model Architecture	227
9.4.2	Loss Functions and the Knowledge Transfer Process	229
9.4.3	Datasets	232
9.4.4	Hyperparameters	232
9.5	Evaluation Metrics	233
9.6	Overall Evaluation	233
9.6.1	Depth Evaluation	233
9.6.2	Pose Estimation	235
9.6.3	Speed and Computation Performance	239
9.7	Conclusions and Further Work	240
10	Conclusion	241
10.0.1	Discussion and Future Work	242

Acknowledgements

I would like to sincerely thank my PhD supervisors Doctor Wai Ho Li and Professor Tom Drummond. You have provided me with invaluable guidance throughout this arduous and often times enjoyable process. Wai Ho, your advice and insights into work-life balance as well as the fields of computer vision and robotics has help shape me as a researcher. Tom your guidance, breadth of technical knowledge, and wealth of practical and theoretical experience has propelled me forward through this journey. My appreciation for your supervision cannot be overstated.

Thank you to the assessors I have had throughout this process, who have provided guidance and support for this research. Thank you to the Australian Centre for Robotic Vision, which has provided support financially and morally to pursue avenues of interesting and useful research in the ever changing fields of robotics and vision.

Thank you to Ben Harwood, Ben Meyer, Thanuja Dharmasiri, Vincent Lui, Winston Yii, Yan-Ming Pei and Yan Zuo, who have assisted in my understanding of countless aspects of computer vision and machine learning. In particular Vincent and Thanuja who are always willing to discuss a difficult problem. Thank you to Thanuja who has become a continued collaborator, and who has ultimately contributed significantly to the work contained in this thesis.

Thank you to my parents you have always supported me and my academic endeavours over my life. I will always be grateful to you both for providing me with the love and support I needed to achieve the things I have achieved.

Lastly a huge thank you to my long suffering partner Sisi, who I couldn't have done this without, and provided me with so much motivation and support I only hope a can provide a fraction of that back to her over the rest of our lives together.

Introduction

The goal of the research conducted in this thesis was primarily to advance the field of tracking and mapping. In an effort to achieve this, several works were produced that attempt to improve the quality of surface feature estimation, and the ability to extract relevant information from colour and depth imagery. The use of low-cost depth sensors (particularly the Kinect) motivated a great deal of the research presented in this thesis.

A novel calibration approach was proposed to improve the quality of the output data from the low-cost depth sensor, increasing the sensors usability for the robotics community (Chapter 4). A real-time approach to estimating surface curvature (Chapter 5), that subsequently was used to improve mapping accuracy (Chapter 6) was produced. This allowed a more general approach to surface alignment, and has the potential to operate in a high-accuracy mapping system in the future.

The massive resurgence in popularity of machine learning also motivated the back half of this research. Several novel approaches to depth estimation were produced (Chapters 7-9), in an effort to potentially complement the ability of the low-cost depth sensor in real robotic systems. This research moved from state-of-the-art (Chapter 8) to real-time (Chapter 9) in an effort to provide a real-time system that could be used in cooperation with a low-cost depth sensor and potentially improve the ability of a real-time mapping system.

As mentioned above, in this thesis several novel approaches are presented, relating to dense geometric data processing. In order to provide context to these contributions, some of the motivations and choices for this work are introduced in this section. The intention of Chapter 2, is to provide context to understand where this research fits in the current literature. Chapter 3 is intended to provide a sufficient level of background information to understand the subsequent theory and results. Chapters 4-9 provide a detailed description of the results and theory of several submissions that were completed during this thesis. Finally Chapter 10 provides a summary of what has been achieved, as well as some possible useful future directions this research should/could be taken.

1.1 Simultaneous Localisation and Mapping (SLAM)

Simultaneous localisation and mapping is a method of generating human and machine usable maps, that attempts to solve the chicken and egg problem of acquiring a map of a scene, and navigating a new environment. The key insight introduced in the seminal paper [1], was to simultaneously build a current best guess of a map of the environment while localising against the map being generated. This allows a robot to generate maps faster and potentially more accurately, and is still considered an extremely useful task in robotics, that remains unsolved.

This section is intended as a general overview of some popular techniques as a background to motivate the work contained in this thesis, please see Section 2.2.2 for a more detailed discussion. In this thesis our contribution was to demonstrate an implementation that improves the accuracy of relative pose alignment. Although SLAM isn't directly targeted in this thesis, several of the chapters provide improved techniques that can be generally applied to SLAM systems. In Chapter 5, I attempt to tackle the issue of producing good initial alignments for a dense alignment approach. In Chapter 6, we provide an improved alignment strategy using a more general surface representation. In Chapter 8, we attempt to get a machine to estimate the alignment between frames using ideas borrowed from previous SLAM approaches. Finally in Chapter 9, we demonstrate an approach that improves the performance of a standard SLAM algorithm without using any additional information except that encoded in a network, and demonstrate a path towards machine learning and SLAM integration. This section is intended to provide a context for the motivation of these works, and to demonstrate the variation in current approaches.

1.1.1 Sparse Feature Based

Monocular (or mono) SLAM is a technique for performing SLAM that only relies on a single camera moving through a scene. The most common approach to solving this problem is to match structures in the scene visible in multiple overlapping views, and computing the transformations that would be required to generate the observed changes between views. A common choice of matching structure is image features such as the popular Scale-Invariant Feature Transform (SIFT) feature [2] or the more recent machine learned Oriented Rotated Binary Robust Independent Elementary Features (ORB) [3] [4]. These methods are fairly similar in their approach, they wish to take an image patch around a point and compress that patch into a fixed size vector of values that is highly repeatable across multiple viewpoints of the same point. An image feature allows points to be aligned across different views, and correspondences to be formed. As these points are real points in the world, they must respect geometric constraints and this can be used to triangulate their 3D position, allowing the construction of maps.

1 INTRODUCTION

Another consideration in earlier systems [5] was computational, we may only want to match a small number of points so the approach will remain real-time. In order to cut down the number of structures to convert to features salient points in the image are chosen as the best candidates for features extraction. A very popular choice is to extract features only at the corners of images, a popular choice of corner detector is Features from Accelerated Segment Test (FAST) corner extractor [6]. Rosten *et al.* shows that these points are extremely quick to compute and provide highly discriminative image patches across multiple viewpoints. The approach in [7] uses FAST corners and ORB-features as part of their state-of-the-art tracking pipeline ORB-SLAM2. This off-the-shelf system had proven to be incredibly useful for the SLAM community and features several times in this thesis.

1.1.2 Semi-Dense Featureless Monocular

As computational speed has increased, more recent approaches attempt to use more information as part of the optimisation [8, 9, 10, 11], in an effort to both increase robustness and accuracy. These approaches have been labelled semi-dense monocular methods, as they operate densely on image edge pixels (which are semi-dense as a proportional to the image), and minimise an alignment based on minimising the photometric edge pixel re-projection error. Forster *et al.* demonstrated in [9] that by only using the edges this method reduces the overall runtime of the algorithm while keeping the most salient information for the tracking algorithm. Engel *et al.* extended this idea in [11] to model the lighting of the scene and relative brightness of pixels across the lens of a camera in order to improve the overall accuracy, due to the previous methods sensitivity to lighting changes. Additionally they use the Schurr-complement trick (see Section 3.5.6) to improve the speed of the algorithm by marginalising parameters that contribute minimally to the optimisation at each time-step. One key take-away from the work in [11] is that it emphasises the importance of modelling the sensor accurately to the task of the localisation and mapping. This suggests the importance of accurate sensor calibration, which is explored in Chapter 4. This also suggests that joint optimisation approaches can be used to more effectively solve problems jointly by using more of the available information.

1.1.3 Dense Depth Based

Since the introduction of GPU targeted parallel computing libraries such as Khronos' OpenCL and Nvidia's CUDA, the use of General Purpose GPU (GPGPU) programming has become common in robotics. GPU's are in general slower in serial clock-rate but contain many parallel processing units, which can be used to speed up computations that are parallelisable. This is true of many optimisation and computer vision problems, including surface alignments. As was shown in Section 1.1.2 using more data, even noisy



Figure 1.1: Demonstrates the alignment of multiple overlapping point clouds from a Kinect style depth sensor. Note the slight curve in the walls in the top-down view (**top-right**) caused by using an uncalibrated sensor to generate the point clouds. This alignment is performed using a dense implementation of point-to-plane ICP which is discussed in Section 3.7

data, can be used to improve the accuracy of a tracking system given a robust approach to modelling the noise. However it has traditionally been challenging to maintain real-time performance as the amount of data increases, as CPU’s haven’t increasing in speed sufficiently to allow this. Using GPGPU programming mitigates this issue and enables dense approaches to run in real-time on low-cost consumer hardware. One of the most popular techniques for point-cloud alignment (also surface/scan matching) is the ICP algorithm [12]. As ICP is used several times in this thesis as a fundamental part of some algorithms, we provide a more detailed introduction in Section 3.7, and a short one below. Figure 1.1 shows the result of aligning multiple overlapping point-clouds from a low-cost depth sensor using point-to-plane ICP, and already indicates the issue of alignment using uncalibrated data, where the model contains a slight curve (**top-right**) in the back wall. This does however demonstrate the impressive accuracy possible from the sensors.

1.1.4 Iterative Closest Point (ICP)

The goal of Iterative Closest Point (ICP) is to align multiple overlapping depth scans accurately and efficiently. The idea behind this algorithm is to iteratively reduce the distance between corresponding/closest points in multiple overlapping point-clouds. This is a popular *dense depth based* approach to surface alignment [13, 14, 15, 16, 17, 18], and was used in several works of this thesis including the works of Chapters 4 and 7. As improved modelling was a key goal of this thesis, and dense real-time surface alignment was the most accurate approach to alignment [17], reducing the issues with this

1 INTRODUCTION

method was a key motivation of the initial research. One of the main issues with ICP is the wide-baseline alignment issue. This is caused by the non-convexity of the error function defined for ICP, which allows for many local minima, resulting in sub-optimal relative pose alignments given poor initial conditions. A more detailed discussion of this problem can be found in Section 3.7.

1.2 Low-Cost Depth Sensors

The introduction of low-cost depth sensors has had a significant effect on the fields of robotics and computer vision research. Although depth sensors have existed for many years [19, 20], they had primarily been used for offline modelling. Advances in the technology improved accuracy and reduced capture time, but used expensive bulky specialised hardware [21]. The release of the Microsoft Kinect provided a low-cost all in one sensor, that operated at frame rate with an accuracy comparable to previous approaches [22]. This provided a method of collected registered depth and colour images, in real-time without requiring externally calibrated custom projection systems. This led to the introduction of other new low-cost depth sensors that also took different approaches but each produced the same result, a real-time depth/colour pair.

Devices that produce depth use a variety of approaches, the two broad categories would be stereo and timing approaches. The stereo approaches (of which Kinect falls into) use a standard geometric approach to estimating depth in the image, effectively densely triangulating points across a stereo pair. The timing based approaches, effectively work the same way a radar works, measuring the amount of time it takes for a light wave to leave and return to a camera and using the speed of light as a constant. For a more detailed explanation of both approaches please refer to Section 3.3.

The Kinect and other similar sensors were also enabled by the introduction of low-cost powerful Application Specific Integrated Circuits (ASICs). This shift towards ASICs has been increasing in recent years [23], as they provide a low-cost, low-power solution. This has become an industry standard, moving away from the more general hardware being applied to a variety of common problems, such as encryption, video encoding, pattern matching and signal processing.

Although these low-cost structured-light depth sensors (such as the Microsoft Kinect) have proved to be extremely popular in robotics and vision, particularly in a mapping context [14, 15, 16, 18, 17], the sensor’s accuracy is known to contain structured errors. As part of this research and in an effort to make this sensor more usable, a novel parametric calibration approach was developed that can be used to quickly and effectively calibrate the majority of the structural sensor noise. This approach is described and

evaluated in Chapter 4.

1.3 Surface Curvature Estimation from Low-cost Depth Sensor Data

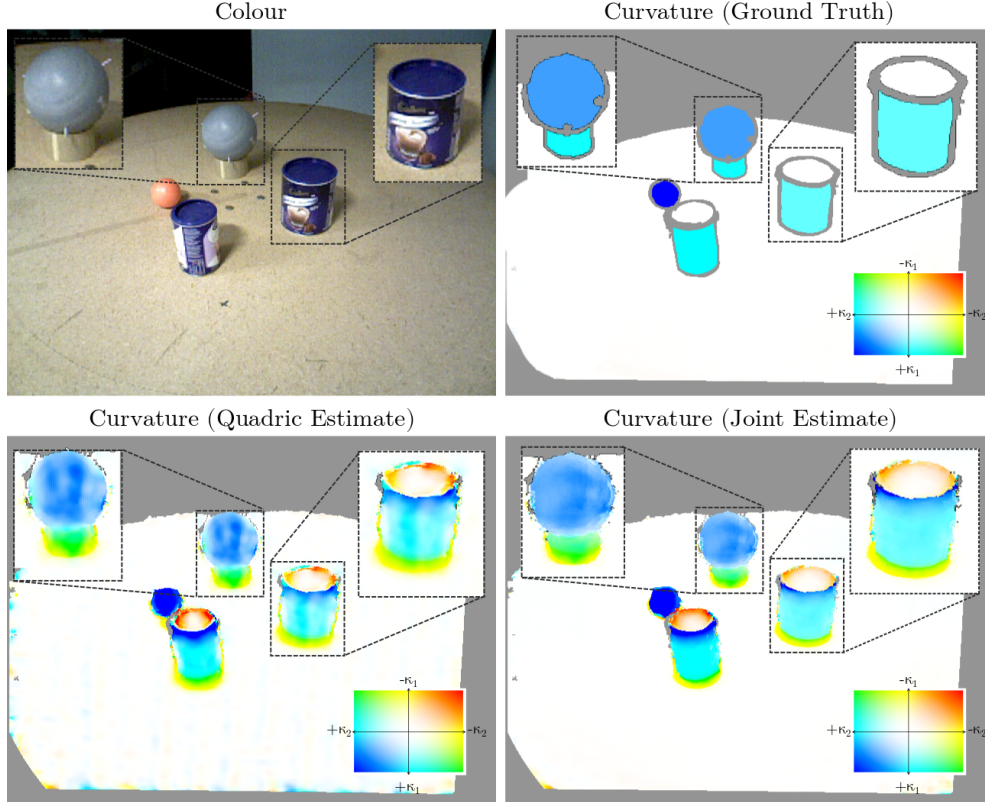


Figure 1.2: *Top Left:* shows the original colour image, *Top Right:* shows the manually labelled ground truth curvature value, *Bottom Left:* shows the result of estimating curvature using a single frame [24], *Bottom Right:* shows the result from [25], using the joint-full multi-frame optimisation. A key is included to indicate how the values of the principal curvatures (κ_1, κ_2) relate to the mapped colours, in the bottom corner of each image.

This section is intended to provide a background for the work in Chapters 5 and 6.

In order to solve the wide-baseline problem of ICP we aimed for a geometric solution. Some previous approaches tackled this issue through the use of feature alignment using photometric based features, such as SIFT [2] and ORB [3]. This is certainly a good approach, but fails when the scene texture is low and a lack of robust photometric features are computed. In order to provide a simple, geometry inspired feature, that is the basis of many geometry based features we focused on surface curvature estimation.

1 INTRODUCTION

Surface curvature is a measure of the rate of change of the surface normal direction as you move along the surface. More specifically the principle curvature, we denote this by κ_1 and κ_2 , which are the maximal and minimal changes at even given point. We denote the direction of the maximal and minimal curvature by the principle curvature directions $\bar{\kappa}_1$ and $\bar{\kappa}_2$. Although perhaps not completely intuitive, these maximal and minimal curvature directions are always orthogonal to one another. In this thesis a quadric surface approximation was used to fit a local surface and extract the curvature directly from the fitted quadric in a process described in Chapter 5.

1.3.1 Real-time Surface Curvature

The computation of surface curvature is challenging, particularly in the case of data extracted from low-cost depth sensors. As discussed previously, the data is noisy and we are trying to estimate a second-order derivative. This ends up quadratically amplifying the noise term, and potentially lowering the signal to noise ratio (SNR) to the point where the estimate becomes unreliable. The typical approach to combating noise is to use more data, which leads to increased computation but more reliable estimates. However computation time is important in a robotics context that requires real-time performance, and in this case we are attempting to use the surface curvature to align overlapping frames in a mapping system, which is certainly a real-time target.

In order to combat these opposing constraints we explored a relatively recently broadened technology at the time, namely General Purpose Graphics Processing Unit (GPGPU) programming. GPU programming has existed for many years, but only recently had the manufacturers begun supporting libraries that allowed direct interaction with GPUs for general computation. This is a paradigm shift in programming from an essentially serial processing model on the CPU to a parallel processing strategy on the GPU. The details of this paradigm are explored in Section 3.9. As this method of programming was a key enabler for large portions of this thesis, I provide its own separate description. The resulting real-time curvature computation system is detailed in Chapter 5.

1.3.2 Surface Curvature for Improved Surface Alignment

One of the most common situations a wide-baseline alignment is required is during re-localisation, which is required when tracking fails and the system becomes lost relative to the current map being built. An additional scenario when a potentially wide-baseline alignment occurs, is during a loop-closure event. This happens when the camera moves back position such that the system views as part of the map that it has previously seen before. Given an ideal mapping the current position will match up perfectly to the current map, but given the system accumulates error in pose estimates along any trajectory it can be expected that some misalignment will be present which requires compensa-

1 INTRODUCTION

tion. This detection can be performed using regions of smooth unique curvature (as shown in Section 5.6), or other photometric feature based approaches. However we can also try to reduce the initial alignment error by improving the relative pose alignments along trajectories to the point where more complex approaches, that compensate for this misalignment are not required.

In this thesis a method of reducing drift is proposed, as an extension of the previous real-time curvature estimation system. This method uses a point-to-surface implementation of ICP (as described in Section 3.7) that used quadrics as the surface representation. This improved the alignment quality of relative poses and massively reduced the drift caused by accumulation error, which could potentially allow loop-closures to be detected simply, reducing the need for wide-baseline alignment approaches. This approach leads to reduced alignment error over previous approaches, even those that are considered to be state of the art. The details of our approach are in Chapter 6. Additionally, an example of the improvement to surface curvature estimation is shown in Figure 1.2.

1.4 Machine Learning Using Convolutional Neural Networks (CNNs)

This section is intended to provide a brief introduction into machine learning, with a focus on robotic applications and in particular applications that relate to estimating geometric quantities from colour images. We provide a more in-depth exploration of machine learning approaches for robotics applications in Chapter 2, in this section we primarily focus on CNNs as they were the machine learning approach chosen in this research. The contents of this section relate primarily to the Chapters 7,8 and 9.

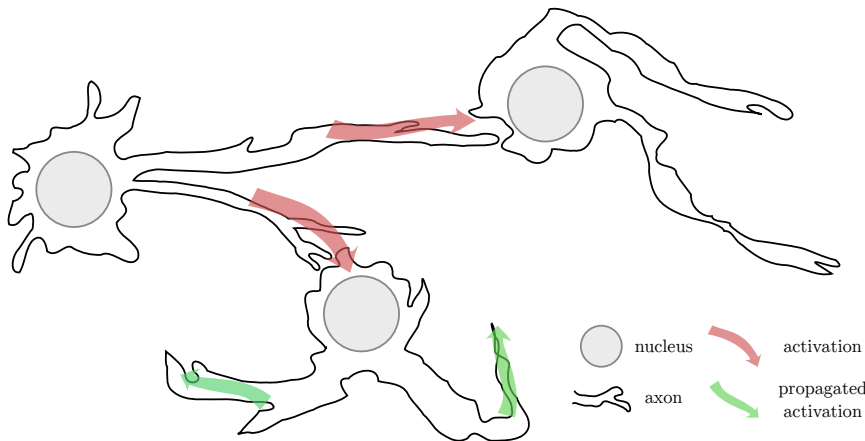


Figure 1.3: Highly simplified structure of three neurons forming a section of a neural network. We also loosely demonstrate the process of activation and how this can subsequently activate some neurons while terminating on others.

1 INTRODUCTION

1.4.1 Neural Networks

The rationale behind neural networks is to attempt to approximate the function that neurons serve in a brain, in an efficient manner. Neurons are the building blocks of brains, they are cells with a nucleus and axons that branch out, electrically connecting them to the nucleus of many other cells. This forms a *network* of neurons, where many interconnections exist between neurons and complex pathways are able to form. The neuron has the ability to send a signal along the axon, to the connected neurons, if the signal is sufficient or of a particular nature this can cause a connected neuron to activate. We show a highly simplified example of this in Figure 1.3. In practice this very simple structure exhibits emergent behaviour, in that the complex routing and activation of particular neurons causes processes to perform in a brain, which ultimately performs thoughts and recognition in minds, a highly complex behaviour. The connections that produce the desired outcome and are used frequently strengthen and potentially more efficient paths are formed, which is potentially how brains can learn new things. Emergent behaviour is very common in nature, but takes millions of years of evolution by natural selection to take place. We attempt to approximate this functionality using the idea of the network combined with back-propagation of gradients through that network to optimise for a desired outcome.

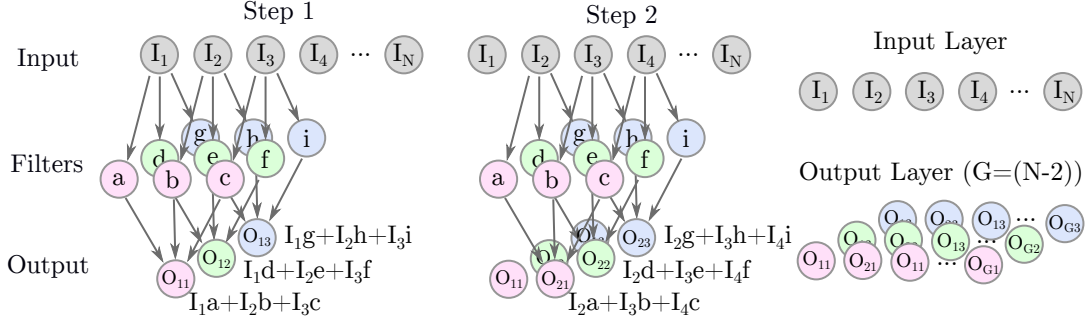


Figure 1.4: A simple example of a 1D convolution on some data, using 3; 1×3 filters. **Left:** shows the first step as part of the convolution computation, computing the output for O_{11} . **Centre:** shows the same three filters are used for the entire input data. **Right:** the resulting layer given the convolution performed on the input data. This effectively increases the channel count of the resulting data, now the 1-dimensional data has become 3-dimensional through the convolution, as each convolutional filter acts independently, generating its own channel shown using the colour coding. Also note the reduction in the number of elements by 2, because the convolution was performed without padding the data.

1.4.2 Convolutional Neural Networks (CNNs)

In recent years the field of machine learning has been dominated by the use of Convolutional Neural Networks (CNNs) which have proven to be some of the most flexible and powerful methods of solving any problem in machine learning [26, 27, 28, 29, 30, 31, 32,

1 INTRODUCTION

33]. CNNs work by performing layers of subsequent convolutions on input data using so-called filters. These convolutions form *activations*, which are then convolved upon to form a network. Some activations will be terminated in the same way as they are for neurons, as the convolutions will result in an insufficient activation level to propagate a signal. In practice this termination is implemented using a rectified-unit operation, for example the Rectified Linear Unit (ReLU) which zeros all negative valued activations. The rationale behind using rectified-units is related to the way in which a CNN *learns* through back-propagation of loss gradients. This is explained in greater detail in Section 3.10.

A typical application of convolutional neural networks is to the task of semantic segmentation, a simplified illustration of which is shown in Figure 1.5. This demonstrates that a set of successive convolutions can encode simple spatial information and combine it to form more informed features. These information rich features can then be used as a basis for classifying individual pixels with a particular label. This example is vastly over-simplified but illustrates the basic principles behind such a network.

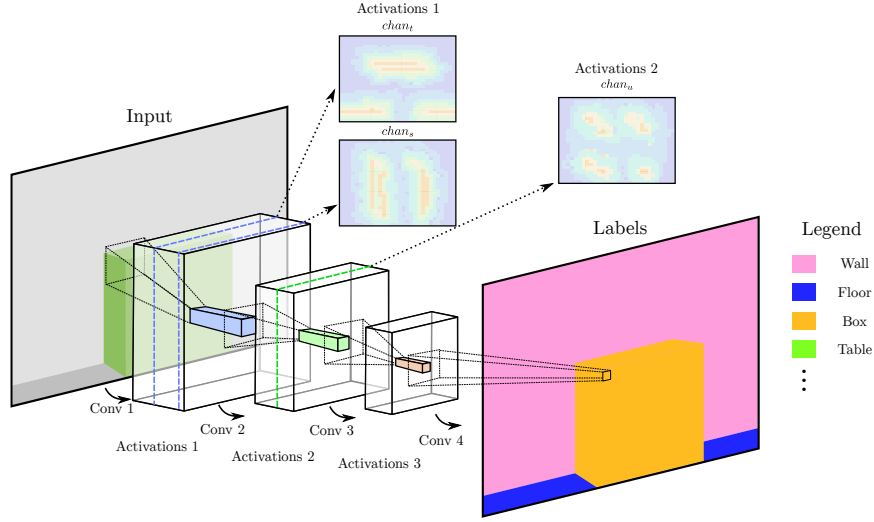


Figure 1.5: A very basic Convolutional Neural Network (CNN), where the dashed box indicates the what section of the previous layer was convolved on to generate the coloured activation block. To illustrate what the filters may learn during training we expand a few activation channels ($chan_s, chan_t, chan_u$). $Chan_s$ responds to vertical edges, while $chan_t$ responds to horizontal edges, which could be the initial sorts of image features that are encoded. Subsequent layers encode more complex information, where $chan_u$ seems to respond to the presence of both vertical and horizontal edges. This demonstrates how subsequent convolutions can perform more complex tasks than individual layers, by successively building upon the previous layers.

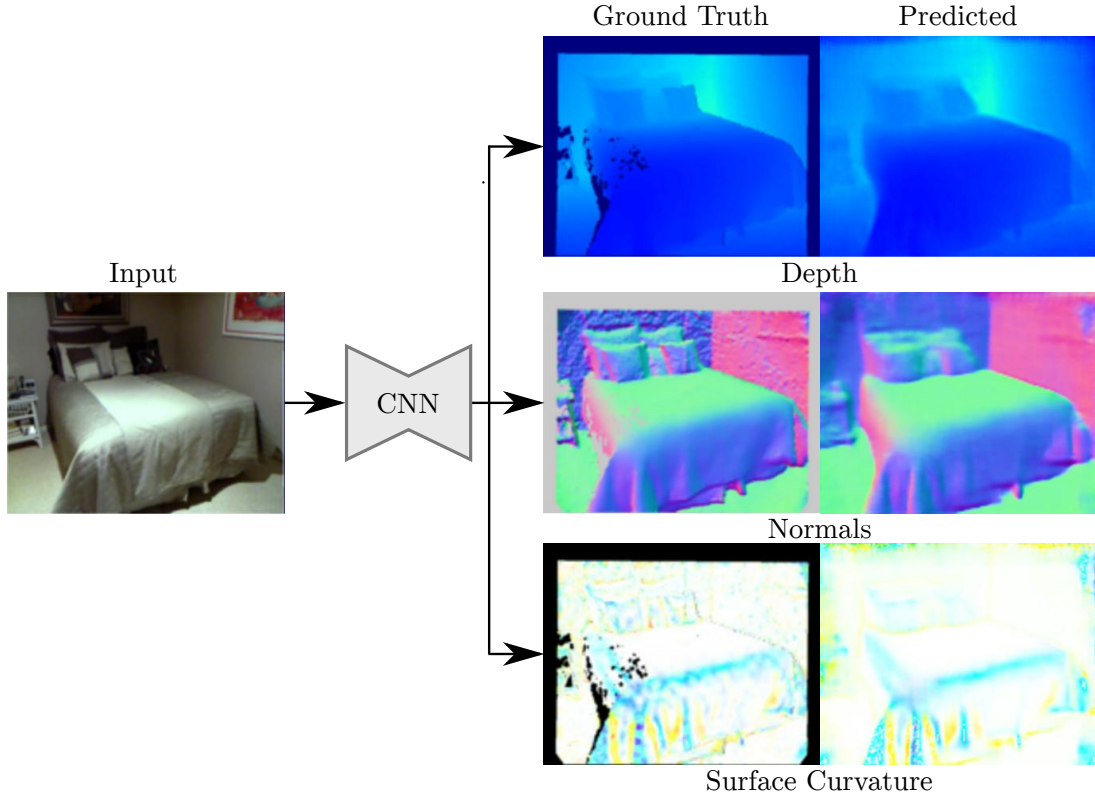


Figure 1.6: An example of the output of a geometric estimation network, given the input of a colour image. This clearly demonstrates how convincing the depths, normals and curvature are for such a network.

1.4.3 Extracting Geometry from Colour

Inspired by the works of semantic segmentation and enabled by the vast amounts of ground truth data low-cost depth sensors can produce, some researchers turned their attention towards the task of estimating geometric quantities from a single view. This manifested itself in works that estimated depth[34] and normals[35] given only colour information as input, and produced very plausible depths. The original network architectures were similar to those used for a very different task, that of semantic image labelling. This is essentially training a network to assign 1 or more plain text labels to an image [36, 32], and is the field that has led to massive improvements in image search technology. In fact the neural networks now outperform humans at this task [37]. Although the original depth estimation networks were based on architectures adapted to perform recognition and in fact initialised with weights from these networks the resulting estimations were still convincing. This does indicate a plausible relationship between the features required to estimate an image label and those required to estimate a depth.

1 INTRODUCTION

They also demonstrated that getting a network to estimate related but different tasks, forces it to devote energy into improving the underlying features it uses to represent quantities [35, 38, 39]. More concretely if a network is attempting to estimate depth vs depth and normals in the same network without increasing model capacity, the network will have to learn features that encode information relating to both. Somewhat surprisingly this compromise, instead of hurting the networks performance actually seems to improve it.

Extending upon these previous approaches we constructed an depth estimation approach and used the curvature and normal data generated by the system described in Chapter 5, to generate further training data from NYUv2 dataset [40]. The system was then trained to estimate all three quantities in parallel. An example output of such a system is shown in Figure 1.6. The details of our approach are in Chapter 7. We too found that estimating related quantities with a common framework was able to consistently improve our networks performance on all tasks.

1.4.4 Choosing the Correct Objective Functions

As has been shown in the previous section CNNs benefit from being given related quantities to estimate. In addition to this having a set of loss functions that attempt to balance the various goals of a network also has a positive effect on the overall performance [35]. For example in estimating depths, it is clear we want to minimise the difference between the estimated depths and the ground truth depths. However we may also want adjacent estimated depths to be similar during inference, and so may want to enforce this with an additional loss function during training. This will result in a balance between these goals, where the network will essentially preference minimising which ever has the higher loss during training. This can result in better overall depths if the errors are properly balanced. Additionally we may want to use additional loss functions when it comes to estimating a quantity with sparse ground truth but a weak extrinsic constraint.

An example of this is for stereo imagery datasets like KITTI [41], where the ground truth data comes in the form of highly sparse LIDAR data, but the dataset was captured in stereo. One approach may be to use a method that computes accurate depth estimates given stereo imagery, but this would potentially bake-in any failure cases of the stereo approach into the networks prediction. The preferred method is to take a so-called semi-supervised approach [42, 43, 44], and use left-right photometric consistency to improve depth estimation quality, even when we only have a single image available during inference. Photometric consistency is the constraint that the same thing in two different photos should look basically the same. Further generalising this, instead of fixing the relative pose to be a known transformation, namely the left or right relative

1 INTRODUCTION

shift of the camera between a stereo pair, one could estimate the general transformation between adjacent frames in a sequential dataset. This would provide the same additional constraints as the left-right consistency in terms of the photometric consistency, but now it can be applied more generally to any sequential dataset which was done in [45] and [46].

Expanding and improving upon these previous approaches we introduce further error functions. Additionally, inspired by the works like [47] we design a network that estimates quantities CNNs in general are better suited for, as opposed to trying to force it to estimate something that is highly abstract like estimating relative pose given two input images based solely on the pixel values. It is known for example that neural networks are able to produce state of the art optical flow estimates. Optical flow is a vector valued quantity that describes the motion of any pixel from one frame to another. This quantity is highly related to the relative distance of the point from the camera, as well as the nature of the camera motion between frames. Acknowledging that a CNN is good at estimating quantities such as metric depth, and optical flow, we were able to build an iterative pose estimation from the estimated values, and outperform networks that attempt to get the network to estimate the pose directly. We also used the estimated pose to enforce photometric consistency across general transformations, and improve our overall depth estimates. This highlights two things about the nature of CNNs, the first is that the design of loss/objective functions is incredible important and can largely determine the resulting performance of your approach, and secondly its important to consider the nature of CNNs when designing your system to ensure the tasks you are attempting to solve align with the strengths of the network.

1.4.5 Towards Real-time Performance

Recently the desire for real-time performance has lead to many innovations in machine learning approaches. Many of these focus on approximating the structure of larger accurate approaches using mathematically similar formulations that sacrifice performance for speed. A popular example was *mobilenets* [48], where they were able to achieve real-time performance for semantic labelling of images on a mobile device. To achieve this they use depth-wise separable convolutions, which attempt to emulate the performance of a regular convolution, by separating it into two stages. They take a set of $j \times j$ filters equal to the number of input channels, and convolve across the input, with each filter acting only one channel. This reduces the computational cost massively but now complex features can not form across channels. To address this they convolve the resulting activations from the previous step, with a number of $1 \times 1 \times M$ filters equal to the output channel count, where M is the number of input channels. This is mathematically comparable to traditional convolutions and the results indicate similar performance to existing networks but with large gains in speed.

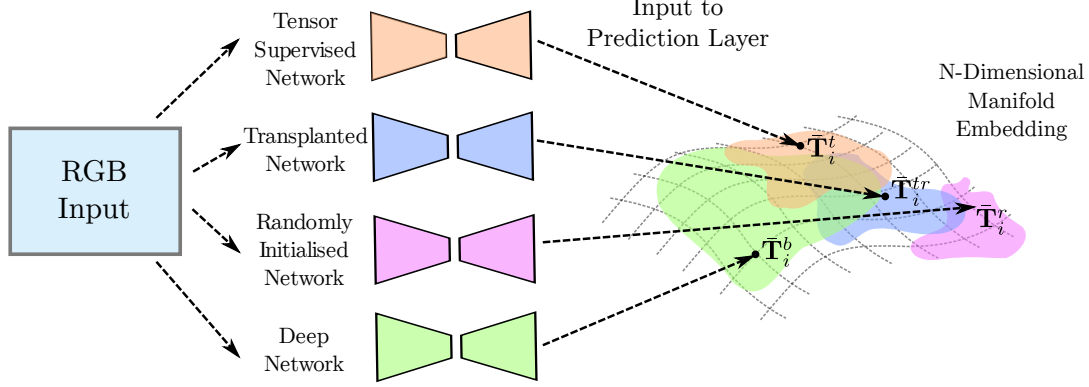


Figure 1.7: A representation of the possible manifold embedding generated by a set of networks, including the networks that are being trained to emulate the performance of a deeper network. The imagined possible embedding space positions for input images are shown in the networks respective colour, the size largely indicates the model capacity and how that might relate to embedding. The distance from the deep network attempts to express how similar the resulting embeddings appear to be. This is purely intended to be illustrative of the rationale behind the resulting outcomes.

Inspired by the work [49] we aimed to generate a real-time depth network on a mobile platform using knowledge distillation or model compression. What Hinton *et al.* demonstrated in their approach, was that a larger more accurate network can be used as a teacher to a weaker faster network, and improve its overall performance beyond what a traditional training approach to achieve. They show this for the problem of classification, training a network to emulate the average probability distribution of a ensemble of strong networks. This provides a stronger training signal to the network and overall produces a better classification than training on the ground-truth data alone. This is similar to the adding a additional related loss functions that provide the network with more information about what it needs to express. Extending this approach to depth estimation is less straight forward, as its no longer a classification problem the network doesn't output a probability distribution. It does however generate an embedding at every layer, in terms of the activations generated by the network. Taking the final layer to be a pseudo-classification layer, we attempted to get a smaller network to emulate the input of a larger state-of-the-art approach to this last layer.

We took two approaches to emulating the input to the last layer of the network, transplantation and supervision, both generating a networks that outperform randomly initialised networks. Transplantation simply takes the final layer of state-of-the-art network and transplants it on to the end of the fast network, the weights are fixed and the remainder of the network is trained, meaning the knowledge of is simply transplanted into the faster network. Supervision was based on trying to minimise the difference between the activations of the large network and the small fast network, this can be com-

1 INTRODUCTION

bined with a transplantation of the final layer. Interestingly this approach to knowledge distillation produced some possibly counter intuitive results, including the transplanted network consistently performing the strongest of all tested approaches.

The Figure 1.7, attempts to demonstrate the relationship between the training methodology and the resulting embedding. This is also intended to demonstrate that the model capacity is largely what determines the resulting size of the embedding, not size of possible outputs, which is a much larger. This was despite the supervised approach generating activations more similar to the state-of-the-art network, indicating the topology of the manifold is less convex than desirable. These results highlight the need to further investigate the relationship to model-capacity and this method of knowledge distillation. In addition to interesting results of the process we also demonstrate the practical application of this approach to an existing SLAM system [50], and demonstrate a qualitative and quantitative improvement over using purely monocular data, by introducing our estimated depths. This approach is discussed in detail in Chapter 9.

1.5 Publications

- A Compact Parametric Solution to Depth Sensor Calibration, **Andrew Spek & Tom Drummond** *In British Machine Vision Conference (BMVC)*, September 2017
- A Fast Method For Computing Principal Curvatures From Range Images, **Andrew Spek & Tom Drummond** *In Australasian Conference on Robots and Automation(ACRA)*, November 2015
- Joint Pose and Principal Curvature Refinement Using Quadrics, **Andrew Spek & Tom Drummond** *In International Conference on Robotics and Automation(ICRA)*, June 2017
- Joint Prediction of Depths, Normals and Surface Curvature from RGB Images using CNNs, **Thanuja Dharmasiri***, **Andrew Spek*** & **Tom Drummond** *In International Conference on Intelligent Robots and Systems (IROS)*, September 2017
- ENG: End-to-end Neural Geometry for Robust Depth and Pose Estimation using CNNs, **Thanuja Dharmasiri***, **Andrew Spek*** & **Tom Drummond** *Under review: European Conference on Computer Vision (ECCV)*, September 2018
- CReaM: Condensed Real-time Models for Depth Prediction using CNNs, **Andrew Spek***, **Thanuja Dharmasiri*** & **Tom Drummond** *Under review: International Conference on Intelligent Robots and Systems (IROS)*, October 2018

* Indicates equal contribution from those authors.

1.6 Software and Hardware Libraries

1.6.1 Software

Computer Vision Library (libCVD) - libCVD is a very portable and high performance C++ library for computer vision, image, and video processing. - <https://www.edwardrosten.com/cvd/index.html>

Tom's Object Oriented Numerics (TooN) - TooN is a C++ numerics library which is designed to operate efficiently on large numbers of small matrices, and provides easy access to a number of algorithms including matrix decompositions and optimizations. - <https://www.edwardrosten.com/cvd/toon.html>

CUDA - CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). - <https://developer.nvidia.com/cuda-toolkit-archive>

Tensorflow - TensorFlow is an open source software library for high performance numerical computation. - <https://www.tensorflow.org/>

JetPack - NVIDIA JetPack SDK is the most comprehensive solution for building AI applications. - <https://developer.nvidia.com/embedded/jetpack>

TensorRT - NVIDIA TensorRT is a high-performance deep learning inference optimizer and runtime that delivers low latency and high-throughput for deep learning inference applications. - <https://developer.nvidia.com/tensorrt>

ORB-SLAM2 - Real-Time SLAM for Monocular, Stereo and RGB-D Cameras, with Loop Detection and Relocalization Capabilities [50, 7] - https://github.com/raulmur/ORB_SLAM2

Pangolin - Pangolin is a lightweight portable rapid development library for managing OpenGL display / interaction and abstracting video input. - <https://github.com/stevenlovegrove/Pangolin>

1.6.2 Hardware

Jetson TX2 - Jetson TX2 is the fastest, most power-efficient embedded AI computing device. - <https://developer.nvidia.com/embedded/buy/jetson-tx2>

Related Work

This chapter provides a review of previous work, in order to provide a context for the research presented in this thesis.

2.1 Low-cost Depth Sensors

2.1.1 Sensor Types

A large portion of this thesis relies on the use of low-cost depth sensors in particular the Microsoft Kinect, and as such the context of challenges and opportunities afforded to the user of these sensors is provided here. Additionally information on various varieties of depth sensor is provided to emphasise the wider applicability of the work performed in this thesis. As indicated in the subsequent sections, these sensors are very widely used in robotics [51, 21, 52, 53, 54, 55], computer vision [14, 17, 16, 18] and machine learning [40, 34, 35].

2.1.1.1 Stereo Camera Systems

Stereo matching has been used in computer vision for decades [56], with the initial efforts devoted towards automated matching in aerial photography. Stereo algorithms function by estimating dense correspondences between frames that are separated by a known transformation, in most cases a simple horizontal or vertical translation, to simplify the matching procedure. Matching is generally performed using patch based correlations [56, 57, 58], but more complex approaches have also been developed [59, 60]. In [59], Hirschmuller presents an approach that has become the quasi-standard [61], where correspondences are estimated based on mutual information and an approximation of a global smoothness function. In [60], Tola *et al.* present a dense feature based approach that attempts to improve the parallelism of a feature based approach, exploiting the overlap between points in the computation and separating convolutions across precomputed orientation maps, allowing for a future fast parallel implementation.

2 RELATED WORK

As computer speeds have increased the ability to perform stereo matching has as well. This has resulted in real-time practically sized stereo vision systems [61, 62], but in general have historically required special hardware including high performance Field Programmable Gate Arrays (FPGAs) and accompanying CPUs making them expensive and specialised. The introduction of low-cost ASICs [63, 23] has increased the performance of these approaches. However, these ASIC solutions have become unavailable publicly and commercialised. As stereo approaches are passive they don't require active interaction with the environment to generate a depth. This also means their range and resolution is only limited by the sensor size, and the camera separation. As a consequence of this passive nature, stereo systems find low-texture surfaces very challenging [64], resulting in sub-optimal depth estimation.

At the time this research was conducted, stereo systems were readily available in robotics, but in general were expensive and required external computation and special interfaces to operate, making them a less attractive option. Since then real-time stereo devices have become more available [65, 66]. However, these systems are in general more costly or require addition hardware to perform some of the processing via GPUs.

2.1.1.2 Time-of-Flight (ToF)

Time of Flight (ToF) sensors are a popular choice in robotics [67, 68, 69, 70, 71, 72, 73, 74]. As described in [64], this is because of a number of favourable properties of this devices. It requires no extrinsic calibration (single sensor), it can generate reliable depth measurements without requiring a complex correspondence measurement, and can function in low-texture environments comfortably at relatively high frame-rates ($>30\text{fps}$). However, these devices also exhibit less favourable traits that are unique to these sorts of sensors, including a noise that is proportional to the reflectivity of a surface, integration along edges creating floating points that don't exist, an effective depth resolution and unambiguous range that are inversely proportional [64] (in some sensor types), and in general a lower output image resolution.

Many approaches have been made to address these issues, as well as new methods of actually computing depth [72, 71, 73]. In [72], Niclass *et al.* present an alternative approach to estimating the depth that is effectively a higher resolution lidar scanner. The device presented has such a high time resolution it can effectively count the number of individual photons that hit the sensors. This allows for the device to highly accurately measure the depth to very large distances with a very high degree of accuracy. This sensor was improved upon in [71], increasing the frame-rate and reducing the cost by replacing the externally required FPGA (used for the DSP required to reconstruct the

2 RELATED WORK

depth), with a System on Chip (SoC) all-in-one solution. The system produced is still relatively bulky, specialised, and contains a number of mechanical elements that decrease the overall expected reliability while increasing the cost. Additionally, these devices still exhibiting some of the classic drawbacks of ToF sensors, such as the object boundary ambiguity [75], and relatively low-resolution. Finally, neither device provides a registered colour value which is provided by other low-cost depth sensors.

An alternative to modifying the design of the sensor is presented in [69], where Zhu *et al.* globally fuse the resulting depths from a stereo and ToF system in order to produce registered depth and colour images with a higher accuracy than either approach alone. This demonstrates a promising approach to resolving some of the flaws that exist in the ToF approach to depth estimation by treating stereo as a complementary approach. They expand upon the previous work in [70] to provide an in depth analysis of the global fusion function they use, which is used to estimate a per-pixel confidence estimate they use to guide the fusion of the data from the two sensor types. However, the system produced is highly customised and adds a significant bulk in hardware, as it includes both a stereo pair of cameras and a ToF sensor. The reliability estimate is something that could be significantly useful in broad sense for ToF and stereo devices in general to be able to exclude low-confidence points, but is not widely available on many current commodity ToF sensors such as the Kinect 2.

2.1.1.3 Structured Light

Structured-light sensors work on the same principal as stereo camera systems, points are corresponded very quickly between two views. The difference is one of the *views* is projected onto the scene and the other view triangulates depth values by corresponding to known points in the projected image, providing a stronger link between correspondences and making matching more reliable. Structured light and ToF sensors are considered active sensors, as they project a signal onto the world in order to detect depth, which means both are greatly effected by environmental conditions [76], such as ambient lighting.

Many early approaches to depth estimation using a structured light approach relied a early digital projectors [20, 21] or custom laser systems [77]. Scharstein *et al.* present an approach that uses a projector based approach, that is considered to be so accurate as to provide a ground-truth image approximation to be used to evaluate the performance of competing stereo approaches. A fixed projector is used to project a series of gray-codes, which are like alternating bars that lead to extremely accurate binary line identification in the image. Projecting a series of these at alternative angles and positions can be used to very accurately correspond points across multiple views. However, this approach

2 RELATED WORK

cannot provide registered colour information at the same time, and performs very slowly. Weise *et al.* proposes a near real-time approach in [21], which works on the same principal but also incorporates a closed form motion compensation function that allows it to capture moving objects and produce reliable reconstructions. These early works produced highly accurate depth estimation, but require expensive specialised hardware, complex calibration procedures, custom software to compute the desired depth, and still don't produce colour imagery.

The introduction of the structured light based low-cost depth sensor, the Microsoft Kinect, created a significant stir in the robotics and vision community. This sensor used compact mini-projector, that projected a fixed pseudo-random grid pattern onto a scene. The device was soon made accessible by the open-source community [78], and subsequently support to developers was made official. The device provides registered colour and depth imagery, at 30Hz through the use of a custom ASIC that is used to match regions of the projected pattern. The device still has a number of issues [76] (including structured error which requires intrinsic calibration), but has the attractive property that it reports depth conservatively, removing points that don't match with a high confidence. This means the data that is produced is very reliable, and comparable in usefulness to approaches that directly estimate confidence [58].

2.1.2 Calibration Approaches

In order to combat some of the intrinsic problems present in each of the sensor types, many calibration methods have been developed. The approaches presented here are primarily applicable to the active sensor types such as ToF and structured light. Calibration of a stereo pair is approached slightly differently, where each camera should be calibrated separately for the intrinsic parameters (see Section 3.2) and then calibrating for the extrinsic parameters between the cameras (baseline, rotation, etc...).

2.1.2.1 Supervised and Semi-supervised

An early example of planar calibration is shown in [79]. Herrera *et al.* present a method for improving the depth to colour registration provided by the manufacturer of the Kinect and then attempt to improve the depth estimates. They use a joint checker-board approach to register depth and colour images, then using the checker-board on a planar surface, they optimise for a solution that corrects depth to the plane. In [80], Raposo *et al.* provide an extension to [79] and improve the performance of the calibration in terms of the number of required frames and speed of calibration. Both methods show improved accuracy but both require the additional step of calibrating

2 RELATED WORK

the colour to the depth camera. In [81], Jin *et al.* simplify this supervised calibration method by having the user identify cuboids in the scene and iteratively optimise an error function over the *planar* regions in the selected cuboids. One of the drawbacks of these methods is an emphasis on supervision, and external ground truth estimates which take additional time and/or require accurate measurements to be made by the user in order to function.

A similar planar approach used in [82] where Di Cicco *et al.* present a calibration approach that uses machine learning. An Artificial Neural Network (ANN) is used to calibrate the sensor, removing the need for any specific knowledge or sensor modelling. They also describe a method for collecting large planar surfaces for calibration that eliminates non-planar data from collected datasets. This method greatly underestimates the difficulty of capturing a planar surface from more than 6 meters away which limits the effective range of the calibration.

2.1.2.2 Mapping Based

One of the most successful unsupervised calibration method for the Kinect was [83]. Teichman *et al.* use an approach which relies on building a globally consistent SLAM model of a scene prior to calibration. They attempt to ensure the model quality and pose accuracy by rejecting depth values greater than 2 meters when optimising the model. This is because they require very accurate initial pose estimates. Using the pre-built SLAM model they then attempt to minimise the error between the projected depth and the model using a maximum likelihood across all corresponding depths patches. The correction is a linear function applied to depth calculated for all 8x6 image patches for discrete depths and interpolated between adjacent patches. The process of modelling-calibration process can be repeated, but reportedly this is largely not required for a good calibration.

Zhou and Koltun took a similar approach to calibration in [84], but instead of requiring the user to manually cycle the modelling and calibration steps, it automatically recomputes a more accurate SLAM model by using the corrected depth images in their model generation stage after each iteration. Again they use a set of patch centred linear scaling factors for discrete depths, however they use tri-linear interpolation between patches of a particular depth and between specific depths. Both [83] and [84] use a patch based approach to save computational time and memory usage for the resulting calibration model, and avoid over-fitting the data as the devices used produce high frequency noise. Both methods are reportedly *overnight* computational processes on modern hardware. However, both methods can suffer from over fitting to the noise due to the extremely high model capacity using thousands of parameters.

2.2 Depth in Robotics

As referenced above low-cost depth sensors have proven to be extremely useful in a robotics and vision context. This section provides a context for the motivation in using low-cost depth sensors in this work, including the importance of geometric based features (Section 2.2.1) to many applications such as SLAM (Section 2.2.2), scene segmentation (Section 2.2.3), and machine learning (Section 2.3).

2.2.1 Geometric Feature Estimation

In the context of this thesis geometric feature estimation is considered to be deriving features from depth information alone. This includes geometry properties such as normals and surface curvature, but also includes hand-crafted point features. In this thesis research primarily focused on the former of these two feature types, and improving a system’s ability to estimate them. The goal of computing these features in this thesis is for the task of mapping alignment and to a lesser extent segmentation.

2.2.1.1 Normals from Depth

The normal is the direction that is perpendicular to the surface at any point. Normals and planes are tightly coupled, a tangent plane to a point on a surface and the point can be used to define the equation of a plane, which can be used to quickly check if other points lie on that plane. A common robotics application for normals is plane fitting, which is helpful in navigation and scene segmentation for a robot. Normal estimation is important for computer graphics [85, 86, 87] and robotics [53, 54, 88]. For the computer graphics and vision communities normals are crucial to many standard lighting models [89], additionally a normal of a point can be used to define whether that point is expected to be visible or not given a change in viewpoint. An early example of a variety of normal estimation approaches is in [85], where Yagel *et al.* provide methods for computing so-called discrete normals from point based surfaces in a graphics context. This work using small neighbourhoods around points in the world space, or pixel space to compute an estimate for the normal through differentiation of the surface. The surface derivative is effectively equivalent to the surface normal, but these approaches are somewhat more susceptible to the noise in the depth information, which can lead to poor estimates of the normals.

In [87], Amenta *et al.* introduce a method for computing normals based on a Voronoi-based filtering approach in point fitting, initially taking the surface normal to be the

2 RELATED WORK

vector that connects this point to the furthest Voronoi vertex in Voronoi diagram surrounding that point. Using this as a basis for redefining a new Voronoi diagram, and subsequent Delauney triangulation of the surface. Dey *et al.* extend this idea in [86], to use a Voronoi based method in an effort to handle noisy point-clouds which are more common in real-world applications. The normal is first estimated as with [87] then refined using a least squares approach, which results in normals that are fairly robust to noise in the points. This approach requires multiple steps, and could result in increased compute times.

2.2.1.2 Curvature from Depth

As surface curvature is such a fundamental surface quantity, there exists a significant body of research concerning its computation [19, 90, 91, 92]. Many of these methods are computed using dense point cloud data [19] and high quality meshes [93, 91, 92, 94] allowing high quality estimates of curvature to be made using a relatively small amount of the data, due to the high inlier rates. In [19] Besl introduces the concept of local surface fitting in order to compute curvature. Besl *et al.* examines fitting range image data densely using least-squares regression to fit to quadratic, cubic and quartic surface parametrizations. Surface fitting has been a popular choice for surface curvature estimation [95, 96], in particular quadric surface fitting which has been shown to be effective for curvature based segmentation [97]. Douros *et al.* present a work [98] that attempts to fit local surface patches to point cloud data directly. However, in their work they use a least squares approximation to produce a closed form solution. Although this approach can deal with noise in the input, it doesn't fully account for outliers resulting in potentially erroneous curvature estimates.

Curvature estimation from quadric surface fitting is based on fitting an implicit quadric surface around a central point. Curvature is then estimated from calculated surface gradient estimations of this fitted surface. In order to estimate surface curvature some methods [55, 99] also employ a technique that requires the estimation of second order partial surface derivatives about the point. This creates an estimate highly susceptible to noise in the point cloud, as the estimate of the second derivative will be quadratically effected by the noise, if estimated directly. For a sensor like the Microsoft Kinect with quite a noisy local signal [76], this can prove to be quite problematic.

Many common methods employ techniques that require high quality surface meshes or point clouds to estimate normals and curvature fast and accurately. In [94] Rusinkiewicz *et al.* use a patch of connected vertices taken around each vertex in a triangular mesh to estimate curvature. The patch used is made of a 1-ring selection of triangles that share a vertex with that point. Using this small neighbourhood of triangles, normals are

2 RELATED WORK

estimated for each vertex and their relative weight is computed based on the proportion of the area of the surface that lies closest to the vertex. Using this 1-ring of normals all pair-wise normal differences are calculated and the curvature of the surface is estimated from the weighted sum of these normal differences. This method is roughly equivalent to a discrete differentiation of the surface normal field and gives a reportedly reliable estimate of the principal surface curvatures across the mesh. However as the patches used in this 1-ring method only uses direct neighbours to the center vertex it is heavily sensitive to noise in the surface. This method was extended in [92], where Griffin *et al.* used a GPU implementation to make it run in realtime (30fps) for mesh models of the order a million vertices. However this requires a pre-processing step to sort the vertices via an iso-surface extraction which takes on the order of hundreds of ms (using NVIDIA GTX480) for models of that size. This approach is also more geared towards vertex-based models, as opposed to point cloud data.

There also exist other open-source methods for computing curvature [99], namely the method developed for the open-source Point Cloud Library (PCL) by Rusu *et al.* This library provides an estimate for surface curvature in a point-cloud and can be applied generically to any point-cloud. The original method employs a KD-tree which allows a metric based surface patch which is proportional to the density of the point cloud, making it suitable for noisy point clouds. They also created a GPU method which operates in real-time and operates in view-space (as our method does), with both computing an estimate for principal curvature values given an input point cloud. However both methods compute curvature via an robust double differentiation using PCA of the surface differences and then normal differences, which should be able to cope with a small amount of noise in the data, but again struggle with outliers. This double differentiation substantially amplifies the noise and outliers in the point cloud data and is therefore unsuitable for quite noisy devices such as the Microsoft Kinect.

2.2.1.3 Features from Geometry

Geometry based features generally use a few common approaches that are borrowed from the earlier photometric feature work, such as being patch based, using pair-wise point comparisons to form histograms [100, 101] and even using gradient information (in this case surface gradient/curvature) [102]. Almost all the approaches that demonstrated robustness in a photometric case have been translated to function on geometric data and seem to demonstrate that the features used seem to be salient in both domains (photometric and geometric). A few notable examples of geometry features include Point Feature Histograms, Spin Images, SIFT3D and NARF. A general geometric feature is surface curvature, which is not completely discriminative for a point on its own, but has proven to be effective when summarised across a neighbourhood around a point

2 RELATED WORK

[53, 54, 103] and generally forms the basis of most computed features indirectly.

Rusu *et al.* first demonstrates the Point Feature Histogram (PFH) in [100] as a method for providing a generic surface feature that is based solely on the relative orientation of normals in an N-neighbour area around a key-point. This is a method that essentially uses surface curvature of a small neighbourhood around a point to provide a sampled descriptor. The PFH actually gives a histogram based on the similarity in the per-point normals for a small neighbourhood (fixed on the number of neighbours) of points around each point. The basic technique transforms the points and normals into a frame/basis formed around the central point and the angle between the axes of the frame and the distance between the point is used to construct the histogram. Rusu *et al.* expanded on this work in [101] to produce Fast Point Feature Histograms (FPFH) which differ mainly in computation efficiency but they also remove the distance between the points in the calculation of the histogram, as they found it to be unhelpful. A practical application of similar work can be found in [103], where Drost *et al.* define surface features of a point cloud based on normal differences. These features are defined for entire objects and a look up table is created to store each object required for recognition. Using this principal they need to sample a selection of points in order to match to the desired image using these correspondences. These sort of geometric features have proven popular in certain situations, but fail to extend well to large scale applications due to the incomplete nature of the data they use.

In his thesis [104], Johnson analyses the feature known as Spin Images. Spin Images are defined useful for 3D point cloud data. Spin Image use a region of points around the point of interest in the point cloud. Using these points a dominant direction of the points on the region is defined, which takes into account the relative position of all points in the region. Using this central point a sampling plane is aligned to the dominant direction and then rotated about the central point sampling the location of points that intersect the sampling plane. These points are binned based on the cylindrical rotation and radial coordinates giving a descriptor for that central point that uses the sampled point density as its defining characteristic. These features again have issues in the case of missing data, this is particularly true when self occlusions occur, the features really need to be computed from more complete models, but this in some sense would defeat their purpose if one wished to use them for wide-baseline alignment, as you want to align incomplete images to the current model.

SIFT3D is very similar in principle to SIFT but requires 3-Dimensional data which are all spatial for MRI and CT scans [105, 106] or two spatial and one temporal dimension such as in [102, 107]. The 3D data component makes it almost the same as the original SIFT but the histograms are now segmenting over cubic regions instead of squares and the dominant gradient direction is determined in terms of 3D dimensions, as are the image gradients. This has shown to be very effective for aligning medical imaging data

2 RELATED WORK

achieving high accuracy scan matching [105], and has been shown to even be effective at object recognition for security purposes [106]. The temporal version of SIFT3D is shown in [102] which uses the same SIFT3D descriptor but instead of using the third spatial dimension, the third dimension is now time. Scovanner *et al.* have shown this is quite effective at determining actions performed in video sequences. Although this provides reasonable results in matching and object recognition [106], the data requirements are 3D which are not available when using a low-cost depth sensor. Additionally, this thesis is not concerned with temporally based action recognition.

The Normally Aligned Radial Feature (NARF) descriptor proposed by Steder *et al.* in [108] uses *stable* surface points at boundary locations to form features based on changes in a patch. These stable points are defined as those that have normals in a neighbourhood around it that can be accurately estimated. This is not generally true of points on physical edges in a depth scan as these are some of the least stable points due to the inaccuracies in the device that performs the scan. The descriptor is formed by applying a fan shape to a point aligned to its normal giving a descriptor based on how much points vary along each line in the fan shape. This is then rotated to align the descriptor with the dominant fan line. This descriptor essentially tries to find regions of stable surface curvature in order to ensure its robustness. This is based on the idea that the least reliable points to compute a descriptor for are those that occur on object boundaries. This is because these are the exact locations depth sensors find most challenging, meaning a fast method of computing surface curvature to allow stable regions to be computed more quickly could be highly beneficial to this approach. However, potentially region level segmentation could also be sufficient, given the viewpoint dependence inherent in many of the geometric features when applied to low-cost depth sensor data.

2.2.2 Localisation and Mapping

Localisation has been a significant problem in the robotics and computer vision community for a number of decades now. The term *localisation* is used in computer vision and robotics to mean one of two things, finding the location of objects in the world relative to you and finding your location relative to the world (ego-motion). The method of localisation can vary greatly, but one major distinction can be made between sparse [1, 50, 7, 109, 110, 111], semi-dense [10, 11, 112] and dense techniques [14, 15, 16, 17, 18, 113, 114, 13, 25]. Mapping is the process of generating or creating a *map* of the *world* that could be used to accurately navigate that *world*. The *map* produced is generally constructed from points, lines, primitives, feature point locations, or a combination of these.

Localisation and Mapping have a *chicken and egg* relationship, where good mapping

2 RELATED WORK

requires a way to accurately localise, and good localisation requires an accurate map. One of the initial efforts to combine these tasks and solve them simultaneously was in [1]. In this work by Durrant-Whyte *et al.* localisation is performed based on the location of feature points in a 2D sonar map of the environment, this was also the first use of the term Simultaneous Localisation and Mapping (SLAM). In the two-part follow on papers [109] and [110], Durrant-Whyte and Bailey give a detailed introduction to the basic principals behind SLAM. This includes using landmarks to build a map of the environment, using the landmarks to form correspondences across map views to align multiple scans for map building and also localisation. In [110] the focus of the paper is on the extensions to the framework, such as performing the expensive global map update infrequently while introducing a local sub-map that can be updated very quickly but provide accurate location information. Bailey *et al.* show loop-closure via bundle-adjustment can be performed very efficiently on a sparse matrix, establishing the idea that a sparse matrix representation of your map will result in a large improvement in speed. This also indicates that combining information from multiple views allows a global optimisation that's more accurate, due to the redistribution and subsequent reduction the system errors.

2.2.2.1 Feature-based Approaches to Mapping

This thesis focuses on photometric and geometric tracking including sparse and dense techniques. This section is included to provide a context for SLAM approaches in general. In this thesis we also move in the direction of *machine learning* for navigation using some off-the-shelf SLAM approaches [50, 7] in the evaluations, as these approaches are also entirely photometric. Photometric based tracking refers to methods that use the colour or luminance values of an image to compute relative alignment information. Many methods that use this form of tracking have been formulated including the use of gradient based intensity features [2, 115], textural based and trained descriptors [116, 3, 117].

Scale Invariant Feature Transform (SIFT) features [2] are still one of the most powerful photometric based features in terms of descriptive ability, and are used [118] for this reason. In David Lowe's original 1999 publication he outlines the original SIFT descriptor and an implementation of feature matching using his new formulation. A SIFT feature is an aligned histogram patch of the image gradients for each feature point. The SIFT feature is first aligned to the direction of the dominant intensity-based patch gradient, providing some invariance to rotation in the viewpoint as the camera moves. The feature vector is the result of binning the gradients over the patch giving a 128 dimensional vector. It's key drawback is it's computation time, which is comparatively high compared to more recent approaches [3, 60, 117]. This led to the introduction of

2 RELATED WORK

Speeded Up Robust Features (SURF) [115]. SURF developed by Bay *et al.* is essentially an approximation of a SIFT descriptor, but replaces gradient calculations with fast *Haar Wavelet* filter convolutions. This shows an improvement in speed with only a minor sacrifice in robustness. However, the computational savings are probably not sufficient to justify the loss in view-point invariance and thus re-localisation ability.

Rublee *et al.* propose the Oriented FAST and Rotated BRIEF (ORB) descriptor in [3]. The ORB descriptor relies on a machine learning approach for defining features and has shown significant improvements in speed with little sacrifice in matching robustness across many datasets. Rublee *et al.* have used a point feature pair-wise comparison strategy for defining a feature, using a rotated BRIEF kernel that is aligned to the direction of the *intensity centroid*. The Binary Robust Independent Elementary Feature (BRIEF) [4] fixed length binary feature vector and is produced by performing pair-wise intensity tests that are optimised to give the most information for each question. This is computed based on choosing the question that would most evenly separate features in the training dataset.

Recent successful SLAM solutions [50, 7] have shown that ORB features are well suited to a wide variety of contexts, with evaluation across a number of varied datasets and even performance that rivals dense and semi-dense approaches. More recent approaches to producing learnt descriptors use machine learning throughout the extraction pipeline [117]. Yi *et al.* produce Learned Invariant Feature Transform (LIFT), which is an approach that attempts to learn how the patch around a key-point should be cropped, rotated and transformed into a discriminative feature space, by ensuring the distance between the same patch from different viewpoints is small in feature-space. Something that almost all these approaches do is to assume that the patches around a point lie on a locally *visually* planar region. This is largely not true in practice, as the features are in general extracted from corners (using FAST [116] or some similar method) which produces large numbers of corners on object boundaries. Extracting patches from these regions is significantly viewpoint dependent, as background pixels in a patch will include data from objects that the point is currently occluding. This can make re-localisation given a wide-baseline change more challenging.

2.2.2.2 Using Geometry Approaches to Mapping

Geometry based mapping and localisation uses the geometric information in the input data to compute the information required for mapping and localisation. This geometric information is in general related to relationships between points, lines and planes/normals. There are two primary ways in which the geometry information is used, patch-based geometry features and direct surface matching. The appropriate method

2 RELATED WORK

of matching can depend on the way in which the data is collected, and its format. For example laser scanning techniques are able to provide high resolution point-clouds of a scanned area, while in the case of Computed Topography (CT) scans the information is a 3D *intensity image*. These two different forms of 3D data have different characteristics and thus in general a different strategy of mapping and tracking should be beneficial.

An extremely popular technique for aligning overlapping depth scans is the so-called Iterative Closest Point (ICP) algorithm. The original implementation of ICP was proposed simultaneously by [12] and [119] approximates a relationship between points in scans and iteratively repositions them to minimise error across the system. Chen *et al.* and Besl *et al.* both introduced ICP independently but Besl *et al.* proposed the name that stuck and formed the initial point-point, point-line and point-plane error minimisation strategies. Since the original paper ICP has become one of the most popular choices [120, 121, 122, 113, 114, 123, 124, 125, 126, 127, 13, 128, 129, 18, 14, 17, 16] for scan matching due to its high level of precision and the simplicity of the implementation.

The original implementation was exhaustive and offline, as was the acquisition of dense point clouds, although later modifications were made to speed up the original algorithm [120, 121], allowing for real-time ICP at frame rates of 30-50fps. Rusinkiewicz *et al.* showed that the sampling method used could be important depending on the choice of scene. Where largely planar scenes contain very little salient information for scan matching, hence sampling the points with the greatest amount of variation are going to give you the best chance of converging on the global minimum. They also showed that the project and walk strategy gives some of the best results in convergence rate and is also one of the quickest methods to compute correspondences. Rusinkiewicz *et al.* also showed that outlier rejection was important for improving convergence radius but had little effect on the rate of convergence.

The introduction of the low-cost depth sensor has changed the landscape for the robotics and SLAM communities as it has made the collection of real-time accurate depth scans much easier. As such many current systems [129, 122, 125, 126, 13, 14, 128] have been enabled by this new sensor. One such system [122] utilized the format of the data returned by the original Kinect sensor. The disparity information returned by the Kinect is proportional to the inverse depth according to the relationship $q(d_{ij}) = k_1 d_{ij} + k_2$. Lui *et al.* showed that by performing ICP directly over inverse depth coordinates the error becomes isotropic instead of increasing with distance, and it also avoids the unnecessary conversion to Euclidean space. This demonstrates a case when consideration of the nature of the error in the input data, aids in modelling and compensating for that error.

One successful low-cost depth sensor based system, Kinect-fusion introduced in [14] uses

2 RELATED WORK

a dense multi-scale ICP approach to track the live camera to the currently produced model/map of the environment. One contribution of this paper was the novel model representation, that used a volumetric Truncated Signed Distance Function (TSDF) to robustly handle free space in a voxel representation and dynamic elements as the TSDF is constantly updated per-voxel. Newcombe *et al.* combine this with the added accuracy of the dense ICP approach demonstrating that the error accumulation can be avoided to a large degree without explicitly compensating for it. Error accumulation is most noticeable when tracking or mapping frame to frame over a long series of frames, as the errors between each frame accumulate to cause the global consistency of your model to drop and lead to poor mapping and tracking quality. This is especially apparent for uncalibrated sensors, and less accurate approaches to alignment.

In SLAM++ [129], Salas-Moreno *et al.* expand upon the works [103, 14], using the idea of tracking from a model to reduce the accumulated error. In this work, objects that are expected to appear in a scene perhaps multiple times are loaded as models and detected during runtime using the method developed in [103]. Then a high quality model can be swapped into the scene in the place of the object to improve the quality of the tracking by including more prior information about the shape. The tracking is then performed keyframe-to-keyframe and keyframe-to-object, allowing for greater connectivity in the graph. Further to this the objects/models are used to more quickly detect loop-closures to trigger bundle-adjustment to run across the current graph/map. Additionally a ground plane constraint is used to reduce the degrees of freedom in which the object can be moved creating improved matches. This system has shown impressive real-time performance, however it requires accurate models to be scanned in and provided to the system, and a lengthy training step is performed to allow for recognition of the object at runtime. Additionally, the constraints imposed by using a ground plane approximation are not valid in many scenarios, particularly outdoor, or uneven environments or even potentially multi-level buildings.

An interesting and powerful extension to the choice of distance metric was proposed by Gelfand *et al.* in [130]. The authors recognised that the distance metrics, point-to-point and point-to-plane, and the subsequent optimisation methods were using a first order approximation of the surfaces. They show that this can lead to suboptimal solutions even when using advanced optimisation strategies. As a way to improve upon this Gelfand *et al.* propose an alternative quadratic (second-order) surface approximation for the distance metric, by fitting a local quadric to neighbourhoods of points and optimising a pose that minimising a point-to-quadric residual function. However, this work requires significantly more computation time, and the computation of the quadrics is fixed for a scan, indicating a poor initial quadric estimate may significantly effect performance. Additionally, this was only applied to synthetically generated models, which is significantly different to the data produced by low-cost depth sensors.

2 RELATED WORK

2.2.2.3 Combining Depth and Colour for Alignment

Using the combination of both photometric and geometric information is beneficial to trackers, as they are able to form a more complete and accurate joint solution. The rise in fused systems [125, 131, 127, 132, 126, 125, 17, 133] can largely be linked to the introduction of the low-cost RGB-D sensor, which provides synchronised depth and colour frames. These approaches taken can be divided into those that attempt to introduce a production line for the colour and depth, where the information from each is processed sequentially and what’s learned is passed along to the next stage, and those that attempt to optimise directly on all the data simultaneously.

An initial approach by Henry *et al.* in [125], was to first use a feature based approach on the colour information such as extracting SIFT or SURF features and then computing an initial alignment. This is then generally followed by a more accurate ICP calculation using the depth data. This has the benefit of overcoming the limitations of ICP which can not handle wide baselines, which feature based approaches are quite good at. Additionally this may be simpler to implement than a totally new system, as both of SIFT and ICP are quite mature. As well this can be made quite light weight as it is computed sparsely and on the CPU.

In [113, 114] Meilland *et al.* show modelling with a wider baseline using a custom spherical stereo depth image. They employ a novel saliency map to quickly determine the points that provide the most information for mapping, by examining the effect on the total error by adjusting each motion parameter individually and choosing those points that resulted in the largest change, corresponding to the largest reduction in global error. Another contribution was their implementation of a novel spherical depth map loop-closure strategy.

A more recent approach [127, 132, 133, 126, 17] is to attempt to directly minimise the error in the depth and photometric data. This requires the use of all data points (dense solutions) in the optimisation to provide the most accurate solutions. ICP can be a rather expensive operation to compute densely, and when combined with a dense optimisation over colour this would be pushing it outside real-time on a CPU and has led a significant number of researchers to move to GPU implementations. This is the case certainly for [132], where the system uses a bi-jjective optimisation strategy, that simultaneously minimises projected depth and photometric error to a reference frame. This process needs to be performed on a GPU to ensure real-time performance. This method is essentially the same for [126], which uses the idea of matching to the model to avoid drift. The contribution from Henry *et al.* was to use the novel Patch-Volumes. Which can be used to represent large sections of data (possibly objects) based on segmentation results and bringing them in and out of memory, allowing for

2 RELATED WORK

a more extensible system than that in [14]. Whelan *et al.* extend further upon these previous approaches in [17], where they propose an approach to modelling that focuses on resulting models reconstruction accuracy by continuously making incremental model updates through small model deformations to local regions. This approach again fuses the information from depth and colour to improve tracking and modelling accuracy. Each of these approaches can be improved with a better calibration than provided by the factory. Instead these approaches compensate for minor structured error in the sensor by in general adapting the model, or removing points from consideration, which could be used more effectively.

2.2.3 Scene Segmentation

Scene segmentation is the process of *background/foreground* separation, where objects or instances are segmented from the background. As is the case for localisation and mapping, there exist many photometric and geometric approaches, and naturally combined photometric/geometric approaches to improve segmentation. Scene segmentation is useful in computer vision and robotics [100, 54, 134] with accurate, robust segmentation being the basis of many other computer vision technologies including mapping, semantic labelling and object tracking. One common aspect is that combining further sources of geometric and photometric information can increase an approaches segmentation performance. Although scene segmentation is not a primary goal of this thesis, many of the approaches in machine learning that focus on scene segmentation have used/created important architectural frameworks that are applicable to later of half of the thesis. Additionally, as surface curvature, and normals are geometric features that are able to improve segmentation quality, we provide a discussion of some of the most relevant techniques.

2.2.3.1 Colour Based Segmentation

Photometric based segmentation uses the colour data in order to segment the scene, this can be using just the intensity of the image or by incorporating colour information to improve the distinguishing power of the the segmentation algorithm. Segmentation methods vary significantly on a case-by-case basis, where the direction of the field seems to moving towards complex models of the pixel relationships, including complex energy minimisation strategies [135, 136] to the use of super-pixel tree structures [137]. Segmentation by photometric data or *2D segmentation* is a long established field and some of the best performing segmentation algorithms perform almost as well as humans [138].

2 RELATED WORK

In [135], Gould *et al.* use a segmentation function that relies on global refinement of a function that minimizes over the horizon location, region, boundary, object, and context. The region term attempts to weight the region that most represents the pixel, and penalizes misclassification. The boundary term penalises adjacent regions with similar appearance or lack of boundary contrast, which helps merge similar regions. This scores the likelihood of a group of given region classifications being part of an object (regions form objects e.g. cups, bowls, chairs). The context term relates the object classification and the local background. In their implementation they consider the *background* as a single object, hence the context term is related only to one other object making computational growth linear instead of quadratic as is the case with taking the relationship to all other objects pair-wise. This method showed promising performance in terms of segmentation accuracy, but the computation is still very expensive and perhaps not yet possible in real-time, with a reported computation time of a few minutes.

A significant shift in the field of robotics and computer vision has been towards using machine learning techniques for problems like this. Early machine learning segmentation approaches such as [137], use convolutional neural networks (CNNs) to produce a segmentation. The system presented in [137] uses a similarly advanced segmentation strategy to perform a pixel-wise segmentation of the scene. Farabet *et al.* initially over-segment the scene, giving vastly more segments than objects, then a trained CNN classifier is run on each of the of segments to join the segments into super-pixels. Following this initial step the super-pixels are used to form a conditional random field (CRF), in which the joint-probabilities at the scene level are modelled between super-pixels. This is finally used to form a hierarchical segmentation of the scene for analysis purposes. They show this allows a more flexible segmentation representation and also allows for the training of a more general classifier which is the purpose of the work they conducted. The resultant segmentations were highly accurate, but this is largely due to the use of a trained classifier in determining different scene types. A more recent system presented by Prisacariu *et al.* in [136] uses no explicit per frame segmentation of the scene to allow for execution, instead performing a very simple colour based histogram segmentation of foreground and background. This is an example of where a simple method is perhaps all that's required as the application is real-time and as such one would expect to be able to improve your estimate in subsequent frames, and allows you to ignore the problems that are explicitly handled by single frame solutions such as those in [135, 137].

More recent approaches [34, 138, 139] use largely entirely CNN based approaches, allowing the machine to learn the useful aspects of segmentation and design architectural components as well as complementary loss functions to aid in network performance. In [34], Eigen *et al.* presents an approach for segmentation based on the popular VGG architecture [27], that includes complementary loss functions, that requires the network to optimise for not only segmentations but also depth and normals. This is shown

2 RELATED WORK

to improve the overall network performance, as well as the inclusion of auxiliary loss terms, which attempt to force the network to generate a low-resolution result from an earlier layer of the network, effectively forcing the network to encode the information earlier in the network. In [138], Badrinarayanan *et al.*, demonstrate a state of the art approach, that again uses a similar architecture to VGG but in this case, the network is used to encode and then a flipped version is placed on the end as a decoder. This encoder/decoder model has become popular in segmentation [138, 139] and dense labelling tasks. This approach uses un-pooling with a learnable convolution on the unpooling layer, which they claim improves performance. Another technique that has become very common is the notion of using skip-layers [140], that effectively reintroduce lower level feature maps back into the current network layer, effectively allowing this layer to ‘skip’ to a later section of the network. This has the effect of reintroducing high frequency information that can be lost in the encoding stage, allowing for sharper segmentations. These original architectures designed for segmentation, have since been found to be incredibly strong performers in geometric quantity estimation [141].

2.2.3.2 Geometry Based Segmentations

Geometry based segmentation uses the geometry information to segment the scene into its elements. Typical approaches include using physical edges (regions of strong curvature) to separate elements combined with an estimate of large geometric features in the scene such as planes [53, 40] or using local geometric information between points in a neighbourhood to determine their local similarity [100, 101].

An example of a simple geometric segmentation is the classic planar technique, which is found in multiple works such as [53, 40]. In the work by Rusu *et al.* the initial scene segmentation is based on a simple planar based segmentation, where they show this method of segmentation on a point cloud is very effective at retrieving a good initial estimate of where the object in the scene might be located. The method uses a simple RANSAC based approach to find the possible planes in the scene, and any points not lying on one of the planes found is said to be an object and points in these regions are segmented for examination. Another example of such a system can be found in [40], where Silberman *et al.* produced a system that not only segments the scene into major surfaces and objects it also attempts to determine support relationships. This method showed that planar based segmentation is useful for separating objects from planar surfaces, but showed that it couldn’t be heavily relied upon for accurate segmentation, ultimately showing they required colour information in order to improve segmentations. In the vast majority of situations this method is appropriate for finding objects that you wish to segment in the scene but was shown to be vastly improved upon by including a colour information in the segmentation method.

2 RELATED WORK

Rusu *et al.* demonstrated that the method presented in [101] can be extended to segment the scene and was shown to be viable but perhaps expensive in real-time in [53, 54]. This extension of their previous work showed that simpler geometric approaches are able to give good segmentations based on simple planar heuristics, that objects probably lie on planes, and that planes form surfaces and/or larger objects. These are regions of low surface curvature and demonstrated the importance of measuring surface curvature to segmentation and object manipulation in a robotics context.

2.2.3.3 Combining Depth and Colour Information

For the case of tracking and mapping fusion based methods that incorporate depth and colour information have been found to improve performance significantly over using them alone. Not surprisingly this has also been found for segmentation where improvements have been made clear in many recent works [142, 143].

Kim *et al.* use a *state-of-the-art* segmentation algorithm in [143] based on colour, demonstrating its use in a larger system. They first use a sliding window approach to estimate possible bounding boxes for segmentation candidates, these are used by the system to produce a list of K candidate segmentations with the use of a constrained parametric min-cut algorithm [144] which is a trained classifier of pixel segments, that uses the contour and colour information for training. This segmentation method provides a number of hypotheses for the *true segmentation*, which they use as a basis for their strategy. They demonstrate impressive segmentation performance using the depth information to give a final segmentation that will *hopefully* incorporate all the information available.

Another implementation that attempts to use as much of the information available can be found in [142]. Mishra *et al.* use colour and depth information to form a probabilistic boundary edge map. In their first step they use a series of convolutional masks which compute the edges at discrete angles as an initial estimate of object boundaries. From this they examine the 3D points on either side of the edge, estimating the plane and the nature of the plane intersection is used to determine whether the pixel is a boundary edge or an internal edge. This is largely based on a threshold of the probability function. The resulting probabilistic border map is then used with a fixation based minimum contour fit and the object boundary is extracted. This implementation only relies on valid depth estimates as the seed points to the segmentation method. Additionally duplicate edges are found and eliminated. One issue in this implementation is how they deal with object boundaries that perhaps might be interpreted as internal edges, this could result in the redundant calculation of the same edge in multiple objects, so this

2 RELATED WORK

case is explicitly handled by only dealing with edges for exactly one object in terms of classifying borders as inside on outside.

Again these previous works have largely been superseded by machine learning approaches [40, 145, 146]. All these approaches use different CNN architectures to perform segmentation based on colour and depth information. One interesting aspect of these approaches is the same architectures can perform significantly differently depending on how the information is presented to the network [146]. A common approach is to convert the depth to a three channel input HHA, which many pre-trained networks expect. **HHA** stands for **H**orizontal disparity, **H**eight above ground and **A**ngle with gravity. In [147], Holder *et al.* investigates the contribution of each input and concludes that the addition of depth information provides little improvement to performance, in general it also complicates the approach as most common architectures do not expect more than 3-channel inputs for image based machine learning.

2.3 Machine-Learning

Machine learning has become an essential element of many computer vision applications, including semantic labelling [148, 149, 150, 151, 134], scene/image understanding [152, 153, 149, 154, 36, 27, 29, 155], human robot interaction [156] and geometric feature extraction [34, 35, 157, 158, 159, 160]. It is the process of *teaching* a machine to solve a problem by providing it with input data it will have available during operation and the expected output of the machine, and attempting to get the computer to converge upon the solution to the problem without any interference by a programmer. Common machine learning approaches have often borrowed from the ideas about human intelligence, in particular attempting to approximate the function of the human brain or its individual elements. Some historically popular algorithms are restricted boltzman machines and decision trees/forests, non-linear SVMs, de-noising auto-encoders and neural networks. In recent years computers have become more powerful and high-performance computational libraries have been made widely available [161], largely narrowing the focus of machine learning to Convolutional Neural Networks (CNNs). In this section the applicability of registered RGB-D data to the task of machine learning is discussed. As suggested previous sections combining multiple sources of information provides improved performance on a variety of tasks.

In solving the classification problem, all learning algorithms essentially do the same thing in a mathematical sense, each takes the high-dimensional space in which the input data exists (pixel values for images for example), encode this information in some n-dimensional embedding, and attempt to group the data by separating the data with n-dimensional hyper-planes [162, 163]. Linear Support Vector Machines (SVMs) were a very popular choice in the computer vision community for semantic labelling

2 RELATED WORK

[162, 100, 164, 163, 165, 143, 166, 167, 54, 150, 168, 154, 153, 149, 135]. This is most likely due to their ease of implementation and their ability to handle n-dimensional data very simply [162]. This allows one to train an SVM with very large input vectors. In general these input vectors take the form of feature values that are computed over the raw input data, as SVMs are expected fixed dimensional input data. However, SVMs are considered rather limited, as they are primarily for classifying data, and generally don't work well on highly non-linear problems.

Some early approaches to machine learning begun by preconditioning the input data between dimensional spaces [100, 164, 100, 167], sometimes as a method of making all the input data a consistent dimensionality, other-times as a way to apply a linear learning algorithm for a non-linear problem by transforming the non-linear data to a linear feature space which can be learnt. This transformation can be desirable in a segmentation context, if the original data contains insufficient dimensionality to be separated easily. A simple example of this would be transforming from pixel space into feature space by taking a local descriptor around a feature. However, this reduction in dimensionality of the data can lead the machine learning approach to not see some part of the data that is removed in the conversion to a feature. The choice to use the reduced dimensionality data was originally made in order save computational time, and previously hand-crafted features were seen as containing the majority of salient information. Advances in parallel computing [161] have increased the utility of Graphics Processing Units (GPU) have made them the compute method of choice for almost all machine learning. This has largely moved the field away from this approach, as its impossible to know what information is important prior to training, so using everything available and allowing the machine to learn saliency is ideal.

Datasets [51, 153, 169, 170, 40] are a significant contribution to the computer vision and machine learning community as it allows for standardisation across testing results. Additionally a good dataset is difficult and time-consuming to obtain, things like ground truth segmentation and labels or accurate camera poses are generally challenging to obtain and require expensive hand labelling or external equipment. Sturm *et al.* provide one of the original RGB-D datasets [51], which provides a set of Kinect captured scenes with ground-truth pose information, allowing for a new SLAM benchmark to be defined. Silberman *et al.* present a popular dataset NYUv2 [40] to complement NYU's original segmentation dataset [170]. This dataset has been used by many other systems [165, 34, 35, 171, 172, 159] and was the largest such dataset until recently. Lai *et al.* produced an object focused RGB-D dataset [153], which consists of three hundred small objects arranged into fifty-one different categories, and taken at a set number of angles. The purpose of this dataset was for learning scene labels for objects for semantic labelling purposes. Xiao *et al.* present SUN3D [169], a dataset which provides labelled data for much larger scenes than for NYUv2. In their implementation they design a user input system that requires only minimal intervention to label a large number of frames.

2 RELATED WORK

With the system using a novel-generalised bundle adjustment to transform the 3D points selected by the user as the outline of the object. This allows for all subsequent frames to be given an estimated label using the correspondences, and then these can be quickly refined by the user. These datasets have varying applications but can be useful for standardised comparisons across multiple systems and each of them has been made possible largely due to the ease at which data can be captured with the Kinect. Recently these datasets have been used extensively as source of information for training machine learning approaches. They are now being used as the basis for learning tasks such as predicted depth from single colour image [34, 35, 159], without requiring any multi-view geometry computations.

2.3.0.4 Supervised vs Unsupervised Learning

Supervised Learning is based around providing the learning algorithm with a large amount of labelled data and using the difference between output and the ground truth (referred to as estimation error) to adjust the internal parameters of the system in an effort to reduce the error. This method of machine learning in general will require a large amount of labelled data [152]. A key unresolved issue of supervised approaches is where does all the labelled data come from? This is becoming less of a problem as data collection and sharing becomes easier however, the accuracy of the labels provided can also vary greatly, making it challenging to build accurate models. Unsupervised learning is perhaps a bit of a misnomer, and should probably be referred to as semi-supervised learning in almost all cases. It is a method of machine learning that doesn't use explicit labelled data in the error function. One approach to this method is to provide a very large number of positive examples to the network, for example images of a single classification label, say *cats*. Instead of training the model to distinguish a direct labelling the system may be trained to learn features that separate the embeddings of feature vectors. This is really semi-supervised as the learner is being provided with the examples that have a common *label*.

2.3.0.5 Semantic Representations

The effectiveness of unsupervised learning was demonstrated by Hinton *et al.* in [173], where they used an adaption of Boltzmann Machines known as Restricted Boltzmann Machines (RBMs) to learn the features required to represent hand written digits. The implementation by Hinton *et al.* even allows one to view "what the machine is thinking", as it can represent what it currently predicts the digit might look like. The machine is trained based on observing the correlation between the input and hidden layer outputs,

2 RELATED WORK

and then adjusting the weights based on the assumption that all the weights are linked. This work was expanded upon in [174] where Lee *et al.* demonstrated the method was extensible to more complex image processing problems such as face recognition, with a high level of effectiveness. They also visualise the learnt information, at each level demonstrating that the features learnt are very similar to those that we train them with, edges and image gradients for example. The machine is trained by applying filters at each level across each of the sample images. The lowest level learns directed edge filters, while the convolutional and pooling layers demonstrate that the edges are then combined into arrangements that look like rough representations of different objects in the training data. This process of combining simple features into more complex ones through successive layers of convolutions in the basis of CNNs. They believe this demonstrates that the human mind may function in a similar way in its recognition and perhaps even learning algorithm, at least in a forward pass sense.

Expanding upon the findings in [174] Coates *et al.* instead use K-means clustering [175] which they demonstrate in [176] has basically equivalent effectiveness as other learning methods given enough data. Using this principal they train the classifier on a very large unlabelled dataset collected to examine the results. What they find is the classifier learns the same edge type filters, and that the higher level features correspond to common objects such as people and cats as you would expect. This method has the advantage of being very easy to implement, but takes a very long time to learn.

Rusu *et al.* present an example of supervised machine learning in [100]. They introduce the Point Feature Histogram (PFH), a method for loosely classifying any point in a point cloud into generic geometrical classification such as planar, spherical, etc... using only the normal information in a local patch. This is formed in an initial training phase where they explore which learning methods were most appropriate, including K-nearest neighbours (KNN) and SVM, ultimately deciding on using the SVM. They then use this per-point classification to train an additional SVM for object recognition using the histograms of point-classifications of labelled regions, using a techniques borrowed from [162]. They extended this initial work to create the Fast Point Feature Histograms (FPPH) [101] which are largely the same as PFH's differing in the fact that they no longer consider distance between sample points in the descriptor as they found the information to be of little value to the discriminative power. Rusu *et al.* extend upon the idea of the PFH in [54] to create a Global-FPPH (GFPPH) which is again a general measure of the local similarity of the point-wise classifications, forming a histogram of the pair-wise point classifications in a region. The region selection is based upon an efficient oct-tree segmentation of the points that have been labelled. This is again used to train a linear SVM which has shown to give very good classifications, and enabled their system to work out complex object affordances (where a cup can be grabbed from). Having accurate depth information actually gives more information than humans have, hence its not surprising that systems are now able to surpass human ability to segment

2 RELATED WORK

scenes. However, these approaches are heavily reliant on the quality of the data, and are not truly view-point invariant, as the data may contain holes or occlusions.

In [149] Lai *et al.* employ a sliding window approach to segmentation and labelling. Instead of learning from a full scene model they use scene views to train the detector, based on the view in frames of the training data. By first oversegmenting the data then classifying on the over segmented data, this approach will make the classification and detection process more symmetric resulting in better performance, and simpler computation. They use a HOG descriptor to represent the features for matching, filling in any missing data and removing noise with a median filter. Using a Markov Random Field (MRF) over the descriptors they produce an effective classifier. Expanding upon this work [165] take a similar approach to form descriptors, but instead use a cuboid to provide a bound for the object, and using this cuboid as input to the feature vector to improve matching. In [177], Russell *et al.* use a variant of MRFs known as Conditional Random Field (CRF) which further improved classification accuracy. A CRF was shown to be replaceable by a separately trainable Recurrent Neural Networks (RNN) that acts as a surrogate CRF in [178], and show again improved performance. More recent segmentation networks remove the CRF altogether, instead opting to encode the data in successively concatenating layer activations, where higher spatial information activations are constantly reintroduced to the filters through concatenation. These neural networks have been called DenseNets, and were first introduced in [155] for semantic labelling. This architecture has been incorporated as a decoder in some of the strongest architectures in many dense estimation tasks including depth estimation [141, 171].

2.4 Depth without a Low-cost Depth Sensor

The use of low-cost depth sensors for robotics has proven to be extremely useful, but sometimes its not possible to use a special sensor like a low-cost depth sensor. As mentioned in Section 2.1 there are many modes of operation that are unsuitable, for example outdoor scenes for many active sensors due to ambient lighting. Many competing approaches have made significant steps to possibly replacing or removing the need for a depth sensor, demonstrating that knowledge of geometry and camera motion is all that's required to robustly estimate depths from an monocular camera. More recently, impressive strides have been made in estimating accurate metric depths from colour images alone [35] demonstrating that the colour image may contain enough information on its own. In this thesis we demonstrate the value of higher order geometric features to the process of estimating depths, such as curvature, normals, optical flow, and relative pose.

2 RELATED WORK

2.4.1 Semi-Dense Depth Estimation

Semi-dense depth estimation is related directly to semi-dense monocular tracking approaches [179, 10, 11]. I separate these approaches from the feature based approaches [111, 50, 7], purely because they don't use explicit photometric features in the localisation. Instead these approaches all optimise based on photometric error minimisation. In [179], Engel *et al.* propose a system that maintains a dense estimate of depth for all points in a frame with sufficient image gradient for tracking. This has the dual effect of reducing the amount of computation to be much more manageable as a CPU implementation, but also robustifies the solution as these points have increased saliency. These estimates of depth are propagated through time, and subsequently refined upon increasing tracking and modelling accuracy. Engel *et al.* extend this approach in [10], to optimise for not only relative pose but over the group of scaled pose transformations, allowing the direct optimisation of scale and the reduction in error due to scale-drift. They extend this work again in [11], including a detailed model of camera intrinsics and extrinsics in the optimisation, resulting in further improvements to accuracy. This logical progression of incorporating more and more information into an optimisation, while eliminating ambiguous data points with low signal-to-noise ratio, is demonstrated to improve performance.

2.4.2 Dense Depth Estimation

Dense depth estimation using a single monocular camera, in general relies on stereo triangulation techniques [180, 181, 60, 15, 8]. This requires the motion of a camera be tracked across a sequence of frames using correspondences between frames. By estimating a sufficient number of correct correspondences across multiple views, a relative pose between frames can be computed. This is done using the geometric properties of a transformation, allowing an estimate for the depth to be triangulated (see Section 3.3). Strecha *et al.* present an approach that uses a stereo pair of cameras and the motion between two frames to optimise for an improved dense depth estimate. They extend this work in [181] to work for an arbitrary set of captured images, that need not be stereo pairs, and treat the depth estimation problem as an expectation maximisation (EM) problem over all views. These approaches both produce dense depths, but require multiple view of the same scene and offline processing due to their complexity.

The feature matching method proposed in [60], provides a *potentially* real-time approach to stereo depth estimation. The method uses the DAISY descriptor to compute dense feature matches across a set of overlapping view-points. Tola *et al.* demonstrate their approach is more discriminative across wide-baselines and more simply calculable. This is *potentially* real-time only because they only demonstrate that the proposed method is highly parallelisable but leave it as future work. This approach could be made real-time

2 RELATED WORK

and a dense depth estimate computed for each frame, due to the parallel nature of the convolutions integral to the computation of the feature. They do extend this work in [182], to estimate dense point clouds for *large* datasets and show impressive multiple core CPU performance.

Real-time dense monocular depth estimators [15, 8] use similar approaches to estimate depth, relying on photometric consistency across frames to densely triangulate points. In [15], Newcombe *et al.* use a novel cost volume implementation to estimate depth while tracking to the current estimate of the volume. Although this may appear to be more challenging, to try track in real-time to an incomplete model, this approach actually produces a much more robust solution. The initial depth estimate is poor, but a strong estimate for the initial relative alignment (given by the approach in [183]) allows the refinement of these depths. This approach uses a primal-dual approach, to provide an upper-bound to an error function that is used to triangulate points across the epipolar lines, in order to significantly increase the search speed without compromising on accuracy. This allows for a real-time dense depth estimation to be made using a GPU implementation, which can be extremely reliably tracked against in varying lighting and focus conditions. Pizzoli *et al.* present a similar approach in [8], where the cost-volume is replaced by a Bayesian depth estimation, using planar region estimates to perform depth filling. The proposed solution is also implemented on the GPU, and runs in real-time. Both approaches are real-time but ultimately require a massive amount of information to compute, requiring multiple seconds of videos, with literally 100's of overlapping images. A good estimate for depth can be made with as little as two images using geometric constraints, but can an estimate be performed using only a single image?

2.4.3 Machine-Learning Approaches

Using the information and architectures learned from tasks like pose detection [184, 185], image classification [26, 37], semantic segmentation [146], can a CNN be trained to estimate depth given a single frame? The intuitive answer should be yes, as an individual with one eye open can perceive depth to a limited extent [186], meaning there is information that can be learned to provide clues about depth. 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 the right direction. Predicting depth from a single RGB image using learning based approaches has been explored even prior to the resurgence of CNNs. In [187], Saxena *et al.* employed a Markov Random Field (MRF) to combine global and local image features. Eigen *et al.* introduced a common CNN architecture [35] capable of predicting depth maps for both indoor and outdoor environments. This concept was later extended to a multi-stage coarse to fine network in [34].

2 RELATED WORK

Advances in depth estimation have been made by combining graphical models with CNNs [157] to further improve the accuracy of depth maps, through the use of related geometric tasks [159] and by making architectural improvements specifically designed for depth prediction [158]. Kendall *et al.* demonstrated that predicting depths and uncertainties improve the overall accuracy in [141]. This indicates training a network to estimate the how certain it's estimates are, not only improves it's performance but supplies meaningful data that can be useful for traditional optimisations. While previous approaches have demonstrated impressive results in single image depth estimation, explicit notion of geometry was not used during any stage of the pipeline.

Using explicit geometry constraints has been shown multi-view stereo problems to provide high quality depth estimates [15, 8]. One of the earliest works to predict depth using geometry in an unsupervised fashion, Garg *et al.* used the photometric constraints between a stereo image pair, where the target image was synthesized using the predicted disparity and the known baseline[42], in a fashion similar to [133]. Left-right consistency was also explicitly enforced in the unsupervised framework of Goddard *et al.* [43] as well as in the semi-supervised framework of Kuznietsov *et al.* [44], which was found to be beneficial during training, particularly on sparse ground truth datasets[41]. These approaches all rely on stereo constraints to be known at training time to allow for the additional loss functions. To have a known relationship between pairs of images can be challenging, and many datasets were not captured with known poses. Increasing training data is known to improve performance, and produce more general solutions. Can more a general training regime be developed to allow the use of more generic training data for depth estimation?

Zhou *et al.* address the issue of general training data in [46], where they train a neural network in an unsupervised to jointly estimate a pose transformation between frames and single image disparity. As opposed to assuming a fixed relationship between frames [42, 43], they allow for a general transformation and again minimise photometric error based on the geometric constraints. Similarly in [45], Ummenhofer *et al.* produce an approach that also attempts to solve for general transformations in a network bootstrap approach. They alternate between estimating flow and depth, to compute a pose and using the pose to generate a flow that can transform the depth and compute a measure of photometric consistency. In contrast to [46], this approach is optimised in an almost fully supervised fashion, with ground truth depth, flow and pose used in training. Both of these approaches also predict a single confidence map which is able to improve their results, again demonstrating the importance of uncertainty in machine learning solutions. They are essentially a proof of concept, as neither approach achieves state-of-the-art performance in the tasks, but both illustrate that an approach that used pose information can be used to improve the performance of a depth estimation network. Using pose, forces the network to learn more robust image features that allow the network to fulfil the subsequent implied geometric constraints.

2.4.4 Towards Real-time

Although these single image depth estimation networks can produce extremely impressive results [171], they are in general not able to run in real-time without large high powered hardware. As robotics is a field that requires light-weight, low-cost solutions to move forward, this is less than ideal. Light-weight high quality real-time approaches are highly desirable in many applications such as unmanned aerial vehicles and self-driving electric vehicles. A significant push has been made to create networks that run in real-time [48, 188, 189]. Due to the attractive qualities such as low power consumption and high mobility, researchers have been keen to examine the possibility of building smaller architectures.

In [48], Howard *et al.* present an method that is targeted at mobile devices, MobileNets. Their approach significantly reduces computation time through the use of separable convolutions, which separate large convolutions into multiple passes, the first pass sets the change in channel count and the second pass attempts to capture cross-channel features. Zhao *et al.* present an approach ICnet [188], which computes convolutions at multiple scales in parallel to reduce the time required for inference. The auxiliary losses force the network to learn features earlier in the feature embedding, allowing a shorter network and improved performance. In [190], Paszke *et al.* present a method that uses asymmetric and dilated convolutions to reduce the overall computation without sacrificing much in terms of performance. Dilated convolutions are particularly helpful in this case, as they allow a large increase in the networks receptive field, while maintaining relatively constant computational cost. Deng *et al.* refine this idea further in ERFnet [189], further reducing the model size and computational cost, without reducing performance a great deal. In this work, they target a particular frame-rate on a particular hardware platform, in this case 15fps on an NVIDIA TX1. In essence, all these approaches show with a good choice of architecture, computational time can be gained with small sacrifices in performance. Additionally even when these approaches are allowed to grow, and take a large amount of extra computation, they still fail to outperform the state-of-the-art approaches.

The machine learning community has also investigated the problem of model compression or emulating the performance of a larger network. Hinton *et al.* in [49] introduced a concept called distillation which aimed to replicate the class probabilities of a larger model using a smaller model. In the case of depth estimation, this is a regression problem not a classification problem, training a smaller network to replicate the prediction layer of a larger model becomes difficult with no notion of class probability. Could a network be trained to mimic the latent space or the embedding of the penultimate layer of the larger model in an unsupervised manner? Similar to [49] Bucila *et al.* [191] showed that it is possible to replicate the performance of an ensemble of classifiers using a single model. Their method relied on generating synthetic data using an ensemble

2 RELATED WORK

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 [192] for image classification. The approach in [192] demonstrates that networks certainly contain redundancy which can be removed without compromising performance. Using each of these still has the effect of largely capping the possible network performance to the ability of the large networks, or ensemble of networks, but shows that more information can be conveyed to the small fast networks in a number of ways and this improves their performance significantly over standard regression.

Fundamentals

3.1 Points, Vectors, Matrices, and Tensors

This section provides a short note on notation used throughout this thesis, as well as some important basics of matrix, discrete and continuous operations.

3.1.1 Points

A point p_i can be N -dimensional, is represented using the notation

$$p_i = (x_1 \ x_2 \ \cdots \ x_N)^T, \quad (3.1)$$

where a lower-case is used to indicate a point, or a value (which is a 1D point).

3.1.2 Vectors

a vector is a set of values, points or vectors of length N given by

$$\bar{\gamma} = [\gamma_0, \cdots, \gamma_N]^T, \quad (3.2)$$

where the $\bar{}$ notation is used to indicate a variable is a vector.

3.1.3 Matrices and Tensors

A matrix is a set of values stored in a $N \times M$, 2-dimensional grid as follows

$$\mathbf{A} = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0M} \\ a_{10} & a_{11} & & a_{1M} \\ \vdots & & \ddots & \vdots \\ a_{N0} & a_{N1} & \cdots & a_{NM} \end{bmatrix}, \quad (3.3)$$

where we use both a capital and bold to indicate a matrix. Tensors are a generalisation of a matrix to N -dimensions in this thesis (although they are truly interchangeable

3 FUNDAMENTALS

terms). For three dimensions, a tensor of size $N \times M \times L$ could look like the following

$$\bar{\mathbf{A}} = \left[\begin{array}{c} \left[\begin{array}{ccc} a_{00L} & a_{01L} & \cdots & a_{0ML} \\ a_{10L} & a_{11L} & & a_{1ML} \\ \vdots & & & \vdots \\ a_{N0L} & a_{N1L} & \cdots & a_{NML} \end{array} \right] \\ \left[\begin{array}{ccc} a_{000} & a_{010} & \cdots & a_{0M0} \\ a_{100} & a_{110} & & a_{1M0} \\ \vdots & & & \vdots \\ a_{N00} & a_{N10} & \cdots & a_{NM0} \end{array} \right]^{T1} \end{array} \right] \begin{array}{c} \vdots \\ a_{NML} \end{array}, \quad (3.4)$$

where multi-dimensional tensors are represented with bold, capital and the $\bar{}$ notation.

3.1.4 Matrix Transpose

The matrix transpose for a 2×2 matrix \mathbf{B} is given by

$$\mathbf{B} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \mathbf{B}^T = \begin{bmatrix} a & c \\ b & d \end{bmatrix}, \quad (3.5)$$

where T is used to indicate the transpose operation.

3.1.5 Matrix Inverse

The inverse of a matrix is defined by the following matrix equation

$$\mathbf{A}\mathbf{B} = \mathbf{B}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I} \quad (3.6)$$

where in $\mathbf{B} = \mathbf{A}^{-1}$ is the inverse of A , and $\mathbf{A}, \mathbf{B}, \mathbf{I} \in \mathbb{R}^{N \times N}$ is only defined for square matrices of size N . Only square matrices are invertible, and only if their determinant $\det(A) \neq 0$. For a matrix $\mathbf{A} \in \mathbb{R}^{2 \times 2}$

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad (3.7)$$

the determinant is given by

$$\det(\mathbf{A}) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = 1 * a * \det(d) + -1 * b * \det(c) = ad - bc. \quad (3.8)$$

This pattern extends for larger matrices, for a matrix $\mathbf{A} \in \mathbb{R}^{3 \times 3}$

$$\mathbf{A}_{3 \times 3} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad (3.9)$$

3 FUNDAMENTALS

the determinant is given by

$$\begin{aligned}\det(\mathbf{A}_{3 \times 3}) &= 1 * a * \begin{vmatrix} e & f \\ h & i \end{vmatrix} + -1 * b * \begin{vmatrix} d & f \\ g & i \end{vmatrix} + 1 * c * \begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ &= (aei - afh) - (bdi - bfg) + (cdh - ceg)\end{aligned}\quad (3.10)$$

The inverse of a 2×2 matrix is given by

$$\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}. \quad (3.11)$$

The general inverse of an $N \times N$ matrix is given by

$$\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})} \text{adj}(\mathbf{A}) \quad (3.12)$$

where $\text{adj}(\mathbf{A})$ is the adjunct matrix of \mathbf{A} . The adjunct of a matrix is given by the transpose of the co-factor matrix \mathbf{C} . The co-factor matrix of $\mathbf{A}_{3 \times 3}$ is given by

$$\mathbf{C} = \begin{bmatrix} + \begin{vmatrix} e & f \\ h & i \end{vmatrix} & - \begin{vmatrix} d & f \\ g & i \end{vmatrix} & + \begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ - \begin{vmatrix} b & c \\ h & i \end{vmatrix} & + \begin{vmatrix} a & c \\ g & i \end{vmatrix} & - \begin{vmatrix} a & b \\ g & h \end{vmatrix} \\ + \begin{vmatrix} b & c \\ e & f \end{vmatrix} & - \begin{vmatrix} a & c \\ d & f \end{vmatrix} & + \begin{vmatrix} a & b \\ d & e \end{vmatrix} \end{bmatrix} \quad (3.13)$$

Therefore the adjunct of \mathbf{A} is given by

$$\text{adj}(\mathbf{A}) = \mathbf{C}^T. \quad (3.14)$$

3.1.6 Matrix Decomposition

Matrix decomposition is used frequently in order to transform a matrix into a form that is easier to work with in some way, often allowing one to avoid explicitly calculating an inverse matrix. There are many decompositions, the most relevant to this thesis is the Cholesky decomposition. Cholesky decomposition is only applicable to matrices that are symmetric positive-definite, it decomposes a matrix \mathbf{A} as

$$\begin{aligned}\mathbf{A} = \mathbf{LDL}^T &= \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \begin{bmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & D_3 \end{bmatrix} \begin{bmatrix} 1 & L_{21} & L_{31} \\ 0 & 1 & L_{32} \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} D_1 & L_{21}D_1 & L_{31}D_1 \\ L_{21}D_1 & L_{21}^2D_1 + D_2 & L_{31}L_{21}D_1 + L_{32}D_2 \\ L_{31}D_1 & L_{31}L_{21}D_1 + L_{32}D_2 & L_{31}^2D_1 + L_{32}^2D_2 + D_3 \end{bmatrix}\end{aligned}\quad (3.15)$$

3 FUNDAMENTALS

where \mathbf{L} is a lower triangular matrix, and \mathbf{D} is a diagonal matrix. The elements can be computed iteratively using the following equations

$$\begin{aligned} L_{ij} &= \frac{1}{D_j} \left(A_{ij} \sum_{k=1}^{j-1} L_{ik} L_{jk} D_k \right) \text{ for } i > j, \\ D_j &= A_{jj} - \sum_{k=1}^{j-1} L_{jk}^2 D_k. \end{aligned} \quad (3.16)$$

3.1.7 Convolution

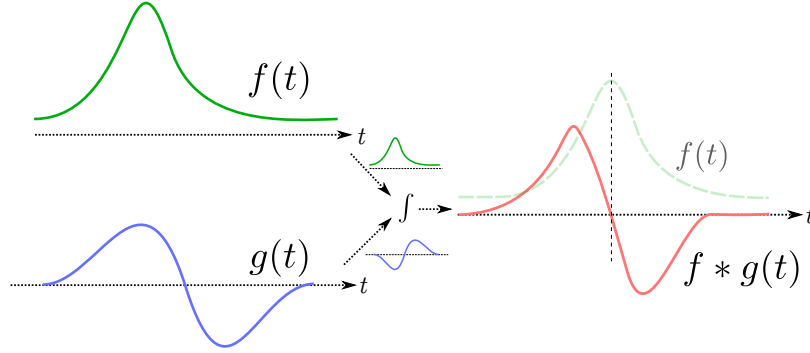


Figure 3.1: Demonstrates the convolution of two functions f and g , and shows the original function f in an overlay to indicate the relationship. The flip in the convolved function g is also performed, before the integration.

Convolution is a linear operation between two discrete or continuous signals denoted by the $*$ operator. The operation is the integral product of two functions, after one has been flipped and shifted, defined by the following equation

$$f(t) * g(t) = \int_{-\infty}^{\infty} (f(\tau)g(t - \tau))d\tau \quad (3.17)$$

where f and g are continuous functions of t , and in this case g is the function that is flipped and shifted. This is demonstrated visually in Figure 3.1, where the convolutional signal $f * g(t)$ essentially displays the derivative of the input signal $f(t)$, demonstrating one of the uses of convolution. Figure 3.1 is shown with two 1-dimensional signals, but convolution is possible between signals of equivalent dimension. An example of convolution with a 2D discrete signal is shown in Figure 3.2, which shows a gray-scale mapping of the values. The resulting convolutional signal highlights the presence of edges in the 2D signal.

3 FUNDAMENTALS

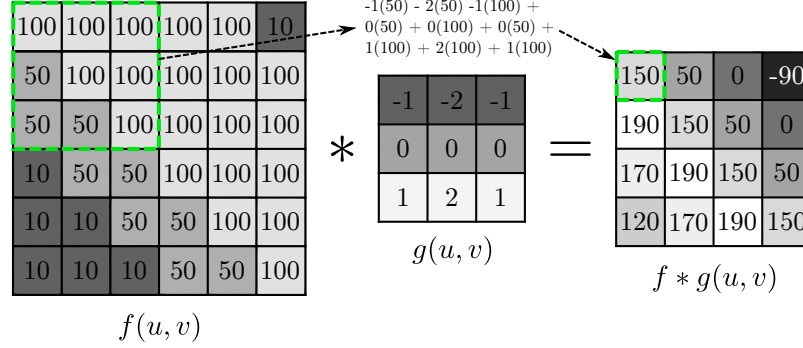


Figure 3.2: Demonstrates a discrete 2D convolution over a fixed domain and range, this shows that in the case of discrete convolutions. In this case we treat the borders to be undefined and thus the resulting convolved signal is $(N - 1)$ smaller in both height and width, where N is the size of the filter/convolutional signal $g(u, v)$. The resulting convolution for the top-left square is shown explicitly, and again the filtering signal g is flipped before performing the convolution.

3.2 Camera Models and Image Projection

This section is included to give the reader a brief background in camera projection, and correction techniques. For a more detailed explanation of all the concepts in this section, we would refer the reader to [193], this section is intended to act as a guide to the notation used through out this thesis.

3.2.1 Pinhole-Camera

The simplest camera model is pin-hole, which can be thought of as an ideal camera. Additionally this formulation makes it easy to illustrate image and camera coordinates. This can be done with an example shown in Figure 3.3, which shows the projection of a simple cube onto the image plane. As the light passes through the pin-hole it will invert the light and thus the image as it projects onto the image plane. In this thesis the diagrams show the virtual image behind the camera, it is also equivalent to show the un-inverted image in front of the camera, but both formulations can be considered equivalent. Note the origin of the camera

$$C_{origin} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad (3.18)$$

is at the position of pin-hole, while the origin of the image

$$I_{origin} = \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad (3.19)$$

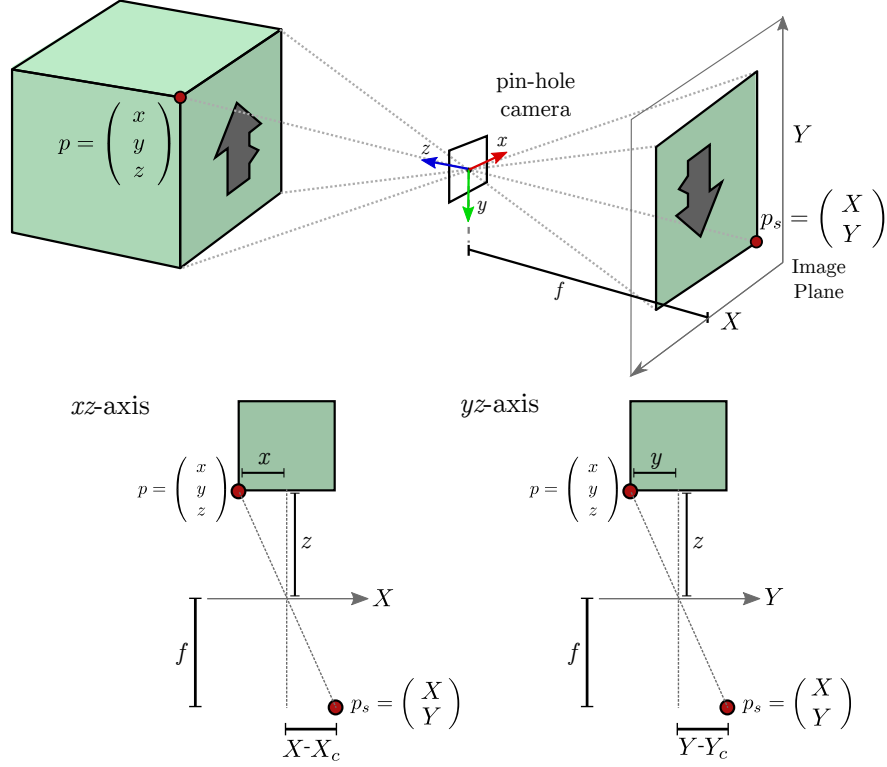


Figure 3.3: **Top:** A diagram of a pin-hole camera projecting the image of a cube onto the image plane. **Bottom:** Two diagrams showing the projection of a single ray to the image, along the xz -axis and yz -axis.

is at the top-left of the image (inverted). The bottom half of Figure 3.3 shows the projection of a single ray onto the xz and yz axis. Using the principal of similar triangles its simple to show the relationship between the world and the image coordinates given by

$$\frac{X - X_c}{f} = \frac{x}{z} \quad \frac{Y - Y_c}{f} = \frac{y}{z} \quad (3.20)$$

where $(X_c, Y_c)^T$ marks the centre of the image in pixel coordinates.

3.2.2 Homogeneous Coordinates

A more convenient way to represent coordinates is in a homogeneous fashion. This allows multiple benefits including a simpler expression for linear equations involving the coordinates, and also allows an easy way to express points at infinity using finite

3 FUNDAMENTALS

coordinates. Homogeneous coordinates take the form

$$\begin{pmatrix} \lambda u \\ \lambda v \\ \lambda \end{pmatrix} = \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \quad (3.21)$$

for image coordinates, and

$$\begin{pmatrix} \lambda x \\ \lambda y \\ \lambda z \\ \lambda \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (3.22)$$

in Cartesian coordinates in \mathbb{R}^3 , where λ in this case is some constant.

3.2.3 Camera / Projection via Matrices

Using the homogeneous coordinates we can express the projection of the camera to image coordinates as

$$\overbrace{\begin{bmatrix} f_u & 0 & X_c \\ 0 & f_v & Y_c \\ 0 & 0 & 1 \end{bmatrix}}^{\mathbf{K}} \overbrace{\begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix}}^{p_n} = \mathbf{K}p_n = \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \quad (3.23)$$

where $p_n = (x/z \ y/z \ 1)^T = (u \ v \ 1)^T$ are known as normalised camera coordinates, which is equivalent to projecting to an image plane with a focal length equal to 1. In this thesis the matrix \mathbf{K} is also called the camera matrix. In order to project from image (X, Y) to normalised-camera coordinates (u, v) to homogeneous world-coordinates the following inverse operation is performed

$$z\mathbf{K}^{-1} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (3.24)$$

which is sometimes referred to as an un-projection.

3.2.4 Lens Distortion

The pin-hole camera is one of the first types of imaging devices created for photography. One of its key issues is caused by using a point light source as the focus mechanism, which filters out most of the input light and makes imaging more challenging. Cameras today almost exclusively use lenses to focus the incoming light in a similar fashion. This method also has issues, but they can be more subtle. The two most significant

3 FUNDAMENTALS

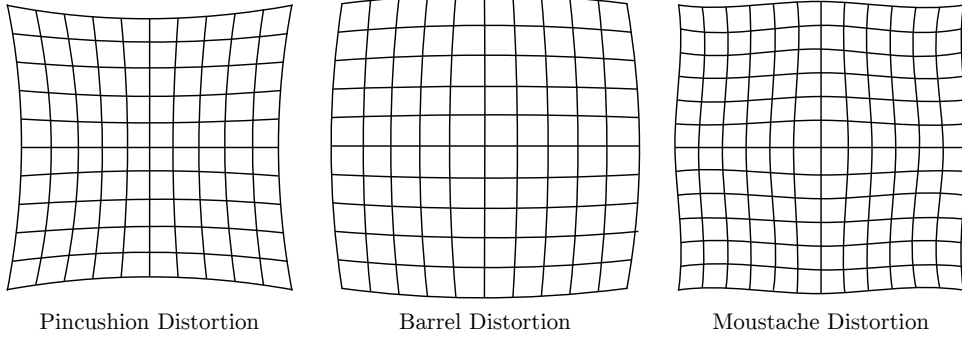


Figure 3.4: Demonstrates three common types of lens distortion, that need to be compensated for in a standard camera.

effects on accurate imaging are radial and tangential distortion, which are types of lens distortion, caused by imperfections in the lens or the geometry of the design.

Figure 3.4 shows three common types of distortion found in lens. All three distortions result from uneven magnification of the scene across the lens. With pincushion the image is magnified more as it moves away from the image centre. Barrel distortion magnifies the image the most in the centre and this results in a fish-eye type image, and is often present in wide-angle lenses. Moustache distortion is a combination of above distortions, this is more barrel distortion in the centre of the image, gradually becoming pincushion as you move away from the image centre. This results in straight lines at the top of the image resembling a handle-bar moustache.

The distortion model we use is

$$\begin{aligned}
 u_0 &= \overbrace{u(1 + k_1 r^2 + k_2 r^4 + \dots)}^{\text{Radial}} + \overbrace{(t_1(r^2 + 2u^2) + 2t_2 uv)(1 + t_3 r^2 + t_4 r^4 + \dots)}^{\text{Tangential}} \\
 v_0 &= \overbrace{v(1 + k_1 r^2 + k_2 r^4 + \dots)}^{\text{Radial}} + \overbrace{(2t_1 uv + t_2(r^2 + 2v^2))(1 + t_3 r^2 + t_4 r^4 + \dots)}^{\text{Tangential}}
 \end{aligned} \quad (3.25)$$

where k_1, k_2 are the radial distortion parameters, t_1, t_2, t_3 and t_4 are the tangential distortion parameters, u, v are the normalised camera coordinates, $r^2 = u^2 + v^2$ is the squared radial distance of the normalised camera coordinate, and u_0, v_0 are the corrected image coordinates. The corrected image coordinates are the image coordinates you would expect if the camera followed the ideal pinhole model.

3 FUNDAMENTALS

3.2.4.1 Radial

There are multiple methods of modelling radial distortion, this thesis focuses on Brown's method [194]. This approach uses the radial distance (r) from the centre of the camera to model the distortion shown in Equation 3.25. In general the first two calibration parameters of the correction function (k_1, k_2) are the most significant. This is because the higher order parameters can be unstable and in general contribute minimally to correcting the image.

3.2.4.2 Tangential

The tangential component of standard camera correction is in general less significant than the radial component, this can be seen in the formulation in Equation 3.25 where the polynomial terms are higher order.

3.3 Depth Camera Variants

This section is intended to explain some of the popular available options in low-cost depth cameras and effectively justify the choice of sensor used in this research.

3.3.1 Time-of-flight (ToF)

3.3.1.1 Operation

Time-of-Flight (ToF) sensors can operate a number of ways. The principle behind its operation is to estimate the time it takes for a beam of light to leave the origin (somewhere near the camera) and return to the camera. Using the known speed of light as a constant, the device directly translates this information into a distance. A ToF flight camera such as the Microsoft Kinect V2 does this by using a ranged gate imager. This works by broadcasting a modulated IR signal out from a point very close to the camera's origin, and having the gates of the image sensor open and close in unison with the transmitted signal. By measuring the integrated returned signal (as shown in Figure 3.5) at each gate, an estimate from the depth using the ratio of the received signal during the two time periods. This allows the sensor to compensate somewhat for ambient light pollution as this will effect both periods evenly, and also signal absorption

3 FUNDAMENTALS

for the same reason. To resolve distance ambiguities created by this process the gating can be performed at several frequencies to a specified depth resolution.

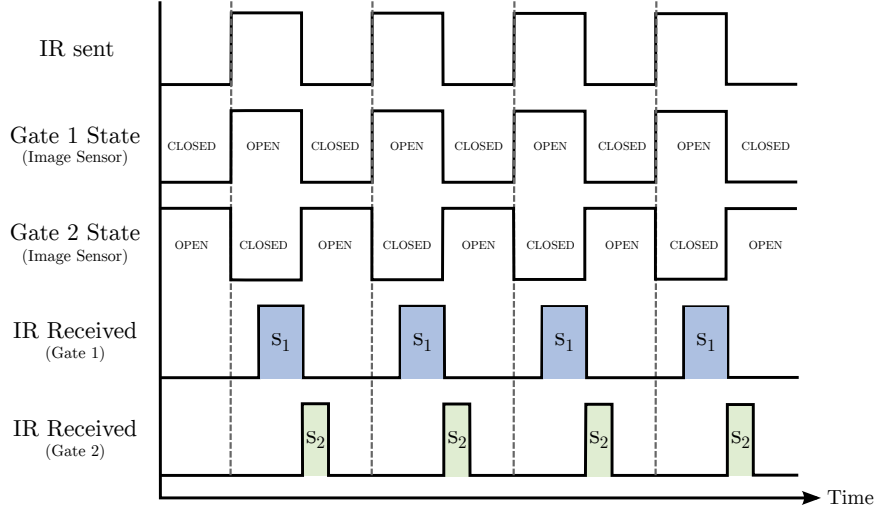


Figure 3.5: Demonstration of the basic operation of a Time-of-Flight (ToF) sensor similar to that used in Microsoft Kinect V2. **Top:** The modulated Infra-red (IR) signal that is broadcast onto the scene, **2nd and 3rd row:** The state of the gates in the sensor chip, each pixel has two gates that open and close 180 degrees out of phase with each other. **4th and 5th row:** The integrated returned signal for the respective gates. The values S_1 and S_2 indicate the integrated area while the signal is broadcast, and while it isn't respectively.

3.3.1.2 Sensor Issues

This design can be made highly accurate and recent advances have made them cheap to produce at higher resolutions and frame rates, 512x424 pixels at 30 frames per second in the case of the Microsoft Kinect©(version 2). However, this sensor and other ToF sensors suffer from several issues [75] that make it a less desirable choice in many robotics and mapping applications. The first notable issue is the multi-path problem, which results in the sensor believing points are closer than they actually are. This is demonstrated in Figure 3.6, the multi-path error makes all points in the corner of a reflective surface (including brushed aluminium) appear much closer to the camera, and makes these corners bulge towards the camera. The main issue with this type of error is its hard to detect that its even an error. What this means is, the sensor doesn't know when it's data is wrong.

Another known issue is the floating pixel problem (waterfall effect), which means the sensor naturally smoothes the depth boundaries of objects, smoothly connecting everything into a smooth surface from the viewpoint of the camera, demonstrated in Figure

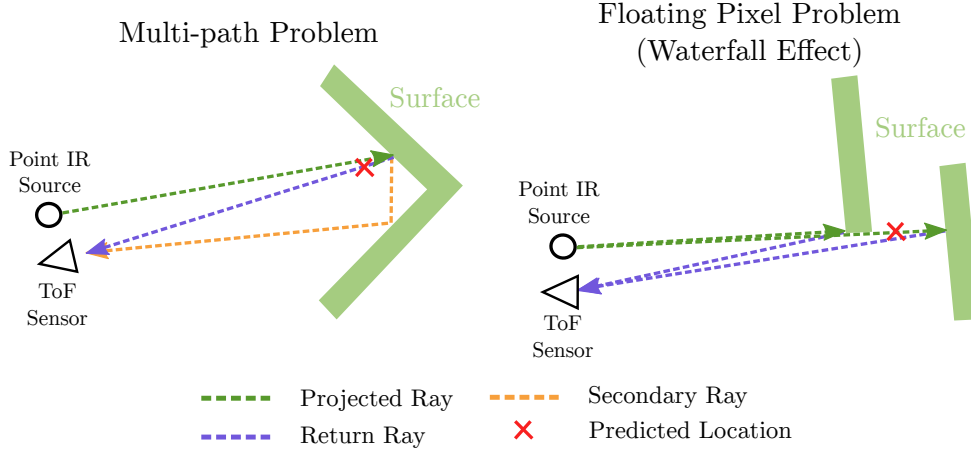


Figure 3.6: Left: An example of the multi-path problem, as a ToF sensor faces a corner it receives multiple signals back to the sensor, causing the point to appear closer. **Right:** An example of the floating pixel problem, where the sensor naturally integrates smoothly along object edges creating floating points between the surface and the background.

3.6. This is again a result of the integration of the sensor readings at object boundaries. This results in objects being smoothly connected to the table they are sitting on for example. A related issue to the multi-path problem is floating pixels, where some points appear to float in mid-air as a result of integrating stray reflections. Lastly, another serious issue is the light absorbency of some surfaces, which cause less return signal and result in these surfaces appearing further away. Again this is extremely difficult to detect as the points are uniformly moved further away and the surfaces that absorb light do so linearly proportionally to the incident angle, which means changing the viewpoint doesn't help. The key issue is that the sensor reports incorrect values in the situations described above. If it were to simply fail to return any values this would be much more desirable, as it avoids incorporating large amounts of error into any solution that use this sensor. A good summary of these issues found in this sensor type can be found in [64] and [75].

3.3.2 Stereo

We include a description of stereo imaging as a motivation/inspiration for the structured-light type sensors, as we do not use stereo colour cameras in this thesis. A familiar reader should move quickly (or skip) through this section.

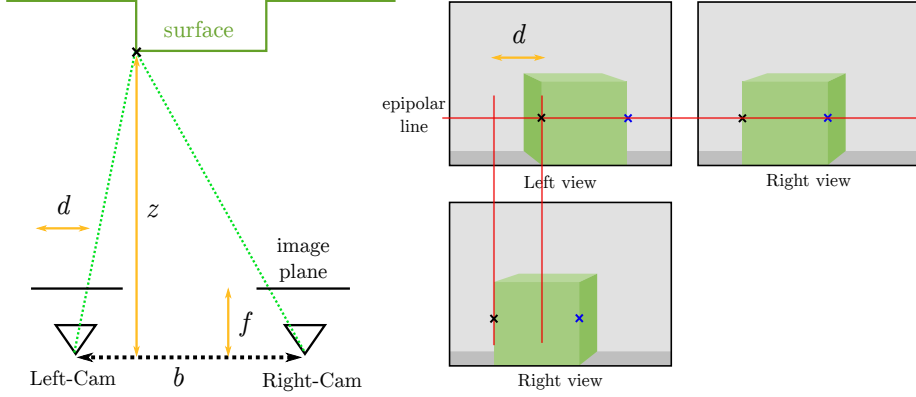


Figure 3.7: Left: The basic setup of a stereo camera rig, with the disparity (d), depth (z), focal length (f) and physical baseline (b) marked. **Right:** Demonstrates that following that for a stereo camera it's epipolar lines are horizontal, and the disparity corresponds to the horizontal shift of a particular pixel in image space.

3.3.2.1 Operation

Figure 3.7 above demonstrates the behaviour and operation of a stereo-rig. Generally stereo cameras are set up horizontally aligned, this means that in an ideal case the rows of the left image will be aligned with the rows of the right image. The distance between matching (corresponding) points given in the left and right images is known as the disparity (d). This relates inversely to the depth via the following relationship

$$\frac{d}{f} = \frac{b}{z} \implies z = \frac{bf}{d} \quad (3.26)$$

which is a result of applying similar triangles. This implies the epipolar lines, which are lines that point in the direction of another camera given two cameras are viewing the same scene, all point horizontally. These epipolar lines, turn out to be the only lines required to search along for matching (corresponding) points in two images. In practice a stereo algorithm will match individual pixels by taking a small patch/window around each pixel and trying to find a matching patch in the other image, usually using the sum of differences (SAD) [195], or normalised cross correlation (NCC) [196] along the epipolar lines. More recent semi-global approaches [59] produce more accurate estimates for correspondences and have been shown to be real-time using GPUs [61] but remain computationally expensive.

3 FUNDAMENTALS

3.3.2.2 Issues

Stereo requires robust and sometimes expensive matching methods to generate the disparity values accurately. This expense is becoming less of an issue as compute resource and power improves.

Scene texture is genuinely the most serious issue with a stereo camera, as it is a passive sensor, the correspondence estimates rely on a sufficiently textured set of images to generate a depth. This means regions of low texture will generally give inaccurate depths. In an effort to compensate for this, many stereo estimation rigs provide an estimate of confidence as well as the depth of each point. Given the cameras in a standard stereo rig are horizontally aligned any low-texture regions that align with the camera horizontally (table edges for example) will be difficult to correspond to get a disparity and thus depth will be extremely challenging to compute. Additionally the colour and exposure between identical cameras can still be different enough to make matching challenging.

3.3.3 Structured-Light

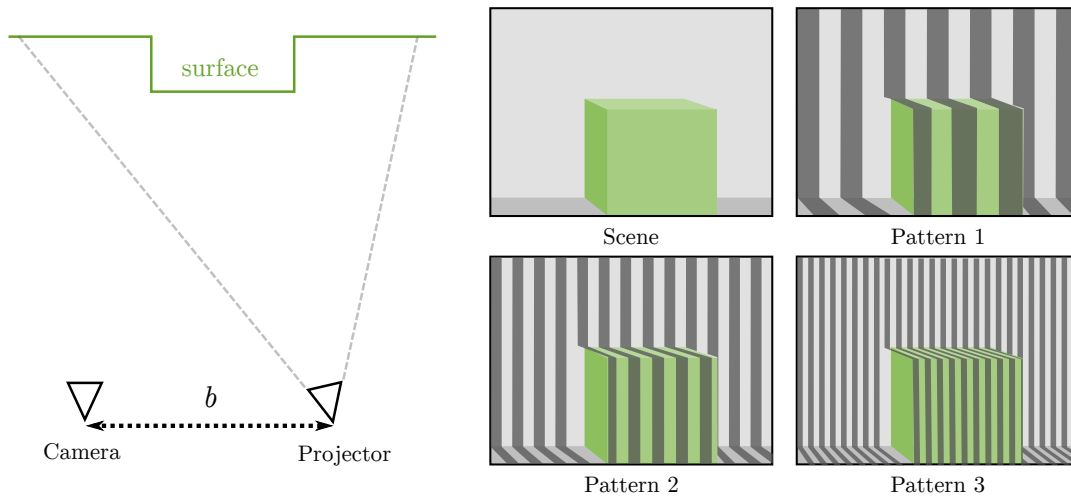


Figure 3.8: Left: The basic setup of a structured light 3D sensor. **Right:** Demonstrates how the projected pattern will predictably deform based on the surface geometry which is visible to the calibrated camera. The pattern cycle will be repeated horizontally as well to resolve some possible ambiguities.

Structured-light 3D scanners use a series of known projections on to a surface and one or more cameras to produce a high quality 3D model. Original implementations of this approach would use large expensive projectors and would project a series of patterns

3 FUNDAMENTALS

(shown in Figure 3.8) that would allow very accurate reconstructions [20] , but this approach is generally slow and expensive and requires a complex calibration procedure to ensure correct operation. More recently (since late 2000s) a variation of this approach is used in low-cost depth sensors and has been used in the Microsoft KinectTM. This type of sensor is very compact and cheap to manufacture making a perfect choice for robotics applications. Due to the reduction in cost and power consumption of hardware capable of high-speed pattern matching, these types of sensors are a very affordable and accurate choice for indoor use. Additionally as these are *active* style sensors the Signal-to-Noise Ratio (SNR) is much higher than for a *passive* stereo camera configuration in general. This has become the sensor of choice in indoor robotics we will focus most of our discussion on this form of structured-light device.

3.3.3.1 Operation

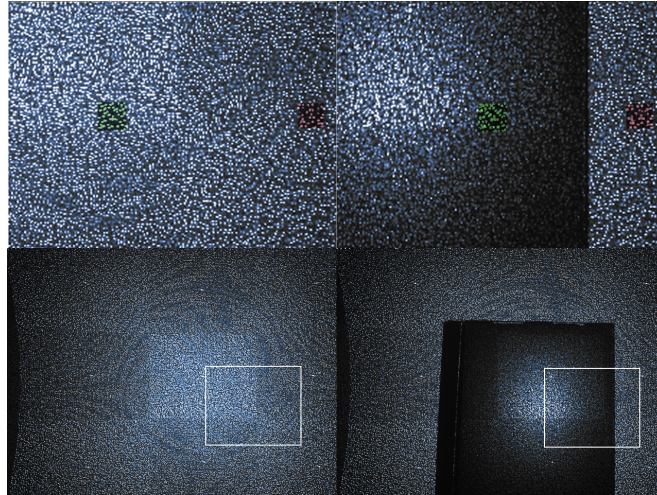


Figure 3.9: Left: The dot pattern under IR light of a flat surface, the white square is enlarged and shown on the top row. **Right:** The same IR dot pattern but with an object in front of the surface, and again the white square is enlarged on the top-row. The top row shows the matching patterns in the red and green, highlighted against the dots of the pattern and demonstrates the change in disparity caused by the object. Also of interest is the dark section on the far left of each image in the bottom row, which shows the limit of overlap between the projection and camera.

The original structured-light 3D scanners work by projecting a series of patterns onto a surface, namely alternating evenly spaces vertical black and white stripes as shown in Figure 3.8. The surface distorts the stripes based on its geometry and given the known angle and separation between the camera(s) and projector a depth can be estimated for each region of an image. By projecting a series of different thickness bands more accurate estimates can be made. This made traditional structured-light 3D sensors

3 FUNDAMENTALS

very accurate [20, 21], but also more expensive. Low-cost depth sensors, like the Kinect, produce dense estimates of depth using a similar method to the original structured-light scanners. However, they project a fixed pseudo-random infra-red (IR) dot pattern onto the scene and capture the resulting projection using an IR camera as shown in Figure 3.9.

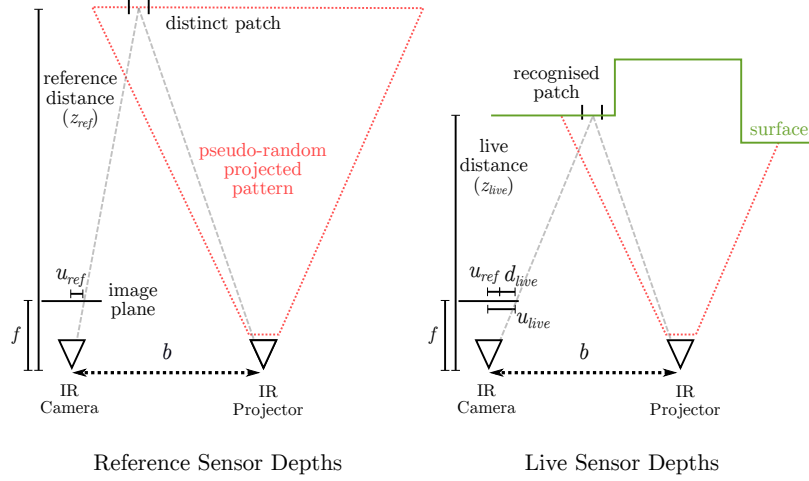


Figure 3.10: Left: A pseudo-random grid pattern is projected and its appearance is known at a reference distance, meaning reference image coordinates (u_{ref}) are stored for each pseudo-random pattern **Right:** These patterns will predictably vary in disparity based on the distance from the device to a live surface, the recorded image coordinates (u_{live}) can be compared to the reference coordinates to compute a depth.

The process used by the Kinect is summarised in Figure 3.10. A pseudo-random pattern is projected for a reference distance originally and the reference image coordinate (u_{ref}) is computed for a patch around every point. We refer only to the u coordinate because the transformation between the camera and projector is assumed to be axis aligned, along the u -axis. These reference image coordinates are stored and during runtime the live image coordinate (u_{live}) of every point is estimated. As the projector and camera pair respect the previously mentioned stereo image constraints, the depth (z_{live}) can be estimated using the equation

$$z_{live} = \frac{bf}{d'} \quad (3.27)$$

where b is the known baseline between the camera and projector, f is the focal length of the camera and d' is the actual disparity. The actual disparity is computed using the

reference image coordinate (u_{ref}) and the current image coordinate (u_{live}) using

$$\begin{aligned} d_{live} &= u_{ref} - u_{live} \\ d_{ref} &= \frac{bf}{z_{ref}} = u_{ref} \\ d' &= d_{ref} + d_{live} \end{aligned} \tag{3.28}$$

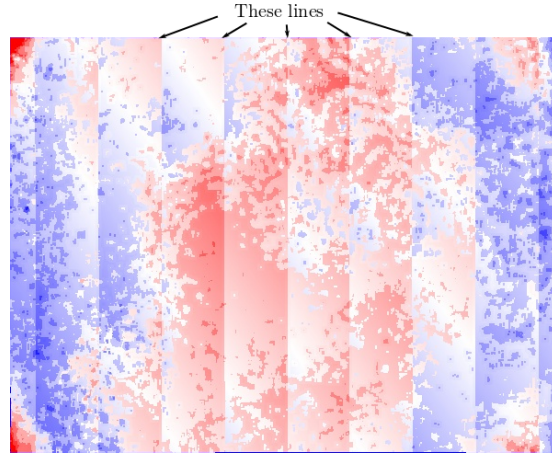


Figure 3.11: Plane deviation for light-coding device, clearly shows the vertical banding that occurs with this type of sensor

This reference image approach is chosen as it avoids many possible issues with a projector camera set up, including much of the issues related to distortion in projecting patterns. However it does create a number of possible issues discussed in Section 3.3.3.3.

3.3.3.2 Use in Robotics

A key motivation for choosing this sensor type has been its extensive use in robotics applications [14, 40, 34, 7]. This has shown the sensor has a wide range of applicability in robotics. One of the main uses is in mapping for robotics, and due to the volume of data this device is able to generate accurate dense 3D models of novel scenes in real-time [14], which can be used for a wide array of applications including augmented and virtual reality. Perhaps more applicably these 3D models can be used for accurate robotic navigation, where previous approaches used a single line-scan laser range finder which was expensive and specialised, where as a robot equipped with this a structured light depth sensor is provided with a dense depth map and can navigate much more robustly [13, 17]. Another application of these sensors to robotics has been in object detection and grasp affordance estimation. This has been shown to be more accurate and robust than a monocular only sensor [197].

3.3.3.3 Issues

As this type of sensor uses standard cameras, they suffer from lens-distortion (as discussed in Section 3.2) in both colour and depth separately. It is this issue that is primarily addressed in Chapter 4. In addition to the lens distortion the device also suffers from large vertical bands of discontinuous depth steps that appear on all depth images, which change unpredictably. An illustration of this is shown in Figure 3.11. This final type of error is difficult to calibrate for, as the bands change seemingly *randomly* over time, which doesn't seem to be dependent on the environment its imaging. A significant limitation of the device is its restriction to indoor use only. This is because ambient IR will prevent the patterns the projector shines from being discernible by the IR camera. Another possible consideration is the method the sensor uses to report depths, where the values are encoded into an 11 bit inverse depth value (proportional to disparity). This means that the closer depth values will be represented at a much higher resolution than far depths. As an illustration of this issue lets imagine the depths are calculated using the following formula

$$d = \frac{c}{b + v} \quad (3.29)$$

where c and b are some constants that allow us to tune the valid range of depths and v is the value the sensor calculates, lets assume that instead of being 11 bits v is 8 bits. Say we want to represent 0.5m to 5m with our sensor, that is $d \in [0.5, 5]$. So when $v = 0$ $d = 5$ and when $v = 255$ $d = 0.5$, then roughly $b = 28$ and $c = 142$. Now imagine that we have a value of v that might vary by ± 1 , if the value of $v = 4$ this translates to a variation in distance of approximately ± 0.15 m while if $v = 245$ then the variation in distance will be approximately ± 0.002 m nearly 100 times smaller. This demonstrates that far distance values predicted by the sensor should be trusted much less.

3.4 Review of Calculus

3.4.1 Multi-Variate Differentiation

Often functions are the product of multiple independent variables. In order to approximate how these functions change we use partial derivatives, where we compute the derivative of a function with respect to all other parameters assuming all other values remain constant. As an example lets assume the following

$$u(x, y) = 3y + (x + 1)^2 \quad , \quad v(x, y) = 7x^2 + y \quad (3.30)$$

and we know that x and y are both functions of s and t as follows

$$x(s, t) = s + t \quad , \quad y(s, t) = 2st. \quad (3.31)$$

3 FUNDAMENTALS

Then the function u can be differentiated with respect to the variables s and t via the chain rule as follows

$$\begin{aligned}\frac{\partial}{\partial s}(u(x, y)) &= \frac{\partial u}{\partial x} \frac{\partial x}{\partial s} + \frac{\partial u}{\partial y} \frac{\partial y}{\partial s} \\ &= 2 \cdot (x + 1) \cdot 1 + 3 \cdot 2t \\ &= 2(s + t + 1) + 6t \\ &= 2s + 2 + 8t\end{aligned}\tag{3.32}$$

where $\frac{\partial}{\partial s}$ indicates the partial derivative with respect to s . Which is exactly what you get if you were to first substitute $x = s + t$ and $y = 2st$ into the original Equation 3.30.

3.4.1.1 Matrices

The same logic can be applied to equations involving matrices only we need to be careful to respect ordering, as matrices often don't commute ($\mathbf{AB} \neq \mathbf{BA}$). As an example, lets say we have two $N \times N$ matrices $\mathbf{A}(\bar{a})$ and $\mathbf{B}(\bar{a})$ which are functions of some vector $\bar{a} = [a_0, a_1, \dots, a_n]$ of size n , and a point $p_0 = (p_0, p_1, \dots, p_N)$. Given the matrix equation

$$r_0 = \mathbf{A}^T \mathbf{B} \mathbf{A} p_0.\tag{3.33}$$

The derivative of r_0 with respect to the parameter a_i is given by

$$\frac{\partial r_0}{\partial a_i} = \frac{\partial \mathbf{A}^T}{\partial a_i} \mathbf{B} \mathbf{A} p_0 + \mathbf{A}^T \frac{\partial \mathbf{B}}{\partial a_i} \mathbf{A} p_0 + \mathbf{A}^T \mathbf{B} \frac{\partial \mathbf{A}}{\partial a_i} p_0,\tag{3.34}$$

where the derivative of a matrix with respect to a dependent variable is element-wise as such

$$\frac{\partial \mathbf{A}}{\partial a_i} = \begin{bmatrix} \frac{\partial A_{11}}{\partial a_i} & \frac{\partial A_{12}}{\partial a_i} & \dots & \frac{\partial A_{1N}}{\partial a_i} \\ \frac{\partial A_{21}}{\partial a_i} & \frac{\partial A_{22}}{\partial a_i} & \dots & \frac{\partial A_{2N}}{\partial a_i} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial A_{N1}}{\partial a_i} & \frac{\partial A_{N2}}{\partial a_i} & \dots & \frac{\partial A_{NN}}{\partial a_i} \end{bmatrix}.\tag{3.35}$$

3.4.2 Vector/Scalar Fields and The Jacobian Matrix (\mathbf{J})

A vector field is a field in which all points in the field have a corresponding vector. A simple example of a vector field, would be a normal map, where all surfaces in an environment have a corresponding normal vector (vector pointing orthogonal to the surface). Given a vector valued function $f : \mathbb{R}^N \mapsto \mathbb{R}^M$, which takes an input vector

3 FUNDAMENTALS

$\bar{a} \in \mathbb{R}^N$ and produces an output vector $f(\bar{a}) \in \mathbb{R}^M$. Then we can define the derivative of the vector function f at \bar{a} as

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial a_1} & \cdots & \frac{\partial f_1}{\partial a_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_M}{\partial a_1} & \cdots & \frac{\partial f_M}{\partial a_N} \end{bmatrix}, \quad (3.36)$$

where \mathbf{J} is known as the Jacobian matrix. This is often used in linear algebra for example, say we have the following matrix equation

$$e = \mathbf{A}b, \quad (3.37)$$

where e and b are points such that $a, e \in \mathbb{R}^N$, and $\mathbf{A}(\bar{a})$ is an $M \times N$ matrix and $\bar{a} = [a_1, \dots, a_J]$. If we wish to describe the motion of the point e with respect to the parameters that describe the matrix \mathbf{A} , this is described by the Jacobian,

$$\mathbf{J} = \frac{\partial e}{\partial \bar{a}} = \begin{bmatrix} \frac{\partial e_1}{\partial a_1} & \frac{\partial e_1}{\partial a_2} & \cdots & \frac{\partial e_1}{\partial a_J} \\ \frac{\partial e_2}{\partial a_1} & \frac{\partial e_2}{\partial a_2} & \cdots & \frac{\partial e_2}{\partial a_J} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_N}{\partial a_1} & \frac{\partial e_N}{\partial a_2} & \cdots & \frac{\partial e_N}{\partial a_J} \end{bmatrix}, \quad (3.38)$$

this also leads to the following relationship

$$\Delta e = \mathbf{J} \Delta \bar{a}, \quad (3.39)$$

where Δ denotes an infinitesimal change. So the Jacobian, which is a first order approximation of the derivative at the point e , gives an approximation of how the point e would change given an infinitesimal update to the parameters \bar{a} .

3.4.3 The Hessian Matrix (\mathbf{H}) and Approximate Hessian ($\mathbf{J}^T \mathbf{J}$)

The Hessian matrix (\mathbf{H}) is closely related to the Jacobian matrix (\mathbf{J}), but instead of enumerating the first partial derivatives the Hessian is the second order partial derivatives. As a simple example suppose we have the same function f from Section 3.4.2, where $f : \mathbb{R}^N \mapsto \mathbb{R}^M$ with an input vector $\bar{a} \in \mathbb{R}^N$. Then the Hessian matrix \mathbf{H} is a 3-dimensional tensor such that $\mathbf{H} \in \mathbb{R}^{N \times N \times M}$, where the Hessian of the i th output dimension would be

$$\mathbf{H}_i = \begin{bmatrix} \frac{\partial^2 f_i}{\partial a_1^2} & \frac{\partial^2 f_i}{\partial a_1 \partial a_2} & \cdots & \frac{\partial^2 f_i}{\partial a_1 \partial a_N} \\ \frac{\partial^2 f_i}{\partial a_2 \partial a_1} & \frac{\partial^2 f_i}{\partial a_2^2} & \cdots & \frac{\partial^2 f_i}{\partial a_2 \partial a_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f_i}{\partial a_N \partial a_1} & \frac{\partial^2 f_i}{\partial a_N \partial a_2} & \cdots & \frac{\partial^2 f_i}{\partial a_N^2} \end{bmatrix}. \quad (3.40)$$

3 FUNDAMENTALS

This matrix is somewhat expensive to compute, and requires the calculation of all partial second-order derivatives, and is useful in optimisation which is shown in Section 3.5.4.2. Often in order to avoid estimating such a large expensive matrix, a first order approximate for the Hessian is used. We can re-express the elements of the Hessian as

$$\mathbf{H}_{i,j,k} = \frac{\partial^2 f_k}{\partial a_i \partial a_j} = (\mathbf{J}_k^T \mathbf{J}_k)_{ij} + g = \frac{\partial f_k}{\partial a_i} \cdot \frac{\partial f_k}{\partial a_j} + g, \quad (3.41)$$

where g is the difference between the two values expressed by the second order terms of the derivative. For small values of g , which is important to optimisation, this means we can approximate the Hessian by

$$\mathbf{H}_{i,j,k} \approx (\mathbf{J}_k^T \mathbf{J}_k)_{ij}, \quad (3.42)$$

where this is true whenever the second-order terms contribute negligibly to the function. This leads to $\mathbf{J}^T \mathbf{J}$ often being referred to as the approximate Hessian matrix, particularly in an optimisation setting.

3.4.4 Taylor Series Approximations

The Taylor series provides the approximation of a function (f) around a point (x_0) given by:

$$\begin{aligned} f(x) &\approx f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 + \dots \\ &\approx \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n, \end{aligned} \quad (3.43)$$

where the superscript $'$ implies the derivative of the function. This approximation also assumes f to be infinitely differentiable around x_0 . For a polynomial the Taylor series is simply the polynomial itself. The Taylor series is often used to a significantly reduced order, as it provides a very accurate estimate of the function about a point, that is in general trivially differentiable. This is particularly useful for iterative optimisation approaches such as non-linear least-squares (Section 3.5.3). In many cases the approximation will diverge when the function is evaluated at a value of x far from x_0 . Figure 3.12 demonstrates this quite clearly for a simple function (e^x), and contrasts to the third order Taylor series approximation.

3.4.5 Quadrics

Quadrics are the 4-dimensional equivalent of a conic, where a conic can be used to express any second-order 2D *slice* of a 3D cone, a general quadric can be used to

3 FUNDAMENTALS

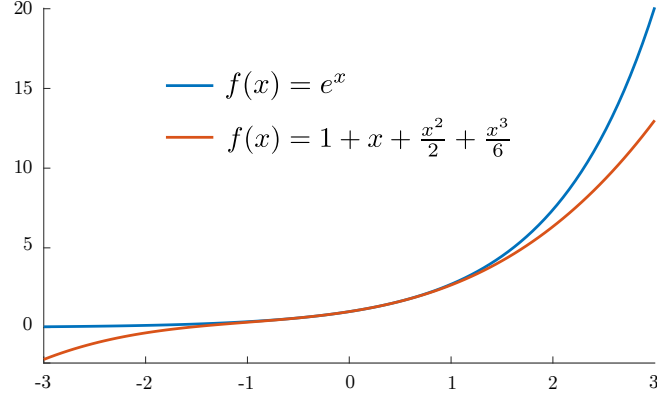


Figure 3.12: Demonstrates the amount to which the Taylor series can differ as the evaluation moves away from the centre point x_0 , in this case the function $f(x) = e^x$ and is evaluated around $x_0 = 0$.

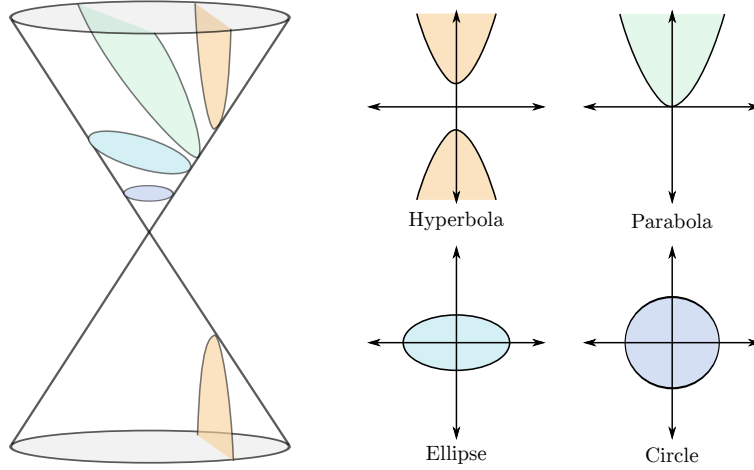


Figure 3.13: Shows the relationship between the different intersections of the cone and the resulting 2D shapes that can be made using conics. This shows the mathematical definition of a cone, which should be considered to extend infinitely far in all directions.

express any second-order 3D *slice* of a 4D surface. In order to guide this explanation we will first provide the background for conics, and extend to quadrics.

A general conic can be defined by the following matrix equation

$$p_i^T \mathbf{C} p_i = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0, \quad (3.44)$$

where $p_i \in \mathbb{R}^2$ is a homogeneous coordinate, and $\mathbf{C} \in \mathbb{R}^{3 \times 3}$ is a symmetric matrix that defines the conic. The set of points \bar{X} where $p_i \in \bar{X}$, that satisfy this equation define the

3 FUNDAMENTALS

intersection, and thus the resulting 2D dimensional line(s). Examples of some possible conics are shown in Figure 3.13. To make it clear how this relates to the equation we'll go through an example for a parabola, and show the subset of $\mathbb{R}^{3 \times 3}$ that can create parabolic intersections. If we expand out Equation 3.44 we get

$$ax^2 + bxy + cy^2 + dx + ey + f = 0, \quad (3.45)$$

which defines the equation of any second-order line in \mathbb{R}^2 , if we focus on just defining a parabola, we can set $b = c = 0$ and $e = -1$, which gives the following equation

$$y = ax^2 + dx + f, \quad (3.46)$$

which is the general form of a parabola, inserting this back into the conic matrix \mathbf{C} gives:

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a & 0 & d/2 \\ 0 & 0 & -1/2 \\ d/2 & -1/2 & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0, \quad (3.47)$$

which defines all 2D parabolas. We can now extend this logic to second-order shapes in \mathbb{R}^3 and define quadrics with the following matrix equation

$$\bar{x}^T \mathbf{Q} \bar{x} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a & b/2 & d/2 & g/2 \\ b/2 & c & e/2 & h/2 \\ d/2 & e/2 & f & i/2 \\ g/2 & h/2 & i/2 & j \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0, \quad (3.48)$$

where again $\bar{x} \in \mathbb{R}^3$ is a homogeneous coordinate, and the matrix $\mathbf{Q} \in \mathbb{R}^{4 \times 4}$ is again symmetric. As with conics, we can expand the Equation 3.48 to

$$ax^2 + cy^2 + fz^2 + bxy + dxz + eyz + gx + hy + iz + j = 0. \quad (3.49)$$

We show some examples of the possible representable shapes using quadrics in Figure 3.14. Again we can take a concrete example to illustrate how the matrix Equation 3.48 relates to the generated shapes, by expressing a sphere centered at the origin ($\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$) using a quadric. Taking Equation 3.49 as a starting point we can now give values to some of the parameters, we can set $b = d = e = g = h = i = 0$, $a = c = f = 1$ and $j = -r^2$, where r is the radius of the sphere. The equation becomes

$$x^2 + y^2 + z^2 = r^2 \quad (3.50)$$

and the corresponding quadric is:

$$\bar{x}^T \mathbf{Q} \bar{x} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -r^2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0. \quad (3.51)$$

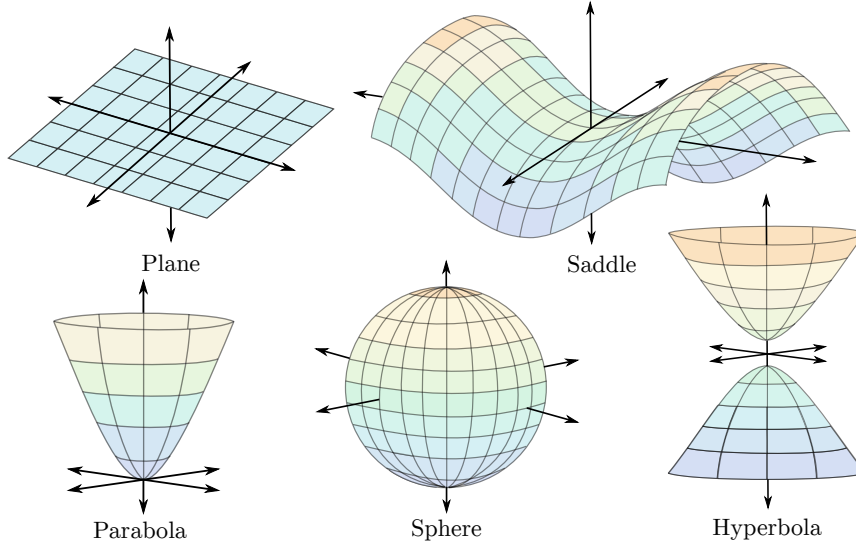


Figure 3.14: Several examples of second-order shapes that can be represented by using quadrics.

This shows that a sphere centered at the origin only requires a single parameter to be fully described, which is the radius. This method of representing shapes using quadrics, is commonly used to describe local surface regions and can even be used to describe entire objects, such as balls, cylinders and planar regions. Additionally we can restrict the number of non-zero parameters we allow to reduce the number of shapes representable by the quadric, this is applicable in a situation where you may wish to represent a small local region of a larger surface, as many surfaces can be approximated over small neighbourhoods accurately with a reduced parameterisation. An example might be to represent local regions using planes, which is in general a close approximation of any surface given the appropriate bounds.

3.5 Optimization Techniques

In this section we provide a sufficiently detailed description of the robust optimisation techniques used throughout this thesis. For a more detailed description of least-squares, weighted least-squares and iterative re-weighted least-squares please see [193, 198, 199, 200]

3.5.1 Gaussian Elimination

Gaussian elimination is step in an approach that solves a set of linear-equations, where the number of equations is sufficient to solve for the number of variables. This is the

3 FUNDAMENTALS

first step that rearranges the equations into a form that can be algorithmically solved using back-substitution (Section 3.5.2). Given a set of linear equations

$$\begin{aligned}x + y + z &= 3 \\2x + 3y + 7z &= 0 \\x + 3y - 2z &= 17\end{aligned} \quad (3.52)$$

These can be expressed by the augmented matrix

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 2 & 3 & 7 & 0 \\ 1 & 3 & -2 & 17 \end{array} \right] \begin{array}{l} \leftarrow r_1 \\ \leftarrow r_2 \\ \leftarrow r_3 \end{array}, \quad (3.53)$$

where row-operations can be performed to convert this augmented matrix into something called row-echelon form. The goal is to form a diagonal matrix on the left-hand side, which can be done by performing the following row operations

$$\begin{aligned}r'_2 &= r_2 - 2 * r_1 \\r'_3 &= r_3 - r_1 \\r''_3 &= r'_3 - 2 * r'_2\end{aligned} \quad (3.54)$$

which results in the following augmented matrix

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & 1 & 5 & -6 \\ 0 & 0 & -13 & 26 \end{array} \right], \quad (3.55)$$

The system of equations has become

$$\begin{aligned}x + y + z &= 3 \\y + 5z &= -6 \\-13z &= 26\end{aligned} \quad (3.56)$$

which is simply solvable through back-substitution as shown in Section 3.5.2.

3.5.2 Back-Substitution

Back-substitution is the process of solving a set of equations from a matrix. It can be greatly simplified by transforming the set of equations into a matrix in row-echelon form. Row-echelon form is primarily for human readability, the important aspect is the diagonal matrix on the left-hand side, which allows a solution to be calculated. As an

3 FUNDAMENTALS

example, using the system defined in Equation 3.56, we can solve for x, y and z by first solving for z using

$$\begin{aligned} -13z &= -26 \\ z &= \frac{26}{-13} = -2, \end{aligned} \quad (3.57)$$

now we can solve for y given the solution for z using

$$\begin{aligned} y + 5z &= y + 5(-2) = -6 \\ y &= -6 + 10 = 4, \end{aligned} \quad (3.58)$$

now we can finally solve for x given y and z using

$$\begin{aligned} x + y + z &= x + (4) + (-2) = 3 \\ x &= 3 - 4 + 2 = 1, \end{aligned} \quad (3.59)$$

hence $x = 1, y = 4$ and $z = -2$. If we were to define the diagonal matrix \mathbf{U} as

$$\mathbf{U} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 5 \\ 0 & 0 & -13 \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix}, \quad (3.60)$$

the vector \bar{x} as

$$\bar{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad (3.61)$$

and the solution vector \bar{y} as

$$\bar{y} = \begin{bmatrix} 3 \\ -6 \\ 26 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}. \quad (3.62)$$

The matrix equation $\mathbf{U}\bar{x} = \bar{y}$ can now be solved algorithmically in terms of the unknowns in \bar{x} using

$$\begin{aligned} x_n &= y_n / u_{nn} \\ x_i &= \left(y_i - \sum_{j=i+1}^n (u_{ij}x_j) \right) / u_{ii}. \end{aligned} \quad (3.63)$$

The above equation gives an iterative solution to the values of \bar{x} . This technique is also relevant to matrices that have been decomposed, using the Cholesky decomposition technique (Section 3.1.6) for example.

3.5.3 Least-Squares

Least-squares is a technique for fitting a model to data. This section provides a background into the basic concepts and extends up to the more complex formulations used in this thesis, while providing a motivation for some of the choices one might make using this technique.

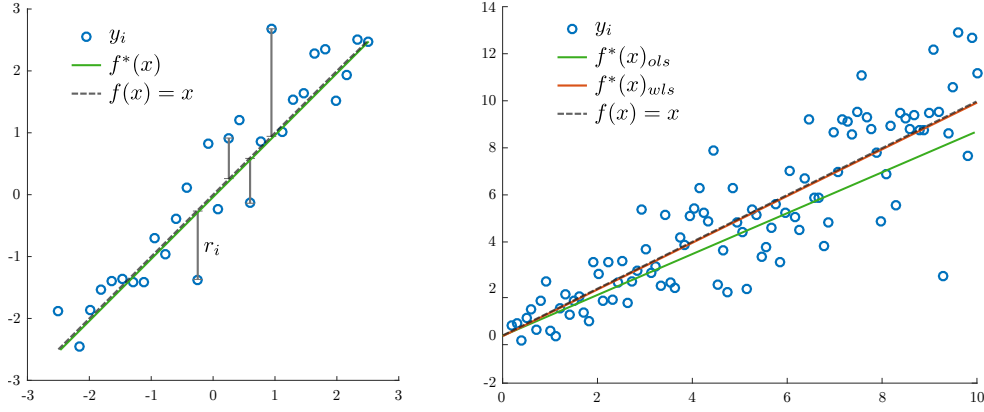


Figure 3.15: Left: A simple example of least-squares to line-fitting, in this case the samples (y_i) are normally distributed around the $y = x$ line. Additionally the residuals (r_i) are shown for several points. This corresponds to an Identity covariance matrix (discussed below). **Right:** A more complex example where the covariance matrix is no longer the identity, and the sample points (y_i) clearly vary proportionally to the independent variable x . This shows a case where weighted least-squares provides a better estimate ($f^*(x)_{wls}$) of the true relationship between the data.

3.5.3.1 Ordinary Least-Squares (OLS)

Ordinary Least-Squares (OLS) can be defined as the process of minimising the sum of the squared residuals. Given a function f that models a set of data points \bar{X} , where $(x_i, y_i) \in \bar{X}$ are data point pairs, we get the residual r_i from the following equation

$$r_i = y_i - f(x_i, \bar{\gamma}), \quad (3.64)$$

where $\bar{\gamma}$ are the estimated model parameters of the function f , x_i is the independent variable and y_i is the dependent variable. For least-squares optimisation we want to know the model parameters γ that minimise the sum of the squared residuals, described by the equation

$$\arg \min_{\gamma_i \in \bar{\gamma}} \left(\sum_{i=0}^N r_i^2 \right) = \arg \min_{\gamma_i \in \bar{\gamma}} \left(\sum_{i=0}^N (y_i - f(x_i, \gamma))^2 \right). \quad (3.65)$$

3 FUNDAMENTALS

A number of formulations exist for solving the least-squares problem, in this thesis we predominantly use the Gauss-Newton and Levenburg-Marquadt approaches. A short explanation of the background leading to these approaches, as well as the approaches themselves is provided here to aid the reader in understanding the motivation behind these choices.

Linear Least-Squares applies to the linear form of least-squares, where the function that is being fit to the data is assumed to be a linear combination of the parameter vector. This problem can be parameterised by the following linear equation

$$\mathbf{A}\bar{\gamma} = \bar{y}, \quad (3.66)$$

where

$$\mathbf{A} = \begin{bmatrix} a_{00} & \cdots & a_{0m} \\ \vdots & \ddots & \vdots \\ a_{n0} & \cdots & a_{nm} \end{bmatrix}, \quad \bar{\gamma} = \begin{bmatrix} \gamma_0 \\ \vdots \\ \gamma_m \end{bmatrix}, \text{ and } \bar{y} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix}, \quad (3.67)$$

represent a general set of linear equations. In this case an objective function (R) can be defined, which is the least-squares objective function, in terms of the model parameters ($\bar{\gamma}$)

$$R(\bar{\gamma}) = \|\bar{y} - \mathbf{A}\bar{\gamma}\|^2. \quad (3.68)$$

This function will be minimum as the gradient approaches zero. We can first expand this equation to

$$\begin{aligned} R(\gamma) &= \|\bar{y} - \mathbf{A}\bar{\gamma}\|^2 = (\bar{y} - \mathbf{A}\bar{\gamma})^T (\bar{y} - \mathbf{A}\bar{\gamma}) \\ &= \bar{y}^T \bar{y} - \bar{y}^T \mathbf{A}\bar{\gamma} - \bar{\gamma}^T \mathbf{A}^T \bar{y} + \bar{\gamma}^T \mathbf{A}^T \mathbf{A} \bar{\gamma} \\ &= \bar{y}^T \bar{y} - 2\bar{\gamma}^T \mathbf{A}^T \bar{y} + \bar{\gamma}^T \mathbf{A}^T \mathbf{A} \bar{\gamma} \quad (\text{given: } \bar{y}^T \mathbf{A}\bar{\gamma} = \bar{\gamma}^T \mathbf{A}^T \bar{y}). \end{aligned} \quad (3.69)$$

Now differentiating this with respect to $\bar{\gamma}$ and setting to zero (where the residual function will be minimised) gives

$$\begin{aligned} 0 &= -2\mathbf{A}^T \bar{y} + \bar{\gamma}^T \mathbf{A}^T \mathbf{A} + \mathbf{A}^T \mathbf{A} \bar{\gamma} \\ &= -\mathbf{A}^T \bar{y} + \mathbf{A}^T \mathbf{A} \bar{\gamma} \quad (\text{given: } \bar{\gamma}^T \mathbf{A}^T \mathbf{A} = \mathbf{A}^T \mathbf{A} \bar{\gamma}). \end{aligned} \quad (3.70)$$

A simple rearrangement will allow solving for the model parameters directly

$$(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \bar{y} = \bar{\gamma}, \quad (3.71)$$

which is the general solution to linear-least squares.

Non-Linear Least-Squares is applied when the relationship between the dependent and independent variable is known to be non-linear, i.e. not able to be expressed as a combination of linear functions. This form of least-squares often has no closed-form solution, and as such a numerical approximation is often substituted and an iterative update to the approximate solution is applied. There are several approaches to solving this type of problem and we detail a few in the following sections.

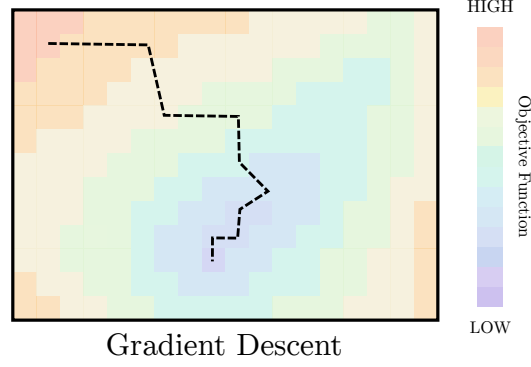


Figure 3.16: Demonstrates the characteristic zig-zagging that results from always moving in the direction of the steepest gradient, in order to minimise the objective function.

3.5.4 Iterative Optimisation Approaches

3.5.4.1 Gradient Descent

Gradient descent can be used to solve a set of equations of the form given by linear and non-linear least-squares. This is an iterative approach that attempts to minimise the objective function given by 3.68, we do this by moving in the direction of the negative gradient such that

$$\bar{\gamma}^{i+1} = \bar{\gamma}^i - \alpha^i \frac{\partial R(\bar{\gamma}^i)}{\partial \bar{\gamma}}, \quad (3.72)$$

where α^i provides an iterative reduction in step size to improve convergence. This method is guaranteed to converge on a convex objective function (bowl-shaped) if the initial conditions are in the neighbourhood of the solution. However as shown in Figure 3.16 it can take a large number of iterations due to its formulation.

3.5.4.2 Newton Method

Newton's method is a efficient approach to gradient descent. Take the least-squares residual function (R) and assume that this function is convex in the neighbourhood of the true solution ($\bar{\gamma}^*$). If we take the gradient of the function around the solution ($\nabla R(\bar{\gamma}^*)$) should be zero, we can now approximate this function using the first-order Taylor series expansion about the current estimate of the model parameters $\bar{\gamma}^i$

$$\nabla R(\bar{\gamma}^*) \approx \nabla R(\bar{\gamma}^i) + \frac{\partial \nabla R(\bar{\gamma}^i)}{\partial \bar{\gamma}} (\bar{\gamma}^* - \bar{\gamma}^i), \quad (3.73)$$

3 FUNDAMENTALS

where

$$\frac{\partial \nabla R(\bar{\gamma}_n)}{\partial \bar{\gamma}} = \mathbf{H} \quad (3.74)$$

is known as the Hessian matrix for a vector field. Noting that $\nabla R(\bar{\gamma}^*) = 0$, this gives the following recursive update condition for the model parameters $\bar{\gamma}$

$$\bar{\gamma}^{i+1} = \bar{\gamma}^i + \mathbf{H}^{-1} \nabla R(\bar{\gamma}^i). \quad (3.75)$$

This update implies a quadratic convergence rate in the neighbourhood of the solution, but it doesn't guarantee convergence, if the initial conditions are poor this method may diverge. In general this will converge much faster than gradient descent, but at an increased computational cost. Additionally and most crucially for a vector field, as is the nature of the optimisations in this thesis, the calculation of the Hessian matrix can quickly become intractable (\mathbf{H} is a 3-dimensional tensor of all second-order derivative pairs for each data-point pair).

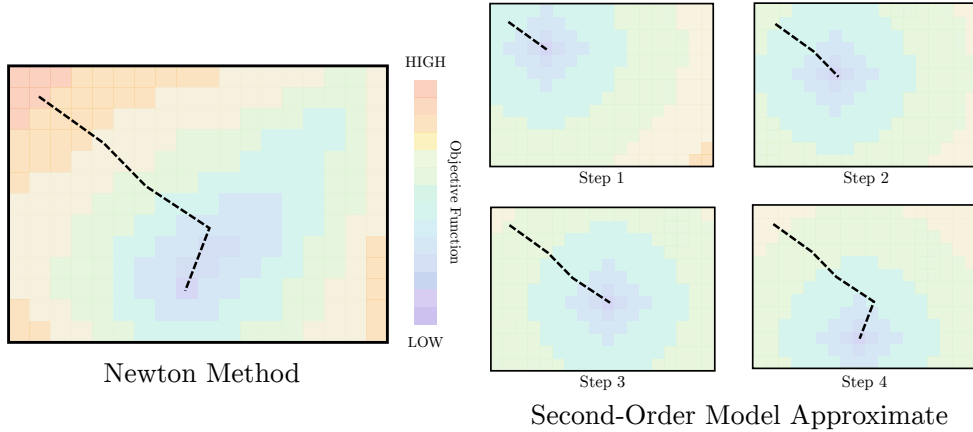


Figure 3.17: Demonstrates the motion of each step to the currently approximate minimum given a second order approximation, this results in faster convergence than gradient descent as the motion

3.5.4.3 Gauss-Newton Method

This is an efficient approximation of Newton's method, without losing much of the robustness but significantly reducing the average computation cost. Gauss-Newton is applied in a similar manner to linear least-squares but now we attempt to solve for an update to the parameters ($\Delta \bar{\gamma}$) as is the case for Newton's method. We want to make an update to $\bar{\gamma}$ that reduces the sum of the squared residuals (r_i). We can define the

3 FUNDAMENTALS

model parameters iteratively $\bar{\gamma}$ as

$$\bar{\gamma}^{j+1} = \bar{\gamma}^j + \Delta\bar{\gamma}^j, \quad (3.76)$$

where the $\bar{\gamma}^j$ gives the model parameters at iteration j . If we now take the first-order Taylor series expansion about the current model parameters $\bar{\gamma}^i$

$$\begin{aligned} f(x_i, \bar{\gamma}) &\approx f^j(x_i, \bar{\gamma}) + \frac{\partial f(x_i, \bar{\gamma})}{\partial \bar{\gamma}} (\bar{\gamma} - \bar{\gamma}^j) \\ &\approx f^j(x_i, \bar{\gamma}) + \bar{J}^T (\bar{\gamma} - \bar{\gamma}^i), \end{aligned} \quad (3.77)$$

where \bar{J} is the Jacobian, which is the vector of first order partial derivatives of the function f w.r.t. the model parameter vector $\bar{\gamma}$. Taking the formulation for the update given in Equation 3.76, and given that the data point values $y_i = f(x_i, \bar{\gamma})$, we can now express the residuals as

$$r_i = y_i - f^j(x_i, \bar{\gamma}) + \bar{J}^T \Delta\bar{\gamma}. \quad (3.78)$$

Given we want to minimise the function R given by Equation 3.65, substituting for this new approximation gives

$$R(\gamma) = \sum_i^N \|\Delta y_i + \bar{J}^T \Delta\bar{\gamma}\|^2. \quad (3.79)$$

Again we can make the assumption that this function is convex in the neighbourhood of the true solution, the current estimate is within this well of convergence, and the function f is twice differentiable. In which case the minimum to the residual function will be when the gradient of R is zero. Differentiating this with respect to $\Delta\bar{\gamma}$ and setting to zero gives

$$0 = -2 \sum_{i=0}^N \mathbf{J}_i (\Delta y_i - \mathbf{J}^T \Delta\bar{\gamma}), \quad (3.80)$$

where \mathbf{J} is the matrix of each Jacobian vector for every data point i . This can be shown to be equal to the following iterative formulation

$$\Delta\bar{\gamma} = (\mathbf{J}^T \mathbf{J})^{-1} \mathbf{J}^T \Delta\bar{y}, \quad (3.81)$$

where $\Delta\bar{y}$ is the vector of residuals from each data point. This formulation is remarkably similar to that of Newton's method in Equation 3.75. In fact Gauss-Newton approximates the Hessian \mathbf{H} as $\mathbf{J}^T \mathbf{J}$ and the gradient descent vector $\nabla R(\bar{\gamma}^i)$ as $\mathbf{J}^T \Delta\bar{y}$. This approximation assumes the second derivative of the Taylor series contributes minimally to the approximation, and in general is a valid assumption in the neighbourhood of the solution.

As with Newton's method, this approach is still vulnerable to starting outside the convergence well. A strong aspect of this approach is that it computes a much cheaper approximation of the Hessian matrix, which makes the practical implementation of this approach much simpler. However, as it is given here, no attempt is made to combat outliers in the data and as such can result in sub-optimal solutions.

3.5.4.4 Outlier rejection

A simple approach to combating outliers is to remove them from the modelled data. This can be done by estimating all the residuals, sorting them and only taking the first m values to use in the optimisation method, algebraically this looks like

$$\sum_0^m \|r_i\|^2 = \beta_0, \quad (3.82)$$

where β_0 is the sum of the first m residuals. This function has no closed-form solution, nor does it have any guarantees of convergence. Additionally in some circumstances it can prove to be quite unstable, as points move in and out of consideration. A more reliable choice is to use an maximum-likelihood estimator (M-estimator), where an objective function is provided that allows for outliers to be weighted appropriately. Using an M-estimator effectively will remove some points from consideration in a stable manner (Section 3.5.5.1).

3.5.4.5 Weighted Least-Squares (WLS)

Weighted Least-Squares (WLS) is an extension of ordinary least squares, where now the covariance matrix of the residuals with respect to each point is no longer assumed to be identity as we have effectively done so far, but still assumed to be diagonal for convenience. That is to say the error of each point is no-longer considered to be independent of the function, but still considered independent of each other. An example of the improved performance of this approach is shown in Figure 3.15, where weighting is able to reduce the influence of the outliers and the final model provides a more accurate fit to the data.

Weighting each data point is actually quite simple. Take the Gauss-Newton approach, it can be generalised by applying a weighting to each residual, using a diagonal weight matrix \mathbf{W}

$$\Delta\bar{\gamma} = (\mathbf{J}^T \mathbf{W} \mathbf{J})^{-1} \mathbf{J}^T \mathbf{W} \Delta\bar{y} \quad (3.83)$$

where

$$\mathbf{W} = \begin{bmatrix} w_0 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & w_N \end{bmatrix}. \quad (3.84)$$

In general the weights w_i are a function of the residual r_i , which modifies the optimisation function based on the choice of weighting function. There are many choices for

3 FUNDAMENTALS

weighting function, but in general a M-estimator function is used as discussed in Section 3.5.5.1.

3.5.4.6 Levenburg-Marquadt

This is a further improvement to the weighted least-squares formulation of the Gauss-Newton approach. We now extend the solution to Gauss-Newton from Equation 3.81, once more

$$\Delta\bar{\gamma} = (\mathbf{J}^T \mathbf{W} \mathbf{J} + \lambda \mathbf{I})^{-1} \mathbf{J}^T \mathbf{W} \Delta\bar{y}, \quad (3.85)$$

where λ is a parameter that increases as the system iterates in general. Another approach is to vary λ based on whether the weighted residual function reduces from one iteration to the next, increasing its value if the function would increase given the predicted update, until either the function reduces or λ reaches a pre-decided maximum value. The rational behind this is that as the value of $\lambda \mathbf{I}$ becomes much larger than $\mathbf{J}^T \mathbf{W} \mathbf{J}$, then the update approaches a small step in the direction of the gradient, which is in general a stable motion. In this way Levenburg-Marquadt attempts to balance the benefit of the fast convergence given by the Gauss-Newton approach with the inherent stability of gradient descent. However, this method still provides no guarantees on convergence, outside the neighbourhood of the solution.

3.5.5 Robust Optimisation

3.5.5.1 Maximum Likelihood Estimators (M-Estimators)

There exist a wide array of choices of weighting function, for weighted least-squares optimisation approaches. A popular technique is to use an M-estimator, in this section we describe a list of some of the most commonly used approaches. A very informative summary of this weighting methodology can be found in [201].

As discussed previously, taking the straight least-squares error for a function (Equation 3.65) can result in sub-optimal solutions in the presence of outliers. To combat this sensitivity, we replace the residual function with the function $\rho(r_i)$, and we now attempt to minimise the new function

$$\arg \min_{\bar{\gamma}} \left(\sum_{i=0}^N \rho(r_i(\bar{\gamma})) \right), \quad (3.86)$$

3 FUNDAMENTALS

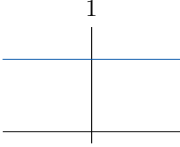
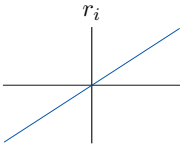
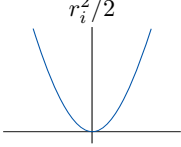
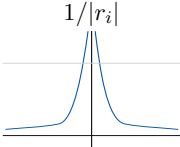
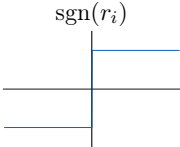
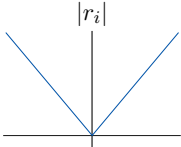
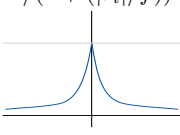
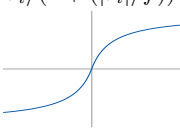
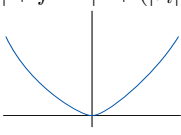
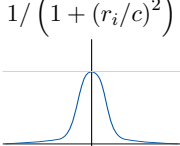
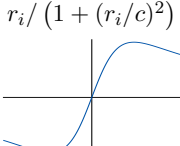
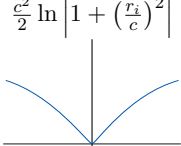
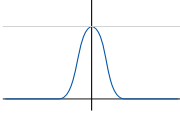
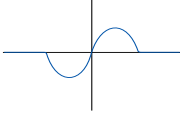
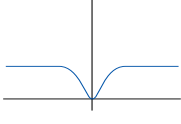
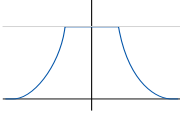
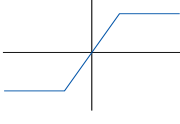
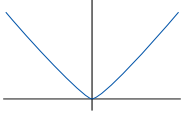
Method	$w(r_i)$	$\psi(r_i)$	$\rho(r_i)$
L_2			
L_1			
"Fair"			
Cauchy			
Tukey	$\begin{cases} \text{if } r_i \leq t \\ \text{if } r_i > t \end{cases}$ 	$\begin{cases} (t^2/6)(1 - (r_i/t)^2)^3 \\ t^2/6 \end{cases}$ 	$\begin{cases} r_i(1 - (r_i/t)^2)^2 \\ 0 \end{cases}$ 
Huber	$\begin{cases} \text{if } r_i \leq h \\ \text{if } r_i > h \end{cases}$ 	$\begin{cases} r_i^2/2 \\ h(r_i - h/2) \end{cases}$ 	$\begin{cases} 1 \\ h/ r_i \end{cases}$ 

Table 3.1: Gives a summary of commonly used M-estimator functions, including the weighting (w), influence (ψ) and objective function (ρ). Each of the functions has a graphical representation, to further illustrate the relationship to the error. The choice of the constants is importantly to ensure good operations, in practice $f = 1.3998$, $c = 2.3849$, $t = 4.6851$ and, $h = 1.345$ are used as tuning constants for their respective ρ -function.

where ρ is a positive-definite function with a unique minimum at zero (in the neighbourhood of the solution) and should increase slower than a square function away from the minimum. As with standard least-squares we can solve this function iteratively. The first step is to solve for the minimum to Equation 3.86, by finding the value of the

3 FUNDAMENTALS

parameter vector $\bar{\gamma}$ when the gradient is zero like so

$$\sum_{i=0}^N \psi(r_i) \frac{\partial r_i}{\partial \bar{\gamma}} = \sum_0^N \frac{\partial \rho(r_i)}{\partial r_i} \frac{\partial r_i}{\partial \bar{\gamma}} = 0, \quad (3.87)$$

where $\psi(r_i)$ is the so-called influence function and is equivalent to $\frac{\partial \rho(r_i)}{\partial r_i}$. Now we can define a weighting function w in terms of the influence function ψ ,

$$w(x) = \frac{\psi(x)}{x} \quad (3.88)$$

which can be substituted back into Equation 3.87

$$\sum_{i=0}^N w(r_i) r_i \frac{\partial r_i}{\partial \bar{\gamma}} = 0. \quad (3.89)$$

This is the same set of optimisation equations we would get if we attempted to minimise the iterative weighted least-squares equation

$$\arg \min_{\bar{\gamma}} \left(\sum_{i=0}^N w \left(r_i^{j-1} \right) \left(r_i^j \right)^2 \right), \quad (3.90)$$

where r_i^j is the i th error at iteration j . We include a number of weighting functions and their respective influence and ρ functions in Table 3.1. This table indicates the effect of outliers using different weighting functions, for example using the standard least-squares (L_2) highlights it's vulnerability to outliers. We can see that with a weighting function of 1 the influence of each error ($\psi(r_i)$) is proportional to the residual r_i , which means outliers assert a greater influence on the optimisation which is undesirable.

This can be solved using the Gauss-Newton or Levenburg-Marquadt as shown previously, using a diagonal weight matrix \mathbf{W} . The ρ -function is now what is being minimised and as such is the modified objective function. There is a small abuse of notation above, as $\bar{\gamma} = [\gamma_1, \gamma_2, \dots, \gamma_m]^T$ is a parameter vector, and as such each derivative would have to be evaluated separately for the above expressions.

3.5.6 Schurr-Complement Trick

The Schur complement trick applies to matrix equations of the form:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \bar{a} \\ \bar{b} \end{bmatrix} = \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix} \quad (3.91)$$

3 FUNDAMENTALS

It is an alternative method of computing the solution to this system of equations for the vectors a, b that uses a simple rearrangement of the matrix to compute its inverse. The solution to this rearrangement is given by

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} (\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} & -(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1} \\ -\mathbf{D}^{-1}\mathbf{C}(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} & \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{C}(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1} \end{bmatrix}. \quad (3.92)$$

At first this may seem like it makes things more complicated, but for certain types of matrices this can be used to greatly simplify the computation. Note that there are only two inverses to compute in this formulation, \mathbf{D}^{-1} and $(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}$.

3.5.6.1 Block-Diagonal Matrices

For a block-diagonal ($N \times N$) matrix \mathbf{A} of the form:

$$\mathbf{A} = \begin{bmatrix} \mathbf{D}_0 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{D}_N \end{bmatrix}, \quad (3.93)$$

where each \mathbf{D}_i is a square matrix of the same size, the inverse is trivial to compute and is given by

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{D}_0^{-1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_1^{-1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{D}_N^{-1} \end{bmatrix}. \quad (3.94)$$

3.5.6.2 Symmetric Matrices

If the matrix is symmetric of the form:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{bmatrix}, \quad (3.95)$$

the inverse can be simplified to

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{G} & -\mathbf{G}^{-1}\mathbf{H} \\ -\mathbf{H}^T\mathbf{G}^{-1} & \mathbf{C}^{-1} + \mathbf{H}^T\mathbf{G}\mathbf{H} \end{bmatrix}, \quad (3.96)$$

3 FUNDAMENTALS

where $\mathbf{G} = \mathbf{A} - \mathbf{B}\mathbf{C}^{-1}\mathbf{B}^T$ and $\mathbf{H} = \mathbf{B}\mathbf{C}^{-1}$. Again we have only two inverses to calculate, \mathbf{G}^{-1} and \mathbf{C}^{-1} . This gives the solution for \bar{a} and \bar{b} as

$$\begin{aligned}\bar{a} &= \mathbf{G}^{-1}(\bar{x} - \mathbf{H}\bar{y}) \text{ and} \\ \bar{b} &= \mathbf{C}^{-1}\bar{y} - \mathbf{H}^T\bar{a}.\end{aligned}\tag{3.97}$$

3.5.6.3 Use in Optimisation

In optimising least-squares problems using the Gauss-Newton approach we require the inverse of the approximate Hessian Matrix $((\mathbf{J}^T\mathbf{J})^{-1})$ which is guaranteed to be a symmetric matrix. For many optimisation problems this matrix takes the form

$$\mathbf{J}^T\mathbf{J} = \begin{bmatrix} \mathbf{A} & \cdots & \mathbf{B} & \cdots \\ \vdots & \mathbf{C}_0 & \mathbf{0} & \mathbf{0} \\ \mathbf{B}^T & \mathbf{0} & \ddots & \mathbf{0} \\ \vdots & \mathbf{0} & \mathbf{0} & \mathbf{C}_N \end{bmatrix},\tag{3.98}$$

where $\mathbf{C} \in \mathbb{R}^{M \times M}$ block-diagonal matrix and $\mathbf{A} \in \mathbb{R}^{L \times L}$ is a dense symmetric matrix, where $L \ll M$. Since \mathbf{C} is symmetric it's inverse is trivial to calculate and as shown in Equation 3.94 it is separable. This means \mathbf{G} and \mathbf{H} can be computed in parallel. More explicitly, given \mathbf{C} is block diagonal, made of a N $K \times K$ matrices, we can also express \mathbf{B} as a set of N $L \times K$ matrices as

$$\mathbf{B} = [\mathbf{B}_0 \quad \cdots \quad \mathbf{B}_N],\tag{3.99}$$

then $\mathbf{B}\mathbf{C}^{-1}$ is given by

$$\mathbf{B}\mathbf{C}^{-1} = [\mathbf{B}_0\mathbf{C}_0^{-1} \quad \cdots \quad \mathbf{B}_N\mathbf{C}_N^{-1}]\tag{3.100}$$

and $\mathbf{B}\mathbf{C}^{-1}\mathbf{B}^T$ is therefore

$$\mathbf{B}\mathbf{C}^{-1}\mathbf{B}^T = \sum_{i=0}^N \mathbf{B}_i\mathbf{C}_i^{-1}\mathbf{B}_i^T,\tag{3.101}$$

where $\mathbf{B}\mathbf{C}^{-1}\mathbf{B}^T \in \mathbb{R}^{L \times L}$, making this type of problem highly suitable to parallel programming, as all these results can be computed separately and sum reduced. This is also true for $\mathbf{H}\bar{y}$, which is also separable.

3.6 Transformation Matrices

Transformation matrices are used extensively throughout the work presented in this thesis. A transformation in this case is considered to be a matrix that represents the motion of a point or basis in N-dimensional space. We provide a brief background on groups and Lie-groups in particular to describe their general usage and properties here. For a more detailed explanation of concepts please consult [193], which provides an in-depth analysis of the concepts presented here.

3.6.1 Groups

A group in mathematics is a set of elements under an *operation*, that when applied to combine any two elements in the set into a third element, the resulting value will and the original values will satisfy the four requirements for a group, namely closure, associativity, identity and invertibility.

closure	:	$\forall a, b \in S, a \circ b = c \mid c \in S$
associativity	:	$\forall a, b, c \in S, (a \circ b) \circ c = a \circ (b \circ c)$
identity	:	$\forall a \in S, \exists I \mid a \circ I = I \circ a = a$
invertibility	:	$\forall a \in S, \exists b \mid a \circ b = b \circ a = I$

Table 3.2: The four axioms that govern whether a set should be considered a group.

These definitions are summarised in table 3.2 above. Closure means that for all a and b in the set S under the operation \circ , the result c will also be in the set S . Associativity means for all a, b and c in the set S , the operation holds associatively, meaning the order doesn't matter. Identity refers the requirement of the set that there be an I in S , such that when applied to any element under the operation \circ , it remains unchanged, that is I is the identity of the set. Invertibility requires that for all elements a in S , there must exist an element b that is the inverse of a ($b = a^{-1}$) and as such under the operation \circ will equal the identity I .

A simple and common case of a group would be Z (set of all integers) under the operation addition. It is trivial to see that it meets all the requirements for a group, but we illustrate for completeness:

closure	:	$1 + (-2) = -1 \mid -2, -1, 1 \in Z$
associativity	:	$(1 + 2) + 3 = 1 + (2 + 3)$
identity	:	$7 + 0 = 7 : I = 0$
invertibility	:	$6 + (-6) = 0 = I$

3 FUNDAMENTALS

3.6.2 Matrix Exponentiation

A matrix exponential is the matrix version of an ordinary exponential and is defined by the following power series

$$e^{\mathbf{A}} = I + \mathbf{A} + \frac{1}{2}\mathbf{A}^2 + \frac{1}{6}\mathbf{A}^3 + \dots = \sum_{k=0}^{\infty} \frac{1}{k!}\mathbf{A}^k, \quad (3.102)$$

where A is a square $N \times N$ matrix. This is well defined for all values of A , both real and complex. This definition of matrix exponentials leads to the following properties:

$$\begin{aligned} e^{\mathbf{0}} &= \mathbf{I} & e^{\mathbf{YAY}^{-1}} &= \mathbf{Y}e^{\mathbf{A}}\mathbf{Y}^{-1} \\ e^{\mathbf{A}^T} &= (e^{\mathbf{A}})^T & e^{a\mathbf{A}}e^{b\mathbf{A}} &= e^{(a+b)\mathbf{A}} \\ e^{\mathbf{A}^*} &= (e^{\mathbf{A}})^* & e^{\mathbf{A}}e^{-\mathbf{A}} &= \mathbf{I} \end{aligned}$$

$\mathbf{0}$ is an $N \times N$ zero matrix, \mathbf{A}^T denotes the transpose of \mathbf{A} and \mathbf{A}^* denotes the conjugate transpose of \mathbf{A} , which is relevant to complex matrices. The matrix \mathbf{Y} is assumed to be invertible and the same size as \mathbf{A} . The final property ($e^{\mathbf{A}}e^{-\mathbf{A}} = \mathbf{I}$) also requires that \mathbf{A} be invertible. These properties also imply that if \mathbf{A} is symmetric then $e^{\mathbf{A}}$ is also symmetric. Additionally matrix exponentiation allows the mapping from one group to another, this is important as it implies that the exponential of a group forms an isometric mapping to another group.

3.6.3 Lie Groups, Lie Algebras and The Tangent Space

This section is intended to explain the background of how we have expressed transformations (rotations and translations) of points, cameras and frames in this thesis.

3.6.3.1 Lie Groups and Lie Algebras

A Lie group is a group \mathbf{G} such that the elements form a smooth differentiable manifold over the range of the group. The elements of this group can be used to represent linear transformations. As with groups these qualities are defined under some operation, in this case we consider groups of matrices under multiplication. A key motivation for using Lie-groups is the ability to map smoothly any point in a complex group, to a local linearized version of the group called the Lie-algebra \mathfrak{g} , which is valid in the neighbourhood of identity of the group. That is using a Lie bracket a Lie algebra can be formed that is equivalent to the tangent space about the identity. The Lie bracket is defined by

$$[\mathbf{A}, \mathbf{B}] = \mathbf{AB} - \mathbf{BA} \text{ where } \mathbf{A} \in \mathfrak{g}, \mathbf{B} \in \mathfrak{g}, [\mathbf{A}, \mathbf{B}] \in \mathfrak{g}. \quad (3.103)$$

3 FUNDAMENTALS

In the case of commutative matrices clearly $[\mathbf{A}, \mathbf{B}] = 0$, but for non-commutative matrices the bracket measures the degree to which the matrices violate the commutativity law.

This thesis only considers a subset of Lie-groups, that are useful in geometry. They are

- Special Orthogonal Group 3 ($\mathbf{SO}(3)$) - Represents all magnitude preserving rotation's possible in linear algebra
- Special Euclidean Group 3 ($\mathbf{SE}(3)$) - All transformation between \mathbb{R}^3 coordinate frames that preserve distances, and relationships between points

In both cases the 3 refers to the fact that these groups operate in \mathbb{R}^3 . These groups and their respective Lie algebras (\mathfrak{so}_3 and \mathfrak{se}_3) form the basic transformations used for many sections of this thesis. The matrix exponential of an element in the group's Lie algebra takes it to back of the original group, and a matrix logarithm takes it back to its Lie algebra. The elements of the Lie algebras (\mathfrak{so}_3 and \mathfrak{se}_3) can be expressed by a sum of the products of infinitesimal generator matrices, which is explained below.

3.6.4 Special Orthogonal Groups ($\mathbf{SO}(N)$)

A set of Lie-groups known as the special orthogonal groups, can be represented in linear algebra as a set of matrices. For the purpose of this thesis, these groups can be considered to be purely rotation matrices. That is a group in $\mathbf{SO}(N)$ will rotate points in \mathbb{R}^N about the origin. The matrices that form this set of groups must be ortho-normal, that is the matrix must have unit magnitude columns and each column must be orthogonal to all others. They have also satisfy the condition $\mathbf{A}^T \mathbf{A} = \mathbf{I}$, that the transpose of a matrix in $\mathbf{SO}(N)$ is its inverse. Which implies they must also be skew-symmetric, which means this also applies to the rows of the matrix.

3.6.4.1 Rotation in Two Dimensions ($\mathbf{SO}(2)$)

This is the most basic of the special orthogonal groups. This group takes the form

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \quad (3.104)$$

where θ defines the magnitude of the rotation. This can be thought of as rotating about a third orthogonal dimension (z) as shown in Figure 3.18 below.

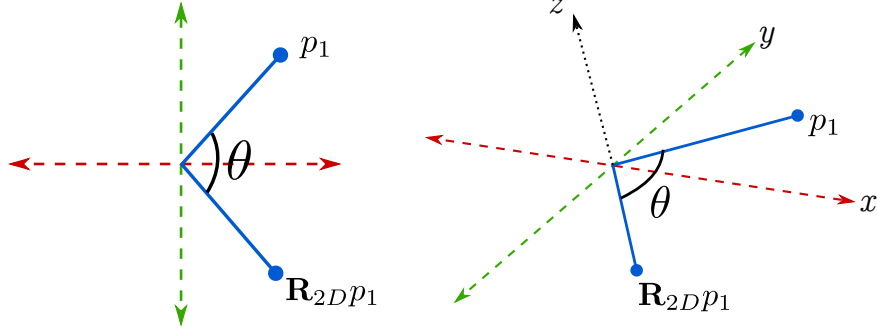


Figure 3.18: **Left:** the rotation of a point (p_i) by a $\mathbf{SO}(2)$ matrix (\mathbf{R}_{2D}). **Right:** Demonstrates that a rotation can actually be thought of as occurring about a separate orthogonal dimension (z in this case).

The Lie group $\mathbf{SO}(2)$ has a Lie algebra \mathfrak{so}_2 defined by the infinitesimal generator matrices that form a basis of all 2D transformations. This is the basis of the tangent space of \mathbf{SO}_2 , which can be shown to be given by the derivative of the rotation with respect to θ at the point $\theta = 0$

$$\mathbf{G} = \frac{\partial \mathbf{R}(\theta)}{\partial \theta} \bigg|_{\theta=0} = \begin{bmatrix} -\sin(\theta) & -\cos(\theta) \\ \cos(\theta) & -\sin(\theta) \end{bmatrix} \bigg|_{\theta=0} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \quad (3.105)$$

where \mathbf{G} is the basis of \mathfrak{so}_2 . We can now show that $e^{\mathfrak{so}_2} \mapsto \mathbf{SO}(2)$ is true for points close to the identity, by taking the exponential of the generator \mathbf{G} multiplied by a constant α ,

$$\begin{aligned} e^{\alpha \mathbf{G}} &= \mathbf{I} + \alpha \mathbf{G} + \frac{1}{2}(\alpha \mathbf{G})^2 + \frac{1}{6}(\alpha \mathbf{G})^3 + \dots \\ &= \begin{bmatrix} -\sin(\alpha) & -\cos(\alpha) \\ \cos(\alpha) & -\sin(\alpha) \end{bmatrix}, \end{aligned} \quad (3.106)$$

which is the original rotation matrix expressed in terms of a rotation by α radians.

3.6.4.2 Rotation in Three Dimensions ($\mathbf{SO}(3)$)

The Lie group $\mathbf{SO}(3)$ defines all 3×3 matrices that respect the orthogonality condition

$$\mathbf{A}^T \mathbf{A} = \mathbf{I}, \quad (3.107)$$

where $A \in \mathbf{SO}(3)$. As mentioned previously we can define the tangent space for $\mathbf{SO}(3)$ as \mathfrak{so}_3 where the elements of this tangent space (Lie algebra) can thought of as the set of infinitesimal generator rotations about a direction vector. A simple approach to find these generators is to take the derivative of the rotation matrices around each of the

3 FUNDAMENTALS

directions (x, y, z) , this commonly used basis for \mathfrak{so}_3 is given by the following generator matrices

$$\mathbf{G}_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \mathbf{G}_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \mathbf{G}_2 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \quad (3.108)$$

This can be related to $\mathbf{SO}(3)$ through a matrix exponentiation as follows,

$$\mathbf{R}(\bar{\gamma}) = e^{\sum_{i=0}^2 \gamma_i \mathbf{G}_i}, \quad (3.109)$$

where $\bar{\gamma} = [\gamma_x, \gamma_y, \gamma_z]^T$ is a vector of rotations in radians about the x, y and z axis respectively.

3.6.5 Special Euclidean Groups ($\mathbf{SE}(N)$)

The special Euclidean groups extend the special orthogonal groups to allow any rigid transformation of a coordinate, vector or shape. The most relevant Lie group to this thesis is $\mathbf{SE}(3)$, which similarly has a tangent space \mathfrak{se}_3 .

3.6.5.1 General 6DOF Transformations ($\mathbf{SE}(3)$)

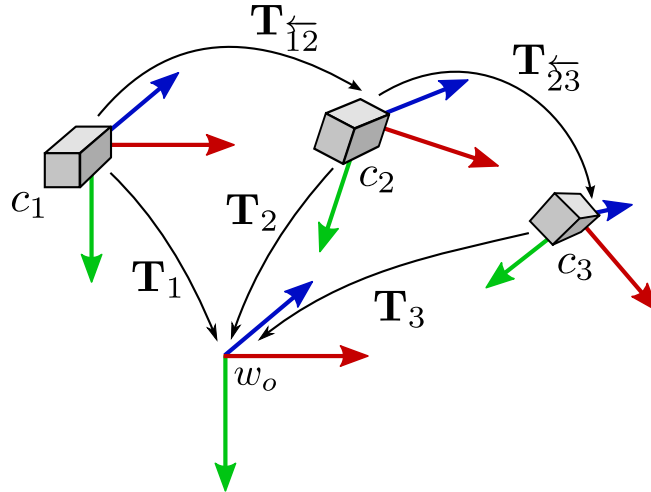


Figure 3.19: Demonstrates the transformation between coordinate frames using $\mathbf{SE}(3)$ transformation matrices \mathbf{T} . The top transformations are relative and show the transformation between cameras c_i and c_j as $\mathbf{T}_{ij}^{\leftarrow}$. The transformations coming from the origin (w_o) are absolute and show the transformation from the origin to a common frame. In practice the world origin will be chosen to be the position of the first camera.

3 FUNDAMENTALS

This is the set of *rigid* transformation matrices for coordinates and frames in \mathbb{R}^3 . We give an example set of transformations in Figure 3.19 above. This is perhaps the most convenient and common method of expressing transformations in geometry. As with $\mathbf{SO}(3)$, $\mathbf{SE}(3)$ has an easily expressible tangent space. However $\mathbf{SE}(3)$ doesn't require that $\mathbf{A}^T \mathbf{A} = \mathbf{I}$, in fact in general this is not satisfied except for the elements of $\mathbf{SE}(3)$ that are pure rotations. The matrices of $\mathbf{SE}(3)$ can be expressed as

$$\mathbf{T} = \begin{bmatrix} \mathbf{R}(\gamma_x, \gamma_y, \gamma_z) & \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (3.110)$$

where $\mathbf{R} \in \mathbf{SO}(3)$ is a rotation matrix and $\bar{t} = [t_x, t_y, t_z]^T$ is a translation vector. This means that the entirety of $\mathbf{SO}(3)$ is contained in $\mathbf{SE}(3)$. This set of matrices can naturally be expressed using a set of infinitesimal generator matrices again, where a common choice of generators is

$$\begin{aligned} \mathbf{G}_0 &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{G}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{G}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{G}_3 &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{G}_4 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{G}_5 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \quad (3.111)$$

where the generators can be used to generate the group using matrix exponentiation

$$\mathbf{T}(\bar{\gamma}) = e^{\sum_{i=0}^5 \gamma_i \mathbf{G}_i}. \quad (3.112)$$

In this case $\bar{\gamma} = [t_x, t_y, t_z, \theta_x, \theta_y, \theta_z]^T$ is a vector of transformation parameters expressing translation (t_i) and rotation (θ_i) about the i axis. These express the 6 Degrees of Freedom (6DoF) that are possible for any rigid transformation.

3.6.6 Differentiating Transformation Matrices

As shown previously we can express any rigid transformation as a matrix exponential, using its tangent space. Assuming the resulting transformation is small, we can say this approximation is valid in the vicinity of the identity. This means our derivative can be approximated from the matrix exponential

$$\frac{\partial}{\partial \gamma_i} e^{\sum_{i=0}^5 \gamma_i \mathbf{G}_i} = \mathbf{G}_i e^{\sum_{i=0}^5 \gamma_i \mathbf{G}_i} = \mathbf{G}_i \mathbf{T}_0 \quad (3.113)$$

where $\mathbf{T}_0 \in \mathbf{SE}(3)$ is the current approximation of the transformation matrix. This means our gradients are very simple to calculate in terms of the motion parameters (γ_i)

3.7 Iterative Closest Point (ICP)

We include this section as it is an integral part of each of at least 2 pieces of work presented in this thesis (see Chapters 4 and 6), making it important that the reader understand the basis behind the algorithm to fully appreciate the goals and contributions contained in those works.

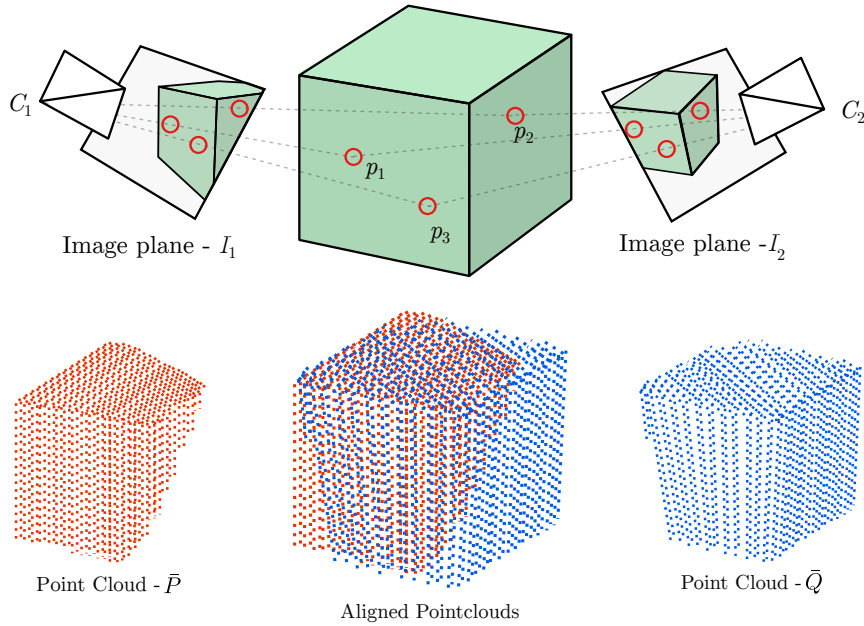


Figure 3.20: Shows the alignment of two over-lapping point clouds P and Q . Also demonstrates how points in the world can be used to relate multiple cameras, by respecting the geometry of the scene. Best viewed in colour.

The goal of Iterative Closest Point (ICP) is to align multiple overlapping depth scans accurately and efficiently. The idea behind this algorithm is to iteratively reduce the distance between corresponding/closest points in multiple overlapping point-clouds. The steps in ICP can be summarised as in [120], as the following:

1. **Selection** of the points p_i and q_i from surfaces P and Q
2. **Correspondence estimation** for all points $p_i \in P$
3. **Weighting** the estimated correspondences
4. **Outlier removal** of the invalid points from the correspondence set
5. Choosing an **error metric**
6. **Minimising** the chosen error metric

3 FUNDAMENTALS

3.7.1 Selection

There are many approaches to point selection, and this choice was particularly relevant when limited compute resources are available.

3.7.1.1 RANdom SAMpling Consensus (RANSAC)

RANSAC is a very popular strategy for selection [202]. The basic principle of RANSAC is to choose a random selection of n points, from the full set of N points, and compute an update to the system, in the case of ICP it would be a transformation. Using this estimated transformation, a consensus score is calculated that measures the number of inliers, or points that agree with the proposed transformation. This is repeated for L iterations, or until a predefined consensus score is reached. The choice of L is given by

$$L = \frac{\log(0.01)}{\log(1 - \omega^n)}, \quad (3.114)$$

where ω is the fraction of inliers in the set. This choice gives you a 99% chance of selecting at least 1 set of points with all inliers. The value n is usually kept small to keep L small as well, reducing the overall work. The minimum choice of n in the case of ICP is 3, as this is enough to fully constrain a 6DOF transformation (provided that are not all co-planar or form a straight line). In general it's better to choose more than the minimum and use a least-squares approach as it is more robust to noise in the inlier set.

3.7.1.2 Dense Sampling

Since the introduction of parallel or GPGPU computing the constraints on the number of points considered have been loosened dramatically. Now a valid strategy is to use every point in the optimisation. The inlier rates are now particularly relevant to weighting and outlier removal. In this thesis we focus on dense selection techniques and parallel processing. In general robust dense optimisation results in a more accurate and solution [14, 14, 17], than statistical sampling.

3.7.2 Correspondence Estimation

The most relevant method of correspondence estimation to this thesis is to project, and is primarily because of the form of data used. The range scans produced by low cost

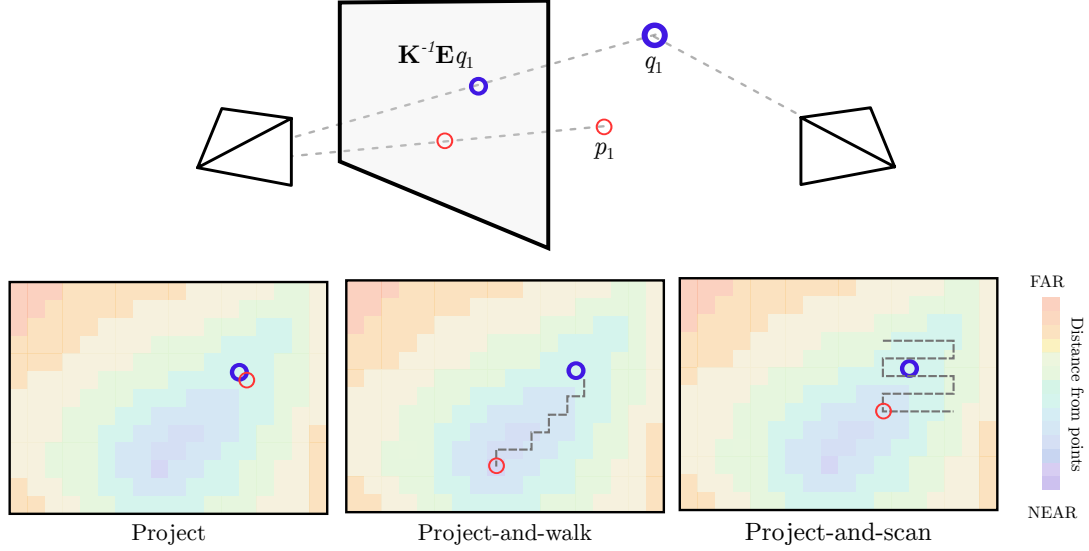


Figure 3.21: Demonstrates the standard approach to projected the transformed point on the image plane of the other depth scan. We show three approaches to matching correspondences between cameras, based on a standard camera. The colour of the image is intended to show the distance to the transformed coordinates. **Project:** the correspondence is simply the closest image coordinate the transformed point (q_i) projects to. **Project-and-walk:** after projecting the point a "greedy" search can be performed in image space to find the closest point according to some error metric. **Project-and-scan:** after projecting the point a fixed size patch is searched for the closest point.

depth scans, can be used to efficiently produce organised point clouds, which can be searched more efficiently in screen space, that is in the original image coordinate space. This also means its convenient to represent the transformations in terms of the camera centres, as opposed to model centres, or the first moment for example.

Several different approaches to matching correspondences between cameras are shown in Figure 3.21. The several methods each have different advantages and disadvantages. Straight **projection** takes the transformed coordinate to image space in the other camera and takes the closest point. This is the fastest method, but the least robust to large viewpoint changes. **Project-and-walk** projects to the other camera and searches for the closest point, generally in a greedy fashion. Beginning at the current closet point and taking the neighbouring point that reduces the distance the greatest amount, repeating until no neighbouring points are closer and taking that as the correspondence. This is a good strategy to balance the additional computational cost of searching to increase the convergence radius, but can result in a sub-optimal choice which as a weakness of the greedy algorithm. **Project-and-scan** projects to the other camera and searches a fixed radius around the projected image coordinate taking the closest as the correspondence. This has a fixed complexity and increases the convergence radius, but with a reduced

risk of being trapped in a local minima.

3.7.3 Weighting & Outlier Removal

As mentioned in Section 3.7.1.2, the weighting of points using dense selection is essential for the overall robustness of the system. The choice of M-estimator can be made experimentally, in this thesis a common choice for weighting function was the Cauchy function due to its computational simplicity and relatively good performance,

$$w_i = \frac{1}{1 + (\epsilon_i^2/c^2)}, \quad (3.115)$$

where c is a constant chosen either *a priori* or computed over a number of trials to match the expected error of the system. A more detailed description of robust weighting and M-estimators can be found in Section 3.5.5.1.

3.7.4 Error Metric

As the name suggests ICP is attempting to align the points that are closest, in an iterative fashion. The question therefore becomes, what does closest mean? There are several popular variants of ICP, including Point-to-Point, Point-to-Plane and Point-to-Surface. The names of the variants indicate the distance metric used in the optimisation, an example of each is shown in Figure 3.22. This demonstrates the relationship between the error metric and points of the surface. These distance metrics are used to define the closest points, and as such correspondences.

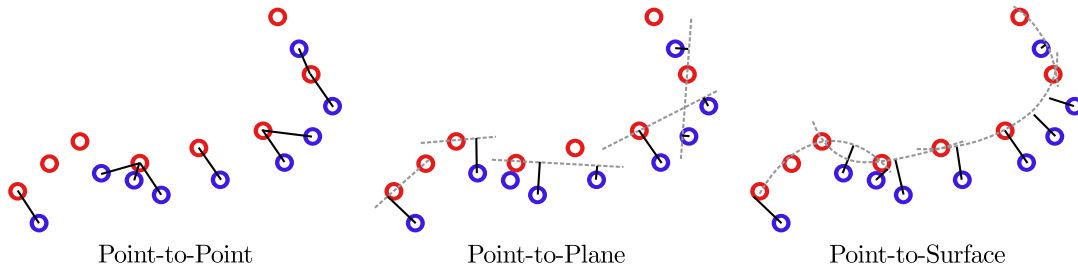


Figure 3.22: **Point-to-Point** computes correspondences based on the point to point Euclidean distance, with closest points connected. **Point-to-Plane** error is measured w.r.t. a locally fit plane (grey dashed lines) for each target point (red), the black lines show the distance to the closest plane for each point. **Point-to-Surface** is a generalisation of Point-to-Plane where the surface representation is extended beyond planes. All diagrams are shown in 2D, making the planes technically lines, but this diagram is intended to be purely illustrative.

3 FUNDAMENTALS

The error metrics can be computed using the following

$$d_{pt-pt}(p_i, q_i) = \|q_i - p_i\|_2 \quad (3.116)$$

$$d_{pt-pl}(p_i, q_i) = \|\hat{n}(p_i) \cdot (q_i - p_i)\|_2 \quad (3.117)$$

$$d_{pt-s}(p_i, q_i) = \|S_p(q_i) \cdot (q_i - S_p(p_i))\|_2 \quad (3.118)$$

where $d_X(p, q)$ is the distance between points \bar{p}, \bar{q} defined by the metric X , $\hat{n}(p_i)$ is the unit normal at the point p_i of the surface P , $S_p(p)$ is the point on the surface approximation in the neighbourhood of p_i closest to q_i , and $S_p(q_i)$ is the closest point of the surface S_p that the point q_i . The point-to-plane metric is the most popular approach in real-time ICP based tracking approaches [14, 14, 17], and is the variant used to perform camera calibration in Chapter 4 of this thesis. The point-to-surface metric is dependent on surface representation and as such is a more nebulous definition. In Chapter 6, we present a variant of this based on extension of the work in Chapter 5. This work performs a point-to-surface alignment while jointly optimising to improve the quality of the surface representation.

3.7.5 Error Minimisation

In mathematical terms ICP is trying to compute the pose alignment that minimises the sum of the distances between all points given by the following error function

$$\epsilon(\bar{\gamma}) = \sum_i^N d(p_i, \mathbf{E}(\bar{\gamma})q_i), \quad (3.119)$$

where $p_i, q_i \in \mathbb{R}^3$ are points in surface P and Q respectively, $E(\bar{\gamma})$ is the transformation required to move the points surface Q , towards P , $\bar{\gamma} = \{\theta_x, \theta_y, \theta_z, t_x, t_y, t_z\}$ is the set of motion parameters that describe the 6 degrees of freedom (6DOF) required to align the surfaces, and d is a chosen distance metric. As described in Section 3.5 we can optimise for the relative pose alignment using a number of standard techniques. This requires the differentiation of the loss function with respect to the motion parameters, and this is described in Section 3.6.

3.7.6 Wide Baseline Problem

The ICP algorithm is very powerful and can provide very accurate surface-to-surface alignments in a short time. However, some aspects of the alignment problem can cause it to fail. The so-called wide baseline problem is a key issue for ICP, which is caused when attempting to align two scans that have a poor initial alignment, possible due to a reduced surface overlap or challenging initial conditions. An example of this is shown in Figure 3.23, where the scene contains sufficiently unique geometry to align well, but the algorithm can not converge by merely searching for the closest point and iterating.

3 FUNDAMENTALS

The final alignment produced, which is shown, is a local minimum which is also caused by the robust weighting function that penalises distant points and causes the alignment to become stuck in this way.

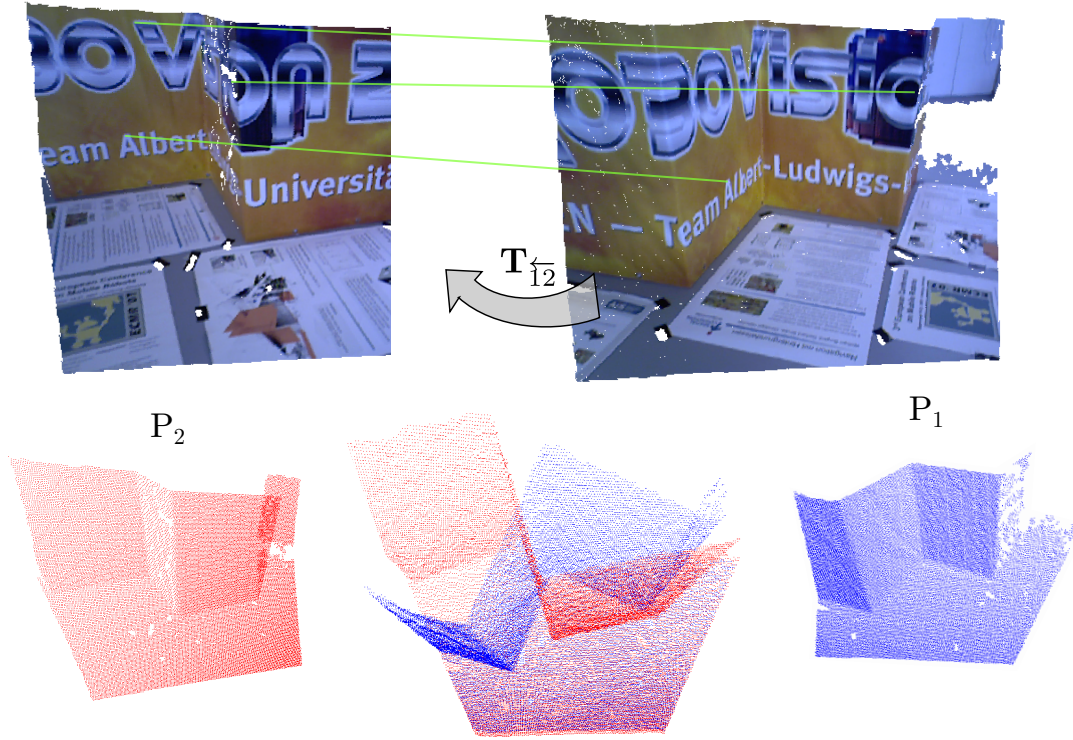


Figure 3.23: Top: Shows the coloured point cloud for two different views of the same scene and the rough transformation between the frames (T_{12}), lines in green show correspondences between the two clouds for clarity. **Bottom:** Shows an attempt to align the point-clouds using the initial assumption of an identity transformation, and shows that the algorithm simply cannot cope with this relatively simple transformation and fails to converge.

Insufficient overlap and a bad initial alignment can easily lead to a sub-optimal solution. This problem is difficult to solve using a purely iterative approach as this simply tries to move the solution in a way that reduces the current error, and geometric alignment is prone to multiple local minima. One approach to solve this, is to use associated colour information to initially provide an estimated alignment and then iterate from there, using a feature based approach. This approach is very effective in highly textured images, and point clouds that actually have associated colour, but can be made difficult when there are repeating or low textured regions such as walls or furniture. This approach is also contingent on lighting, which the geometric alignment is not. Another common approach is to find unique geometric features, such as large planar regions, or feature vectors that examine the changes in a neighbourhood of a point. Extracting geometric feature vectors can be a quite effective way to identify correspondences across

point-clouds. Many of these features use a combination of normal and local surface curvature information, which means accurate estimates of surface normals and surface curvature (Section 3.8) can be used to improve geometric based alignment approaches [203]. One of the contributions of this thesis is to improve the accuracy of the surface curvature and surface normal estimates as well as the relative alignment of surfaces using quadrics (see Chapters 5 and 6).

3.8 Surface Curvature

Surface curvature is a measure of the rate of change of a surface if you were to walk along it. Principal surface curvature measures the rate of change in the principal directions ($\bar{\kappa}_1$ and $\bar{\kappa}_2$), which are aligned to the direction maximum and minimum curvature respectively. Since curvature can be negative or positive the direction of minimum curvature could have a higher magnitude than the maximum. This is demonstrated in Figure 3.24, for the case of the inverted cylinder, where the maximum curvature is zero but the minimum is negative.

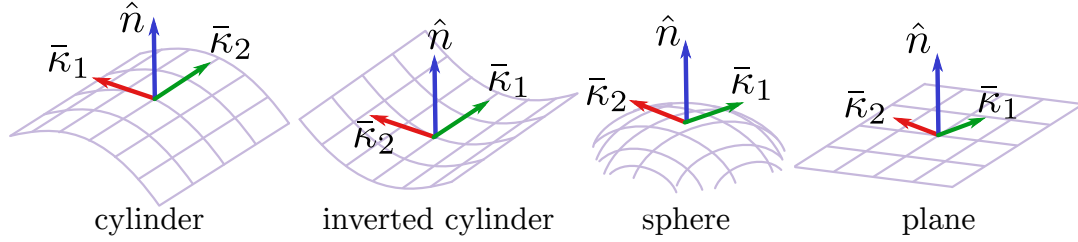


Figure 3.24: Examples of principal curvature directions for basic shapes. **Cylinder:** has the principal curvature direction $\bar{\kappa}_1$ aligned with the direction of greatest positive curvature. **Inverted Cylinder:** An inverse cylinder, has the principal curvature direction $\bar{\kappa}_1$ aligned with the zero-curvature direction, as this is the most positive. **Sphere/Plane:** the alignment of the principal curvature directions is arbitrary, as all directions have the same curvature for a sphere or plane.

3.8.1 Relation to the Second Fundamental Form (II)

The curvature of a two dimensional surface embedded in three dimensions is defined by the rate at which a unit surface normal changes with respect to motion across the surface. For a twice differentiable surface M , let \hat{u} and \hat{v} be orthogonal unit vectors in the tangent plane to the surface at a point $p \in \mathbb{R}^3$. The surface normal is then given by $\hat{n} = \hat{u} \wedge \hat{v}$. \hat{u} , \hat{v} and \hat{n} form an orthonormal basis about p . The surface in the neighbourhood of p can then be defined as

$$p = x\hat{u} + y\hat{v} + z\hat{n},$$

3 FUNDAMENTALS

where $z = z(x, y)$, shown in Figure 3.25. The surface normal can then also be defined in the neighbourhood of p as $\hat{n}(x, y)$.

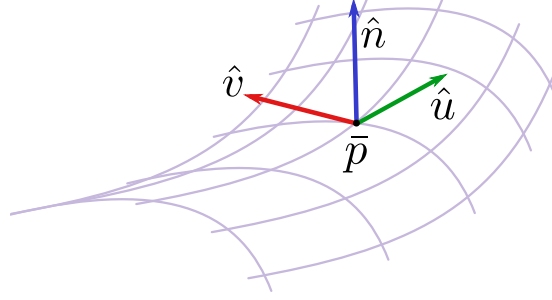


Figure 3.25: The surface M in terms of its basis vectors \hat{u} and \hat{v} .

The curvature of the surface can now be defined in terms of the second fundamental form \mathbf{II} . This is a symmetric two-form matrix which can be represented by a 2×2 matrix in the surface coordinate frame defined by our original basis vectors \hat{u} and \hat{v} as

$$\mathbf{II} = \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix}, \quad (3.120)$$

where A , B and C can be defined in terms of derivatives of the surface normal given by

$$\begin{aligned} A &= -\frac{\partial \hat{n}(x, y)}{\partial x} \cdot \hat{u}, \\ B &= -\frac{\partial \hat{n}(x, y)}{\partial x} \cdot \hat{v} = -\frac{\partial \hat{n}(x, y)}{\partial y} \cdot \hat{u}, \text{ and} \\ C &= -\frac{\partial \hat{n}(x, y)}{\partial y} \cdot \hat{v}. \end{aligned}$$

This is the usual method used to compute the curvature. However A , B and C can be equivalently defined as

$$A = \frac{\partial^2 z(x, y)}{\partial x^2}, \quad B = \frac{\partial^2 z(x, y)}{\partial x \partial y}, \text{ and } C = \frac{\partial^2 z(x, y)}{\partial y^2} \quad (3.121)$$

Using this definition, it is easy to construct a parabolic surface that has this curvature at the origin as

$$z = \frac{A}{2}x^2 + Bxy + \frac{C}{2}y^2. \quad (3.122)$$

Computing the Eigenvalues of the second fundamental form \mathbf{II} from Equation 3.120 gives the principal curvature values κ_1 and κ_2 . Diagonalising this matrix with a rotation

is equivalent to aligning the surface to the principal directions ($\bar{\kappa}_1$ and $\bar{\kappa}_2$), which is given by

$$\mathbf{II} = \begin{bmatrix} du' & dv' \end{bmatrix} \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} \begin{bmatrix} du' \\ dv' \end{bmatrix}. \quad (3.123)$$

3.9 General Purpose Graphics Processing Unit (GPGPU) Programming

This is intended as a brief overview of GPU programming, as the real-time aspect was an important contribution to the work in Chapter 5, and was made possible by this programming paradigm. Additionally this programming paradigm was also applied to the work of Chapter 6 in order to greatly accelerate the performance. Chapters 7, 8 and 9 all heavily rely on GPGPU programming in their underlying machine learning libraries, as GPGPU programming has largely also been the enabler for the current explosion in machine learning. In particular Chapter 9 uses an accelerated inference approach developed by NVIDIA, in order to run in real-time.

The wide spread adoption of GPGPU programming has been led by two main factors, the development of software frameworks such as Khronos' OpenCL and NVIDIA's CUDA, to allow easy interaction with GPU hardware, and the massive increase in parallel performance in recent GPUs from both NVIDIA and AMD. This has led to more than just massive gains in speed allowing the use of dense approaches in real-time, it has largely enabled the massive popularity of machine learning as parallel GPU programming is the cornerstone of convolutional neural networks (see Section 1.4). We include this chapter as a background for understanding software design considerations based on the hardware characteristics of GPUs and as a brief explanation of their operation. The majority of this Section will apply to all GPGPU software frameworks, but the majority of the work from this thesis was implemented using CUDA, and so the discussion of this thesis will be limited to NVIDIA's CUDA. As this thesis is not purely focused on CUDA programming and optimisation, this is beyond the scope, for a more detailed discussion of the CUDA fundamentals and GPGPU programming in general please refer to [204].

3.9.1 Serial vs Parallel Programming

The main difference between developing for the CPU and GPU is the programming paradigm, on a CPU there is extremely limited parallel resources and so the fastest approach is in general to process things serially, while on a GPU the amount of parallelism is high but the individual workers are generally slower making parallel processing essential on a GPU to maintain a high throughput. An example of the difference in paradigms is illustrated in Figure 3.26, where this provides the simple case of a sum reduce.

3 FUNDAMENTALS

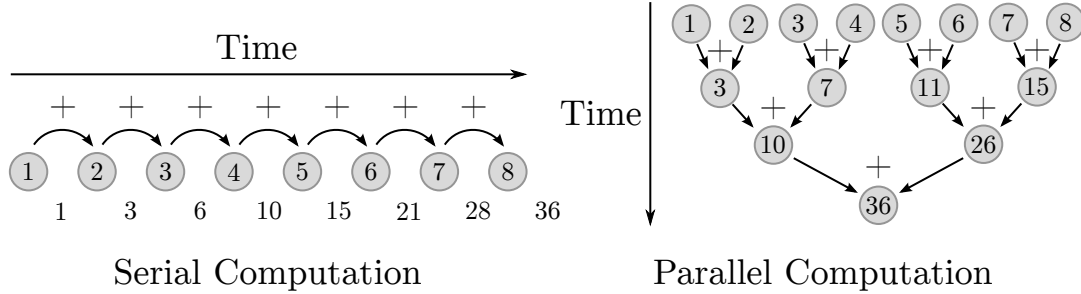


Figure 3.26: **Left:** The serial computation has to make a running total of the numbers to add them up **Right:** The parallel computation uses additional hardware in order to compute the sum in parallel, which allows it to do the same amount of work in less time.

```

1  __global__ void add(float** array_in, float** array_out, const int
    half_length)
2  {
3  // compute the a unique index into the array
4  // threadIdx, blockIdx and blockDim are populated by the SM during the
    call
5  int idx = threadIdx.x + blockIdx.x*blockDim.x;
6
7  // bail early if index exceeds length
8  if(idx >= half_length) return;
9
10 // perform one-level of addition
11 array_out[idx] = array_in[idx] + array_in[idx + half_length];
12 }
13
14 float addGPU(float** array1, float** array2, int full_length)
15 {
16 dim3 blockSize(N,1,1); // specify block size - generally based on the
    number of CUDA cores
17 int M = (full_length/(2*N)) + 1; // need enough blocks to cover half of
    the original length
18 dim3 gridSize(M,1,1);
19
20 int it = 0;
21 for(int l = full_length/2; l > 1; l /= 2) {
22 if((it%2) == 0)
23 add<<<gridSize, blockSize>>>(array1, array2, l);
24 else
25 add<<<gridSize, blockSize>>>(array2, array1, l);
26 ++it;
27 }
28 if((it % 2) == 0) return array1[0];
29 else return array2[0];
30 }

```

Listing 3.1: Naive code example of computing GPU sum

3 FUNDAMENTALS

It's easy to see the amount of actual work done is the same for both approaches, but the number of clock-cycles required is much lower for the GPU. In an ideal case the relationship is roughly $\log(N)$ vs N for the GPU and CPU respectively, in terms of time complexity, which provides a significant speed-up. In practice this will not hold, as the amount of hardware is limited in size on the GPU, but in general this sort of ratio is applicable when thinking about the relative speed-ups. In order to execute instructions in parallel, they are broken down into individual functions called kernels. In general they compute a small amount of work, for example the kernel to add a set of numbers in shown the following code snip-it Listing 3.1.

This demonstrates a naive summing based on the parallel computation, which is shown in Figure 3.26 - **Right**. In practice there are a number of optimisations that would be used, in particular cross-thread communication and shared memory [204].

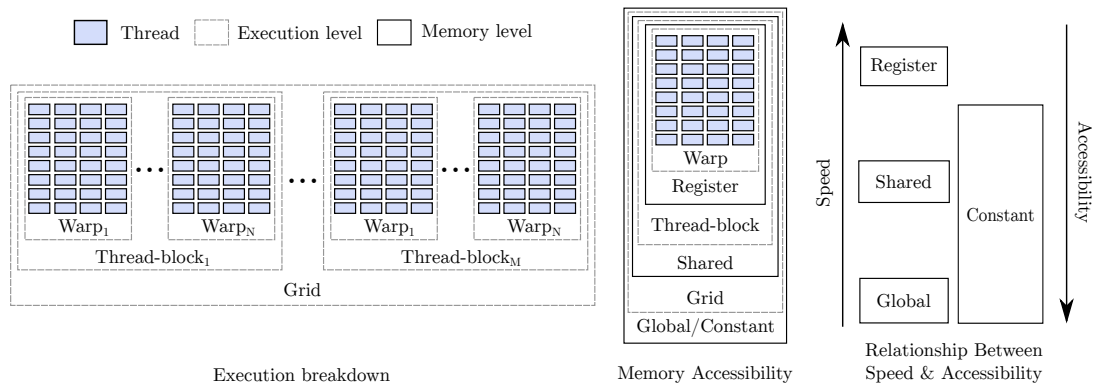


Figure 3.27: Left: The execution breakdown as it is issued to the GPU, threads are divided into warps, which are grouped into thread blocks, and these are arranged in an execution grid, which in this case is 2-dimensional. Both thread-blocks and grids can be specified up to 3 dimensions, and this arrangement is specified during the kernel call by the issuer. The arrangement of thread-blocks and the grid, can have a significant effect on execution time, given the way in this the threads may access memory. **Centre:** Shows the relative memory accessibility of the execution blocks to memory levels, a warp can only interact with *register* memory within the warp, a thread-block can interact through *shared* memory across warps, and any thread can interact across *global* memory which is globally accessible. There is also a limited amount of *constant* memory, which is globally accessible, but only for read-access.

3.9.2 Streaming Multi-processors and Thread Blocks

The GPU contains units known as Streaming Multi-Processors (SMs) which perform the work. The SMs are enlisted by the GPU to execute the work, and attempt to organise the work in the most efficient way possible. The work is organised into thread-blocks, which are organised into sets of warps, which in general contain 32 threads each. Warps can be efficiently executed in parallel as long as they maintain the same execution path

3 FUNDAMENTALS

across all threads in the warp. That is they execute the same instructions in the same order, otherwise in order to maintain synchronisation across a warp the SM will hold all other threads until they reach a common point of execution in a particular kernel before resuming execution in parallel. We show the general layout of the execution as it is passed to the GPU in Figure 3.27, this would be for the following execution of a generic kernel.

3.9.3 Memory and Coherency

The memory layout on the GPU is critical [205] when considering the performance of an approach programmed using GPGPU programming. A rough outline of the trade-off between speed of access to memory type and accessibility is shown in Figure 3.27-**Right**. There is roughly an order of magnitude ($\times 10$) speed-up moving between the memory types, with *register* memory roughly 100 times faster than *global* memory. *Constant* memory is a bit more complicated, it can be nearly as fast as register memory when reads are broadcast to multiple threads simultaneously, that is when entire thread blocks read the same *constant* memory value at the same time. The system can broadcast the value to all threads simultaneously. Outside of this approach constant memory can be as slow as global if random accesses are made.

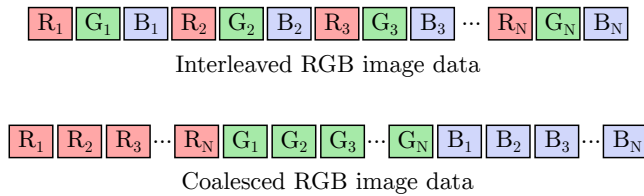


Figure 3.28: Top: The typical data arrangement of a colour image, with red, green and blue values interleaved. **Bottom:** The arrangement preferred by a GPU, where the memory access can be coalesced, by accessing all the read pixels, then all the green and finally all the blue.

An additional important consideration in memory access is random access in *shared* or *global* memory, which is in general very slow. The preferred method of accessing any memory is through memory coalesced calls. This is where threads in the same block will access adjacently located (coalesced) memory locations. An example is shown in Figure 3.28, where we read from the data in a colour image. Although in practice the data for red, green and blue is interleaved, as we will in general access these values sequentially on a CPU, on a GPU we will in general access the red on all threads followed by the blue followed by the green. This leads to a situation where the system will have trouble caching the data effectively, and can massively slow down read performance as it will increase the cache miss-rate, which is a measure of the rate at which uncached (random) data is accessed. This is all because we must keep in mind that the warps execute instructions in parallel, so to read a single colour value (i.e. RGB) the warp must sequentially read in the constituent colours, we want the instructions in a

3 FUNDAMENTALS

warp to access adjacent memory locations to improve performance, so we put all the reads together, then all the greens and then all the blues. Coalesced memory access is important on the CPU as well, and one can get large speed-ups by changing access patterns or altering the structure of the input data, this is particularly true for matrix multiplication operations.

3.9.4 Application to Robotics

GPGPU programming has been applied with great success to many dense approaches in robotics and vision [14, 17, 92, 24]. In particular we use this technology to enable the computation of dense surface curvature estimation on noisy point cloud data, detailed in Chapter 5. In addition this paradigm shift and subsequent massive increase in available compute throughput has directly led to the explosion of machine learning in the robotics community through libraries like Caffe [28] and Tensorflow [206], which makes it the enabler of the latter three chapters of this thesis.

3.10 Convolutional Neural Networks (CNNs)

In this section we provide a description of some of commonly used operations performed inside a CNN, as well as some of the principles of operation.

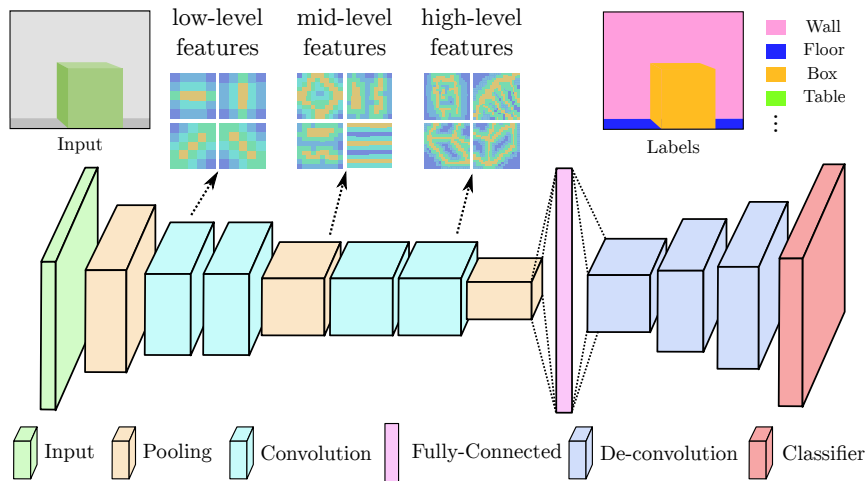


Figure 3.29: A relatively basic CNN (by current standards) with many of the components that appear in some of the systems used inside this thesis. Additionally the concept of features becoming more complex as the network becomes deeper is shown, with simple edge detectors building to detect complex shapes. The features shown are not extracted from the input image, these are a representation of strong responses that would be generated by particular filters at that stage of the network. This is purely intended to be illustrative.

3.10.1 Activations and Weights

Convolutional Neural Networks (CNNs) perform successive layers of convolutions in order to perform machine learning tasks. The convolutions are performed using filters that are formed from the weights of the network, which result in successive levels of activations. The activations, gradients and weights are stored in constructs called tensors inside the network. In a trained CNN the successive convolutional layers use the activations as intermediate storage for responses to input features of increasing complexity. An example of a basic CNN is shown in Figure 3.29, which is a classification network that uses a number of common CNN operations. This is intended to predominantly demonstrate the increasing complexity of features that generate strong activations as one moves deeper into the network. Note, the features are not the activations but are expressed through the weights, which produce strong activations to successively more complex features moving deeper into the network. The weights of a network are the *trainable* parameters, while the activations store the intermediate results moving from layer to layer. The model capacity of a network is essentially given by the number of weights. The amount of computation performed by the network is in general correlated to the overall model capacity.

3.10.2 Convolutional, Pooling and Fully-Connected Layers

In Figure 3.30, several example layers are shown, including convolution, pooling (max in this case) and fully-connected layers. These are all shown for a 1-dimensional input signal, but extend to any dimension. A very good explanation of convolution and transposed convolution can be found in [207]. A very good explanation of pooling and un-pooling can be found in [208]. Fully-connected layers are essentially a large convolutional layer, but have a unique enough purpose to justify a separate distinction.

3.10.2.1 Convolutions and De-convolutions (Transposed)

Convolutional layers simply perform convolutions on input tensor, the number of filters will determine the number of output channels and the size of the filters will determine the final dimensions of the output tensor. For example the convolution in Figure 3.30 has a single 1×3 filter, which means the output will lose 2 elements due to the length of the filter and will also be 1-dimensional. As the input signal is 1D. The filters can also only be 1D, by adding more 1D filters the network can increase the channel count. This is how a network is able to build successively more complex features and extract a number of features from a given input layer. Adding to the channel count allows the network to encode more complex features but increases the complexity of training.

3 FUNDAMENTALS

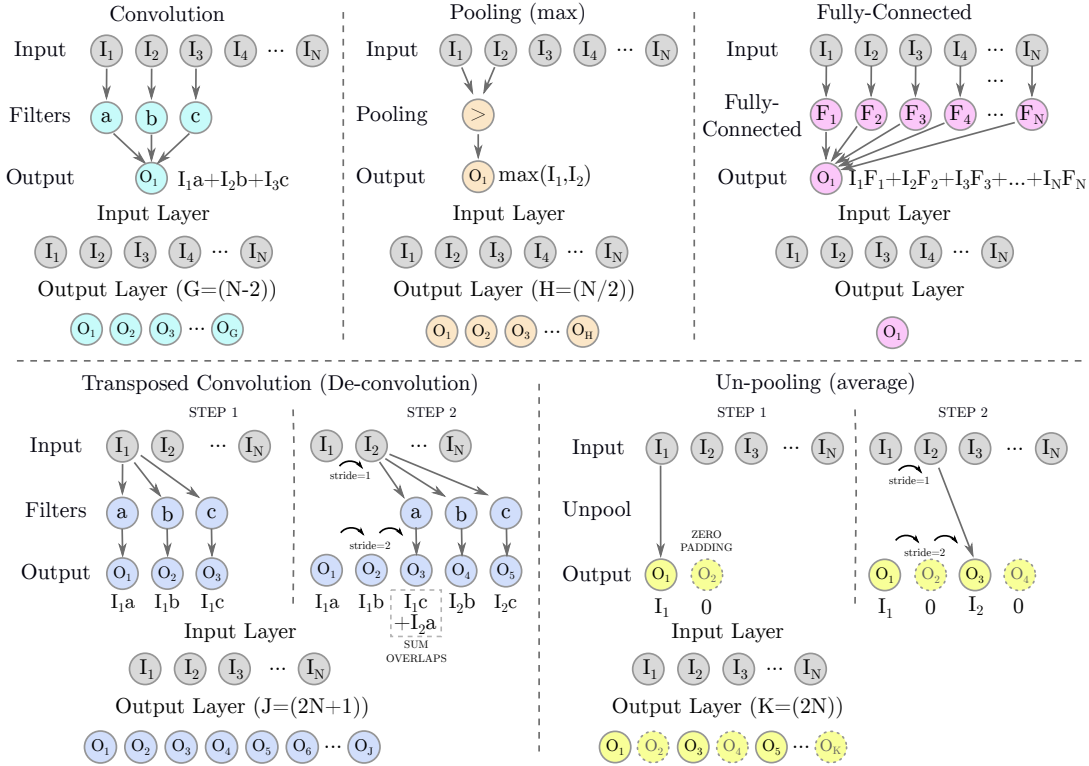


Figure 3.30: Top-Left: Example of convolution on a 1D input. **Top-Middle:** Example pooling operation with a stride of 2, meaning the output size will be halved as indicated. **Top-Right:** Fully-connected layer with a single channel output, this connects to all previous layers with an individual weight per connection. To increase the number of values in the output, one can simply increase the number of fully-connected filters, the current diagram has a single filter, with M filters the output will have M channels. **Bottom-Left:** An example of a transposed convolution (also commonly referred to as deconvolution) for a 1D input. **Bottom-Right:** An example of average un-pooling for a 1D input, which essentially interleaves the input data with zeros to create the output. Both un-pooling and de-convolution are examples of methods to upsample/increase the resolution.

The weights inside each of the convolutional filters are the trainable parameters of the network, and they are optimised to move the network towards the objective function.

Transposed convolutional (also commonly known as de-convolutional) layers are essentially performing the opposite of a convolutional layer. This is the equivalent of how a convolutional layer is back-propagated. As shown in Figure 3.30 for a 1D input example, the transposed convolution works by multiplying the filter values by the input value and adding the result to the output. This is a 1×3 filter with a stride of 2, where the stride applies to the movement of the filter across the output, and moves a constant unit stride across the input. Overlapping values in the output are simply summed. As the

3 FUNDAMENTALS

filter is also made up of learnable weights which are trained during back-propagation, this approach is a way of learning an up-sampling as it increases the output resolution. Again this channel count will vary based on the number of the filters used.

3.10.2.2 Pooling and Un-pooling

Pooling layers/operations in general perform a comparison or simple combination operation. In the case of Figure 3.30, the pooling is a max type, which means only the maximum input tensor is passed through the pooling layer. Pooling can also be over the minimum, average, modal, or any operation that reduces a set of values to a single result without the need for any trainable parameters. The filter is of size 2, with a stride of 2, where the stride dictates the step length the filter takes across the input tensor. With a stride of 1 it will move to the next value, comparing the same value twice, while with a stride of 2 the filter will move to the next pair of values. This means the maximum values only get propagated as most once, it also reduces the tensor size by half. This is a standard approach for distilling information and attempting to reduce the subsequent computational load on the latter layers of the network, as from this point on it will have half the data to deal with. That is pooling-layers *pool* the data from the input tensor.

Un-pooling is again essentially the operation of pooling, as shown in Figure 3.30. This is another method of upsampling and like pooling is not generally treated as a learnable operation making it very quick. The very basic approach shown, is average un-pooling, this simply assumes the input value is the average, it then passes this value through and pads with a zero, doubling the resolution. If this were in 2D, the padding occurs across each dimension, and so one value would pass through and 3 zeros. Another approach is to pass the value twice, or pass the value and instead of a zero interpolate between the adjacent value (this is bilinear up-sampling). All approaches result in an up-sampled output of double resolution. In the case of max-pooling un-pooling can be important to an up-sample step that may be present in a decoder network, as the system needs to keep track of which path it took in terms of the max, so it knows where to position of the un-pooled value.

3.10.2.3 Fully-connected

Fully-connected layers are connected to all values of the input tensor with an individual weight per connection. This means increases the receptive field of this layer to the entire input size, meaning this layer has the potential to encapsulate information from the entire input, which can be incredibly for classification tasks as it provides the network

with the most possible context. However the operation is only linear, and as such to be useful to input tensor must contain a large amount of high-level feature information. In Figure 3.30 we show a fully-connected layer with a single output channel, in practice the fully-connected layer in a classifier will contain perhaps a number of channels equivalent to the number of classes it is trying to predict against. This can become a very expensive operation, as the input layers can be quite large the and number of classes being predicted quite high, and there needs to be a weight between every input and every output. This often leads to fully-connected layers accounting for significant portions of the networks overall model capacity, which isn't particularly desirable as it is still just a single linear operation. This makes pooling type operations essential in order to constrain the overall computation of the network. All weights in the fully-connected filters are also trainable.

3.10.3 Rectifiers and Drop-out

Rectifiers and drop-out have similar purposes, and have the same biologically inspired basis, that some signals inside a neural network should fail to propagate. A common unit for rectification is the Rectified-Linear-Unit, these block all input signals less than zero and can be expressed as

$$f(x) = \max(0, x) \quad (3.124)$$

where x is the input activation, and would be performed element wise on the tensor input to the ReLU layer. This is very efficient to compute (a single comparison) and has a number of other desirable traits and will reduce the number of propagating signals randomly in a randomly initialised network. An another popular alternative is leaky ReLUs, which instead of blocking all signals less than zero, simply reduce their influence massively, reducing them by a factor of 100 typically. Another rectifier is the parametric ReLU, which makes this reduction using a trainable parameter, adding to the computational expense. This allows negative values to always generate a training signal, but maintains most of the benefits. A key motivation in using ReLU functions is to mitigate the vanishing and exploding gradient problem [209] (see Section 3.10.4.2). The previous alternative for activation was to use a sigmoid or hyperbolic-tan function, which would result in gradients exponentially decreasing during back-propagation, by using a ReLU it prevents this as the gradients remain linearly proportional.

Drop-out is another method for cutting some of the propagated activations, but instead of being based on the input activation level, its a random decision to propagate a certain percentage of the time. The main motivation behind drop-out is to reduce the systems capacity to overfit to the training data, which is highly undesirable, see Section 3.10.4.3. It prevents this from happening by essentially forcing the network to make predictions while being denied a significant proportion of the data, forcing the network to generalise. This is a method of network regularisation. Drop-out is particularly

helpful for networks that contain fully-connected layers [35], but in fully-convolutional networks it has largely been supplanted by the use of batch-normalisation (see Section 3.10.5).

3.10.4 Training Through Back-propagation

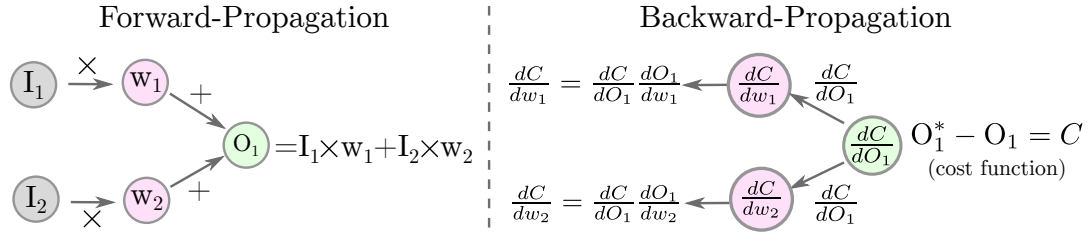


Figure 3.31: **Left:** forward propagation on a section of a neural network, input activation (I_i) are propagated through the network by multiplication with trainable weights (w_i) **Right:** back-propagation propagates derivatives back through the network to the respective weights through the application of the chain-rule (see Section 3.4.1).

The method of training a CNN, is to treat the machine learning problem as an optimisation problem. The simplest method for training is to use a basic gradient descent (or stochastic gradient descent) approach, where it takes small steps in a direction that minimises the cost function. The cost function(s) of a network is essentially what the network is trying to minimise, in the case of a depth estimation the loss could be the difference between the network prediction and the ground truth depth. To optimise through gradient descent (as described in Section 3.5.4) the network simply needs to optimise the trainable parameters with respect to the loss function, by moving the trainable parameters in the direction that minimises loss. This requires we compute the gradient of each parameter with respect to the loss function, for a neural network with millions of parameters this seems intractable, but as shown in Figure 3.31 the computation of the gradients can be calculated through back-propagation of the chain rule as described in Section 3.4.1.

3.10.4.1 Momentum, Learning Rate and Weight-decay

The choice of momentum [210] and learning-rate in a network is incredibly significant to the overall performance. We can, by way of example understand their significance to training, firstly consider a simple network that is trained using gradient descent, in general during gradient descent the step we take is simply towards the direction that the gradients predict will minimise the cost the most. The learning rate is a multiplier that we would put on this step, moving approximately that proportion of what the

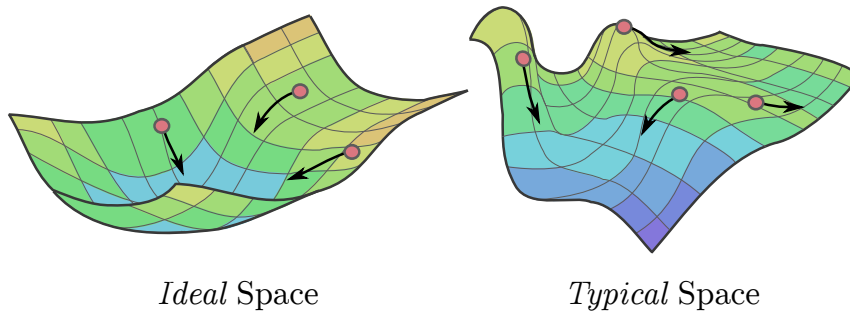


Figure 3.32: **Left:** theoretical *ideal* optimisation space, where its entirely convex and moving in the direction of the loss gradient takes the solution to a global minimum **Right:** a more realistic interpretation of the optimisation space where there are many local minimas and the direction is highly dependent on the current state of the system shown with red circles, the arrow shows the direction the state will move in order to reduce the cost.

gradient predicts will take the cost to zero. The reason we only want to move a small step is, the network back-propagates gradients both traversing the entire set of input data, making a technically stochastic approach. This makes it much more desirable to take very tiny steps, particularly given the nature of the loss function. A higher learning rate will move faster along the objective space which would result in faster convergence in the case of a highly convex objective space. However, this is very unlikely due to the complexity of the system resulting a large number of local minima, with a learning rate that is too high it can even prevent convergence altogether.

In practice learning rates are reduced gradually during training, and the hope is that the network get stuck in a highly optimal local minima, which is visualised in Figure 3.32. Momentum is an acknowledgement that the optimisation space of a neural network is highly non-convex and unstable, and that in general it may be better to *integrate* over the previous steps taken by the network and move in a direction that biases towards the direction the network has been moving previously. This has the benefit of potentially increasing the convergent rate, but also reduces the overall chance that the network will get stuck in a local minima. Both the momentum and learning-rate are in general externally set values between 0 and 1. Another common form of regularisation is weight decay [211], which is commonly implemented by attempting to minimise the L2 norm of the summed total of the weights in the network. The goal of this is to reduce large uneven weight distributions through-out a network, thus making the network more stable. This is typically given a very low priority by the network compared to the main loss function, by multiplying the weight loss term by a very small fraction of 1.0.

3.10.4.2 Vanishing and Exploding Gradient Problems

These are two problems that were present in the initial stages of neural networks. The vanishing gradient problem was related to the choice of sigmoid and hyperbolic-tan rectifier layers which would result in the gradients exponentially decreasing as they were back-propagated through the network, resulting in practical maximum depths, and slow training of the initial layers of the network. The exploding gradient problem is the exact opposite issue where possible initially large gradients are propagated back through the network causing unstable behaviour, this is generally caused by unfortunate random initialisation combined with an excessively high learning-rate. The obvious fix is to use lower learning rates, and also the choice of ReLU layers over sigmoid and hyperbolic-tan functions improves stability in this regard.

Another break-through in machine learning was to train the network to learn residual functions [29], which makes the network more mathematically stable and largely prevents gradient problems from arising. Residual layer networks attempt to force layers to try and learn the most information that it is currently unable to represent, they do this by creating groups of layers often called blocks, that will perform a series of convolution and ReLU operations followed by an addition of the original input tensor onto the output tensor. In this way the block attempts to force the network to learn important new features it can introduce to the original tensor, as the original signal is still present in the output.

3.10.4.3 Over-fitting

Model over-fitting is a highly undesirable state for a neural network, it means that the network has learned to fit very specific traits of the training data rather than actually learning general trends from the training data. This will cause the network to perform very well during training, but far less optimally than possible during deployment on unseen data [212, 213]. Over-fitting is common in situations where ground truth data is scarce and so the network has insufficient training data, especially when the model capacity is high. A typical method for determining a network has over-fit is to use separate training and validation data, training data is used for back-propagation while the validation purely for forward-propagation (also known as inference), the training and validation cost can be evaluated separately and if the validation loss begins to increase while the training loss continues to reduce this is a good indication of over-fitting.

As mentioned previously a good method of reducing over-fitting is to use drop-out [33], which will randomly terminate activation during any given forward pass, forcing the

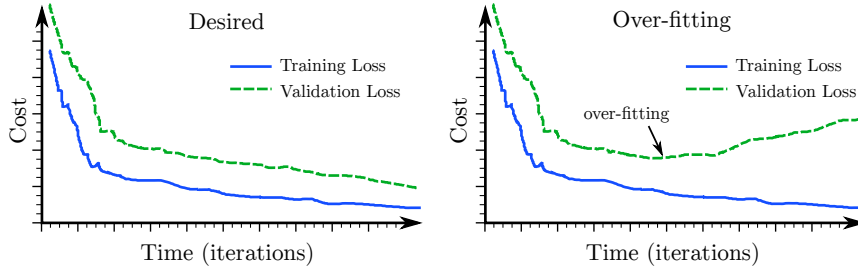


Figure 3.33: **Left:** Desired performance of a CNN on training and validation loss curves, as the training loss continues to reduce so does the validation loss. **Right:** Shows signs of a CNN over-fitting to the training data, the validation loss begins to diverge and then increase, indicating the network has overfit to the training data.

network to learn more general features to continue to minimise the cost function as training data will always be missing. Drop-out is an aggressive form of adding noise to training data, which is more generally what is done to reduce over-fitting, drop-out adds essentially 100% noise to some percentage of the activations. Another common approach in fully-convolutional networks (networks that contain no fully-connected layers) is the use of batch normalisation described in Section 3.10.5.

3.10.5 Batches and Normalisation

Batches are used in CNNs to reduce training time and improve stability, by reducing the number of update steps performed per input. To generate a batch inputs are grouped into input tensors, for example several images are stacked into a batch. The batch is then passed through the network simultaneously in a single forward pass which generates gradients for back-propagation. This means the weights will be fixed for the entire forward pass for a batch, the gradients will then move the state in an average direction of the batch during the update.

Batch normalisation is the process of normalising across a so-called mini-batch in order to improve network training speed and gradient stability. It was first proposed by Ioffe and Szegedy in [214], the first step is to normalise the values across a mini-batch to have a mean of 0 and a variance of 1, which has shown to increase the rate of convergence and stability in neural networks [215]. This can be expressed as performing the following transformation

$$\hat{x} = \frac{\bar{x} - \bar{\mu}_B}{\sigma_B^2}, \quad (3.125)$$

where \bar{x} is an input tensor, \hat{x} is a normalised output tensor, and $\bar{\mu}_B$ and σ_B^2 are the mean and variance of the input batch. This has the effect of potentially changing what the activations represent and may hurt performance when the tensor is intended to be passed

3 FUNDAMENTALS

to a sigmoid activation function, to address this they add two trainable parameters that allow the layer to retain the ability to perform an identity transformation. This takes the following form

$$\bar{y} = \hat{x}^T \bar{\gamma} + \bar{\beta}, \quad (3.126)$$

where $\bar{\gamma}$ and $\bar{\beta}$ represent a trainable scale and offset tensor respectively. This optional and only required in the very specific circumstances that normalising the tensor would effectively make the following layer redundant or ineffectual. During training the batch normalisation layer will also keep track of a global mean and variance, that it can use during inference only situations. In this form each of the elements of the layer is differentiable, and so back-propagation is simple and the trainable parameters can be optimised. Batch normalisation has become standard practice and stabilises network training and further allows the depth of networks to increase.

Low-Cost Depth Camera Calibration

In this chapter we present a method for calibration of low-cost depth sensors such as the Microsoft Kinect. We show our method is effective at correcting the structured sensor error using a simple compact parametric solution, that uses only a small fraction of the number of parameters used in many existing approaches. We have released this calibration method as an open-source implementation, with limited external library dependencies, in an attempt to make it more widely used. The material and results presented in this chapter are largely drawn from the paper [216].

4.1 Motivation

The introduction of low-cost structured-light depth sensors (such as the Microsoft Kinect) and time-of-flight (ToF) such as the Microsoft Kinect v2 has made the collection of accurate real-time depth scans trivial. As described in Section 3.3 This has led to the wide-spread adoption of both sensor types in a range of robotics and computer vision applications, including These sensors provide a registered stream of colour and depth images at 30Hz, allowing fast and accurate modelling of complex indoor scenes [13, 14]. While some previous methods calibrate for time-of-flight (ToF) sensors [217, 218], which have interesting but different noise characteristics, the focus of this paper is on the variety of depth sensors known as *light coding* depth sensors. This type of depth sensor projects a known pseudo random infra-red grid pattern onto the environment, then detects it using an infra-red camera. Using the known displacement of the camera and projector, and finding corresponding patches in the projected pattern allows the device to estimate a dense depth image. A more detailed explanation of this process can be found in [219]. Although these sensors have proven useful for modelling and tracking they have been shown to contain significant calibration errors [219, 76] (Figure 4.9).

Previous approaches have been used to correct the error present in these types of sensors [79, 82, 80, 84] with varying degrees of success. The approaches seem to fall into two main categories, planar and SLAM based calibration. The former approach uses a known planar surface and attempts to correct the reported depth to align more accurately to a fitted plane. While the SLAM/model based approaches attempt to accurately model a complex scene and correct the depth such that the model produced will minimise some global error function across all depth images.

4.2 Contributions

In this paper we present a compact polynomial (tens of parameters) method of depth sensor calibration using an unsupervised SLAM based approach. We provide an open-source implementation that uses few external libraries and provides a simple and reliable method of depth sensor calibration using a relatively small number of keyframes (10-15). The advantages of the presented calibration method are:

- It is highly compact, requiring a greatly reduced number of parameters than existing methods (see Section 4.4).
- It requires minimal supervision from the user (see Section 4.7).
- It provides noticeably improved depth estimates for a range of depth values (see Section 4.8).
- It outperforms previous approaches qualitatively and quantitatively (see Section 4.8.3)

4.3 Related Work

The most popular methods of depth sensor calibration seem to break into two main categories planar based approaches including [79, 80, 81, 82] and SLAM/mapping based approaches including [83, 84]. The planar methods attempt to calibrate the sensor in a more traditional way, by placing known scene elements in front of the scene at potentially known locations, and attempting to correct the signal to match the known features of the scene elements. SLAM or mapping based approaches try to remove the requirement for known scene elements as part of the calibration method. These approaches attempt to minimise modelling error across a scene by optimising a depth correction and resolving for the relative alignments. This method of calibration is highly desirable as it requires no special objects and can even be performed on sensor data from datasets, where the initial sensor is not available but the dataset remains uncalibrated [40].

The method presented in this chapter falls into the latter category, but we attempt to use much fewer parameters to reduce the models chances of over-fitting. Additionally we perform our optimisation jointly over the pose alignment and correction function, while previous approaches treat this process as a multi-stage approach, fixing poses while optimising with correction functions, or vice versa. Additionally our approach includes the radial distortion parameters as part of the correction, allowing greater flexibility in correction, not present in previous modelling based approaches.

4.4 Correction Function

In order to calibrate low-cost depth sensors we propose a polynomial correction function, which can be used to jointly approximate the geometric and lens distortion present in projective type low-cost depth sensors. We express the calibration as both a radial distortion factor $\mathbf{r}(u', v')$, and a multiplicative depth correction factor $\mathbf{c} = C(\mathbf{r}u', \mathbf{r}v', d)$. The corrected points p are computed from the normalised camera coordinates u, v and the sensed depth d as

$$p_i(k) = \text{proj}(u', v', \mathbf{c}) = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{c}d\mathbf{r}u' \\ \mathbf{c}d\mathbf{r}v' \\ \mathbf{c}d \\ 1 \end{pmatrix}, \quad (4.1)$$

where $\text{proj}(u', v', \mathbf{c})$ is the projection of the radially corrected parameters for used in the geometric correction function and $d = d_i(u', v')$ is the depth at u', v' in depth scan i . The normalised camera coordinates u', v' are computed using:

$$u' = \frac{X - u_0}{f_x}, \quad v' = \frac{Y - v_0}{f_y}, \quad (4.2)$$

where u_0 and v_0 are the coordinates of the principal point, f_x and f_y are the focal lengths in the horizontal and vertical dimensions respectively, and X and Y are pixel coordinates. The radial distortion correction is given by

$$\mathbf{r}(u', v') = 1 + \beta_0(u'^2 + v'^2) + \beta_1(u'^2 + v'^2)^2, \quad (4.3)$$

which as shown is only defined for the first two radial distortion parameters. This allow the computation of the corrected normalised camera coordinates (u, v) via the following relationship

$$u = \mathbf{r}u' \text{ and } v = \mathbf{r}v'. \quad (4.4)$$

These corrected normalised camera coordinates are then passed to the depth correction function $C(u, v, d) = \mathbf{c}$, which is calculated as

$$\mathbf{c}^o = C^o(u, v, d) = 1 + \alpha_0u + \alpha_1v + \alpha_2uv + \dots + d(\alpha_nu + \alpha_{n+1}v + \alpha_{n+2}uv + \dots), \quad (4.5)$$

which enumerates the initial terms of the radial $(\alpha_0, \alpha_1, \dots, \alpha_n)$ and depth $(\alpha_n, \alpha_{n+1}, \dots)$ geometric correction coefficients, and u, v are the radially corrected normalised camera coordinates. We continue this pattern of enumerating all possible polynomial combinations $(u, v, uv, u^2, v^2, \dots, du, dv, duv, \dots)$ with a separate coefficient α_i for each up to the desired order given by o . In the case of the fifth-order polynomial the full calibration

function is

$$\begin{aligned}
\mathbf{c}^5 = C^5(u, v, d) = 1 & \\
& \alpha_0 u + \alpha_1 v + \\
& \alpha_2 u^2 + \alpha_3 uv + \alpha_4 v^2 + \\
& \alpha_5 u^3 + \alpha_6 u^2 v + \alpha_7 uv^2 + \alpha_8 v^3 + \\
& \alpha_9 u^4 + \alpha_{10} u^3 v + \alpha_{11} u^2 v^2 + \alpha_{12} uv^3 + \alpha_{13} v^4 + \\
& \alpha_{14} u^5 + \alpha_{15} u^4 v + \alpha_{16} u^3 v^2 + \alpha_{17} u^2 v^3 + \alpha_{18} uv^4 + \alpha_{19} v^5 + \\
& d(\alpha_{20} u + \alpha_{21} v + \\
& \alpha_{22} u^2 + \alpha_{23} uv + \alpha_{24} v^2 + \\
& \alpha_{25} u^3 + \alpha_{26} u^2 v + \alpha_{27} uv^2 + \alpha_{28} v^3 + \\
& \alpha_{29} u^4 + \alpha_{30} u^3 v + \alpha_{31} u^2 v^2 + \alpha_{32} uv^3 + \alpha_{33} v^4 + \\
& \alpha_{34} u^5 + \alpha_{35} u^4 v + \alpha_{36} u^3 v^2 + \alpha_{37} u^2 v^3 + \alpha_{38} uv^4 + \alpha_{39} v^5),
\end{aligned} \tag{4.6}$$

which clearly demonstrates how the number of terms increases approximately quadratically with respect to the order of the polynomial model. Experimentally a 7th-order correction function was found to produce the best results for calibration, which has 70 parameters in total.

4.5 Frame-to-frame Alignment

The basis of any modelling based alignment approach is frame-to-frame alignment, and this is required to produce the initial alignment estimates used in the larger optimisation including the calibration parameters. As described in Section 3.7, we can align pairs of point-clouds using ICP. One of the most commonly used approaches is point-to-plane ICP [12, 120, 13, 15, 17]. This iteratively optimises a relative pose alignment that attempts to minimise the point-to-plane distance between correspondences given by the following residual function

$$r_i = \hat{n}_i \cdot (p_i - \mathbf{E}(\bar{\gamma}_j)q_i), \tag{4.7}$$

where $p_i, q_i \in \mathbb{R}^3$ are corresponding points in point-clouds P_1 and P_2 respectively, and \mathbf{E} is the estimated transformation from P_2 to P_1 given by the motion parameters $\gamma_j \text{in} \mathfrak{se}_3$. This forms the following cost function

$$\epsilon_{ftf} = \sum_{i=0}^N r_i^2, \tag{4.8}$$

where N is the number of corresponding point in both P_1 and P_2 . This function can be optimised in terms of the transformation parameters using a Gauss-Newton approach as described in Section 3.5. This requires the differentiation of the cost function with

respect to the motion parameters. The transformation is expressed as a matrix exponential as described in Section 3.6.2, which is simpler to differentiate. The derivative of the cost function with respect to the motion parameters can be expressed as

$$\frac{\partial r_i}{\partial \gamma_j^k} = -\hat{n}_i \cdot \left(\frac{\partial \mathbf{E}}{\partial \gamma_j^k} q_i \right) = -\hat{n}_i \cdot (\mathbf{G}_k \mathbf{E}_0 q_i), \quad (4.9)$$

where γ_j^k is the k^{th} motion parameter and \mathbf{E}_0 is the current transformation estimate. Evaluating this gives the following Jacobian matrix in terms of the motion parameters

$$\begin{aligned} \mathbf{J}_i &= \begin{bmatrix} -\hat{n}_{ix} & -\hat{n}_{iy} & -\hat{n}_{iz} & \hat{n}_{iz}y_i - \hat{n}_{iy}z_i & \hat{n}_{ix}z_i - \hat{n}_{iz}x_i & \hat{n}_{iy}x_i - \hat{n}_{ix}y_i \end{bmatrix} \\ &= \begin{bmatrix} -\hat{n}_i & -\hat{n}_i \wedge q'_i \end{bmatrix} \end{aligned} \quad (4.10)$$

where $\mathbf{E}_0 q_i = q'_i = \begin{pmatrix} x_i & y_i & z_i & 1 \end{pmatrix}^T$ is the estimated transformed coordinates, \hat{n}_{ix} is the x -component of the i^{th} unit normal \hat{n}_i and \wedge is used to represent the cross-product operator. We can concatenate these Jacobian matrices to form a single large matrix

$$\mathbf{J}_{ftf} = \begin{bmatrix} \mathbf{J}_1^T & \mathbf{J}_2^T & \dots & \mathbf{J}_N^T \end{bmatrix}^T. \quad (4.11)$$

This can also be done for the residuals to form a single residual vector

$$\bar{r}_{ftf} = \begin{bmatrix} r_1 & r_2 & \dots & r_N \end{bmatrix}^T. \quad (4.12)$$

Now the update to the pose $\Delta \bar{\beta}_{ftf}$ can be calculated using

$$\Delta \bar{\beta}_{ftf} = (\mathbf{J}_{ftf}^T \mathbf{W} \mathbf{J}_{ftf})^{-1} \mathbf{J}_{ftf}^T \mathbf{W} \bar{r}_{ftf}, \quad (4.13)$$

where \mathbf{W} is a diagonal weighting matrix that where the weights are given by the following function

$$w_i = \frac{c^2}{c^2 + r_i^2}, \quad (4.14)$$

where c is a constant based chosen to match the typical computed error. This is used to form the initial alignment estimates, which is are required in order to begin the optimisation, as the cost function becomes highly non-convex moving away from the true solution. The updated pose is given by a simple matrix exponential equation, as shown in [220]

$$\mathbf{E}_{t+1} = e^{\sum_{k=1}^1 \gamma_k G_k} \mathbf{E}_t \quad (4.15)$$

where \mathbf{E}_t is the pose at time t .

4.6 Joint Multi-frame Alignment and Calibration

We define a residual function $\hat{r}(i, j, k)$ for each frame pair (i, j) in the input data, for every correspondence between frame pairs k as

$$r'_{ijk} = \mathbf{R}_i \hat{n}_k \cdot (\mathbf{E}_i p_{ki} - \mathbf{E}_j p_{kj}), \quad (4.16)$$

where $i \neq j$ and $\mathbf{E}_i = \mathbf{E}(\bar{\gamma}_i)$ is the transformation described by the motion parameters $\bar{\gamma}_i$ that would take the projection p_i (as described in Equation 4.1) of the i th point-cloud (P_i) to a common world coordinate frame. In this form, Equation 4.16 gives the residual in terms of absolute poses instead of a single relative pose as in Equation 4.7. Points $p_i \in P_i$ and $p_j \in P_j$ are calibrated corresponding points computed from the depth images d_i and d_j respectively, which have been corrected using the current estimated correction function (Equation 4.1). Correspondences are estimated using a project-and-scan strategy as described in Section 3.7.2. All correspondences are estimated on points after the current calibration estimate is applied. This defines a cost function for point-cloud pairs in terms of the absolute pose estimates ($\bar{\Gamma}$), and the geometric($\bar{\alpha}$) and radial distortion($\bar{\beta}$) parameters

$$\epsilon(\bar{\Gamma}, \bar{\alpha}, \bar{\beta}) = \sum_{i=1}^{\hat{N}} \sum_{j=1}^{\hat{N}} \sum_{k=1}^N (\hat{r}(i, j, k))^2, \quad (4.17)$$

where $i \neq j$, \hat{N} is the number of depth images used in the joint optimisation, N is the number of correspondences in the point-cloud pair, and $\bar{\Gamma} = \{\bar{\gamma}_1, \bar{\gamma}_2, \dots, \bar{\gamma}_{\hat{N}}\}$ are pose parameters for every point-cloud.

Again a standard Gauss-Newton approach was used to minimise error across this function with respect to all motion and calibration parameters jointly. This requires the differentiation of the error function given by

$$\begin{aligned} \frac{\partial r'_{ijk}}{\partial \bar{c}} &= \left(\frac{\partial \mathbf{R}_i}{\partial \bar{c}} \right) \hat{n}_{ki} \cdot (\mathbf{E}_i p_{ki} - \mathbf{E}_j p_{kj}) \\ &+ (\mathbf{R}_i \hat{n}_{ki}) \cdot \left(\frac{\partial \mathbf{E}_i}{\partial \bar{c}} p_{ki} + \mathbf{E}_i \frac{\partial p_{ki}}{\partial \bar{c}} - \frac{\partial \mathbf{E}_j}{\partial \bar{c}} p_{kj} + \mathbf{E}_j \frac{\partial p_{kj}}{\partial \bar{c}} \right), \end{aligned} \quad (4.18)$$

where $\bar{c} \in \{\bar{\Gamma}, \bar{\alpha}, \bar{\beta}\}$ is a generic parameter vector, that can be used to represent the separate motion parameters ($\bar{\gamma}_i$) for each frame and the joint calibration parameters ($\bar{\alpha}$ and $\bar{\beta}$) for simplicity. To more simply in clarify Equation 4.18 the calculation for a single point-cloud pair is examined and separate Jacobian calculations performed.

4.6.1 Pose Jacobians

Each point-cloud pair P_i and P_j has two sets of motion parameters $\bar{\gamma}_i$ and $\bar{\gamma}_j$ corresponding to their respective absolute pose, and a joint/shared set of calibration parameters ($\bar{\alpha}$ and $\bar{\beta}$) as they were captured using the same sensor. The derivative with respect to

a the set of motion parameters $\bar{\gamma}_i$ is given by

$$\begin{aligned}
 \frac{\partial r'_{ijk}}{\partial \gamma_{li}} &= \left(\frac{\partial \mathbf{R}_i}{\partial \gamma_{li}} \hat{n}_i \right) \cdot (\mathbf{E}_i p_{ki} - \mathbf{E}_j p_{kj}) \\
 &\quad + (\mathbf{R}_i \hat{n}_{ki}) \cdot \left(\frac{\partial \mathbf{E}_i}{\partial \gamma_{li}} p_i + \mathbf{E}_i \frac{\partial p_{ki}}{\partial \gamma_{li}} - \frac{\partial \mathbf{E}_j}{\partial \gamma_{li}} p_{kj} + \mathbf{E}_j \frac{\partial p_{kj}}{\partial \gamma_{li}} \right) \\
 &= (\mathbf{G}'_{ki} \mathbf{R}_i^0 \hat{n}_{ki}) \cdot (\mathbf{E}_i^0 p_{ki} - \mathbf{E}_j^0 p_{kj}) + \mathbf{R}_i^0 \hat{n}_{ki} \cdot (\mathbf{G}_{li} \mathbf{E}_i^0 p_{ki} + \mathbf{E}_i^0 \mathbf{0} - \mathbf{0} p_{kj} + \mathbf{E}_j^0 \mathbf{0}) \\
 &= (\mathbf{G}'_{li} \mathbf{R}_i^0 \hat{n}_{ki}) \cdot (\mathbf{E}_i^0 p_{ki} - \mathbf{E}_j^0 p_{kj}) + \mathbf{R}_i^0 \hat{n}_{ki} \cdot (\mathbf{G}_{li} \mathbf{E}_i^0 p_{ki}),
 \end{aligned} \tag{4.19}$$

where γ_{li} is the l^{th} motion parameter of the motion parameter vector $\bar{\gamma}$, \mathbf{G}'_{li} is the l^{th} generator of the $\mathbf{SO}(3)$ matrix \mathbf{R}_i , \mathbf{G}_{li} is the l^{th} generator of the $\mathbf{SE}(3)$ matrix \mathbf{E}_i , \mathbf{R}_i^0 and \mathbf{E}_i^0 are the current i^{th} estimated rotation and full transformation respectively, $\mathbf{0}$ is used to represent a zero matrix, and \cdot represents the dot-product operator. To simplify the this slightly we can perform the following substitutions

$$\hat{n}'_{ki} = \mathbf{R}_i^0 \hat{n}_{ki}, p'_{ki} = \mathbf{E}_i^0 p_{ki} \text{ and } p'_{kj} = \mathbf{E}_j^0 p_{kj}. \tag{4.20}$$

This simplifies Equation 4.19 to

$$\frac{\partial r'_{ijk}}{\partial \gamma_{li}} = (\mathbf{G}'_{li} \hat{n}'_i) \cdot (p'_{ki} - p'_{kj}) + \hat{n}'_i \cdot (\mathbf{G}_{li} p'_{ki}). \tag{4.21}$$

In this formulation we assume

$$\hat{n}'_{ki} = \begin{pmatrix} n'_{kix} \\ n'_{kiy} \\ n'_{kiz} \\ 0 \end{pmatrix} \text{ and } p'_i = \begin{pmatrix} p'_{kix} \\ p'_{kiy} \\ p'_{kiz} \\ 1 \end{pmatrix} \tag{4.22}$$

are homogeneous. Also the generators are closely related in this case $\mathbf{G}'_{li} = \mathbf{G}_{li}$ for $l = \{\theta_x, \theta_y, \theta_z\}$ and $\mathbf{G}'_{li} = \mathbf{0}$ for $l = \{t_x, t_y, t_z\}$ as there is no relationship to translation for the rotation matrix. The resulting Jacobian for $\bar{\gamma}_i$ is

$$\mathbf{J}_{\bar{\gamma}_{ki}} = \begin{bmatrix} \hat{n}'_{ki} & \hat{n}'_{ki} \wedge p'_{kj} \end{bmatrix}. \tag{4.23}$$

Repeating this process for the other set of motion parameters $\bar{\gamma}_j$ gives

$$\frac{\partial r'_{ijk}}{\partial \gamma_{lj}} = -\hat{n}'_{ki} \cdot (\mathbf{G}_{lj} p'_{kj}), \tag{4.24}$$

which is essentially the same as Equation 4.9 hence the Jacobian will be the negative of Equation 4.23

$$\mathbf{J}_{\bar{\gamma}_{kj}} = -\mathbf{J}_{\bar{\gamma}_{ki}} = \begin{bmatrix} -\hat{n}'_{ki} & -(\hat{n}'_{ki} \wedge p'_{kj}) \end{bmatrix}. \tag{4.25}$$

This is obvious logically too, if moving P_i towards P_j decreases the residual, moving P_j in the opposite direction back towards P_i will have the same predicted effect and should increase the speed of convergence. This does mean that the poses of the point-clouds are free to drift, but this can be constrained by consistently moving all point-clouds back so a particular point-cloud is centred at the origin.

4.6.2 Depth Correction Jacobians

Returning to Equation 4.18, the derivative with respect to the depth correction parameters $\alpha_l \in \bar{\alpha}$ can be computed as

$$\begin{aligned}
 \frac{\partial r'_{ijk}}{\partial \alpha_l} &= \left(\frac{\partial \mathbf{R}_i}{\partial \alpha_l} \hat{n}_{ki} \right) \cdot (p'_{ki} - p'_{kj}) + \hat{n}'_{ki} \cdot \left(\frac{\partial \mathbf{E}_i}{\partial \alpha_l} p_{ki} + \mathbf{E}_i \frac{\partial p_{ki}}{\partial \alpha_l} - \frac{\partial \mathbf{E}_j}{\partial \alpha_l} p_{kj} + \mathbf{E}_j \frac{\partial p_{kj}}{\partial \alpha_l} \right) \\
 &= (\mathbf{0} \hat{n}_{ki}) \cdot (p'_{ki} - p'_{kj}) + \mathbf{R}_i^0 \hat{n}_{ki} \cdot \left(\mathbf{0} p_{ki} + \mathbf{E}_i^0 \frac{\partial p_{ki}}{\partial \alpha_l} - \mathbf{0} p_{kj} - \mathbf{E}_j^0 \frac{\partial p_{kj}}{\partial \alpha_l} \right) \\
 &= \hat{n}'_{ki} \cdot \left(\mathbf{E}_i^0 \frac{\partial p_{ki}}{\partial \alpha_l} - \mathbf{E}_j^0 \frac{\partial p_{kj}}{\partial \alpha_l} \right).
 \end{aligned} \tag{4.26}$$

The derivative $\frac{\partial p_{ki}}{\partial \alpha_l}$ can be computed using the chain-rule given the relationship defined in Equation 4.1

$$\frac{\partial p_{ki}}{\partial \alpha_l} = \frac{\partial p_{ki}}{\partial \mathbf{c}_{ki}} \frac{\partial \mathbf{c}_{ki}}{\partial \alpha_l} = \begin{pmatrix} d_{ki} \mathbf{r}_{ki} u_{ki} \\ d_{ki} \mathbf{r}_{ki} v_{ki} \\ d_{ki} \\ 0 \end{pmatrix} \frac{\partial \mathbf{c}_{ki}}{\partial \alpha_l} = \begin{pmatrix} \tilde{p}_{kix} \\ \tilde{p}_{kiy} \\ \tilde{p}_{kiz} \\ 0 \end{pmatrix} \frac{\partial \mathbf{c}_{ki}}{\partial \alpha_l}, \tag{4.27}$$

where \tilde{p} are the uncorrected point-cloud coordinates, and the derivative $\frac{\partial \mathbf{c}_{ki}}{\partial \alpha_l}$ is dependent on the degree of the selected corrective polynomial. Given a 2nd order correction function it is given by

$$\frac{\partial \mathbf{c}_{ki}}{\partial \bar{\alpha}} = \begin{bmatrix} u_{ki} & v_{ki} & u_{ki}^2 & u_{ki}v_{ki} & v_{ki}^2 & d_i u_{ki} & d_{ki} v_{ki} & d_{ki} u_{ki}^2 & d_{ki} u_{ki} v_{ki} & d_{ki} v_{ki}^2 \end{bmatrix}. \tag{4.28}$$

The resulting correction function Jacobian $\mathbf{J}_{\bar{\alpha}_k}$ is given by

$$\mathbf{J}_{\bar{\alpha}_k} = \hat{n}'_{ki} \cdot \left(\mathbf{E}_i^0 \begin{pmatrix} \tilde{p}_{kix} \\ \tilde{p}_{kiy} \\ \tilde{p}_{kiz} \\ 0 \end{pmatrix} \frac{\partial \mathbf{c}_{ki}}{\partial \bar{\alpha}} - \mathbf{E}_j^0 \begin{pmatrix} \tilde{p}_{kjx} \\ \tilde{p}_{k jy} \\ \tilde{p}_{k jz} \\ 0 \end{pmatrix} \frac{\partial \mathbf{c}_{kj}}{\partial \bar{\alpha}} \right), \tag{4.29}$$

which can be simplified to

$$\mathbf{J}_{\bar{\alpha}_k} = (\hat{n}'_{ki} \cdot (\tilde{p}'_{ki} \frac{\partial \mathbf{c}_{ki}}{\partial \bar{\alpha}} - \tilde{p}'_{kj} \frac{\partial \mathbf{c}_{kj}}{\partial \bar{\alpha}})), \tag{4.30}$$

where \tilde{p}'_{ki} and \tilde{p}'_{kj} are k^{th} the rotated uncorrected coordinates from P_i and P_j respectively.

4.6.3 Radial Distortion Correction Jacobian

Similar to Equation 4.26 the derivative of the residual function with respect to a radial distortion $\beta_l \in \bar{\beta}$ is given by

$$\hat{n}'_{ki} \cdot (\mathbf{E}_i^0 \frac{\partial p_{ki}}{\partial \beta_l} - \mathbf{E}_j^0 \frac{\partial p_{kj}}{\partial \beta_l}). \quad (4.31)$$

The derivative $\frac{\partial p_{ki}}{\partial \beta_l}$ can once again be calculated using the chain-rule

$$\frac{\partial p_{ki}}{\partial \beta_l} = \frac{\partial p_{ki}}{\partial \mathbf{r}_{ki}} \frac{\partial \mathbf{r}_{ki}}{\partial \beta_l} = \begin{pmatrix} \mathbf{c}_{ki} d_{ki} u_{ki} \\ \mathbf{c}_{ki} d_{ki} v_{ki} \\ 0 \\ 0 \end{pmatrix} \begin{bmatrix} u_{ki}^2 + v_{ki}^2 & (u_{ki}^2 + v_{ki}^2)^2 \end{bmatrix}. \quad (4.32)$$

The Jacobian with respect to the radial distortion $\mathbf{J}_{\bar{\beta}}$ parameters is therefore given by

$$\mathbf{J}_{\bar{\beta}_k} = \hat{n}'_{ki} \cdot \left(\mathbf{E}_i^0 \begin{pmatrix} \mathbf{c}_{ki} d_{ki} u_{ki} \\ \mathbf{c}_{ki} d_{ki} v_{ki} \\ 0 \\ 0 \end{pmatrix} - \mathbf{E}_j^0 \begin{pmatrix} \mathbf{c}_{kj} d_{kj} u_{kj} \\ \mathbf{c}_{kj} d_{kj} v_{kj} \\ 0 \\ 0 \end{pmatrix} \right) \begin{bmatrix} u_{kj}^2 + v_{kj}^2 & (u_{kj}^2 + v_{kj}^2)^2 \end{bmatrix}. \quad (4.33)$$

4.6.4 Full Calibration Jacobian Matrix

In order to solve jointly for the absolute pose, depth correction and radial distortion parameters a joint Jacobian matrix is formed. This is formed by concatenating a number of sparse Jacobian matrices given by

$$\mathbf{J}_{ijk} = \begin{bmatrix} \mathbf{J}_{\alpha_k} & \mathbf{J}_{\beta_k} & \mathbf{J}_{\gamma_{k0}} & \mathbf{J}_{\gamma_{k1}} & \dots & \mathbf{J}_{\gamma_{kN}} \end{bmatrix}, \quad (4.34)$$

where only two $\mathbf{J}_{\gamma_{ki}}$ matrices will be non-zero, given the residual is expressed in terms of two sets of pose parameters $\bar{\gamma}_i$. These Jacobians are also concatenated to form the full joint calibration Jacobian

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_{000} & \mathbf{J}_{001} & \dots & \mathbf{J}_{00N} & \mathbf{J}_{010} & \dots & \mathbf{J}_{\hat{N}\hat{N}N} \end{bmatrix}^T. \quad (4.35)$$

The residuals are also concatenated to form a residual vector

$$\bar{r} = \begin{bmatrix} r_{000} & r_{001} & \dots & r_{00N} & r_{010} & \dots & r_{\hat{N}\hat{N}N} \end{bmatrix}^T. \quad (4.36)$$

This is a slight abuse of notation as \hat{N} , which is the number of correspondences, will vary for each point-cloud pair (i, j) . However this allows the application of a Gauss-Newton optimisation, to solve for all parameters jointly given by

$$\Delta \bar{\zeta} = (\mathbf{J}^T \mathbf{W} \mathbf{J})^{-1} \mathbf{J}^T \mathbf{W} \bar{r}, \quad (4.37)$$

where $\Delta \bar{\zeta}$ is an update vector that is applied to all parameters, and \mathbf{W} is a diagonal weight matrix the same as the one described by Equation 4.14. This process is further demonstrated in Figure 4.1.

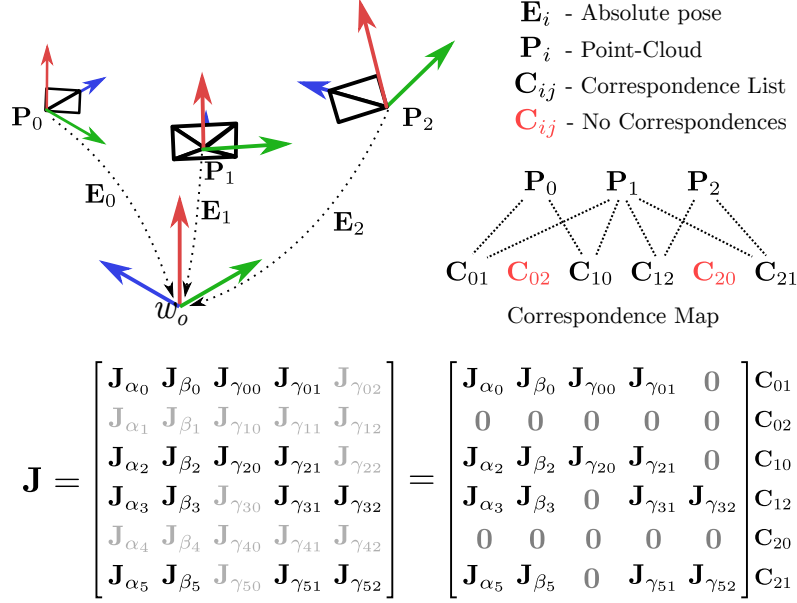


Figure 4.1: Demonstrates the relationship between the absolute poses \mathbf{E}_i , correspondences \mathbf{C}_{ij} and the resulting Jacobian \mathbf{J} for a calibration with 3 images. The red correspondences indicate these two point-clouds contain no overlapping information, which results in empty/zero entries in the resulting Jacobian \mathbf{J} . This also illustrates the fact that correspondences are estimates from P_i to P_j and vice versa, this is related asymmetry of the of the point-cloud resolution, some point-clouds will have a much higher resolution on certain areas meaning more accurate correspondences can be estimated in one direction.

4.7 System Implementation Details

In this section the system implementation is outlined including directions for increasing the chances of a strong resulting calibration.

4.7.1 Calibration Optimisation

As described in Section 4.6 we use a Gauss-Newton optimisation to jointly solve for an absolute pose per point-cloud and a common depth/camera calibration for the depth sensor. As the residual function is defined pair-wise, for each frame pair i, j correspondences are compute \mathbf{C}_{ij} by using a relative pose estimate \mathbf{E}_{ij} . The points in P_j are projected into the frame of P_i and project-and-scan search strategy used to find the closest point, which is assigned as the correspondence. Each set of correspondence \mathbf{C}_{ij} is used to compute the full Jacobian (see Section 4.6) as shown in Figure 4.1. This is used in a Gauss-Newton solver to compute an update jointly for all poses and the common calibration, and iterated until convergence (update is lower than a specified

threshold).

The example shown in Figure 4.1 is intended to be illustrative for three views of a scene. In the example not all point-clouds are overlapping, that is some views share no correspondences, this results in some sparsity in \mathbf{J} where this joint information is not present for in \mathbf{C}_{02} and \mathbf{C}_{20} . The ideal calibration scene will have a nearly dense matrix as this would provide the most mutual information. Figure 4.2, shows some characteristics that should be present in a set of frames that will be used for calibration. This is elaborated upon further in the following section.

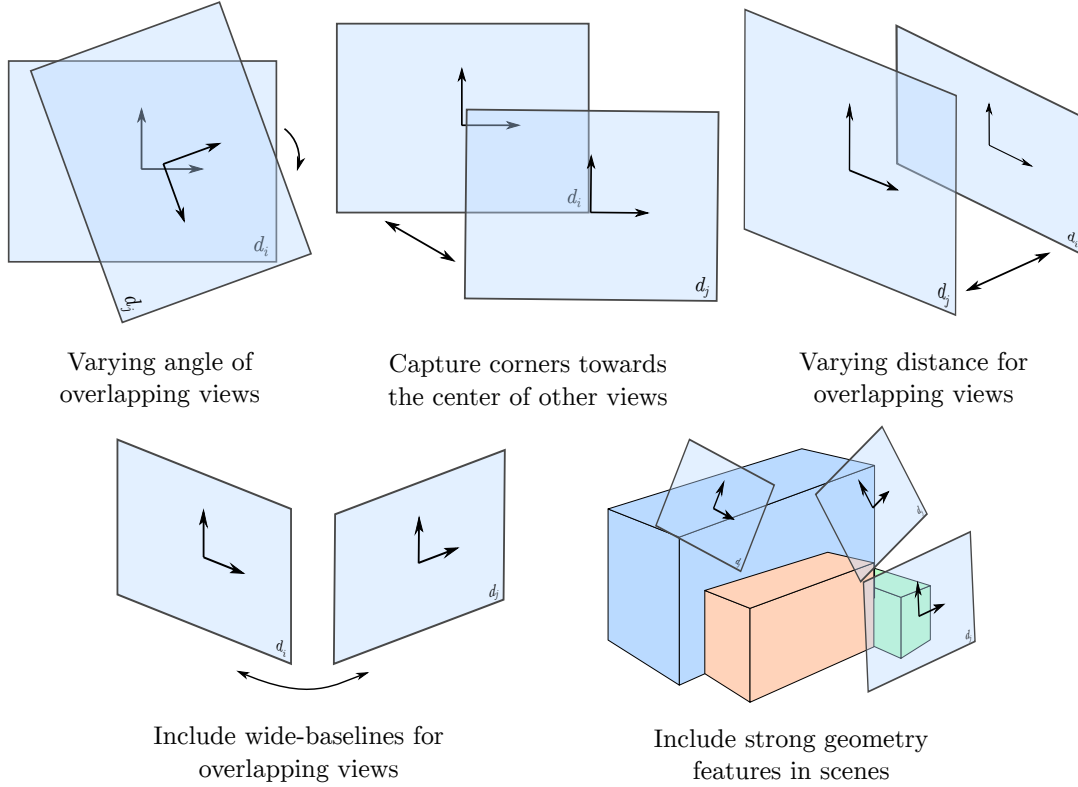


Figure 4.2: Three examples of desirable relationships between the frames to be used for calibration by this system. All contain significant overlap in addition to rotation, translation or varying the distance to a surface. Both the rotation and translation help to constrain the calibration function, while the varying of distance helps to improve the overall performance over a range of distances.

4.7.2 Maximising Calibration Performance

A good scene for calibration should have large visible planes in multiple overlapping frames as these are the most stable source of depth correction. If the scene contains

little or no geometric features ICP will fail to produce good alignments and calibration will fail. Choosing frames is an important point for the user to consider when calibrating, as such a short guide to some desirable properties is shown in Figure 4.2. Varying the angle between frames helps to constrain the overall calibration function by removing a degree of freedom that would be introduced if all frames were for example fronto-parallel. In this situation the calibration function would be free to *bend* the depths orthogonal to the capture orientation without penalising the residual error, which is highly undesirable. Including overlapping frames with the corners of some depth frame centres, attempts to more effectively constrain the edges by balancing the sections of the overlapping frames jointly. Using varying depths for the same points, improves the performance of the calibration over a range of depths. Another subtle point that may affect the stability of our system is the quality of the normal estimates which are effected by sensor noise. To improve stability of our normal estimates we use the method described in Chapter 5 for surface normal estimation.

4.7.3 System overview

The diagram in Figure 4.3 shows a broad overview of the underlying architecture of our calibration system. Our system requires a global pose estimate for each frame as an initial input to the system which is computed as described in Section 4.5. This is provided as is a part of the final system.

The calibration process as shown in Figure 4.3 can be broken down into the following steps:

1. The captured frames are combined into frame pairs.
2. Correspondences are estimated for all frame pairs, given the current calibration and absolute pose estimates.
3. The frame pairs, along with their correspondences are divided evenly amongst available threads.
4. A Jacobian \mathbf{J} for each pair is used to update a local estimate of the approximate weighted Hessian matrix $\mathbf{J}^T \mathbf{W} \mathbf{J}$ for each thread in parallel, as these calculations are separable.
5. The resulting approximate weighted Hessian ($\mathbf{J}^T \mathbf{W} \mathbf{J}$) from each thread is combined into a Joint Hessian matrix.
6. Cholesky decomposition (See Section 3.1.6) is used to calculate an update vector $\bar{\zeta}$.
7. The computed update $\bar{\zeta}$ is applied to all parameters.
8. The corrected coordinates and surface normals are recomputed using the updated correction model.
9. Step 2-8 are repeated until convergence ($|\bar{\zeta}| < \sigma$)

4 DEPTH CAMERA CALIBRATION

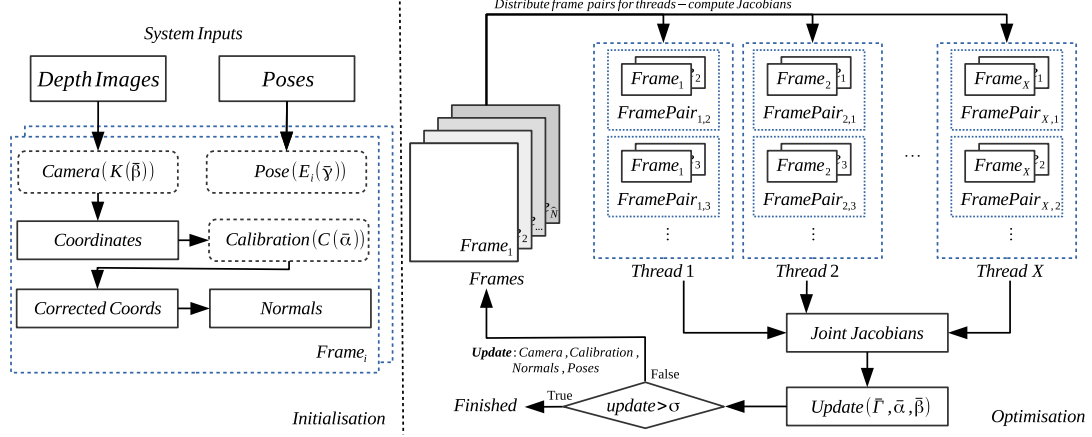


Figure 4.3: The underlying structure of our calibration system, demonstrating the calculation of independent Jacobians across threads which are combined into a Joint Jacobian that contains all jointly available information.

4.8 Calibration Performance Evaluation

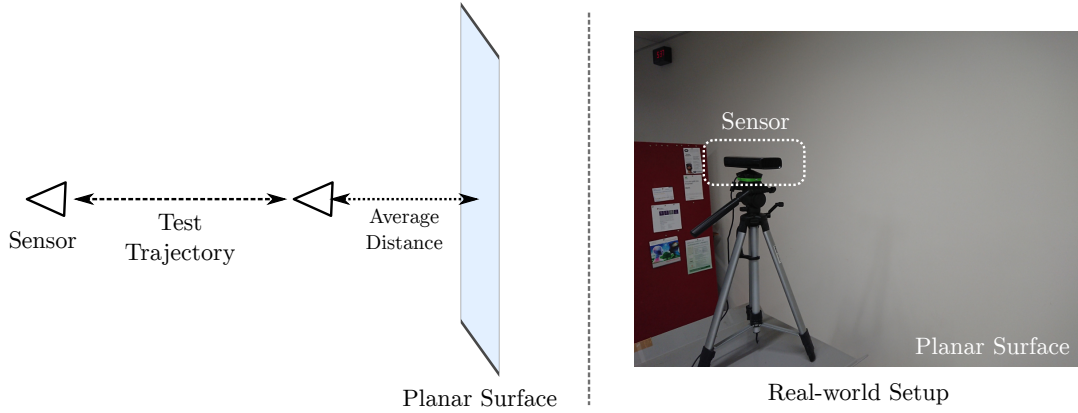


Figure 4.4: **Left:** Planar test setup used to evaluate the performance of the calibration approaches. **Right:** a photo of the real-world test set-up, the sensor is highlighted using a dashed white square, in this case a Kinect V1. The planar surface used was a large flat wall which limited the evaluation distance until the field of view of the camera began to encapsulate more than the wall.

Due to the difficulty of collecting ground truth datasets we use the method of quantitative evaluation proposed in [83] for our calibration method. We use each device to record a test dataset of a flat section of a wall (as shown in Figure 4.4) and measure how well the resulting corrected depth images fit to a plane. This provides a measure of how well the uncalibrated data can be corrected at a range of depths. The results of this experiment for two devices is shown in Figure 4.5 and demonstrates the significant

4 DEPTH CAMERA CALIBRATION

improvement our approach provides over the uncalibrated depth produced by the sensor (particularly for the Kinect V1). We demonstrate a consistent near 50% improvement for our highest order calibration at each depth range.

4.8.1 Choosing the Degree of the Polynomial

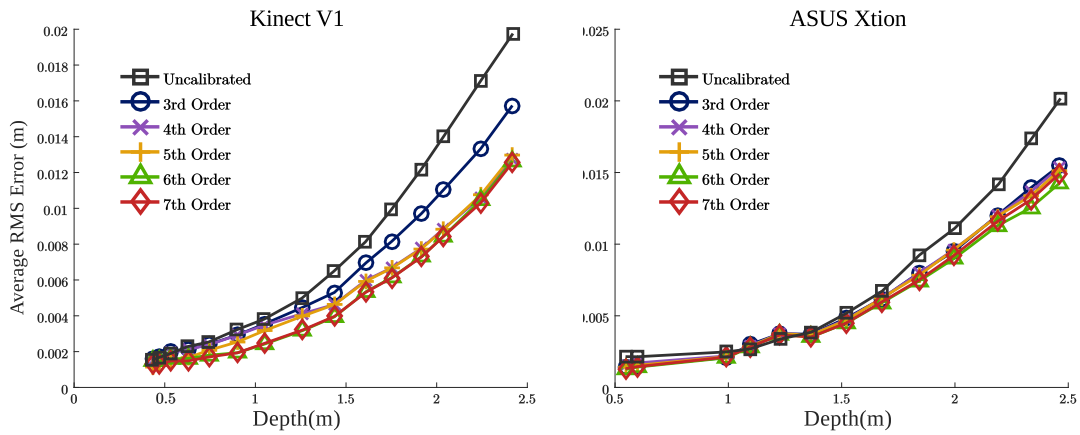


Figure 4.5: The RMS error from fitting a planar surface to corrected data, for Kinect V1(**Left**) and an Asus Xtion(**Right**). This demonstrates the gain in accuracy for higher order polynomials reduces as you increase the degree of the polynomial for our calibration method.

In order to choose the degree of polynomial used by our method we calibrated each device separately with each degree polynomial up to 7th order and examined the improvement on the planar test described in Section 4.8 above. The results of this are shown in Figure 4.5 and demonstrate that increasing the degree of the polynomial above 6th order provides little improvement. Ideally we would like to use the smallest degree polynomial required to give a good calibration as this will reduce computational time, and that appears to be 6th or 7th. Additionally using higher order polynomials can lead to unstable calibrations for areas with less information and larger gradients such as corners.

4.8.2 Visualising Calibration Function

In order to examine how well our calibration function fits the data, we inspect it at various depth values and compare against the uncalibrated error at that same approximate depth from the planar test set. We show the result of this for a depth of 2 meters in Figure 4.6. The correction function fits the data very well, and the calibrated image

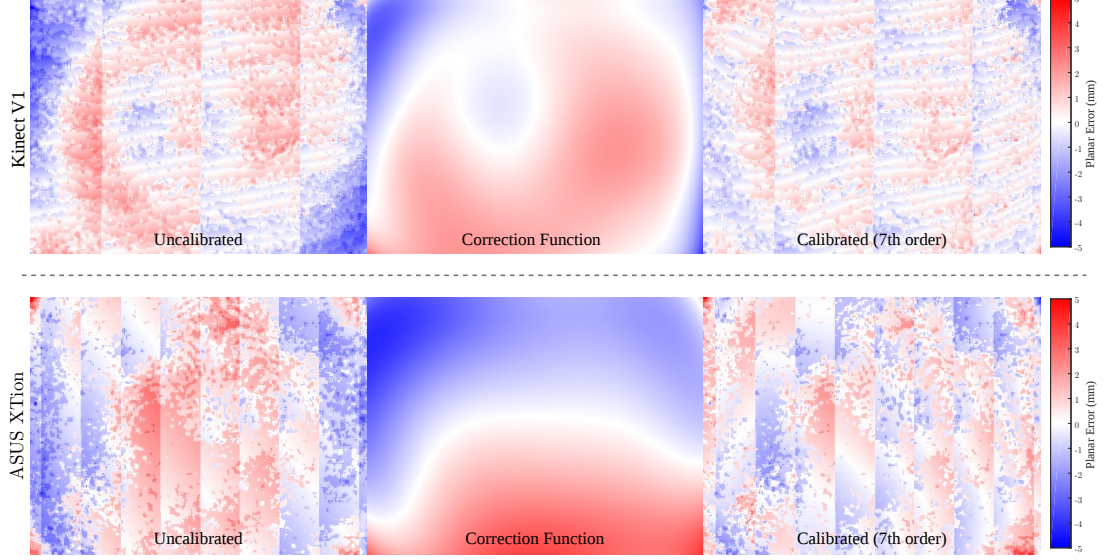


Figure 4.6: Column-wise we have the following data for two different calibrated sensors, **Left:** Planar error for the uncalibrated data. **Middle:** Correction function at the same depth. **Right:** Calibrated depth data. A key is shown to indicate the error at each point in mm.

shows a large reduction in the systematic sensor error, showing this order of the model is appropriate to significantly reduce the structural error. However the data still retains the original high-frequency noise, which varies greatly over time and can be reduced using tradition filtering approaches.

4.8.3 Comparative Performance

We compare the performance of our system against [218] and [83] using the same test described in Section 4.8. We also show a number of qualitative comparisons on real-world data, to demonstrate the comparable performance of our approach. However our approach which is also CPU side only takes minutes, while CLAMS takes many hours. The approach in [218] requires a more specific set of images be collected, including known checker-boards in each image, and each part of the frame must be covered to ensure a successful calibration. Once this dataset is collected this approach does take minutes, given the relatively low amount of data used in the computation. The results are shown in Figure 4.7, and demonstrate our approaches competitive performance against CLAMS([83]) and our improvement upon MIP([218]). For this test we calibrated MIP([218]) using 80 images of checker boards captured at the range of depths and image locations as described in the calibration manual. For the comparison to CLAMS([83]) we calibrated the kinect using 5 minutes of footage, then retrained with a further 5 minutes of footage (10 minutes in total), this final calibration was used in the comparison.

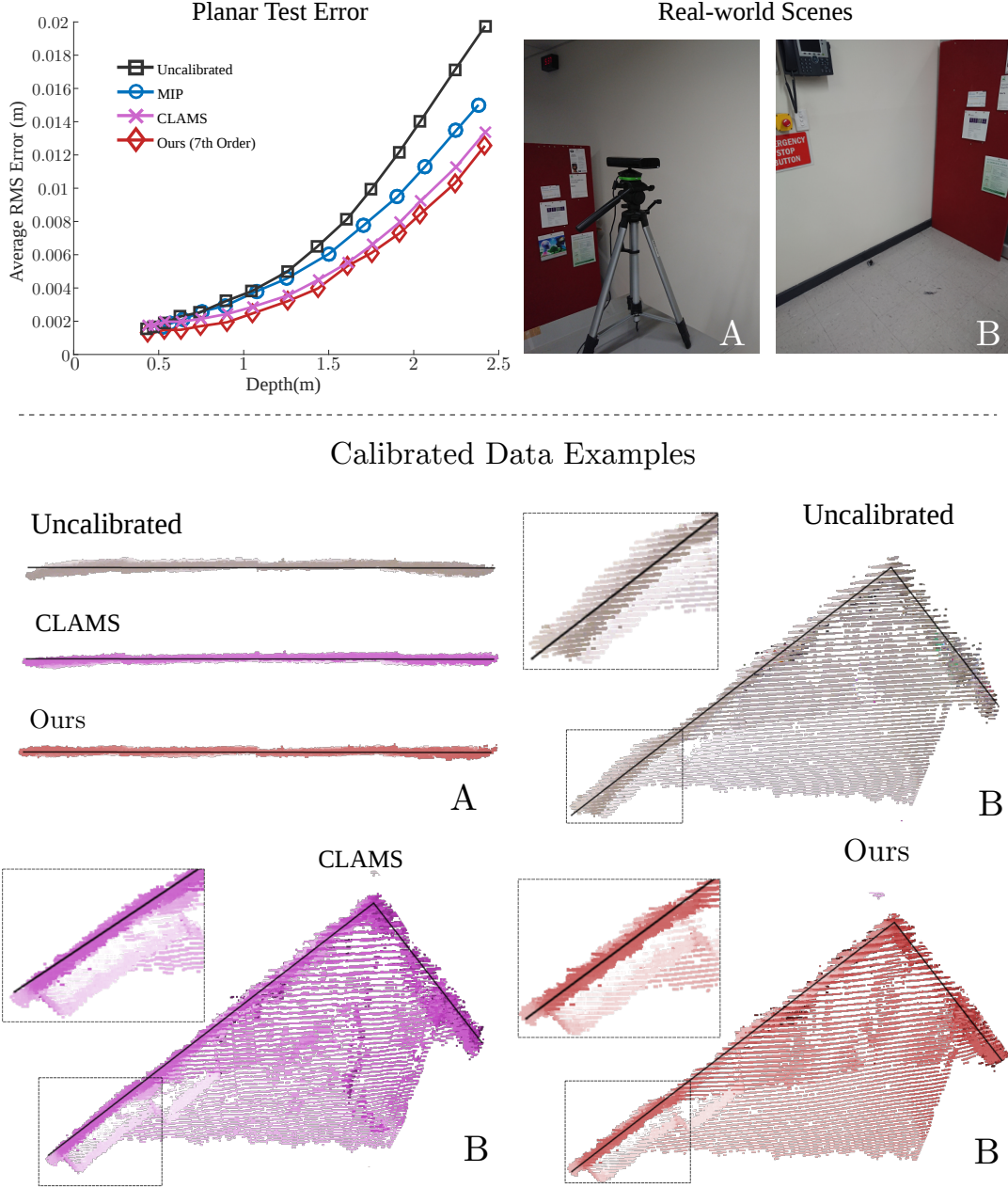


Figure 4.7: Quantitative and qualitative comparisons of our calibration result against the approach from [83] and [218]. **Top-Left:** Results for the same planar test, comparing several approaches each approach. **Top-Right:** Scenes used as qualitative examples for calibration performance. *A* is the planar scene used for planar evaluation, *B* is the corner of a room that contains a prominent right-angle. **Bottom:** Qualitative comparisons of the approach presented in this chapter (Ours) against the approach presented in [83] (CLAMS).

4.8.4 Calibrated Dataset

To further demonstrate the requirement for calibration, we evaluate the accuracy of estimating a known trajectory using set of the uncalibrated and calibrated depth images. We collected a straight line on-rails dataset using the same Kinect used for calibration (the set up is shown in Figure 4.8). We optimise the global poses jointly for this dataset before and after calibration using a joint point-to-plane ICP approach. We show the RMS error in pose rotation in Figure 4.8. The uncalibrated sensor data causes significant pose drift and is visibly reduced after correction using our approach, with approximately a 50% reduction in drift.

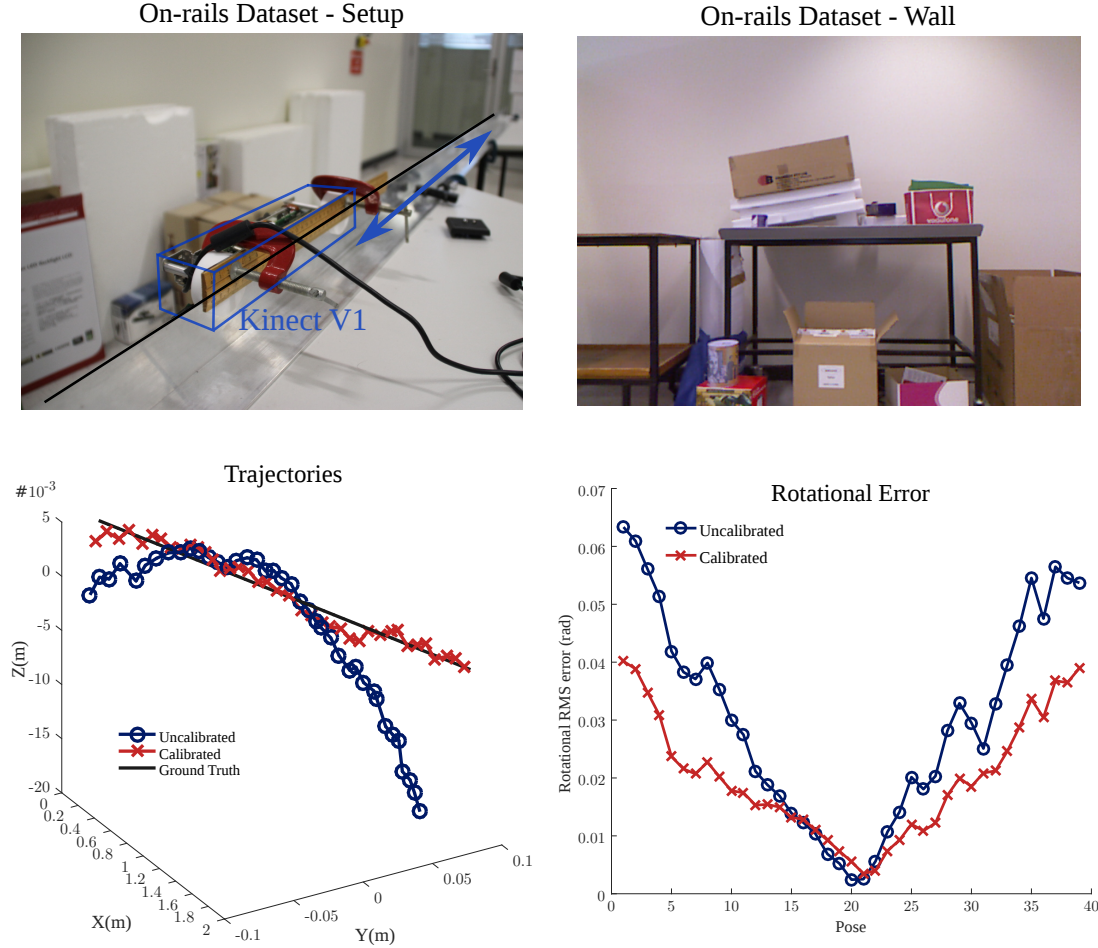


Figure 4.8: **Top-Left:** The on-rails Kinect set up, with an arrow showing the direction of motion. **Top-Right:** A frame from the dataset. **Bottom-left:** Shows the uncalibrated and calibrated trajectories and approximate ground truth. **Bottom-Right:** Rotational error in pose RMS(radians)

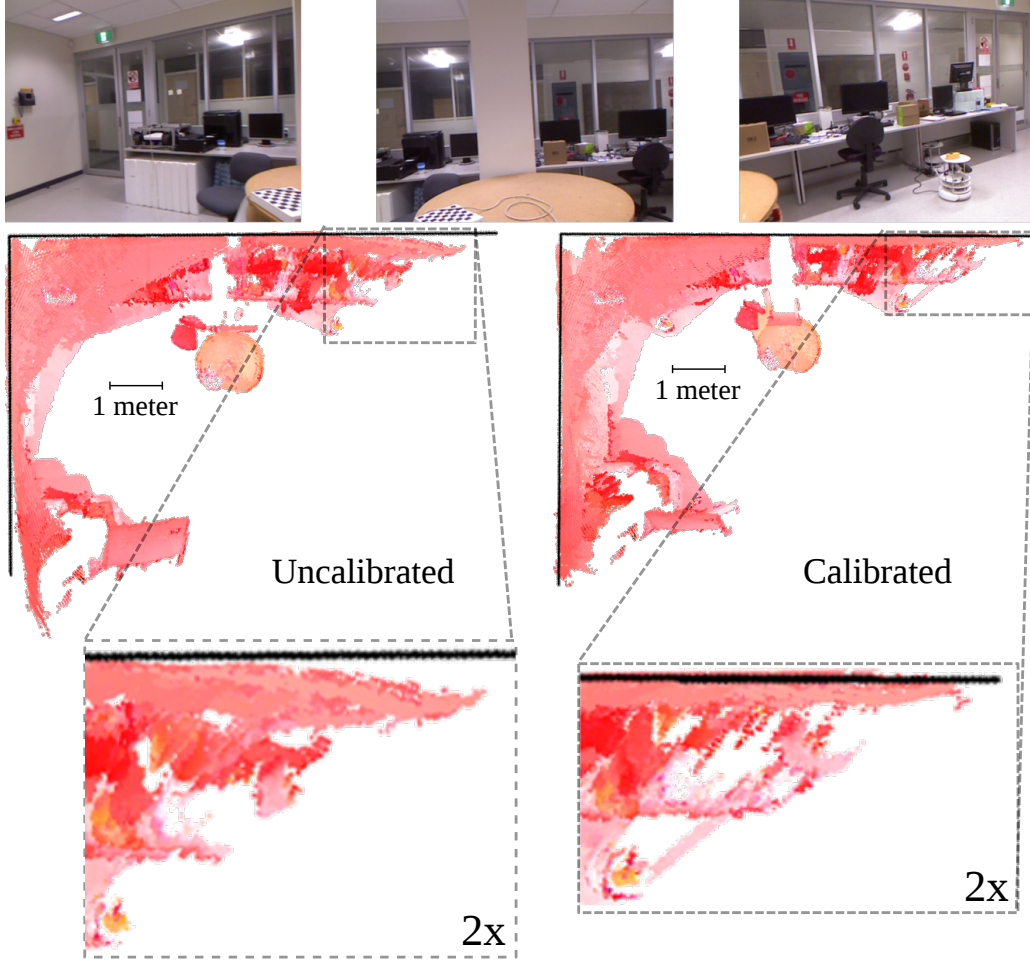


Figure 4.9: **Top:** Several frames from the dataset used to generate the model below. **Bottom-Left:** Model generated using uncalibrated data. **Bottom-Right:** Model generated using our calibrated depth data. The black lines indicate the approximate true position of the room corner and walls, with the scale of the room shown for each model. We include an enlarged section of the corner of each model to demonstrate significant reduction in error using our calibration method.

4.8.5 Improvement to Modelling

Finally we present a qualitative improvement in scene modelling in Figure 4.9. We observe a clear improvement over the original model, after our calibration system has been used to correct the depth sequence. This is most visible in the *curved wall*, which demonstrates the affect of structured sensor noise on model drift and modelling accuracy.

4.9 Conclusions and Future Work

The method of calibration presented in this chapter is a simple and robust approach to low-cost depth sensor calibration. It uses fewer parameters and images than previous similar approaches to produce a robust calibration. Although the quality of the calibration is in general sufficient, there are some unresolved issues. The method is potentially unstable at corner image locations as the calibration relies on a high-order polynomial correction function. A possible remedy to this could be to substitute the polynomial for a more stable but easily differentiable function such as a discrete cosine transform or similar. This could allow a similar number of parameters without introducing the problem of exploding gradients as the normalised camera coordinates increase.

4.10 Code

The source code for the work presented in this chapter was written entirely in C++ and is available at the following repository: <https://github.com/aspek1/RGBDCalibration>

Real-time Principal Curvature Estimation Using Quadrics

This chapter we present a real-time method for dense surface curvature estimation from range data. This is important for a number of tasks in computer vision and robotics, including object segmentation, object recognition and robotic grasping estimation. The approach presented is robust to noise and acts densely across the input depth image, computing accurate metric principal curvature values using GPGPU programming via CUDA. We compare to existing readily available solutions and show comparably favourable performance due to their tendency to amplify noise in the data. Our approach iteratively fits parabolic quadric surface patches to the data, including robust techniques applicable to the highly noisy sensor data. The material and methods presented is largely drawn from the paper [221].

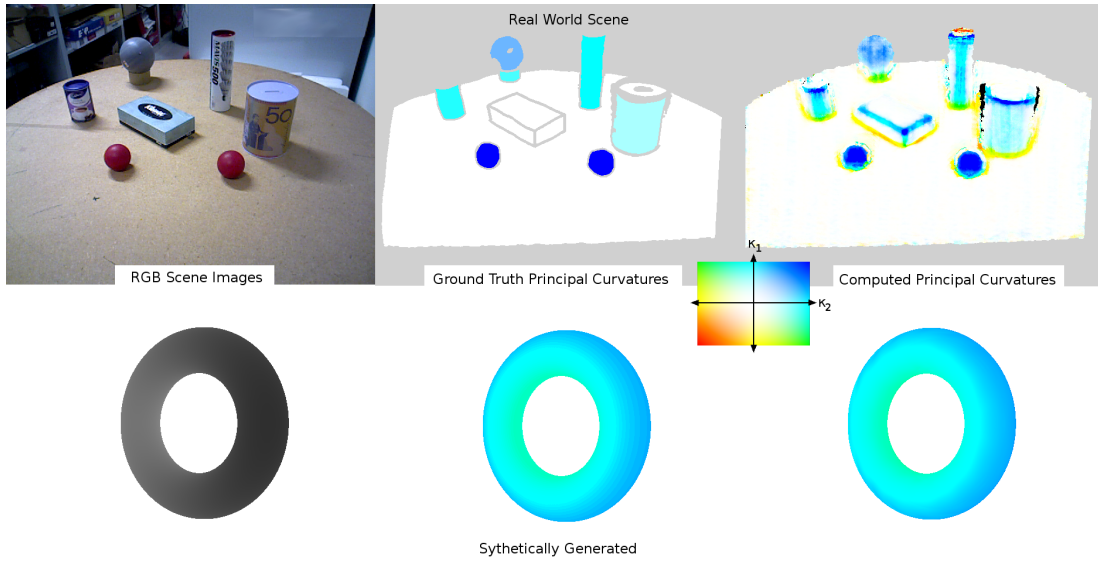


Figure 5.1: The results of curvature estimation using this approach for a real (*top*) and synthetic scene (*bottom*) **Left:** Colour image of the scene. **Middle:** Ground-truth curvature values, coloured using the indicated key. **Right:** Computed curvature values using this approach.

5.1 Motivation and Related Work

As discussed in Sections 1.3 and 3.8 surface curvature is a robust geometric feature that is viewpoint invariant. This makes it useful for various computer vision and robotics applications [203, 93, 92, 98]. In this thesis surface curvature refers the principle curvatures as opposed to Gaussian or mean curvatures. The principal curvatures are the maximal and minimal rates at which the surface normal angle changes at any given point on a surface. This is computable from any 3D surface representation, including meshes, point clouds or and most relevant to this thesis, depth scans from low-cost depth sensors. Surface curvature has been used in computer graphics for applications such as model de-noising and surface alignment [203]. Surface curvature has also been used extensively in robotics for scene/object segmentation [19, 97, 222]. Since the surface curvature is viewpoint invariant it will be the same regardless of the view you observe the surface from, making it ideal for such applications, as well as a basis for geometric feature computation [101, 99].

Existing approaches (at the time this research) that targeted real-time performance compute surface curvature assuming the data used will be of relatively high quality [90, 93, 92] [94], and simply differentiate the surface twice [90], or use very small neighbourhoods with fast approximations [120]. These methods of computing surface curvature are highly susceptible to noise. This led some to either smooth the surface representation [166] or using more robust statistical techniques such as re-weighted least-squares [19] in order to improve the resulting curvature estimates.

Low-cost depth sensors (such as the Microsoft Kinect and ASUS Xtion) have provided a fast method of collecting relatively high resolution dense range images at frame rate for a consumer market. Their operation as well as other variants are described in detail in Section 3.3. Khoshelham *et al.* demonstrated in [76] the data produced by structured-light type sensors is noisy, making it challenging to measure surface curvature accurately using these low-noise methodologies. The approach presented is applicable to all depth sensor types, but will be effected differently by the characteristic noise of the different devices types.

The reason these previous approaches have used very small neighbourhoods is it can be very expensive computationally to accommodate high noise levels in real-time. In order to overcome the computational limitations a General Purpose GPU (GPGPU) programming approach was used. This utilises the large throughput of the GPUs that have been built for graphics applications to perform large numbers of computations in parallel. Lee *et al.* demonstrated in [222] that this enables real-time segmentation and curvature estimation of large scale point clouds. However to enable the real-time section they perform an initial preprocessing step to the data, which is not real-time and must

5 SURFACE CURVATURE FROM QUADRICS

be computed for every point-cloud. The method presented in this thesis requires no additional expensive computations to run in real-time.

Point Cloud Library (PCL) is a widely used open-source library for processing most forms of model data including point clouds and meshes, presented by Rusu *et al.* in [99]. PCL provides multiple methods of computing an estimated curvature value directly from point cloud data, with CPU and GPU implementations. However the estimated metric curvature value is estimated based on a PCA of the change in surface normals (which are also estimated using a PCA of surface changes), and although this is a smoothing method, it can still be greatly effected by surface noise especially when taken to second order. This effectively equates to twice-differentiating the surface, and results in the incorporation of outliers, and points that don't fit the surface greatly effecting the final estimate. To improve the quality of the estimates therefore requires additional computation to separately remove these points from consideration or a *smoothing* of the original data, which increases the computation beyond real-time. Additionally many of the quantities labelled surface curvature are actually the surface variation described by Pauly *et al.* in [223] which is a quantity that is proportional to the mean absolute curvature but non-metric.

5.2 Contributions

The key contribution of this work was a real-time GPU based algorithm for estimating metric curvature values from range images via an iterative quadric surface patch fitting. This method provides several benefits:

- It provides more accurate metric estimates of real-world curvature values than existing methods demonstrated in section 5.5.2.
- It provides smoother and more accurate surface normal estimates compared to surface differentiation by PCA shown in section 5.5.5
- The method is fast and easily able to run at frame-rate as shown in section 5.5.6
- The metric curvature estimates produced by our system can be used to accurately estimate object correspondences across multiple viewpoints as shown in section 5.5.7
- It works well with noisy point cloud data, such as that produced by low-cost RGB-D sensors (like the Microsoft Kinect and ASUS XTion).

5.3 Surface Curvature Using Quadrics

This section describes goes through the specific theory used to create the formulation used in the robust optimisation for surface curvature.

5.3.1 Ordered Point-clouds

A conversion of depth-images to ordered point-clouds is required in order to calculate surface curvature. In this case the point-cloud will be ordered row-wise, or raster-scan image order. We use the standard Camera inverse projection as described in Section 3.2, additionally the depth correction from the calibration described in Chapter 4 is applied. This creates a set of homogeneous coordinates in the frame of the depth sensor.

5.3.2 Second Fundamental Form (II)

As described in Section 3.8, surface curvature can be estimated through something known as the second fundamental form **II**. To reiterate, any local region around a point p on a twice-differentiable surface M can be defined by the equation

$$0 = x\hat{u} + y\hat{v} + z\hat{n} + p, \quad (5.1)$$

where x and y are independent variables, \hat{u} and \hat{v} are orthogonal unit vectors that are tangent to the surface at point p , \hat{n} is the unit-normal vector that is perpendicular to the surface at point p and $z = z(x, y)$ is a function of the independent variables x and y . This allows us to define the second fundamental form of the surface

$$\mathbf{II} = \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix}, \quad (5.2)$$

where

$$A = \frac{\partial^2 z(x, y)}{\partial x^2}, B = \frac{\partial^2 z(x, y)}{\partial x \partial y}, C = \frac{\partial^2 z(x, y)}{\partial y^2}. \quad (5.3)$$

This allows the surface to be redefined in terms of a parabolic surface approximation that passes through $((x, y, z) = (0, 0, 0))$ as

$$z = \frac{A}{2}x^2 + Bxy + \frac{C}{2}y^2, \quad (5.4)$$

which is equivalent to the Taylor series expansion to second-order (Section 3.4.4). Additionally second fundamental form is rotated, such that the vector basis aligns to the principal curvature directions the principal surface curvatures can be extracted directly. This alignment is equivalent to performing the Eigen value decomposition on the matrix as shown

$$\mathbf{II}' = \begin{bmatrix} du' & dv' \end{bmatrix} \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} \begin{bmatrix} du' \\ dv' \end{bmatrix}, \quad (5.5)$$

where the vectors \hat{u}' and \hat{v}' of this surface will be aligned to two the principal curvature directions, and the principal curvature values will be given exactly by the matrix.

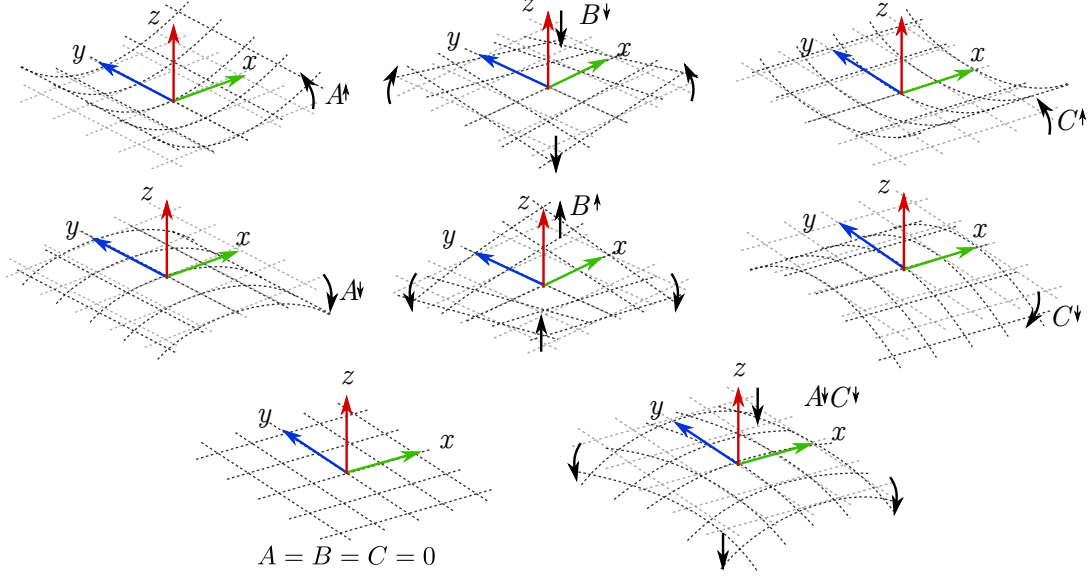


Figure 5.2: Demonstrates the relationship between the quadric coefficients and the effect that has on the resulting quadric surface.

5.3.3 Quadrics

As described in Section 3.4.5, quadrics can be used to represent any second-order surface, and as shown in the previous Section (5.3.2) local surface about a point can be approximated by represented by a parabolic (second-order) surface approximation given by Equation 5.4. This surface approximation can therefore be made using a quadric given by

$$p^T \mathbf{Q} p = \begin{pmatrix} x & y & z & 1 \end{pmatrix} \begin{bmatrix} A & \frac{B}{2} & 0 & 0 \\ \frac{B}{2} & C & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0. \quad (5.6)$$

This formulation limits the quadric to fit to a surface that passes through the origin of the basis, as shown in Figure 5.2. This also demonstrates the effect changing the values of parameters A, B and C has on the resulting quadric surfaces shape.

5.3.4 Creating a Local Coordinate Frame

In order to compute a dense estimate of surface curvature for an ordered point cloud, we take a neighbourhood of points around every valid point in the ordered point-cloud. The neighbourhood ($\bar{N}(p)$) is chosen around any given point p , based on sparse selection

5 SURFACE CURVATURE FROM QUADRICS

filter. This was selected based on an evaluation of performance and speed discussed in Section 5.5.6. To improve the stability of the overall optimisation, the points in the selected neighbourhood are all shifted to be centred around the centre point p_o given by

$$p'_i = p_i - p_o \quad (5.7)$$

where $p_i \in \bar{N}(p_o)$. This is essentially mean-shifting the data and improves many conditions for the optimisation.

In order to define a basis around this point a dense surface normal estimation is required for each point. The normal estimate is computed using a smaller 7×7 neighbourhood $\bar{N}'(p_o)$ of points centred around $p_o = (x_i \ y_i \ z_i \ 1)^T$. This is performed using a simple surface approximation of z_i as a linear function of x_i and y_i using regression. The mean values μ_x , μ_y and μ_z are computed for the patch and then the matrix

$$\mathbf{M} = \begin{bmatrix} \sum_i (x_i - \mu_x)^2 & \sum_i (x_i - \mu_x)(y_i - \mu_y) \\ \sum_i (x_i - \mu_x)(y_i - \mu_y) & \sum_i (y_i - \mu_y)^2 \end{bmatrix}$$

and the vector

$$\bar{v} = \begin{bmatrix} \sum_i (x_i - \mu_x)(z_i - \mu_z) \\ \sum_i (y_i - \mu_y)(z_i - \mu_z) \end{bmatrix}.$$

Setting $\begin{bmatrix} a & b \end{bmatrix}^T = \mathbf{M}^{-1} \bar{v}$ then gives the surface normal as

$$\hat{n} = \frac{1}{\sqrt{1 + a^2 + b^2}} \begin{bmatrix} -a \\ -b \\ 1 \end{bmatrix} \quad (5.8)$$

and this can be cheaply computed from a 2×2 inverse matrix equation. The surface normals are computed densely using a CUDA implementation, and the time taken for this aspect is relatively negligible. The normal allows the computation of an orthogonal basis for the local coordinate frame around each point. The Gram-Schmidt process [224] is used to form an orthogonal basis around every point using the unit normal vector \hat{n} as the z -axis.

5.3.5 Allowing For Further Degrees of Freedom

In it's current form the quadric only has 3 degrees of freedom (3DOF), which technically sufficient to represent any surface approximation, but has some limitations. One key-limitation is the origin of the basis is a fixed-point, all quadric formulations must pass through it. This implies the quadric approximation around p_o must pass through p_o , but in almost all cases there will be some random noise in the selected points position. Additionally the local frame defines around each point is considered to be aligned to

the tangent plane, but the normal computation may be effected by noise or errors. In an effort to combat these issues an intermediate transformation matrix is introduced

$$\mathbf{E}(\theta_x, \theta_y, t_z) = \begin{bmatrix} \ddots & & & 0 \\ & \mathbf{R}(\theta_x, \theta_y) & & 0 \\ & & \ddots & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (5.9)$$

where \mathbf{E} has 3 further degrees of freedom. This transformation is incorporated into the quadric equation from Equation 5.6 as follows

$$p_i'^T \mathbf{E}^T \mathbf{Q} \mathbf{E} p_i' = 0. \quad (5.10)$$

This allows the basis to rotate around the central point p_o as well as move off it to fit the surface quadric. These additional degrees for freedom are illustrated in Figure 5.3, which effectively demonstrates how the translation t_z can be altered to allow the surface to move away from the central point p_o as well as how the rotation allows the surface to rotate but only to a limited degree. Including the rotation parameter θ_z would allow the frame to turn around the normal axis, which is redundant as this can be expressed through the quadrics parameters (A, B and C) as shown in Figure 5.2.

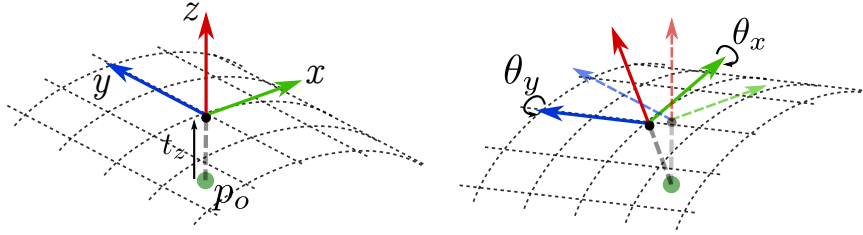


Figure 5.3: Demonstrates the relationship between the translation (t_z) and rotation (θ_x, θ_y) coefficients and the effect that has on the resulting quadric surface. **Left:** the translation t_z allows the surface to move away from the centre point p_o . **Right:** the rotation parameters rotate the surface in order to better align the frame to the points in the neighbourhood $\bar{N}(p_o)$, which additionally compensates for a poor normal estimate.

5.3.6 Iterative Surface Fitting

Given the quadric relationship defined in Equation 5.10, this implies the following error function

$$\epsilon_i = p_i'^T \mathbf{E}^T \mathbf{Q} \mathbf{E} p_i', \quad (5.11)$$

where ϵ_i in this case is the algebraic distance to the quadric surface as shown in Figure 5.4. This allows the definition of the following residual

$$r_k = \sum_{k=0}^{|\hat{N}(p_o)|} |\epsilon_i(p_i)|^2, \quad (5.12)$$

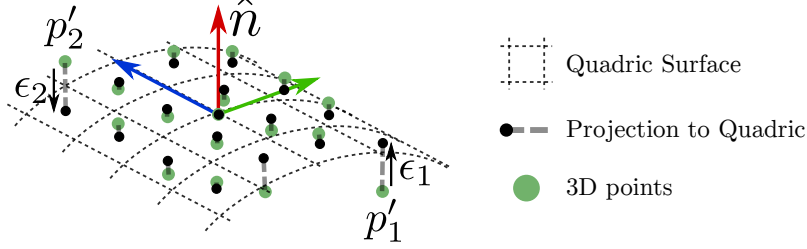


Figure 5.4: A geometric interpretation of the quadric surface \mathbf{Q} , and the individual errors ϵ_i .

which is defined for the size of the neighbourhood ($|\hat{N}(p_o)|$) around the centre point p_o . Once again this can be minimised using a standard Gauss-Newton approach as described in Section 3.5, which requires the derivative of Equation 5.11 with respect to the motion and quadric parameters $\eta_j \in \bar{\eta}$. This derivative can be defined as

$$\frac{\partial \epsilon_i}{\partial \eta_j} = p_i'^T \frac{\partial \mathbf{E}^T}{\partial \eta_j} \mathbf{Q} \mathbf{E} p_i' + p_i'^T \mathbf{E}^T \frac{\partial \mathbf{Q}}{\partial \eta_j} \mathbf{E} p_i' + p_i'^T \mathbf{E}^T \mathbf{Q} \frac{\partial \mathbf{E}}{\partial \eta_j} p_i' \quad (5.13)$$

via the chain rule (see Section 3.4.1), where $\eta_j \in \{A, B, C, \theta_x, \theta_y, t_z\}$. The derivative of the points p_i' is omitted, as it is unrelated to the parameters. This can be separately differentiated by the motion related parameters $\{\theta_x, \theta_y, t_z\}$ and the quadric parameters $\{A, B, C\}$. The derivatives of the residual with respect to the quadric parameters are given by

$$\begin{aligned} \frac{\partial \epsilon_i}{\partial A} &= q_i^T \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} q_i = q_{xi}^2, \\ \frac{\partial \epsilon_i}{\partial B} &= q_i^T \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} q_i = q_{xi} q_{yi}, \\ \frac{\partial \epsilon_i}{\partial C} &= q_i^T \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} q_i = q_{yi}^2, \end{aligned} \quad (5.14)$$

where $q_i = \mathbf{E}_0 p_i'$ is the currently estimate of the point in the neighbourhood given the current transformation estimate \mathbf{E}_0 . The derivatives with respect to the motion parameters can again be computed by using the matrix exponential representation of the transformation \mathbf{E} described in Section 3.6. Allowing the derivatives to be represented

as

$$\begin{aligned}
 \frac{\partial \epsilon_i}{\partial \theta_x} &= q_i^T \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{Q}_0 q_i + q_i^T \mathbf{Q}_0 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} q_i \\
 &= \begin{pmatrix} 0 & -q_{zi} & q_{yi} & 0 \end{pmatrix} q'_i + q_i'^T \begin{pmatrix} 0 \\ -q_{zi} \\ q_{yi} \\ 0 \end{pmatrix} = 2(q'_{zi} q_{yi} - q_{zi} q'_{yi}) \\
 \frac{\partial \epsilon_i}{\partial \theta_y} &= q_i^T \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{Q}_0 q_i + q_i^T \mathbf{Q}_0 \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} q_i \\
 &= \begin{pmatrix} q_{zi} & 0 & -q_{xi} & 0 \end{pmatrix} q'_i + q_i'^T \begin{pmatrix} q_{zi} \\ 0 \\ -q_{xi} \\ 0 \end{pmatrix} = 2(q'_{xi} q_{zi} - q_{xi} q'_{zi}) \\
 \frac{\partial \epsilon_i}{\partial \theta_x} &= q_i^T \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{Q}_0 q_i + q_i^T \mathbf{Q}_0 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} q_i \\
 &= \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} q'_i + q_i'^T \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = 2q'_{zi},
 \end{aligned} \tag{5.15}$$

where $q'_i = \mathbf{Q}_0 q_i$ is the current quadric estimate multiplied by the transformed coordinate. This gives the following Jacobian matrix

$$\mathbf{J}_{\eta j} = \begin{bmatrix} q_{xi}^2 \\ q_{xi} q_{yi} \\ q_{yi}^2 \\ 2(q'_{zi} q_{yi} - q_{zi} q'_{yi}) \\ 2(q'_{xi} q_{zi} - q_{xi} q'_{zi}) \\ 2q'_{zi} \end{bmatrix}^T. \tag{5.16}$$

In order to combat the noise we include a weighting for each point given by

$$w_i(\epsilon_i) = f(p_i) \frac{k^2}{k^2 + \epsilon_i^2}, \tag{5.17}$$

where k is a constant that is proportional to the mean error across previous runs, and $f(p_i)$ is a rejection filter function. The function f is applied to points, in order to reduce

5 SURFACE CURVATURE FROM QUADRICS

the influence of distant outliers, and is given by

$$f(p_i) = \begin{cases} 0 & \text{if } |p_i - p_o| > \iota \\ 1, & \text{otherwise} \end{cases} \quad (5.18)$$

where ι is a fixed maximum allowed distance from the central point. In practice a value of $\iota = 100\text{mm}$ was found to be suitable. As in Section 3.5 the weighting function is used to form the diagonal weight matrix W , the Jacobian matrices are concatenated to form a single matrix, and the residuals are concatenated to form a single residual vector. The update to the motion and quadric parameters is then computed using

$$\Delta \bar{\eta}_k = (\mathbf{J}^T \mathbf{W} \mathbf{J})^{-1} \mathbf{J}^T \mathbf{W} \bar{\epsilon}, \quad (5.19)$$

where the subscript k denotes the k^{th} point in the depth image, meaning a quadric per-point is defined and iterated for separately. The update to A, B and C is purely additive, that is

$$A_{t+1} = A_t + \eta_A, B_{t+1} = B_t + \eta_B, C_{t+1} = C_t + \eta_C, \quad (5.20)$$

while the update to the transformation parameters is applied via a left-multiplication of a matrix exponential as follows

$$\mathbf{E}_{t+1} = e^{\eta_{\theta_x} \mathbf{G}_{\theta_x}} e^{\eta_{\theta_y} \mathbf{G}_{\theta_y}} e^{\eta_{t_z} \mathbf{G}_{t_z}} \mathbf{E}_t. \quad (5.21)$$

5.3.7 Extracting Principal Curvature Values

Once the quadric estimate (\mathbf{Q}) has been computed the principal curvatures κ_1 and κ_2 can be directly obtained using a 2×2 Eigen-value decomposition of the upper-left most sub-matrix of \mathbf{Q} . The principal curvature values can be computed as

$$\begin{aligned} \kappa_1 &= \frac{A+C}{2} + \sqrt{\frac{A+C}{2} - AC + \left(\frac{B}{2}\right)^2}, \\ \kappa_2 &= \frac{A+C}{2} - \sqrt{\frac{A+C}{2} - AC + \left(\frac{B}{2}\right)^2}. \end{aligned} \quad (5.22)$$

5.4 System Summary

The following details the steps performed by our system in computing dense principal curvature estimates using quadric surface fitting.

5 SURFACE CURVATURE FROM QUADRICS

- Point clouds are created from depth images using the inverse projection as described in Section 3.2.
- Surface normals are computed densely for all points using a CUDA implementation of the method described in Section 5.3.4.
- A quadric surface defined in Section 5.3.3 is iteratively fit to an image aligned 37×37 sparse neighbourhood around each point, using a CUDA implementation.
- Principal curvatures are extracted using the method described in Section 5.3.7.
- The initially computed surface normals are refined using the more accurate oriented quadric tangent plane surface alignment.

5.5 System Evaluation

In this section a detailed evaluation of the resulting system is performed, including relevant comparisons to previous approaches.

5.5.1 Ground Truth Curvature Dataset

To evaluate the accuracy of the presented metric curvature estimation method a dataset of ground-truth curvature values was created. The dataset includes both synthetic and manually labelled ground truth images captured using a single low-cost RGB-D sensor to test and compare the robustness of our methodology. The dataset consists of cylinders, spheres and planar surfaces in order to provide an accurate estimate, with out the need for more complex approaches.



Figure 5.5: All objects that appear in the datasets used for real-world evaluation with measured principal curvature values.

The shapes used are pictured in Figure 5.5 and defined in Table 5.1 including their measured principal curvature values. Points without depth measurements and those strong edges and corners in the real-world data was been removed, as they have no corresponding ground truth curvature value. In theory corners and edges are non-differentiable and contain regions of infinite curvature, but in reality this doesn't exist,

5 SURFACE CURVATURE FROM QUADRICS

but at the resolution of the sensor this can-not be expressed adequately. An example of the scenes used in the ground-truth datasets is shown in Figure 5.6, which demonstrates the cluttered scenes used in the evaluation of curvature on the objects and scenes with individual objects focused on evaluating the reliability of this approach against distance.

5.5.2 Measuring System Accuracy

We evaluate our method against several other implementations, including the least-squares method (*Douros*) used in [98], weighted least-squares (*Besl*) from [19] and the method used in Point Cloud Library (*PCL*) compared to our proposed method (*Ours*), to demonstrate the relative improvement of our approach despite its real-time performance. In order to demonstrate a quantitative improvement the approaches are evaluated based on the RMS curvature error defined by

$$\epsilon_{RMS} = \sqrt{\frac{1}{M} \sum_{i=0}^M |(\kappa_1 - \kappa_1^*)^2 + (\kappa_2 - \kappa_2^*)^2|}, \quad (5.23)$$

where κ_i^* and κ_i are the ground truth and estimated i^{th} principal curvature values respectively, and M is the number of points of curvature in the object that is being evaluated. A list of the resulting expected and evaluated curvatures for each object across all datasets is summarised in Table 5.1. This highlights an issue with this approach for spherical objects in real-world scenes, as the approach presented will always align the surface such that the curvature aligns in the maximal directions. This means it will tend to align the data in such a way as to minimise the second principal curvature, making it likely to underestimate κ_2 in practice, which is reflected in the table. However the value κ_1 is consistently well estimated in our configuration.

The results of the evaluation across the real-world dataset items are summarised in Figure 5.7, while the synthetic objects are pictured and the respective approaches evaluated in Figure 5.8. This shows an improvement using our approach in almost all cases. The results of [19] are most similarly to *Ours*, but note this is a more complex fitting strategy and has no real-time approach available. All methods except *PCL* use the same sparsely sampled patch of 37×37 values in view space. The curvature was estimated for *PCL* using a KD-tree search neighbourhood radius of 20mm for real-world and 10mm for synthetic. The choice to use the 20mm radius was made based as it produced the best performance on the dataset. In addition, this size radius corresponds to a patch roughly the same size as the patch used in the other approaches

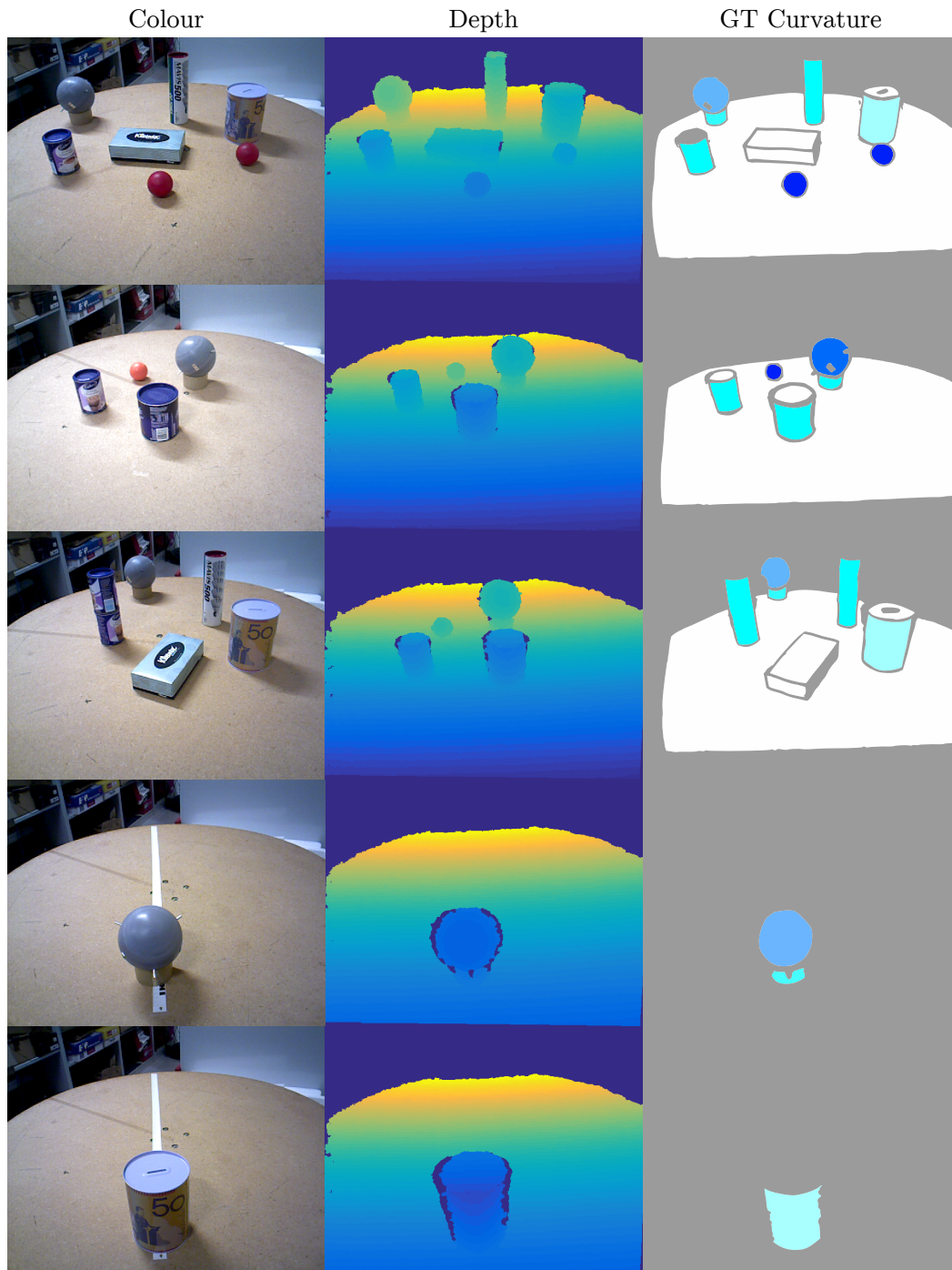


Figure 5.6: Some examples of scenes used in the ground-truth curvature dataset, including the depth and curvature values. This uses the same colour mapping defined in Figure 5.1, where grey is used to represent missing/omitted data.

5 SURFACE CURVATURE FROM QUADRICS

	Expected		Ours		Error	
	κ_2	κ_1	κ_2	κ_1	RMS	σ
Real-World Objects						
Badminton	0	0.029	-9.9e-4	0.023	1.2e-2	8.8e-3
Money	0	0.016	-1.4e-4	0.015	5.5e-3	5.9e-3
ChocB	0	0.019	3.5e-4	0.0172	4.8e-3	3.6e-3
ChocA	0	0.026	-2.3e-4	0.022	9.1e-3	7.4e-3
BallA	0.015	0.015	0.009	0.015	0.01	6.3e-3
Stress	0.032	0.032	0.018	0.029	0.016	0.013
BallB	0.030	0.030	0.015	0.029	0.02	0.013
Synthetic Objects						
Cylinder(9cm)	0.0111	0	0.0111	-1.8e-6	1.2e-4	5.8e-5
Sphere(10cm)	0.010	0.010	0.010	0.010	3.7e-5	1.4e-4
Torus(10cm,3cm)	-	-	-	-	6.3e-4	8.3e-4

Table 5.1: Expected ground truth principal curvature values and those computed by our system averaged across all instances in each dataset. Computed from real-world objects shown in figure 5.5 and synthetic objects shown in figure 5.8.

5.5.3 Evaluating View-Point Invariance

As the approach presented uses a fixed size patch for estimation which may create concerns about consistency over distance and view-point changes of the camera. The results in Figures 5.7 and 5.8 indicate the improved ability of the approach to handle changes in view-point against previous approaches, as these are common in the ground-truth dataset. In order to measure the consistency of the system over large changes in distance we created two datasets from real-world objects, shown in Figure 5.6 in the last two rows. These datasets consist of a fixed camera and an object moving a fixed distance away from the camera between frames. The RMS curvature error is evaluated for each discrete distance and the results summarised in Figure 5.9. The performance of the proposed approach is consistently stronger and more consistent than previous approaches indicating the proposed approach is capable of accurately estimating depth at different distances. This motion has a doubling effect on the noise, as less pixels are used to represent the object reducing the robustness of fitting and the level of quantisation noise in the depth camera will increase as described in Section 3.3.3.3.

5.5.4 Evaluating the Effect of Noise

Robustness to noise was evaluated for each approach by successively increasing surface noise on a synthetically generated sphere while estimating its principal curvature values. The result of this test is shown in Figure 5.10 (Left). Due to the aggressive way the

5 SURFACE CURVATURE FROM QUADRICS

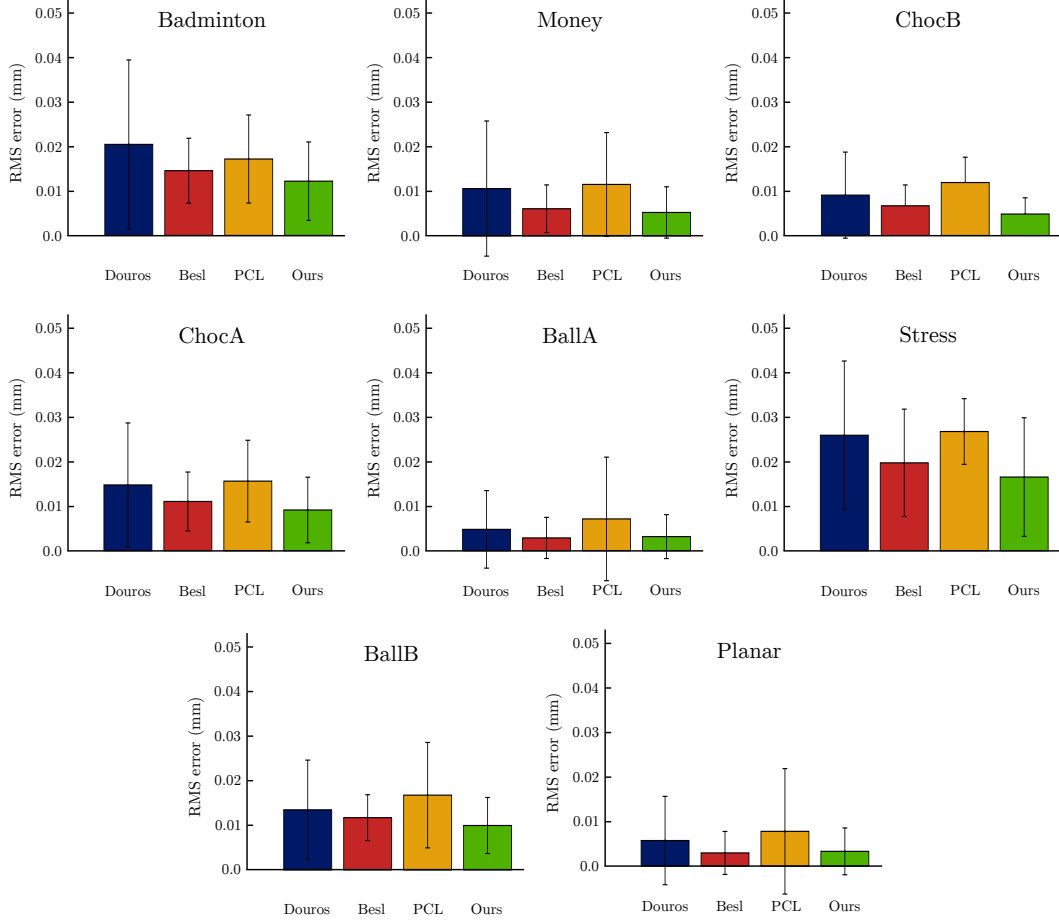


Figure 5.7: RMS error in mm and corresponding standard deviation in estimating principal curvature on real-world data of each object in the dataset for several methods. Least squares (**Douros**), re-weighted least squares (**Besl**), Point Cloud Library, and our iterative method (**Ours**).

noise was added all methods are effected by the added noise. While the quadric based methods provide considerably better estimates for the curvature at each noise level, with PCL consistently underestimating the principal curvatures. Again our method out performs the others in terms of robustness to noise.

5.5.5 Refined Normal Estimation

A qualitative comparison of normal estimations is shown in Figure 5.11. Part of our computation requires the alignment of the quadric surface to fit the points and this is achieved by rotating the surface and altering its curvature. The rotation applied to the

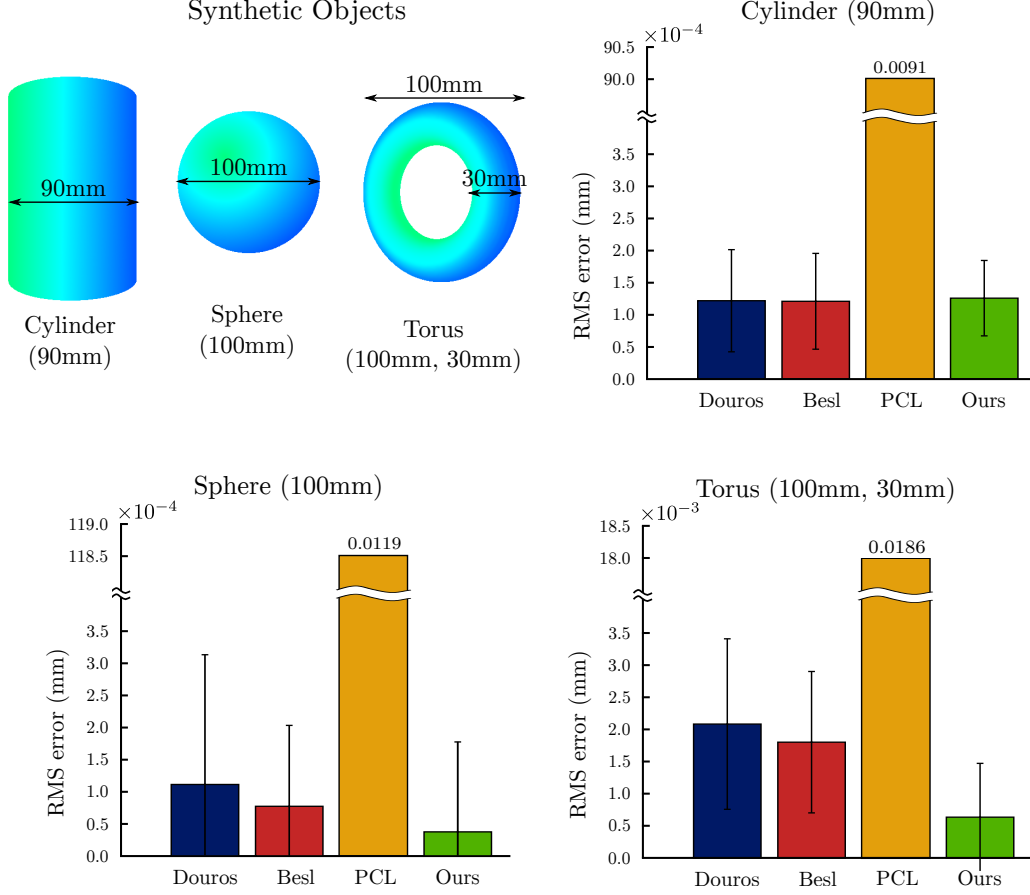


Figure 5.8: RMS error in estimating principal curvature on synthetic objects across a number of synthetically generated viewpoints, comparing each of the evaluated approaches.

quadric surface during the fitting stage can give a more accurate indication of the true direction of the surface normal than the original estimate. Figure 5.11 demonstrates this improvement where **A** shows a comparison to the normals computed by the PCL library and **B** shows the refined normals produced by our method.

A quantitative comparison of the normal estimation robustness was also performed on a synthetically generated sphere, where the known normals were compared for mean absolute error in theta (angle between the normals). The result is shown in Figure 5.10 **Right**. This demonstrates that our system is able to produce smoother and more accurate estimates of normals, than surface differentiation via principal component analysis alone.

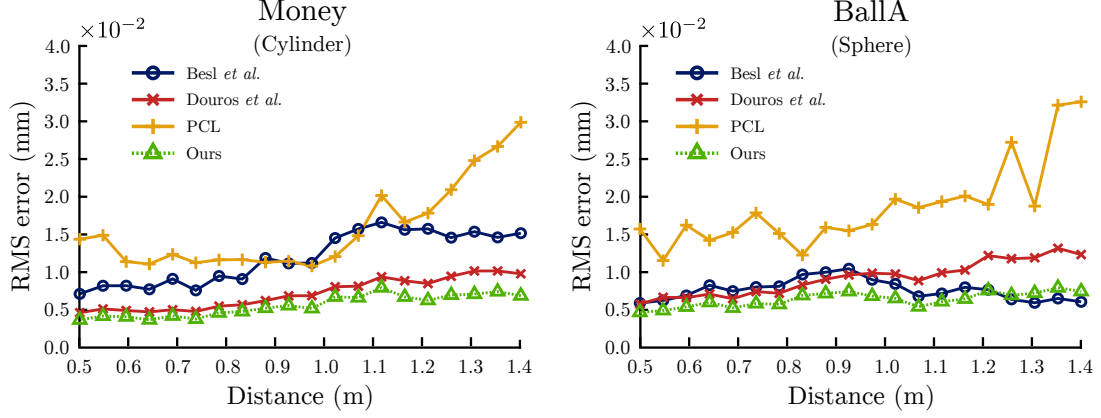


Figure 5.9: RMS error in estimating principal curvature on two real-world scenes where the object of interest was incrementally positioned further and further from the device to investigate how consistent the principal curvature measurements are for the several methods compared.

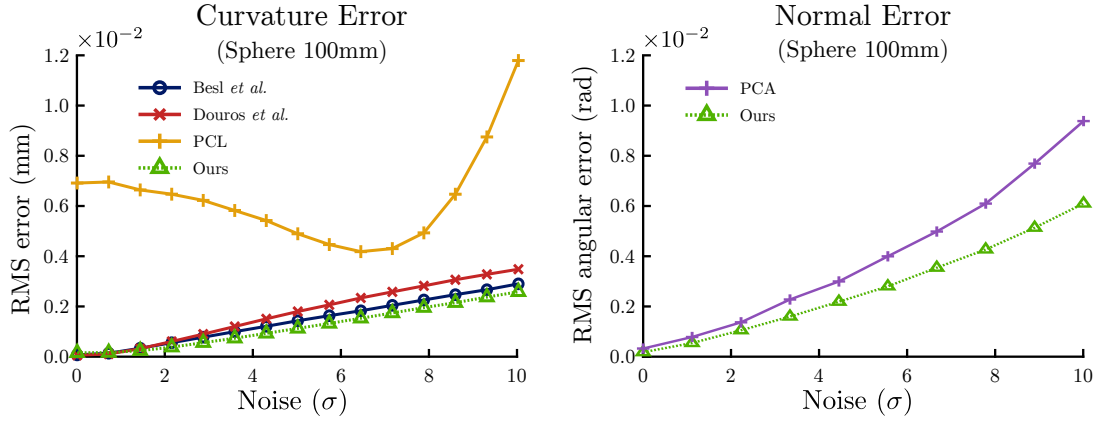


Figure 5.10: Left: RMS error comparison for several methods in estimating principal curvature on a synthetically generated sphere (100mm radius) with increasing random noise added. **Right:** RMS angular error for our method’s normal alignment on the same synthetically generated sphere (100mm radius) with noise added, compared against using the standard PCA approach in PCL. The noise is Gaussian distributed white noise with a standard deviation in mm given by σ .

5.5.6 What’s the Right Size Patch?

The patch-size is very important to the performance of this approach, given it effectively dictates the systems maximum resolution. This is the systems maximum possible curvature value, which is the minimum radius of a sphere or cylinder that can be accurately resolved. The larger the patch size the larger the minimum radius, but the more points the patch can use and thus overcome noisy data. This relationship is demon-

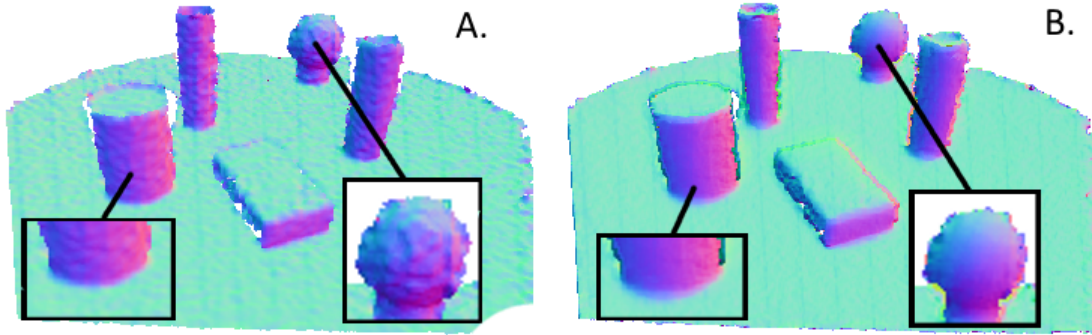


Figure 5.11: Qualitative comparison of normal estimation from real-world data. **A:** Normals estimated using PCA of points differences in a sphere of radius 10mm. **B:** Refined normal estimates from the quadric surface patch calculated in our method. Note that the boundaries still remain sharp even with the added smoothness of the normals.

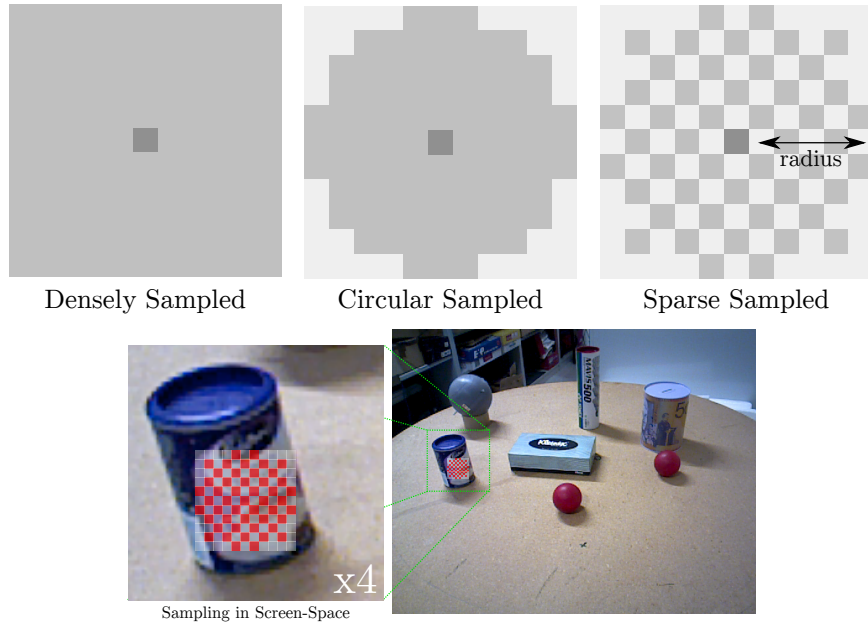


Figure 5.12: Top: three sampling strategies that were each tried. *Dense* samples all points in square patch, *Circular* removes the further away corner points, as they have less influence on the final result anyway, and *Sparse* removes every second point to reduce the computation time further while considering a significantly large neighbourhood. **Bottom:** demonstrates how a patch is sampled from the data, this is actual-size in terms of an 37×37 patch but the sampling pattern is purely illustrative in this case.

strates in Figure 5.13b. The neighbourhood that is used has been described as *sparse* in this chapter, this is demonstrated in Figure 5.12. A sparsely sampled patch is a sparse circular patch. The radius is chosen based on a trade-off between accuracy, resolution

5 SURFACE CURVATURE FROM QUADRICS

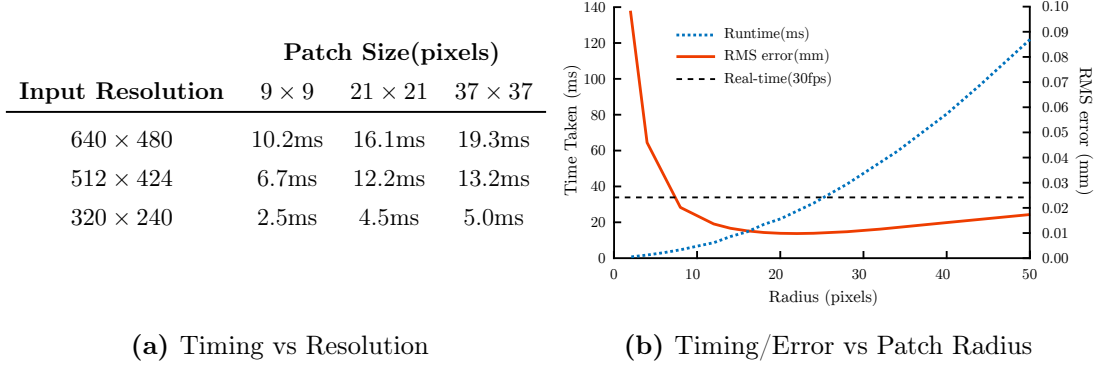


Figure 5.13: 5.13a Timing performance of curvature estimation for our iterative method averaged over several runs at each scale for each patch size.

and time to compute. Figure 5.13b shows that a minimum in the RMS error is between a radius of 16-22, and indicates real-time performance continues up to the top-end of this radius. However to maximise the system’s ability to resolve smaller objects the smallest radius possible should be chosen, as such 18 became the obvious choice.

All tests were performed using a single NVIDIA K40c GPU. Figure 5.13a, demonstrates that the system operates comfortably at frame rate (50-60Hz) for the chosen patch size 37×37 . However, it should be noted that with more accurate data an even smaller patch size would be advisable, allowing for even faster computation times.

5.5.7 Curvature-based Correspondence Estimates

The consistency of this approaches principal curvature estimation was compared to PCL’s using point correspondence estimates on real-world data over different view-points. A CPU PCL implementation, using a K-D tree, with search neighbourhood radius of 20mm was used in all tests. The proposed approach was evaluated with a patch size of 37×37 with *Sparse* sampling for all comparisons.

The test was conducted as follows:

- A random sample point is selected inside each object for every frame in a dataset.
- The closest 400 curvature values to every sample point are computed for each image in the series.
- Each of the closest values will belong to an object in the dataset, part of the table plane or fall on an unclassified region (Background).
- The number of values for each object forms a histogram for each object which shows how consistently each test point matches across multiple views. The ideal

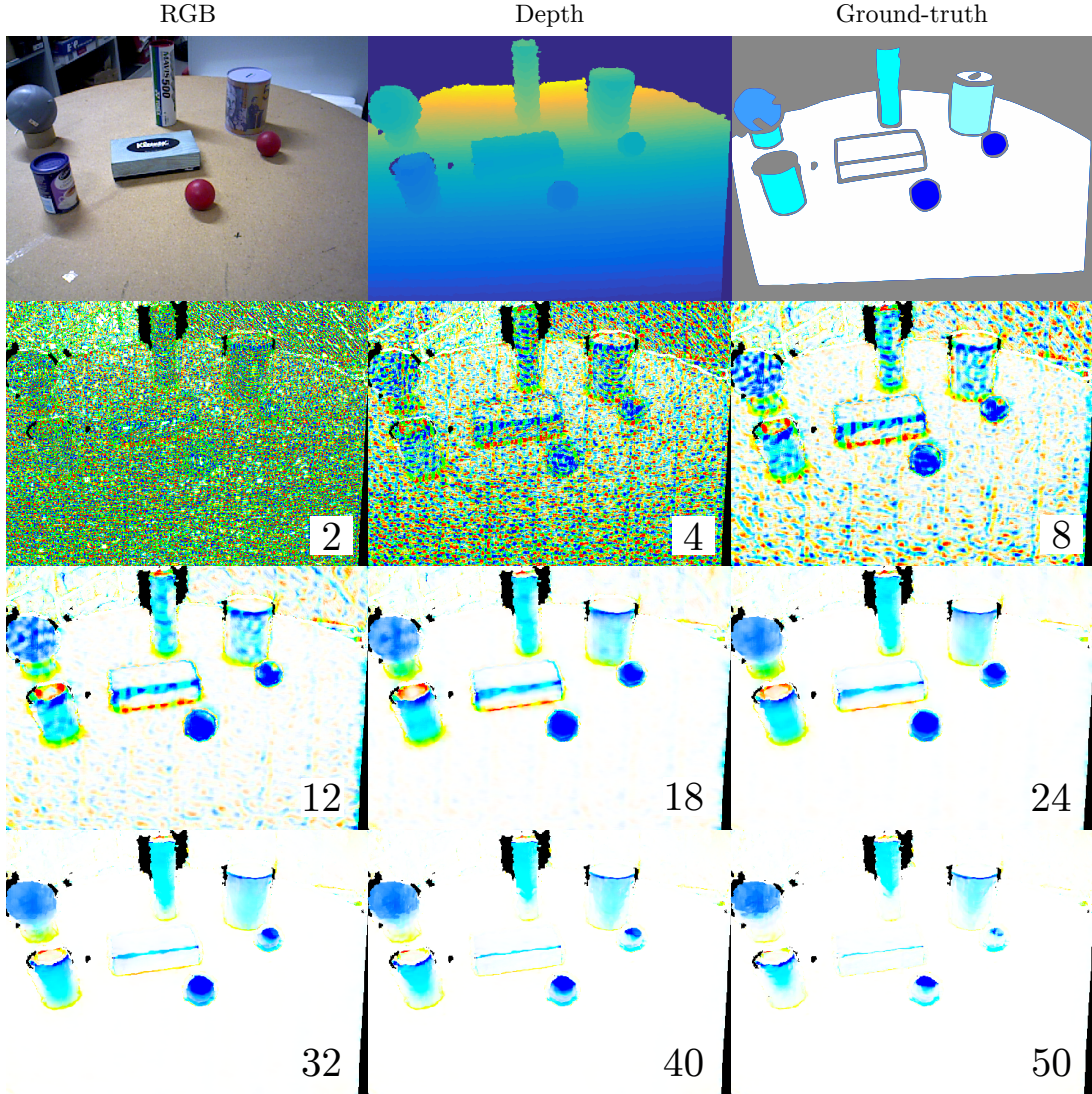


Figure 5.14: Qualitative comparison of the choice of patch size against the quality of the principal curvature estimation. The numbers in the *bottom-right* of each image indicate the sparse-sampled patch radius used. This demonstrates the reduction in resolution as the estimated quadrics become smoother. Between 18-24 strikes a good balance between accuracy and resolution, 18 is quicker to compute and offers the highest resolution and so is the obvious choice.

case is all points belong to the sample point object.

The confusion matrices in Figures 5.15 and 5.16 demonstrate the result of this testing for our method and PCL. These confusion matrices indicate the percentage for each object each of the selected objects matched to. The results in Figure 5.15 clearly indicate

5 SURFACE CURVATURE FROM QUADRICS

	Planar	Background	ChocA	ChocB	Badminton	Money	Stress	BallA	BallB
ChocA	11	16	49	3	25	0	0	0	0
ChocB	5	13	11	66	0	0	0	0	0
Badminton	13	17	25	0	36	2	0	0	2
Money	13	14	5	0	5	58	0	0	0
Stress	5	52	13	5	0	0	22	0	0
BallA	5	10	5	0	2	7	0	67	0
BallB	3	19	3	0	5	3	0	0	63

Figure 5.15: Ours

	Planar	Background	ChocA	ChocB	Badminton	Money	Stress	BallA	BallB
ChocA	36	38	11	2	5	5	0	5	2
ChocB	55	25	7	8	0	0	0	2	0
Badminton	25	38	8	0	11	5	0	7	3
Money	55	22	5	0	2	8	0	2	0
Stress	22	50	5	2	0	0	10	5	0
BallA	41	33	5	0	5	2	0	5	2
BallB	8	27	5	0	8	2	0	8	36

Figure 5.16: PCL

Figure 5.17: The confusion matrix for Ours (Left) and PCL (Right), demonstrating a sensitivity to noise and lack of discriminative power in PCL’s approach with many of the points matching outside the desired regions, while Our method can be used to locate corresponding regions of known or similar curvature across multiple views.

that our system is able to consistently estimate the curvature across different viewpoints and images. This makes it a viable method for estimating corresponding points across a wide-baseline. The results of this test also indicates that smaller objects with high curvature are the most difficult (such as the stress ball) to distinguish for our system and objects with similar curvature profiles will also be more easily confused (such as chocA and badminton). The results for PCL in table 5.16 indicate the relatively poor consistency of their curvature estimation. PCL consistently estimates correspondences to be outside of classified objects (Background). The results of this experiment demonstrates our system could be used to aid object detection or recognition by drastically reducing the size of the search space for correspondence estimation.

5.6 Wide baseline Alignment Through Surface Curvature

The wide baseline alignment problem is an issue that effects systems that use ICP as their underlying alignment method. As described in Section 3.7.6, this occurs when the initial estimate for the aligned surface is outside the narrow convergence well, resulting in a sub-optimal alignment. This situation occurs frequently in practice, as a sensor will often lose track relative to the model and have to relocalise from an unknown position.

As possible solution to this issue a curvature region based alignment strategy was investigated. A simple system was created with the ability to act as a wide-baseline initialiser for the Iterative Closest Point (ICP). This system estimates the wide-baseline transformation using curvature patch correspondences and a rigid body transformation to initialise the ICP computation. We compare this with the performance of ICP without the curvature based transformation initialisation.

5 SURFACE CURVATURE FROM QUADRICS

The system runs as follows:

- Several sample points were randomly selected in reference frame, and used to form suitably sized patches of similar curvature using connected-components.
- Using the curvature values of the randomly selected patches, corresponding patches of similar curvature are found in a target frame.
- Using the patches Cartesian coordinates, moments are extracted to compute the size and center of the patch.
- xyz-centroids for frame pairs are computed using the point cloud for each respective frame.
- Using the corresponding patches a rigid-body transformation is computed using three or more of the corresponding patches.
- The rigid body transformation is used to initialise and an ICP-tracker similar to [18] is used to refine the estimated pose.

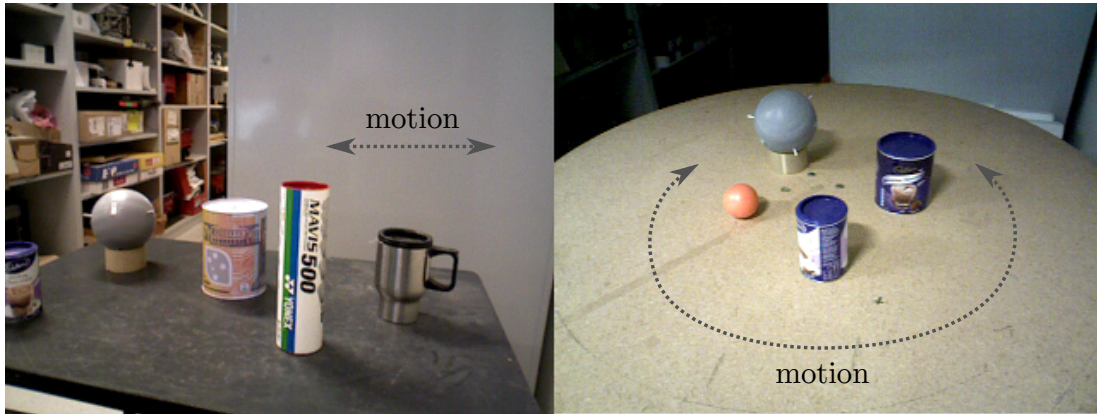


Figure 5.18: **Left:** translation only dataset frame. **Right:** rotation only dataset frame. Points beyond a distance of 1.5m are filtered out of the final point-clouds, as the rotation only dataset rotates the table, and the background remains stationary. The motion of the dataset is indicated in both frames.

The system was tested on two separate toy datasets pictured in Figure 5.18. These datasets experience an isolated motion, translation (Figure 5.18 Left) or rotation (Figure 5.18 Right), to examine the individual effects on the alignment. The resulting contribution of the curvature based alignment is shown in Figure 5.19. In both cases the ICP initialised from identity diverges while the curvature based alignment is able to significantly increase the baseline.

While this validates the basic concept, there is a significant note of caution attached as these are only toy datasets. The datasets used contained large smooth regions with objects neatly separated, and this approach is unlikely to be quite as successful in more realistic scenarios with such a simple approach. The intent of this test was to

5 SURFACE CURVATURE FROM QUADRICS

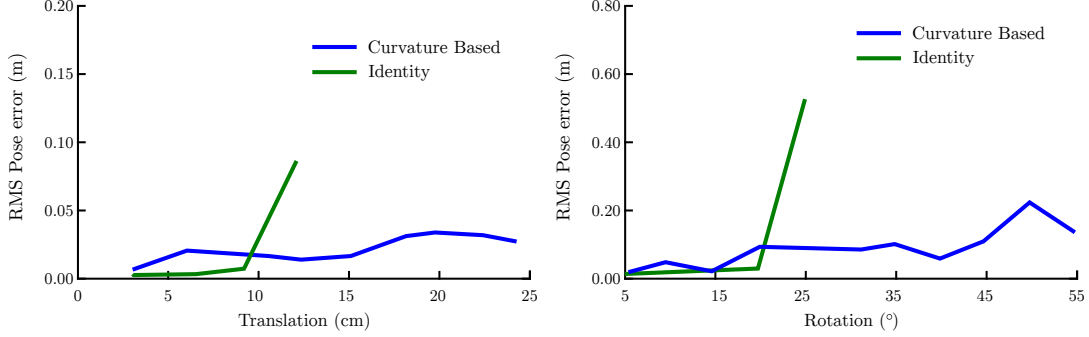


Figure 5.19: Compares the performance of ICP, with and without an initialisation from curvature correspondences for two scenes. **Left:** demonstrates the performance on a scene observed by a translating camera. **Right:** demonstrates the performance on a rotating scene. Both show where the identity initialised test loses track, beyond this point no pose errors are meaningful and are not shown.

demonstrate a feasible example of this work applied to wide baseline estimation.

5.7 Conclusions and Future Work

The method presented in this chapter provides a highly robust real-time solution for surface curvature computation from low-cost depth data. The method provides an improved approach to surface curvature estimation in a compact formulation. The curvature values are shown to have a limited ability to solve the wide-baseline issue in dense mapping approaches. This is a purely illustrative solution in this chapter but could be extended in robustness and a more general approach taken to represent objects using one or more quadrics. As the approach used in this chapter is only intended to be locally correct, a more general quadric would be required for this object representation that is able to represent object level entities.

5.8 Code and Datasets

The code of this project was done in CUDA@/C++ and is available, along with the associated ground truth datasets at the following repository <https://github.com/aspek1/QuadricCurvature>.

Joint Curvature and Pose Optimisation Using Quadrics

This chapter describes the extension of the work in the previous chapter (Chapter 5) on curvature computation from quadrics to include relative pose estimation as part of the optimisation function. The approach presented here jointly solves for both relative pose and an improved estimate of principal curvatures using joint quadric refinement. This is a novel extension of the quadric surface alignment approach, and given previous approaches has already shown the advantages of using quadrics for surface alignment accuracy it seemed an attractive option. We present two implementations to solve this problem, one that uses multiple-frames and jointly solves for all poses and the quadric alignment simultaneously which used a multi-threaded CPU implementation, and a second approach that runs has near real-time ($\approx 5 - 10fps$) performance on pairs of frames.

6.1 Motivation

Surface curvature is a geometric feature that has been shown to be useful for modelling [97, 94, 203], robotic control [55, 225] and segmentation [222]. Surface curvature is a measure of the rate at which the surface normal angle changes while travelling along the surface. This makes curvature a second order derivative, and therefore highly sensitive to noise in the surface. The principal curvatures represent the direction of maximal and minimal curvature at any point. The introduction of low-cost depth sensors, such as the Microsoft Kinect, provided an easy way to collect real-time depth information cheaply. However, the noise present in this sensor [76] has made it challenging to compute accurate curvature values directly [24] due to its high noise sensitivity.

Using quadrics has been shown to improve the robustness of curvature computations [19, 98] and advances in GPGPU programming has allowed for real-time dense estimation of curvature [92, 24]. Using a quadric representation allows the principal curvatures to be extracted directly, through a simple manipulation of quadric parameters described in Section 6.4.2. Additionally constraints can be placed on the quadric representation to reduce the possible representable surfaces to improve the reliability of curvature estimates [24]. A limitation of these methods is that they are designed for use on single-frame of live data or pre-registered low noise dense point clouds. Error in the

registration accuracy or noise in the model can result in poor curvature estimates.

The Iterative Closest Point (ICP) algorithm is considered to be the state of the art method for geometric alignment of point cloud and range data which we briefly summarise in Section 6.4.1. ICP was first described in [12] and [119] and has been extensively used in scan matching and modelling [14, 129], due to its robustness and speed to compute. The algorithm attempts to iteratively minimise the distance between corresponding points in multiple point clouds by rigidly transforming them.

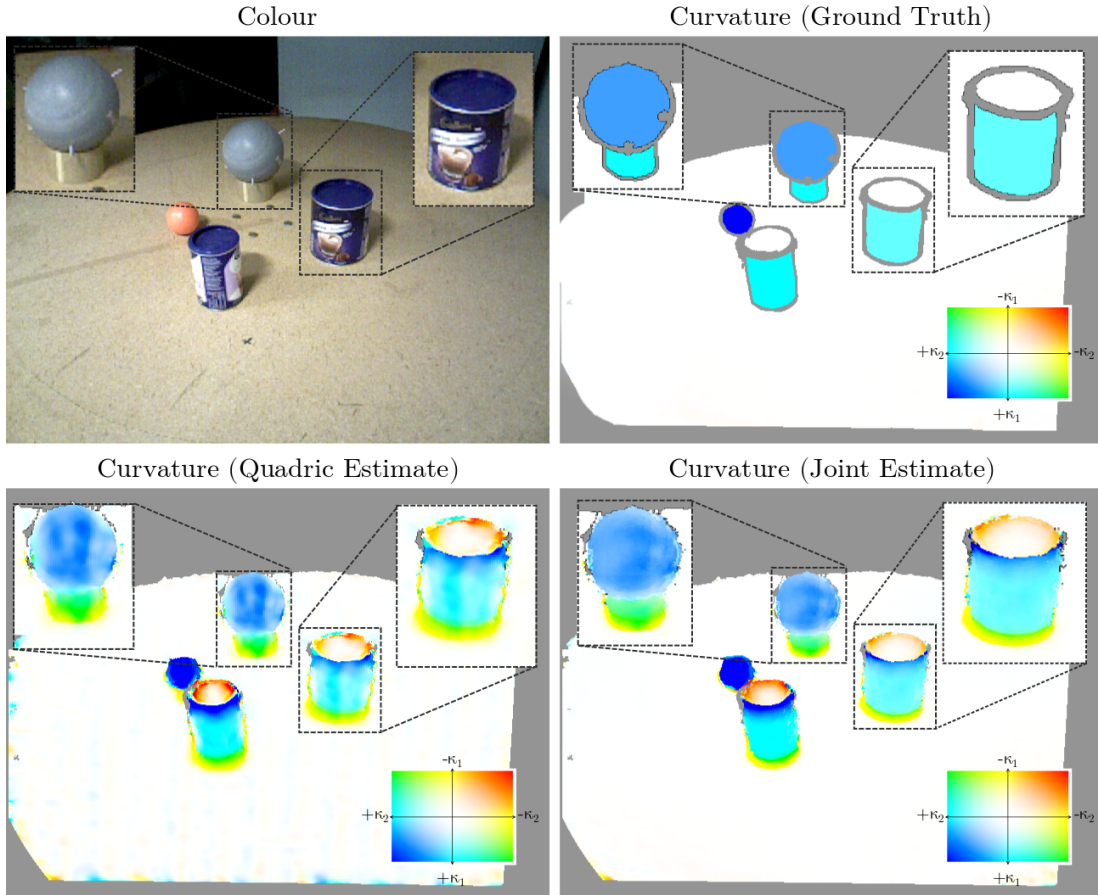


Figure 6.1: *Top Left:* shows the original colour image, *Top Right:* shows the manually labelled ground truth curvature value, *Bottom Left:* shows the result of estimating curvature using a single frame [24], *Bottom Right:* shows the result from this work, using the joint-full multi-frame optimisation. We enlarge the same sections of each image to provide the reader a better view of the differences. Additionally we include a key to indicate how the values of the principal curvatures (κ_1, κ_2) relate to the mapped colours, in the bottom corner of each image.

A more general solution to ICP was described in [203], where Mitra *et al.* show the increased accuracy of using quadric based scan alignment. They also demonstrate the

improved convergence radius of their method over previous implementations of ICP. However, the presented methodology uses a fixed estimate of quadrics for frame-to-frame alignment which can be affected by noise in data, and no real consideration is made for online performance.

6.2 Contributions

In this paper we present a novel joint optimisation approach that:

- Solves for pose alignment and curvature simultaneously using a joint quadric optimisation function (Section 6.5).
- A novel implementation which is intended to be used as an semi-online addition to a tracking system, reducing the relative pose error in adjacent point-clouds while simultaneously improving the current dense estimates of principal curvature values (Section 6.5).
- A novel implementation which is intended to be used as an offline process, performing a joint pose and curvature optimisation across multiple frames after they have been captured and provided with some estimate of their global alignment (Section 6.6).

6.3 Related Work

ICP [12, 119] is a modelling technique important in robotics and computer vision for navigation and mapping [129]. In [12], Besl introduced several different formulations of the minimisation problem including point-to-point, point-to-plane and plane-to-plane, which refer to the method of computing the distance between correspondences. In [120], Rusinkiewicz *et al.* investigate the accuracy and speed of each formulation, finding point-to-plane to be the best choice for most applications leading to its popularity in tracking applications [14, 15, 129, 16]. This method of point-cloud alignment is considered state of the art for aligning multiple overlapping depth scans or point clouds.

ICP relies on largely static scenes in order to operate and can be used to generate both models and accurate ego-motion tracking. Point-to-Plane (Pt-Pl) ICP is used extensively in modern Simultaneous Localisation and Mapping (SLAM) systems as the primary tracking algorithm. In [13] Izadi *et al.* use Pt-Pl ICP to produce a highly accurate and robust dense model, by tracking against the current model, significantly reducing pose drift. This method of tracking is fast to compute using GPUs, while providing reasonably high accuracy. However, this method of tracking against a model requires a large amount of data to be stored and a complex algorithm to move model sections in and out of memory [16].

An improved pose alignment method to point-to-plane ICP was proposed in [203], where Mitra *et al.* use local surface approximations (quadrics) to provide improved pose alignment accuracy. Additionally they demonstrate an improvement in the convergence radius and increased robustness of choosing initial alignment. However, the increase in convergence radius is achieved through a simple heuristic which measures how *correct* the current alignment is and randomly perturbs the current solution if it appears to be converging incorrectly. Additionally their implementation provides no real consideration to online performance of their system.

An important geometric feature used in robotics and computer vision is surface curvature. Curvature has been shown to be useful for many computer vision and robotics applications [97, 94], primarily segmentation [19] and improved modelling [55]. In [19], Besl *et al.* demonstrates that regions of high curvature often occur on object boundaries and this can be used as a reliable basis for segmentation. This idea was extended in [97], where Guillaume *et al.* apply this work to mesh based segmentation. However, these previous approaches either only applied to a single model [93, 94], or rely on the alignment of multiple scan-matches to highly accurate in order to build a single shape model for curvature estimation. Additionally, these previous approaches often assume relatively low-noise in the model, which is not the case when using low-cost depth sensors. In [24], Spek *et al.* presents a method for real-time curvature computation that solves the problem of sensor noise using a quadric based representation.

6.4 Fundamentals

The alignment approach proposed in this chapter is based on the ICP algorithm described in Section 3.7. ICP is also used and detailed in Chapter 4, as part of the joint calibration alignment optimisation defined. In this case the distance function is based on the point-to-surface metric, where the surface is described by a quadric. The quadric surface fitting is identical to the formulation in Chapter 5. For further information please refer to these chapters. As a brief reminder a short explanation of ICP and the quadric surface representation used is provided here.

6.4.1 Iterative Closest Point (ICP)

Iterative Closest Point (ICP) is a technique for aligning two overlapping 3D scans or point clouds of a static scene and is considered state of the art in terms of accuracy and speed. This operates by defining an error function which measures the distance from transformed points in surface 1 (\bar{S}_1), to a small plane patch on surface 2 (\bar{S}_2). In point-to-plane ICP the Euclidean planar distance is used, hence a planar estimate of the surface at any point is required. Thus, a normal is computed for every point $q \in \bar{S}_2$,

using a small neighbourhood $\bar{N}(q)$ around it. For each point $p_i \in \bar{S}_1$ a transformation estimate \mathbf{E} is applied to find the point $\mathbf{E}p_i = q'_i$, which is used to find the closest/-corresponding point $q_i \in \bar{S}_2$. The error is computed using the point-to-plane distance between q_i and p_i defined as

$$r_i = \hat{n}_i^T (\mathbf{E}p_i - q_i), \quad (6.1)$$

where \mathbf{E} is the transformation from the surface \bar{S}_1 to \bar{S}_2 such that the points $p_i \in \bar{S}_1$ are in the same frame as the points $q_i \in \bar{S}_2$. Across all points in \bar{S}_1 this forms the total error

$$R = \sum_{i=0}^{\hat{N}} \hat{n}_i^T (\mathbf{E}p_i - q_i), \quad (6.2)$$

where N is the number of points in \bar{S}_1 . To minimise this a standard Gauss-Newton approach can be used and an update to the motion parameters $\bar{\gamma}$ can be computed as follows

$$\Delta \bar{\gamma} = (\mathbf{J}^T \mathbf{W} \mathbf{J})^{-1} \mathbf{J}^T \mathbf{W} \bar{\epsilon}, \quad (6.3)$$

where individual Jacobians \mathbf{J}_i that are concatenated to form \mathbf{J} are computed as in Chapter 4 as

$$\mathbf{J}_i = \frac{\partial e_i}{\partial \bar{\gamma}} = [\hat{n}^T, (\bar{q}_i \wedge \hat{n})]. \quad (6.4)$$

Finally the subsequent update to the pose estimate is applied via a matrix exponential as follows

$$\mathbf{E}_{t+1} = e^{\sum_{i=k}^1 \Delta \gamma_i G_k} \mathbf{E}_t. \quad (6.5)$$

6.4.2 Quadric Surface Representation

As with the implementation in [24] described in Chapter 5 curvature is computed using a quadric representation, which is defined by

$$p_k^T \mathbf{E}^T \mathbf{Q} \mathbf{E} p_k = 0, \quad (6.6)$$

where

$$\mathbf{Q} = \begin{bmatrix} A & \frac{B}{2} & 0 & 0 \\ \frac{B}{2} & C & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & 0 \end{bmatrix}, \quad \mathbf{E} = \begin{bmatrix} & & & 0 \\ \mathbf{R}(\theta_x, \theta_y) & & & 0 \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.7)$$

are the parabolic quadric matrix and transformation matrix used to align the frames orientation to the true tangent plane. This is iteratively fit using Equation 6.6 as an error function given by

$$\epsilon_k = p_k^T \mathbf{E}^T \mathbf{Q} \mathbf{E} p_k. \quad (6.8)$$

Then the principal curvatures (κ_1, κ_2) can be extracted using a simple Eigen value decomposition of the upper-left 2×2 matrix of \mathbf{Q} . The update to the motion and quadric parameters $\Delta\bar{\eta}$ can also be computed via a Gauss-Newton optimisation.

6.5 Joint Frame-to-Frame Optimisation

The initial motivation to this work was to improve the quality of frame-to-frame alignment in a real-time system. It has been shown in [203] that point-to-surface alignment is possible and improves the quality of ICP over point-to-plane significantly. However the computation of this approach was not real-time and robotic systems have a heavy focus on real-time performance. To this end the goal became an initial implementation that could produce high accuracy frame-to-frame alignments in approximately real-time. This section details the implementation of such a system.

In order to align multiple frames the error function in Equation 6.8 was extended to include a single relative pose as follows

$$\epsilon'_k = q_k^T \mathbf{E}_{12}^T \mathbf{Q}'_{p_i} \mathbf{E}_{12} q_k, \quad (6.9)$$

where \mathbf{E}_{12} is the current estimated transformation from \bar{S}_2 to \bar{S}_1 . In this formulation $\mathbf{Q}'_{p_i} = \mathbf{E}_{p_i}^T \mathbf{Q}_{p_i} \mathbf{E}_{p_i}$ is the quadric at point p_i , expressed in short form for convenience. $q_k \in \bar{S}_2$ are the points in an neighbourhood $\bar{N}(q'_i)$ around the closest point q'_i after projection to the frame of \bar{S}_2 via $\mathbf{E}_{21}p_i$, where p_i is the central point. The transformation is considered to be a function of the motion parameters $\bar{\gamma}_{12}$, while the quadric is a function of the quadric parameters $\bar{\eta}_i$, where each point $p_i \in \bar{S}_1$ has a quadric representation. This defines a residual in terms of the neighbourhood $\bar{N}(q'_i)$ as

$$r'_k = \sum_{q_k \in \bar{N}(q'_i)} (\epsilon'_k(q_k))^2. \quad (6.10)$$

The original error function from Equation 6.8 forms a residual function in terms of the neighbourhood $\bar{N}(p_i)$

$$r_k = \sum_{p_k \in \bar{N}(p_i)} (\epsilon'_k(p_k))^2. \quad (6.11)$$

Both residual functions are in terms of the joint quadric parameters, but only r'_k is in terms of the motion parameters. This implies that a single frame that defines \bar{S}_1 is the source of the quadrics which is the case. The residual functions are used to refine a single set of quadrics and a single relative pose estimate via a Gauss-Newton optimisation as described in Section 3.5.

Gauss-Newton optimisation requires the computation of the error function derivatives with respect to the motion and quadric parameters. This is performed separately for

each residual function, for ϵ_k and ϵ'_k the derivatives with respect to the quadric parameters $\bar{\eta}_{p_i}$ are identical to those defined in Chapter 5 (see Section 5.3.6) so we omit the derivation here. The resulting Jacobian matrix is defined by

$$\mathbf{J}_k^{\bar{\eta}}(p) = \begin{bmatrix} p_x^2 \\ p_x p_y \\ p_y^2 \\ 2(p'_z p_y - p_z p'_y) \\ 2(p'_x p_z - p_x p'_z) \\ 2p'_z \end{bmatrix}^T, \quad (6.12)$$

where $p \in \{p_k, q_k\}$ is either p_k or q_k , and $p' = \mathbf{E}_{p_c} p$ is the point in the neighbourhood $\bar{N}(p_c)$ transformed by the matrix \mathbf{E}_{p_c} of the associated quadric \mathbf{Q}_{p_c} around the central point p_c . The derivatives of ϵ'_k with respect to the motion parameters $\bar{\gamma}_{12}$ are given by

$$\begin{aligned} \frac{\partial \epsilon'_k}{\partial \gamma_i} &= q_k^T \frac{\partial \mathbf{E}_{12}}{\partial \gamma_i}^T \mathbf{Q}'_{p_i} \mathbf{E}_{12} q_k + q_k^T \mathbf{E}_{12}^T \mathbf{Q}'_{p_i} \frac{\partial \mathbf{E}_{12}}{\partial \gamma_i} q_k \\ &= q_k^T \mathbf{E}_{12}^T \mathbf{G}_i \mathbf{Q}'_{p_i} \mathbf{E}_{12} q_k + q_k^T \mathbf{E}_{12}^T \mathbf{Q}'_{p_i} \mathbf{G}_i \mathbf{E}_{12} q_k \end{aligned} \quad (6.13)$$

via the chain-rule (see Section 3.4.1). Where \mathbf{G}_i defines the i^{th} generator of the Lie-group $\mathbf{SE}(3)$ (see Section 3.6). The final computed derivatives can be used to construct the Jacobian matrix $\mathbf{J}_{\bar{\gamma}_{12}}$ are given by

$$\mathbf{J}_{qk}^{\bar{\gamma}} = \begin{bmatrix} 2(q''_{kx}) \\ 2(q''_{ky}) \\ 2(q''_{kz}) \\ 2(q'_{ky} q''_{kz} - q'_{kz} q''_{ky}) \\ 2(q'_{kx} q''_{kz} - q'_{kz} q''_{kx}) \\ 2(q'_{ky} q''_{kx} - q'_{kx} q''_{ky}) \end{bmatrix}^T, \quad (6.14)$$

where q'_{kx} is the x component of the transformed coordinate $q'_k = \mathbf{E}_{12} q_k$, and q''_{kx} is the x component of the transformed and quadric multiplied coordinate $q''_k = \mathbf{Q}'_{p_i} q'_k$. For any point $p_i \in \bar{S}_1$ this provides two sets of Jacobian matrices this provides two sets of Jacobian matrices. The Jacobians of the joint quadric parameters can be formed by concatenation as follows

$$\begin{aligned} \mathbf{J}_{p_i}^{\bar{\eta}} &= \begin{bmatrix} \mathbf{J}_0^{\bar{\eta}}(p_0) & \mathbf{J}_1^{\bar{\eta}}(p_1) & \dots & \mathbf{J}_N^{\bar{\eta}}(p_N) \end{bmatrix}^T \\ \mathbf{J}_{q_i}^{\bar{\eta}} &= \begin{bmatrix} \mathbf{J}_0^{\bar{\eta}}(q_0) & \mathbf{J}_1^{\bar{\eta}}(q_1) & \dots & \mathbf{J}_N^{\bar{\eta}}(q_N) \end{bmatrix}^T, \end{aligned} \quad (6.15)$$

where $\mathbf{J}_{p_i}^{\bar{\eta}}$ and $\mathbf{J}_{q_i}^{\bar{\eta}}$ define the Jacobian for all points in the neighbourhood of the centre points p_i and q'_i respectively, and N is the size of the neighbourhood \bar{N} vector. The

matrix for the motion Jacobian can be expressed as

$$\mathbf{J}_i^{\tilde{\gamma}} = \begin{bmatrix} \mathbf{J}_{q0}^{\tilde{\gamma}} & \mathbf{J}_{q1}^{\tilde{\gamma}} & \dots & \mathbf{J}_{qN}^{\tilde{\gamma}} \end{bmatrix}^T. \quad (6.16)$$

In this case $\mathbf{J}_i^{\tilde{\eta}}$ is twice the length of $\mathbf{J}_i^{\tilde{\gamma}}$, as it involves the points from two neighbourhoods. The Jacobian matrices can once again be combined into the larger Jacobian matrix \mathbf{J}

$$\mathbf{J} = \begin{bmatrix} \mathbf{0} & \mathbf{J}_{p0}^{\tilde{\eta}} & \mathbf{0} & \ddots & \ddots & \mathbf{0} \\ \mathbf{J}_0^{\tilde{\gamma}} & \mathbf{J}_{q0}^{\tilde{\eta}} & \mathbf{0} & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{J}_{p1}^{\tilde{\eta}} & \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{J}_1^{\tilde{\gamma}} & \mathbf{0} & \mathbf{J}_{q1}^{\tilde{\eta}} & \mathbf{0} & \ddots & \mathbf{0} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} & \mathbf{J}_{pN}^{\tilde{\eta}} \\ \mathbf{J}_N^{\tilde{\gamma}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{J}_{qN}^{\tilde{\eta}} \end{bmatrix}. \quad (6.17)$$

This forms a rather sparse matrix, as each quadric is assumed to be independent and the poses only depend on half as many points (i.e. a single neighbourhood as opposed to two). Similarly the residual vectors can be concatenated to form

$$\bar{r}_{p_i} = \begin{bmatrix} r_0 & r_1 & \dots & r_N \end{bmatrix}^T \text{ and } \bar{r}_{q_i} = \begin{bmatrix} r'_0 & r'_1 & \dots & r'_N \end{bmatrix}^T, \quad (6.18)$$

which can be concatenated into a single residual vector

$$\bar{r} = \begin{bmatrix} \bar{r}_{p0} & \bar{r}_{q0} & \bar{r}_{p1} & \bar{r}_{q1} & \dots & \bar{r}_{pN} & \bar{r}_{qN} \end{bmatrix}^T \quad (6.19)$$

This can be used in a standard Gauss-Newton optimisation to compute a joint update to the shared motion parameters $\Delta\tilde{\gamma}_{12}$ and the individual quadric parameters $\Delta\tilde{\eta}_i$ via the update equation

$$\Delta\bar{\zeta} = (\mathbf{J}^T w_i \mathbf{J})^{-1} \mathbf{J}^T \bar{r}. \quad (6.20)$$

However this presents a significant practical issue during computation, the approximate Hessian matrix $\mathbf{J}^T \mathbf{W} \mathbf{J}$ is quite large even for a single point-cloud, and calculating the inverse would require potentially gigabytes of memory and a huge number of compute cycles. Thankfully this matrix is in fact very sparse as indicated in Figure 6.2, and in addition it takes a form that is suitable to apply the Schur-complement trick as

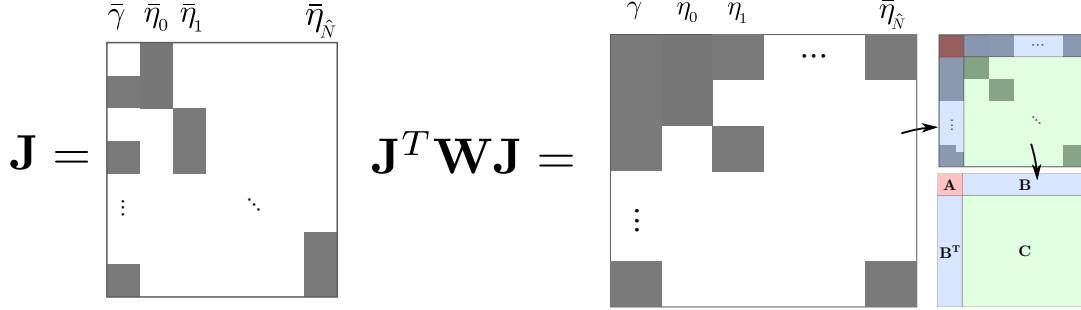


Figure 6.2: **Left:** The joint Jacobian matrix \mathbf{J} of Equation 6.20 and **Right:** the approximate Hessian matrix $\mathbf{J}^T \mathbf{W} \mathbf{J}$. Additionally this demonstrates the approximate Hessian is composed of 3 sub-matrices \mathbf{A} , \mathbf{B} and \mathbf{C} , where \mathbf{C} is block diagonal.

described in Section 3.5.6. Letting

$$\begin{aligned} \mathbf{A} &= \sum_{i=0}^{\hat{N}} ((\mathbf{J}_i^\gamma)^T w_i \mathbf{J}_i^\gamma), \\ \mathbf{B} &= \sum_{i=0}^{\hat{N}} ((\mathbf{J}_i^\gamma)^T w_i \mathbf{J}_{qi}^\eta) \text{ and} \\ \mathbf{C} &= \sum_{i=0}^{\hat{N}} ((\mathbf{J}_{pi}^\eta)^T w_i \mathbf{J}_{pi}^\eta + (\mathbf{J}_{qi}^\eta)^T w_i \mathbf{J}_{qi}^\eta) \end{aligned} \quad (6.21)$$

splits the matrix computation sensibly, as well as letting

$$\bar{x} = \sum_{i=0}^{\hat{N}} ((\mathbf{J}_i^\gamma)^T w_i \bar{r}_{qi}) \text{ and } \bar{y} = \sum_{i=0}^{\hat{N}} ((\mathbf{J}_{pi}^\eta)^T w_i \bar{r}_{pi} + (\mathbf{J}_{qi}^\eta)^T w_i \bar{r}_{qi}). \quad (6.22)$$

This allows the update to be computed using the re-arrangement of the block decomposition as

$$\bar{a} = (\mathbf{A} - \mathbf{B} \mathbf{C}^{-1} \mathbf{B}^T)^{-1} (\bar{x} - \mathbf{B} \mathbf{C}^{-1} \bar{y}) \text{ and } \bar{b} = \mathbf{C}^{-1} \bar{y} - (\mathbf{B} \mathbf{C}^{-1})^T \bar{a}, \quad (6.23)$$

where $\bar{a} = \Delta \gamma_{12}$ is the update to the pose, and $\bar{b} = \Delta \bar{\eta}$ are all the quadric parameter updates.

6.5.1 Implementation Details

The updates can be computed efficiently using CUDA (see Section 3.9] for details of GPGPU programming). A basic implementation outline is shown in Algorithm 1.

Data: Surfaces: \bar{S}_1, \bar{S}_2 , Pose Estimate: \mathbf{E}_{12}
Result: Pose Update: $\Delta\bar{\gamma}_{12}$, Curvature Updates: $\Delta\bar{\eta}_i$

```

foreach  $p_i \in \bar{S}_1$  do
     $\mathbf{Q}_i = \text{getCurrentQuadric}(p_i)$ ;
     $p'_i = \mathbf{E}_{12}^{-1} \cdot p_i$ ;
     $q'_i = \text{project}(\bar{S}_2, p'_i)$ ;
    for  $p_j \in \bar{N}(p_i)$  do
         $\mathbf{J}_{\eta_{p_j}} = \text{computeQuadricJacobian}(\mathbf{Q}_i, p_j)$ ;
         $e_j = \text{computeError}(\mathbf{Q}_i, p_j)$ ;       $w_j = \text{computeWeight}(e_j)$ ;
         $\mathbf{C}_i += \mathbf{J}_{\eta_{p_j}}^T w_j \mathbf{J}_{\eta_{p_j}}$ ;
         $\bar{y}_i += \mathbf{J}_{\eta_{p_j}}^T w_j e_j$ ;
    end
    for  $q_j \in \bar{N}(q'_i)$  do
         $\mathbf{J}_{\gamma_{q_j}} = \text{computePoseJacobian}(\mathbf{Q}_i, q_j)$ ;
         $\mathbf{J}_{\eta_{q_j}} = \text{computeQuadricJacobian}(\mathbf{Q}_i, q_j)$ ;
         $e_j = \text{computeError}(\mathbf{Q}_i, q_j)$ ;       $w_j = \text{computeWeight}(e_j)$ ;
         $\mathbf{A} += \mathbf{J}_{\gamma_{q_j}}^T w_j \mathbf{J}_{\gamma_{q_j}}$ ;       $\mathbf{B}_i += \mathbf{J}_{\gamma_{q_j}}^T w_j \mathbf{J}_{\eta_{q_j}}$ ;       $\mathbf{C}_i += \mathbf{J}_{\eta_{q_j}}^T w_j \mathbf{J}_{\eta_{q_j}}$ ;
         $\bar{x} += \mathbf{J}_{\gamma_{q_j}}^T w_j e_j$ ;       $\bar{y}_i += \mathbf{J}_{\eta_{q_j}}^T w_j e_j$ ;
    end
     $\mathbf{A} -= \mathbf{B}_i \mathbf{C}_i^{-1} \mathbf{B}_i^T$ ;       $\bar{x} -= \mathbf{B}_i \mathbf{C}_i^{-1} \bar{y}_i$ ;
    // Store  $\mathbf{B} \mathbf{C}^{-1}$  and  $\mathbf{C}_i^{-1} \bar{y}$  for use in the final quadric update
     $\mathbf{B}'_i = \mathbf{C}_i^{-1} \mathbf{B}_i^T$ ;       $\bar{y}_i = \mathbf{C}_i^{-1} \bar{y}_i$ ;
end
 $\Delta\bar{\gamma}_{12} = \mathbf{A}^{-1} \bar{x}$ ; // Pose update
foreach  $\mathbf{Q}_i \in \mathbf{Q}$  do
     $\Delta\bar{\eta}_i = \bar{y}_i - \mathbf{B}_i \Delta\bar{\gamma}_{12}$ ; // Compute final quadric update vector
     $\mathbf{Q}_i = \text{updateQuadric}(\mathbf{Q}_i, \Delta\bar{\eta}_i)$ ; // Update quadric
end
    
```

Algorithm 1: Joint Frame-to-Frame Quadric-Pose Optimisation

This doesn't include the optimisations used to make it run in near real-time. Many of these optimisations are CUDA specific such as the warp-wise operations (refer to [226] for further details of this CUDA optimisation). Additionally as was in the case in Chapter 5 the inverse of \mathbf{C} is never explicitly computed, instead a Cholesky decomposition (see Section 3.1.6) is performed which allows for the matrix \mathbf{B} and vector \bar{y} to more efficiently be back-substituted (see Section 3.5.2). The computations are divided such that a single warp is assigned per point p_i , which allows all the Jacobian matrix computations to be performed using register memory and warp reductions to increase efficiency and reduce execution time. Shared memory is used to store interme-

diate results, including those used in the back-substitutions which are also split across a fraction of the threads in the warp. Although this increases thread divergence its faster than returning to global memory and executing a new kernel. Additionally for all symmetric matrices, only the unique values are stored to reduce redundant memory and computation. Finally all computed values are written back to global memory and a separate kernel is launched to compute the final updates, with a single thread per quadric \mathbf{Q}_i . For the full details please refer to the repository.

6.6 Joint Multi-Frame Optimisation

The frame-to-frame is considered a special case of the multi-frame optimisation, as it can be optimised to execute in near real-time using CUDA due to the minimal size of the mutual information matrix. The error function in Equation 6.9 can now be trivially generalised to work for the multiple frames. This slightly modified error function for any particular neighbourhood $\bar{N}(q'_{ij})$ around a transformed central point $q'_{ij} = \mathbf{E}_j^{-1}p_i$ in surface \bar{S}_j is given by

$$\epsilon_{ij} = q_k^T \mathbf{E}_j^T \mathbf{Q}'_{p_i} \mathbf{E}_j q_k, \quad (6.24)$$

where \mathbf{Q}'_{p_i} is the associated quadric for the central point p_i , and \mathbf{E}_j is the transformation that takes points in surface \bar{S}_j to the frame of \bar{S}_0 . This is basically identical to the function in Equation 6.9, but now points in the \bar{S}_0 (this is the surface the quadrics are computed from) aren't considered to be a special case. As such a transformation for all surfaces \bar{S}_j are computed during the optimisation, and a simple normalisation is applied to return this the transformation of \bar{S}_0 to Identity ($\mathbf{E}_0 = \mathbf{I}$). The joint cost function is defined over all points $p_i \in S_0$ as

$$r_{multi}(\bar{\Gamma}, \bar{\eta}) = \sum_{p_i \in S_0} \sum_{j=0}^M (\epsilon_{ij})^2, \quad (6.25)$$

where $\bar{\Gamma} = \{\bar{\gamma}_0, \bar{\gamma}_1 \dots \bar{\gamma}_M\}$ are the respective motion parameters of each surface \bar{S}_j , and $\bar{\eta} = \{\bar{\eta}_1, \bar{\eta}_2 \dots \bar{\eta}_N\}$ are the respective quadric parameters of each quadric \mathbf{Q}'_{p_i} computed from surface \bar{S}_0 .

The cost function in Equation 6.25 was reduced using a standard Gauss-Newton optimisation approach. The gradients and Jacobian matrices are identical to those in the single-frame case, but instead of a single pose now M poses are expressed in the optimisation. As indicated in Equation 6.25, the cost is summed over all neighbourhoods that correspond to the quadric's associated central point. This requires the additional step of computing all associated correspondences across an overlapping set of frames, which is identical to the correspondence estimation in Chapter 4. However, the only correspondences that matter are those that involve surface/frame \bar{S}_0 . As described in Chapter 4 a project-and-scan correspondence estimate was used to compute the correspondences for all surfaces to surface \bar{S}_0 . An example of the Jacobian and Hessian

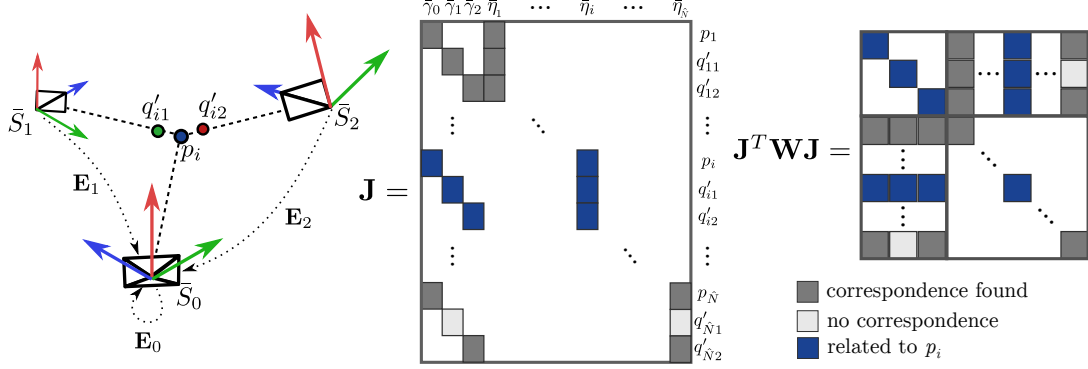


Figure 6.3: **Left:** a scene with three cameras that view a single point p_i , and the corresponding closest points (q'_{i1}, q'_{i2}) in two adjacent views (\bar{S}_1, \bar{S}_2) **Centre:** Jacobian matrix that additionally demonstrates the corresponding sections for the point p_i . A missing correspondence is shown in $p_{\bar{N}}$, where the surface \bar{S}_1 contains no correspondence. The values across the top indicate which parameters this part of the Jacobian relates to, while the values across the right side indicate the points that contribute. **Right:** approximate Hessian matrix that also indicates how the point p_i effects the final matrix.

computation for a set of three frames is shown in Figure 6.3. This demonstrates clearly how the point correspondences effect the final computation, as well as how missing correspondences result in some sparsity in the dense mutual information matrix, through the light greyed-out sections.

As is the case with the frame-to-frame Jacobians, the matrix is block separable. This is indicated in Figure 6.3(Right) by the slightly thicker lines marking the \mathbf{A} , \mathbf{B} and \mathbf{C} matrices that make up the approximate Hessian matrix $\mathbf{J}^T \mathbf{W} \mathbf{J}$, which is ideal for the Schurr-complement trick. However, in this case the size of the mutual information matrix is different, and two block-diagonal matrices \mathbf{A} and \mathbf{C} require inversion. The storage of the intermediate matrix $\mathbf{C}^{-1} \mathbf{B}^T$ also requires a potentially much larger amount of space in memory. These considerations are discussed in the following section on implementation details.

6.6.1 Implementation Details

The joint multi-frame optimisation is implemented using a multi-threaded CPU implementation. The *work* is separated based the correspondences, this is loosely demonstrated in Algorithm 2, where the computation is performed across all correspondences of a single point p_k . This division is identical to that of the work in Chapter 4, however in this case only the correspondences to points in surface \bar{S}_0 are used.

Data: Surfaces: \bar{S}_i , Pose estimates: \mathbf{E}_i , Quadric estimates: \mathbf{Q}_k

Result: Pose updates: $\Delta\bar{\gamma}_i$, Quadric updates: $\Delta\bar{\eta}_k$

```

foreach  $k \in \{1, 2, \dots, \hat{N}\}$  do
    foreach  $i \in \{1, 2, \dots, M\}$  do
        // compute all corresponding points to point  $p_k$ 
        // across all  $M$  surfaces  $\bar{S}_i$ 
         $\mathbf{Q}_k = \text{getCurrentQuadric}(p_k)$ ; // center quadric
         $p'_k = \mathbf{E}_i p_k$ ; // center point
         $q'_k = \text{project}(\bar{S}_i, p'_k)$ ; // closest point in surface  $\bar{S}_i$ 
        foreach  $q_j \in \bar{N}(\bar{S}_i, q'_k)$  do
             $\mathbf{J}_{ij}^\gamma = \text{computePoseJacobian}(\mathbf{Q}_k, q_j)$ ;
             $\mathbf{J}_{kj}^\eta = \text{computeQuadricJacobian}(\mathbf{Q}_k, q_j)$ ;
             $e_j = \text{computeError}(\mathbf{Q}_k, q_j)$ ;     $w_j = \text{computeWeight}(e_j)$ ;
             $\mathbf{A}_i += (\mathbf{J}_{ij}^\gamma)^T w_j \mathbf{J}_{ij}^\gamma$ ;     $\mathbf{B}_{ik} += (\mathbf{J}_{ij}^\gamma)^T w_i \mathbf{J}_{kj}^\eta$ ;  $\mathbf{C}_k += (\mathbf{J}_{kj}^\eta)^T w_j \mathbf{J}_{kj}^\eta$ ;
             $\bar{x}_i += (\mathbf{J}_{ij}^\gamma)^T w_j e_j$ ;     $\bar{y}_k += (\mathbf{J}_{kj}^\eta)^T w_j e_j$ ;
        end
         $\mathbf{A}_i -= \mathbf{B}_{ik} \mathbf{C}_k^{-1} \mathbf{B}_{ik}^T$ ;     $\bar{x}_i -= \mathbf{B}_{ik} \mathbf{C}_k^{-1} \bar{y}_k$ ;
    end
    // store intermediate matrices required in existing  $\mathbf{B}_{jk}$  and  $\bar{y}_k$ 
     $\mathbf{B}_{ik} = \mathbf{C}_k^{-1} \mathbf{B}_{ik}^T$ ;     $\bar{y}_k = \mathbf{C}_k^{-1} \bar{y}_k$ ;
end
foreach  $i \in \{1, 2, \dots, M\}$  do
     $\Delta\bar{\gamma}_i = \mathbf{A}_i^{-1} \bar{x}_i$ ; // compute final pose updates
     $\mathbf{E}_i = \text{updatePose}(\mathbf{E}_i, \Delta\bar{\gamma}_i)$ ; // update poses using matrix exponential
end
foreach  $k \in \{1, 2, \dots, \hat{N}\}$  do
    foreach  $i \in \{1, 2, \dots, M\}$  do
         $\bar{y}_k -= \mathbf{B}_{ik} \Delta\bar{\gamma}_i$ ; // compute final quadric updates
    end
     $\Delta\bar{\eta}_k = \bar{y}_k$ ;
     $\mathbf{Q}_k = \text{updateQuadric}(\mathbf{Q}_k, \Delta\bar{\eta}_k)$ 
end
    
```

Algorithm 2: Joint Full Quadric-Pose Optimisation

The quadrics for each point are estimated using the approach described in Chapter 5, and the estimated poses are computed using the same point-to-plane ICP approach used to initialise poses in Chapter 4. Separate memory per thread is used, to prevent memory contention and avoid the need for *atomics* or *mutexes* in the multi-threaded implementation. This increases the spatial complexity of the approach, which could limit the number of overlapping frames that can be used. In practice the benefit of

using additional frames reduces as more frames are included in the optimisation. This is largely because no new information is really being added after the noise has been compensated for. This relationship between performance and the number of frames, is explored further in Section 6.7. As indicated in Algorithm 2, the matrix $\mathbf{C}^{-1}\mathbf{B}^T$ is stored during the computation, as this matrix is required for the final update computation.

As the system separates the computation such that a single thread will compute all related Jacobians there is no requirement to duplicate this matrix as there can be no memory conflicts, which is also true for the vector $\mathbf{C}^{-1}\bar{y}$. This is not true for the matrix $\mathbf{A} - \mathbf{BC}^{-1}\mathbf{B}^T$ which will be accessed by every thread, as such each thread stores an individual estimate of this matrix and the resulting estimates are sum reduced after all threads finish executing. This has the added benefit of reducing the risk of underflow by maintaining a statistically similar matrix conditioning for each estimate if the work is distributed evenly among threads. Specifically this resulting matrix is the sum of tens of millions of matrix products in a general case, which can be prone to underflow. By dividing the work over threads this increases the effective precision. As $M \ll \hat{N}$ the inversion of the resulting matrix $\mathbf{A} - \mathbf{BC}^{-1}\mathbf{B}^T$ is comparatively cheap and as such this section is performed single threaded. Following this the updates to the quadric estimates are again computed across a single thread, but as the computation is already not real-time this was deemed unnecessary. As is the case in Section 6.5, no explicit inversions are actually performed, all calculations are performed using Cholesky decomposition and back-substitution. The whole multi-frame system could be implemented on the GPU and would potentially run as a background process but at the time of this research was deemed beyond the scope, and can be considered future work.

6.7 System Evaluation

In this section the performance of the novel implementations is compared with state of the art alternatives, and the improvement in accuracy is evaluated. Additionally the approaches are evaluated using real-world low cost depth sensor datasets and their applicability to robotic applications is demonstrated. In Section 6.8 several approaches are evaluated on the task of relative pose alignment on real-world and synthetic data. In Section 6.8.3, the amount of pose drift is measured for several approaches, which is an important quantity in loop-detection. Finally in Section 6.9 several curvature estimation methods are evaluated quantitatively using single and the multi-frame approaches proposed in this chapter.

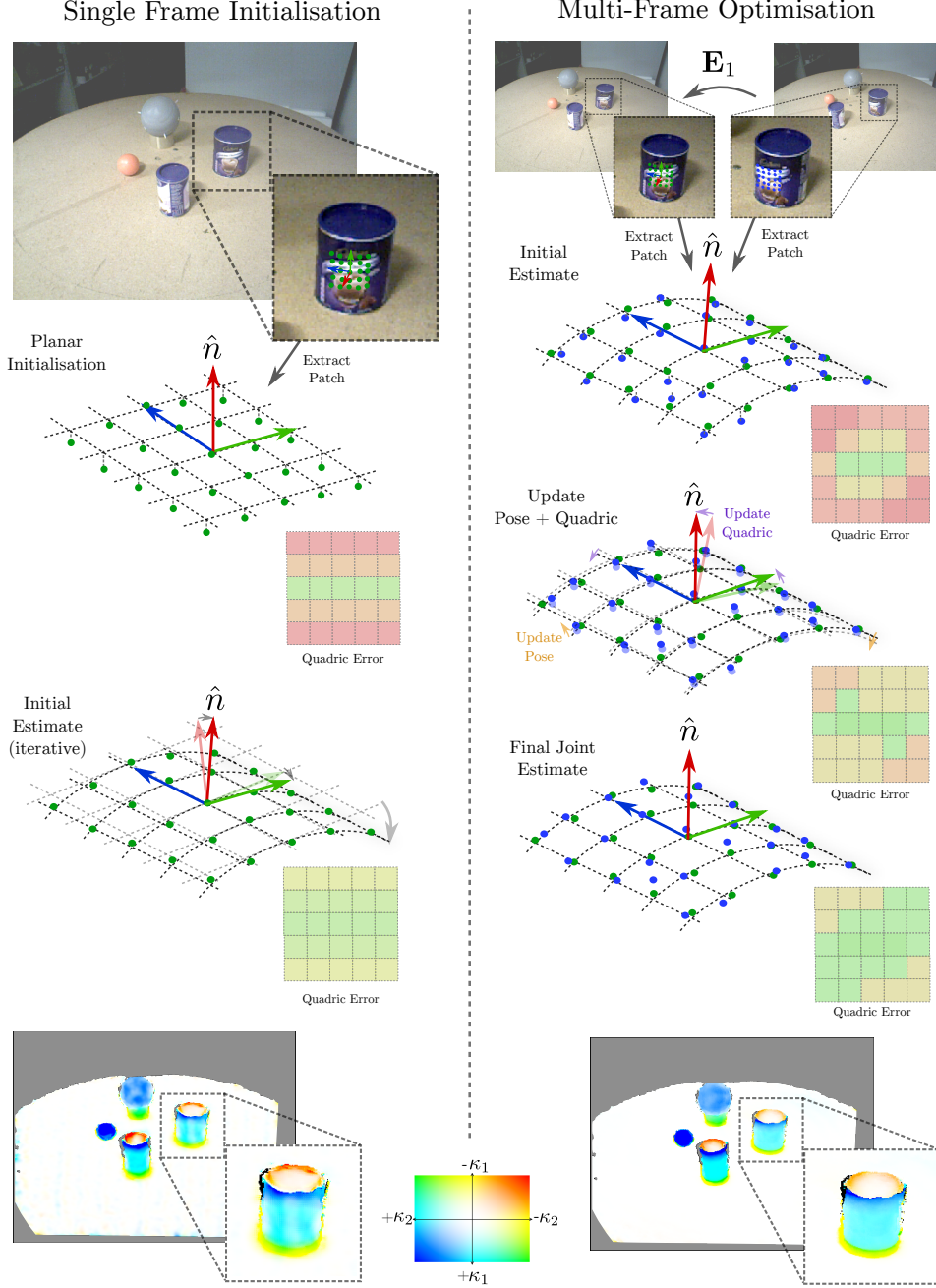


Figure 6.4: **Left:** the initialisation process for a single patch in frame 0 using the quadric surface alignment strategy from Chapter 5. **Right:** the multi-frame alignment for two frames, demonstrating the extracted corresponding patch and the refinement of the quadric and pose alignment to reduce the cost in Equation 6.25. **Bottom:** shows the initial and resulting quadric surface estimates for the whole frame.

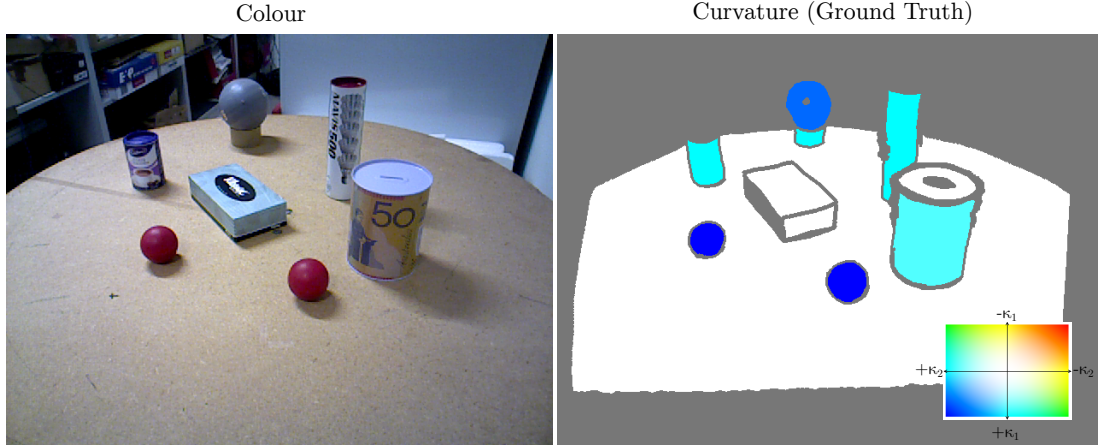


Figure 6.5: An example from one of the real-world curvature datasets (*Real-World 5*). **Right:** the colour image, **Left:** manually labelled ground-truth curvature values. Additionally a key is shown to indicate the false-colouring used for the ground-truth curvature values.

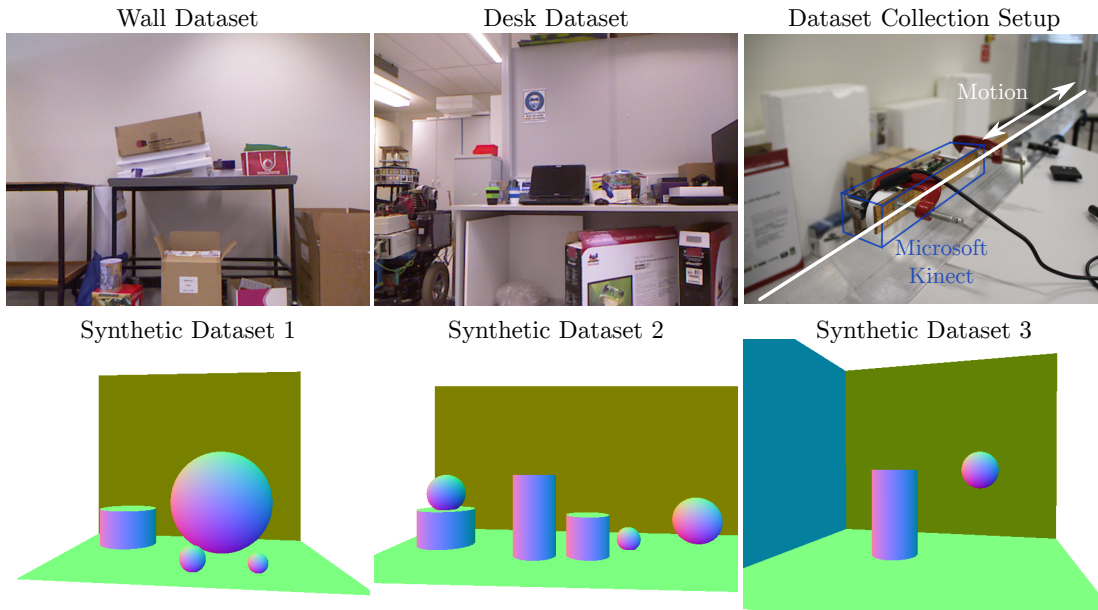


Figure 6.6: The synthetic and real-world datasets used to assess pose accuracy of several approaches. **Top-Right:** Demonstrates the experimental setup used to capture the real-world datasets *Wall* and *Desk*. A Microsoft Kinect Version 1 is rigidly attached to a rail and moved along a straight-line trajectory capturing frames exactly every 10mm.

6.7.1 Evaluation Datasets

Several additional ground truth datasets were created in order to evaluate the systems produced in this work. In addition to the real-world curvature datasets that were created to evaluate the system described in Chapter 5 (an example is shown in Figure 6.5), two additional real-world ground truth pose datasets were created to evaluate the relative pose estimates quantitatively. These datasets are pictured in Figure 6.6 (**Top**). Both *Wall* and *Desk* were collected in the same way, a Kinect (Version 1) was rigidly attached to a rail and moved along it in marked increments of 10mm. While the Kinect was positioned, a single frame was captured that included depth and colour information. A 1.3m straight line trajectory was captured for each dataset, but roughly only 65cm segments of the dataset overlap sufficiently for a multi-frame evaluation. The depths used in the evaluation are corrected using the calibration computed using the approach described in Chapter 4.

To access the approach in a more controlled manner several synthetic datasets were generated. As opposed to the synthetic scenes used to evaluate the system in Chapter 5, the datasets used in this work contain more complex scenes with a mixture of objects. This is intended to make the results from the synthetic data more meaningful. As shown in Figure 6.6(**Bottom**), scenes vary in complexity but always contain a large planar background in order to challenge the curvature estimation, and more accurately represent a more realistic scene. Every image in each dataset includes a ground-truth pose, curvature and depth value. Additionally simulated noise is added in the same manner as described in Chapter 5, to examine the effect on pose and curvature estimation accuracy. The trajectories for each dataset are different, but are all smooth trajectories of simulated motion. The spacing between frames is not necessarily equal are some trajectories accelerate.

6.7.2 Evaluation Metrics

The pose accuracy is evaluated using the absolute trajectory error (ATE) metric proposed in [51] given by

$$\epsilon_{ATE} = \sqrt{\frac{1}{N} \sum_{t=0}^N |\mathbf{E}_t^{-1} \mathbf{E}_t^*|^2}, \quad (6.26)$$

where \mathbf{E}_t and \mathbf{E}_t^* are the ground-truth and estimated pose at time t respectively. This is effectively an RMS error over the magnitude of the log-pose error and this is how the magnitude of the pose is computed using $\log_e(\mathbf{E}_t^{-1} \mathbf{E}_t^*) \in \mathfrak{se}_3$ which operates on the Lie-algebra \mathfrak{se}_3 . The magnitude is separated into the translation and rotation components are evaluated separately, which is trivial in the log pose form.

The curvature accuracy is evaluated in the same manner as Chapter 5 using

$$\epsilon_{curv} = \sqrt{\frac{1}{\hat{N}} \sum_{i=0}^{\hat{N}} |(\kappa_1 - \kappa_1^*)^2 + (\kappa_2 - \kappa_2^*)^2|}, \quad (6.27)$$

where κ_i and κ_i^* are the ground-truth and predicted principal curvature values respectively. This computes the RMS curvature error over the entire set of points p_i in surface \bar{S}_0 .

6.7.3 Compared Approaches

The tested pose alignment methods were:

- *ICP-fff*: frame-to-frame dense ICP, same as the one used in Chapter 4 for the initial alignment estimates
- *ICP-bundle*: dense frame-to-frame ICP bundle-adjustment that minimises the point-to-plane error across all points simultaneously. This is also the same as the one used in Chapter 4 for the calibration, but with fixed calibration parameters.
- *J-fff*: the joint frame-to-frame approach described in Section 6.5
- *J-full*: the full multi-frame joint optimisation described in Section 6.6
- *Q-full*: is the full multi-frame joint optimisation described in Section 6.6, but with the quadric parameters fixed in the optimisation. This is similar to the approach in [203] but using the quadric formulation described in Chapter 5.

The tested curvature approaches were:

- *Quad LS*: the basic quadric least-squares approach described in [98]
- *Quad IT*: the real-time iterative re-weighted least-squares approach described in [24], from Chapter 5.
- *J-fff*: the joint frame-to-frame approach described in Section 6.5
- *J-full*: the full multi-frame joint optimisation described in Section 6.6

6.8 Pose Estimation Accuracy

This section details the quantitative evaluation of pose estimation accuracy using synthetic and real-world data.

6 JOINT CURVATURE POSE OPTIMISATION

Translational Error (m)					
<i>Noise Added</i>	ICP- <i>ftf</i>	ICP- <i>bundle</i>	Ours (<i>Q</i> -full)	Ours (<i>J</i> - <i>ftf</i>)	Ours (<i>J</i> -full)
σ	4.19e-3	6.93e-4	2.79e-4	2.87e-4	2.88e-4
1σ	9.22e-3	1.83e-3	8.15e-4	1.02e-3	6.90e-4
2σ	1.52e-2	3.00e-3	1.68e-3	1.92e-3	1.53e-3
3σ	2.01e-2	4.45e-3	2.53e-3	2.91e-3	2.37e-3
Rotational Error (rad)					
σ	3.36e-3	5.77e-4	2.38e-4	2.31e-4	2.79e-4
1σ	7.35e-3	1.69e-3	3.69e-4	5.05e-4	4.98e-4
2σ	1.31e-2	2.69e-3	6.84e-4	8.70e-4	5.54e-4
3σ	1.72e-2	3.40e-3	1.21e-3	1.22e-3	8.75e-4

Table 6.1: Synthetic Pose Error - Dataset1

6.8.1 Synthetic Dataset Evaluation

The pose alignment accuracy is evaluated using the metrics described in Section 6.7.2, on the synthetic datasets for each of the methods described in Section 6.7.3. The results are presented in Tables 6.1, 6.2 and 6.3. The lowest error is highlighted in each category in **bold**. A consistent order of magnitude improvement over *ICP-ftf* by the proposed *J-ftf* approach is observed. This is significant as this process can be used to massively improve the quality of frame-to-frame alignments during tracking for an online approach. A modest improvement is also visible for the proposed *J-full* system over a full bundle adjustment (*ICP-bundle*) (approximately 30%-50%), despite the proposed approach only aligning to a single frame. This is a significant result as *ICP-bundle*, uses the information of all frame-to-frame alignments to compute a semi-dense pose graph. More surprising is that the *J-ftf* approach is able to consistently out-perform the *ICP-bundle* approach for synthetic data. This indicates that the more accurate surface approximation could be very valuable in a SLAM based system, considering the dense point-to-point bundle-adjustment is in fact similarly computationally expensive to the *J-ftf*.

Another interesting result is the results of *Q-full*, which has essentially the same performance as the *J-ftf* and *J-full* on low noise data, reducing in quality slightly faster than the proposed approaches as more noise is added. This indicates the initial quadric estimates are relatively robust to noise, more so than the normal estimates, and more importantly that having more data improves the quality of those estimates and makes alignment more robust.

6 JOINT CURVATURE POSE OPTIMISATION

Translational Error (m)					
<i>Noise Added</i>	ICP-ftf	ICP-bundle	Q-full	J-ftf	J-full)
σ	3.56e-3	5.42e-5	3.55e-5	3.69e-5	3.58e-5
1σ	4.83e-3	6.22e-4	4.74e-4	4.70e-4	4.56e-4
2σ	8.29e-3	1.41e-3	1.07e-3	9.97e-4	9.86e-4
3σ	1.17e-2	2.60e-3	2.07e-3	2.07e-3	1.97e-3
Rotational Error (rad)					
σ	2.00e-4	2.71e-5	1.14e-5	1.19e-5	8.49e-5
1σ	2.46e-3	1.01e-4	6.33e-5	5.92e-5	9.75e-5
2σ	5.70e-3	5.34e-4	3.54e-4	2.01e-4	2.10e-4
3σ	7.79e-3	2.32e-3	1.99e-4	2.26e-4	1.89e-4

Table 6.2: Synthetic Pose Error - Dataset2

Translational Error (m)					
<i>Noise Added</i>	ICP-ftf	ICP-bundle	Ours (Q-full)	Ours (J-ftf)	Ours (J-full)
σ	1.94e-3	1.27e-4	1.91e-4	1.23e-4	1.22e-4
1σ	4.61e-3	9.78e-4	9.49e-4	9.45e-4	9.39e-4
2σ	8.50e-3	1.95e-3	1.96e-3	1.82e-3	1.83e-3
3σ	1.58e-2	3.04e-3	2.69e-3	2.71e-3	2.72e-3
Rotational Error (rad)					
σ	1.93e-3	7.09e-5	1.74e-4	7.64e-5	7.40e-5
1σ	4.74e-3	2.28e-4	2.73e-4	2.55e-4	2.55e-4
2σ	8.25e-3	3.82e-4	4.99e-4	4.10e-4	3.69e-4
3σ	1.30e-2	7.06e-4	5.52e-4	5.99e-4	5.30e-4

Table 6.3: Synthetic Pose Error - Dataset3

6.8.2 Real-world Dataset Evaluation

The performance of the proposed approaches on real-world datasets collected using a low-cost depth sensor (the Microsoft Kinect) is shown in Figure 6.7. A similar reduction in pose estimation error is observed using the proposed approaches, compared to the dense ICP based approaches. In dataset *Wall* we see a significant improvement using *J-full* approach over other approaches, despite the large number of planar surfaces visible in the scene. This is surprising as these large planar objects should significantly

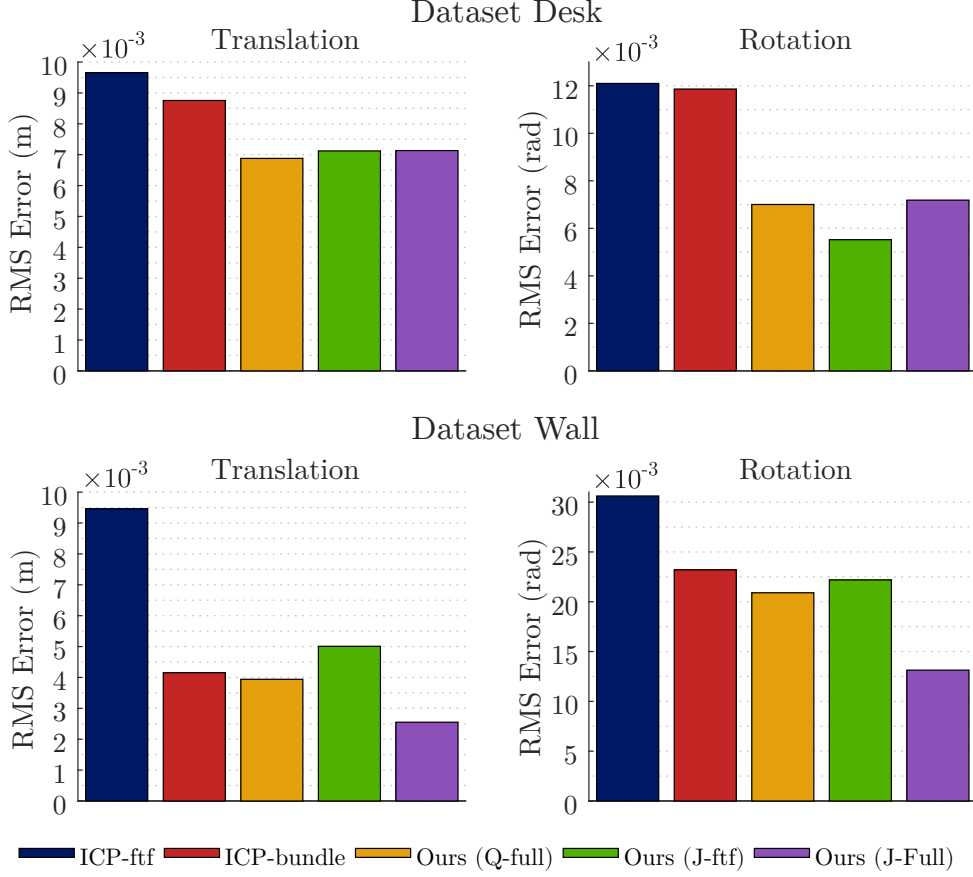


Figure 6.7: Real-world pose error on two ground truth datasets. The datasets were both captured using an on-rails setup with translation only along a straight trajectory. These results demonstrate the improved accuracy of our methods over the standard approaches particularly in the Dataset *Desk* (top), which shows a noticeable reduction in translation and rotational error for the joint approaches.

improve the quality of *ICP-fft* and *ICP-bundle*. In dataset *Desk* the improvement is less significant for the proposed approaches, although all quadric based methods significantly reduce RMS pose error.

6.8.3 Pose Estimation Drift

In order to demonstrate the applicability of our systems to real-world tracking applications we examine the estimated pose drift as we move along a known trajectory. As shown in Figure 6.8 we observe a significant reduction in frame-to-frame drift using quadric based approaches. Most surprising is the reduction in drift using our *J-fft* implementation, achieving comparable pose drift error to *ICP-bundle* and greatly im-

proving upon the drift of *ICP-ftf*. This shows our implementation can be used in a tracking system to greatly reduce the drift of adjacent keyframes, which can accumulate in a system and make the detection of loops very difficult for a non-feature based approach. Additionally our *J-full* approach shows a significant reduction, and increases significantly slower than all other approaches. The main reason for this is that methods like *ICP-bundle* benefit greatly from large loop-closures, which are not present unless the camera revisits a location.

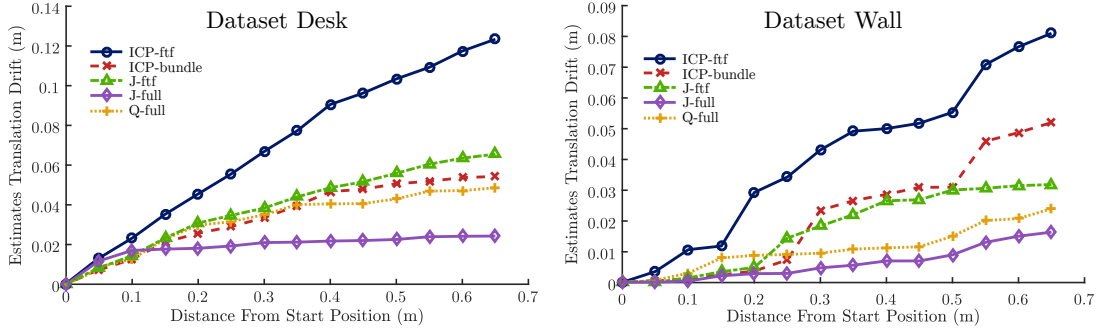


Figure 6.8: Demonstrates the reduction in drift using the proposed approaches, including *J-ftf*, which even manages to match the drift of the global ICP bundle adjustment for Dataset *Wall*.

6.9 Curvature Estimation Evaluation

This section provides a quantitative analysis of the proposed approaches across all datasets. An addition qualitative example can be found in Figure 6.1, at the beginning of this section. This figure demonstrates a dramatic qualitative improvement to curvature estimation for real-world data.

6.9.1 Synthetic Dataset Evaluation

We examine the results of principal curvature estimation using the synthetic datasets shown in Figure 6.5. As described in Section 6.7.2 we compare both of our novel joint approaches to existing methods, including a quadric least-squares (*Quad LS*) approach [98], and our previous iterative quadric fitting (*Quad IT*) approach from [24]. We show the results of testing on synthetic data in Figure 6.9. Not so surprisingly using information from multiple frames results in reduced error, with both *J-full* and *J-ftf* consistently out-performing the previous methods. Our *J-ftf* method only uses two frames and as such the improvements are considerably more modest than the multi-frame *J-full*.

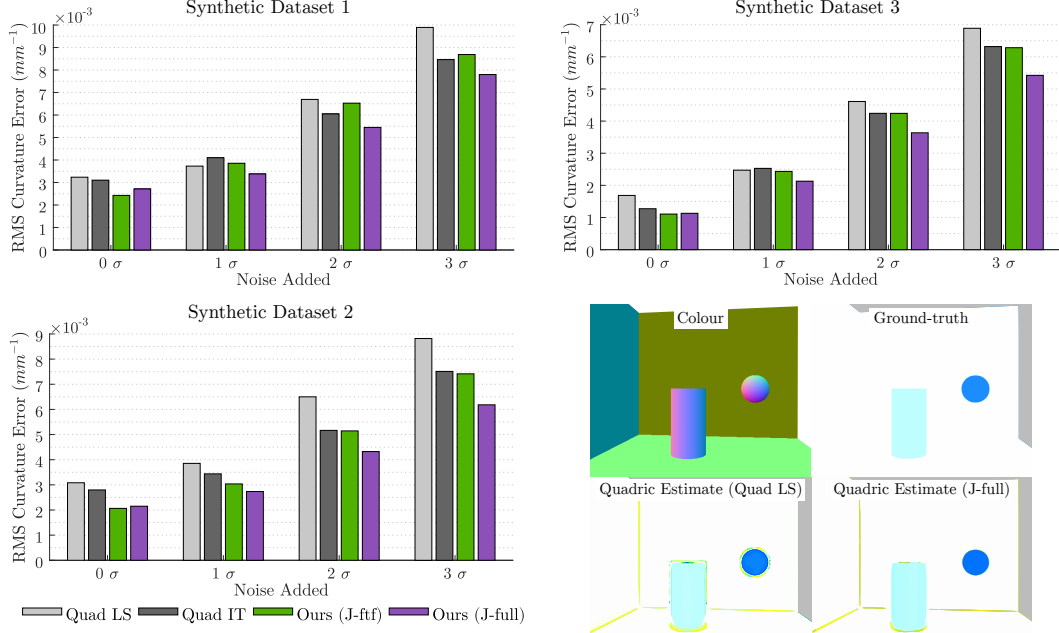


Figure 6.9: Results of testing several curvature estimation methods on 3 synthetic datasets with successively increasing levels of noise added to each. This demonstrates our systems ability to cope with aggressive noise, and shows the improvement upon previous curvature estimation methods by using multiple frames. **Bottom-Right:** shows the qualitative performance for a single frame of the synthetic dataset 3, including the ground-truth curvature values. This shows where the system finds curvature estimation the most challenging, unsurprisingly it’s the edges which technically contain infinite curvature.

6.9.2 Real-World Dataset Evaluation

Finally we compare the implementation on real-world datasets generated for [24], an example is shown in Figure 6.5. The result of this testing shown in Figure 6.10 demonstrates both our joint methods (*J-*ftf**, *J-*full**) can out-perform previous methods at principal curvature estimation on real-world data. This is significant as curvature can be used as a basis for scene segmentation, using our *J-*ftf** method can therefore greatly improve the results of segmentation. We also include a qualitative example of the improvement in Figure 6.1, which clearly demonstrates the improvement using multiple frames has. This frame is from the dataset *Real-World 4*, and contrasts the performance of our previous approach (*Quad IT*) against our full joint optimisation (*J-*full**). A more subtle aspect of the improvement is the reduction in noise on the planar surface, which is much flatter for the joint estimate.

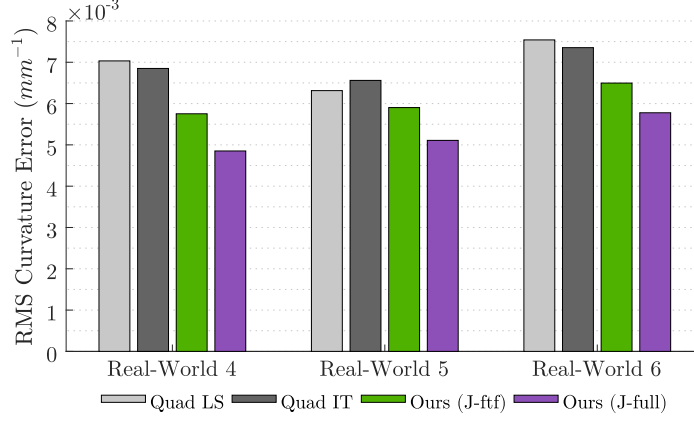


Figure 6.10: RMS curvature error for 3 real-world manually labelled datasets. Note the reduction in error upon previous state-of-the-art techniques across all datasets using our joint approaches. Even with just two frames (*J-fft*) our system shows a significant reduction in error over the approach in [24]

6.10 Conclusions and Future Work

The two systems presented in this chapter use a joint optimisation approach to improve curvature and pose estimates on synthetic and real-world datasets. This approach is shown to not only be useful to improve offline dataset accuracy over state of the art techniques such as dense ICP bundle adjustment but can also be incorporated into a real-time system to greatly reduce pose drift to improve the results of re-localisation and loop-closure. The current implementation of joint optimisation across multiple frames has not been fully optimised, but has been designed such that a GPU implementation should be a relatively simple extension of this work in the future.

6.11 Code/Datasets

The source code and datasets described in this chapter are provided at the following repository <https://github.com/aspek1/JointCurvatureOptimisation.git>.

Machine Learning Approaches for Geometric Quantity Estimation

The following chapter is largely drawn from work performed jointly with Thanuja Dharmasiri, and published in [159] and is likely to appear in a similar form in his thesis. The idea was to use the previous work from Chapter 5 in a machine learning problem. To this end Thanuja Dharmasiri and I collaborated on a novel depth, normal and curvature joint prediction network. The work from Chapter 5 was used to generate training data for a Convolutional Neural Network (CNN) which was trained to replicate the output using only colour information as input. The resulting network was found to demonstrate the benefit of using relating quantities in training a network to overall performance.

A list of elements and the relevant percentages we have agreed in terms of contribution to the production of this work is provided:

- Conception of idea (50%)
- Network architecture design (10%)
- Loss/Objective function design (40%)
- Coding the network ($\approx 1\%$)
- Training data creation (70%)
- Testing and evaluation (50%)

7.1 Motivation

Extracting information from raw data is a well studied problem in robotics. A visual image is one such form of raw data and has been widely used in the community to tackle a range of problems including image segmentation [146], localization and mapping [14], visual servoing [227] etc. and there exist a continuous stream of research which look at maximizing the amount of information extracted. In this chapter it is shown that it is possible to estimate geometric quantities such as surface curvature using only RGB images as input. To our knowledge this was the first work to demonstrate such a capability.

Surface Curvature is an important geometric surface feature, that indicates the rate



Figure 7.1: **Top:** The corresponding RGB image from the NYUv2 test dataset which was used as the only source of information to estimate curvature. **Middle:** The *ground-truth* curvature computed using the depth data using the approach from [24]. **Bottom:** The predicted curvature of this network. The false colouring is as shown in Chapter 5 and 6. *Positive* curvatures are shown in blue, *negative* in red, *saddles* in green and *planes* in white.

at which the direction of the normals change on the surface at any particular point. As discussed in Section 3.8. It has been shown to be particularly useful for the task of segmentation on range image and 3D data [19, 98, 96, 222]. 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 single RGB images [34, 157, 187, 158], including estimating depth and normals. In this work a Convolutional Neural Network (CNN) is used to estimate principal surface curvatures as well as depth and normals, from a single RGB image. This is an extension of the work performed in Chapter 5, as a practical application of the system created in that work.

Contrary to the popular belief that hand-engineered features are inferior compared to learnt features, this work argues that well designed features combined with machine learnt representations provide improved performance. It should be stressed that the features designed are calculated by hand, but rather predicted by the network itself as part of the inference pipeline. More concretely, the network is forced to learn an internal representation that will not just focus on the reduction of loss in one quantity, for example depth, but also other quantities through the use of multiple loss functions and a shared model capacity. This is demonstrated by estimating surface curvature, surface

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

normals and depth in a multi task learning framework which provides superior results compared to training them as individual tasks. The work presented was inspired by that of Eigen *et al.* in [34]. In this work three quantities are also jointly estimated (depth, surface normals and semantic labels) using a single network. Semantic segmentation is clearly related to depth and normal estimation, and they demonstrate an improvement estimating all three. The approach presented in this chapter shows that estimating more tightly coupled quantities, even when the *ground-truth* is computed from noisy data, increases the relative improvement of all three tasks. This is demonstrated both quantitatively and qualitatively as the network is able to achieve better results on depth and surface normals on the NYUv2 dataset [40] by estimating a view point invariant quantity (surface curvature) jointly with depth and normals. Additionally as an application of this work, a simple segmentation is performed using the estimated quantities and a border function.

7.2 Contributions

The contributions of the work presented in this chapter are as follows:

- A novel application of the work presented in Chapter 5 for machine learning.
- A novel technique to estimate surface curvature of objects using purely RGB images.(Method: Section 7.5.3, Results: Section 7.8.3)
- A framework which predicts depth,surface normals and curvature jointly.(Method: Section 7.5, Results: Section 7.8)
- Demonstrate that joint training can improve the accuracy of all three tasks while keeping the model capacity fixed (Method: Section 7.6.3, Results: Tables 7.1, 7.2, 7.3.

7.3 Related Work

In this section we review existing work in the literature that is related to this paper and in turn inspired the ideas presented. We take a look at traditional approaches used to compute surface curvature from raw depth data, then we summarize how deep learning has been used to predict information from images and finally, we discuss how the problem of learning multiple tasks in a single platform was performed using deep learning.

As described in Section 3.8, surface curvature estimation is a well explored topic in robotics and computer vision. It has been shown to be useful for object segmentation [19, 98, 222, 96] 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 [19, 96], 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 [222, 94, 92]. 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 parameterization [98, 203], which has been shown to be robust to noisy data and fast enough to be computed in real-time [24]. In this chapter the approach in [24] is used, to compute surface curvature and surface normals from the training data sourced from the NYUv2 dataset [40] as it has been shown to perform well on range image data of the type present in the dataset.

Convolutional Neural Networks (CNNs) have been very effectively applied to a range of robotic and vision tasks including grasp pose detection [184, 185], image classification [26, 37], semantic segmentation [146], depth estimation [34, 157, 187, 158], surface normal estimation [228, 160, 34]. The work presented in this chapter is more closely related to the latter two tasks as we demonstrate surface curvature can be predicted using RGB images as the only input. This work initially used the VGG architecture [27] as a starting point to predict surface curvature in a standalone network and was then extended to estimate depth and surface normals in a single network which is described in Section 7.4.

Prior to the resurgence of neural networks depth was either computed using a Simultaneous Localisation and Mapping (SLAM) system [111, 15] or directly obtained from a range sensor such as structured light sensors [14, 13] (Microsoft Kinect). As shown in Section 3.3 each of the range sensors has potential drawbacks, and machine learning is seen as a possible complement to these sensors. Saxena *et al.* in [187] used a supervised learning approach that combines local and global image features by using a Markov Random Field (MRF). The idea of using both global and local features was further investigated by Eigen *et al.* [35] using the AlexNet [26] architecture in a multi-scale scheme. Liu *et al.* [157] 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.* [158] proposed to use a far superior fully convolutional residual architecture and obtain state-of-the-art results in single image depth estimation.

Data driven single image surface normal estimation was first tackled by Fouhey *et al.* in [229]. They used a SVM based detector followed by an iterative optimisation 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 [230]. A ConvNet approach to estimating surface normals in global and local scales while incorporating numerous

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

constraints such as room layout and edge labels was taken by Wang *et al.* [228]. Recently, Bansal *et al.* [160] 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.

In one of the earliest works of multi-task learning Caruana *et al.* showed in [231] 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 [232] 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 [233], web search ranking [234], face verification [235] etc. Li *et al.* in their work *Learning Without Forgetting* [236] 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. In this case however, access to training data for all three tasks is available, and it was found that training the prediction stacks jointly, achieved superior performance compared to fine-tuning.

7.4 Model Architecture

A more detailed description of Convolutional Neural Networks (CNN) and their constituent layers can be found in Section 1.4. The current section is intended to provide an overview of the layers used in context, some understanding of layer functionality and use is assumed knowledge.

The model begins with a set of convolutional layers based on the VGG16 architecture [27]. This allows the model to perform an initial *feature extraction*. This is followed by 2 fully connected layers which are used to distil the context of the whole image in a feature embedding. Next an initial stack of convolutional layers corresponding to *coarse level predictions* of depths, normals and curvatures. Each of these with an individual solver to force the network to learn course level features at a lower resolution early in the network. These course level predictions are now treated as feature maps and concatenated with features from a different set of convolutions and convolved separately to predict at a *finer resolution*. The convolutional layers in the coarse and fine level prediction stacks perform 5x5 convolutions with a stride of 1 and a pad of 2, to preserve the input resolution. There is an explicit up-sampling layer which increases the spatial resolution of the coarse level prediction from 74x55 resolution to 147x109 which is maintained

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

throughout. The entire model is summarised in Figure 7.2 and demonstrates the flow of the data explicitly through the network. The individual solvers for each of the training tasks compute the loss and initiate the backward propagation of gradients as described in Section 1.4. During evaluation of the different configurations of this system the model capacity was maintained in order to measure the contribution of each new task. This required several changes to be made during training which are explained further in the Section 7.6.3.

After the completion of this work, different architectural designs were explored and these are discussed in Section 7.8.5.

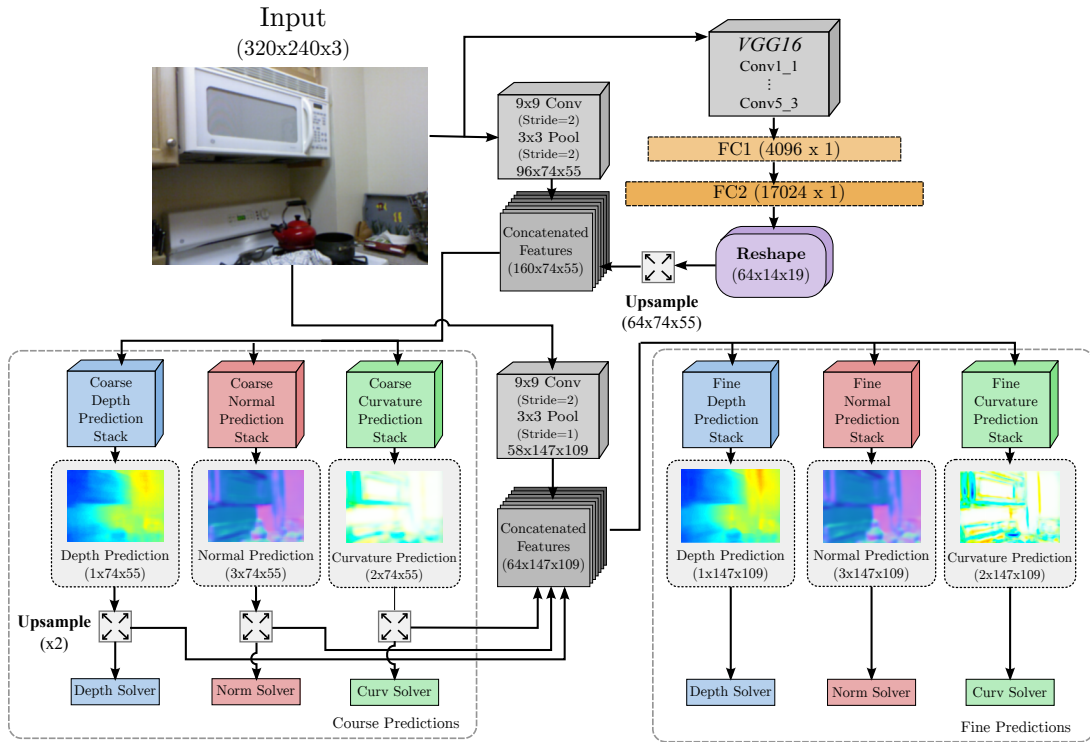


Figure 7.2: Visual Representation of Model Architecture

7.5 System Task Specification

This section is included to provide some background for each of the tasks the network is trained to perform.

7.5.1 Depth

The raw depth data distribution given by the NYUv2 dataset [40] was used for training based on the official train and test scene split (that is 249 training scenes and 219 test scenes). Similar to previous approaches the network was trained to estimate depth at multiple scales, as this was found to benefit the overall performance [34]. The loss function used for calculating the error in the depth estimation included a Euclidean loss term, a scale invariant term, and a gradient term which compares the local rate of change of the predicted and ground truth depth values spatially. Unlike in the work of Eigen *et al.*, the coarse level feature stacks were not fixed during training and were jointly trained along with the fine level feature stacks. The loss function is given by the following equation

$$L(D, D^*) = \sum_{i=1}^N d_i^2 - \frac{1}{2N^2} \left(\sum_{i=1}^N d_i \right)^2 + \frac{1}{N} \sum_{i=1}^N ((\nabla_x d_i)^2 + (\nabla_y d_i)^2), \quad (7.1)$$

where d_i is the difference in predicted log depth and ground truth log depth for the valid pixels N (pixels that contain non-zero depth values in the raw depth data), $\nabla_x d_i$ is the horizontal image gradient of the difference and $\nabla_y d_i$ is its vertical counterpart. This is the same loss criterion employed by Eigen *et al.* in [34].

7.5.2 Surface Normal

Ground truth normals are computed using a variety of techniques in the literature. In this work the normals are estimated by fitting a quadric patch to a set of nearby points in the point cloud, as described in Chapter 5. This provides a more accurate representation of the surface compared to just fitting planar regions, and added little to the overall time complexity as the normals are computed as part of the curvature computation pipeline. A combination of pixel wise Euclidean loss along the three channels corresponding to the three unit vectors $\bar{x}, \bar{y}, \bar{z}$ and the difference in angle between the predicted normals and the ground truth were used as the loss criterion during training. This is expressed as

$$L(\hat{n}, \hat{n}^*) = -\hat{n} \cdot \hat{n}^* + \sum_{i=1}^3 (\hat{n}_i - \hat{n}_i^*)^2, \quad (7.2)$$

where \hat{n} and \hat{n}^* are the predicted and ground truth normal respectively, $\hat{n}_i \in \hat{n}$ and $\hat{n}_i^* \in \hat{n}^*$ are the three components $\{\bar{x}, \bar{y}, \bar{z}\}$ of each of the normals. It was found that including both the Euclidean terms and angular error jointly, improved both the convergence rate and the final accuracy of the system.

7.5.3 Surface Curvature

Principal surface curvatures were estimated using the method from [24], described in Chapter 5. These principal curvatures were extracted using the same settings as described in the original work. The principal curvature values κ_1, κ_2 were limited to a range of $\{-100, 100\}$ in order to avoid the estimation of implausible curvatures, effectively limiting the minimum detectable radius of curvature to be 1cm. This roughly aligns with the precision of the system[76] at the distances present in the training data. The depth data was used as is provided by the NYUv2 dataset [40], and no effort was made to calibrate it using the work from Chapter 4. This was in order to avoid the ambiguity of the evaluation to previous approaches that train and test on uncalibrated data. The curvature estimation is provided for every point in the input depth image (640x480), which was then bicubically down-sampled to 120x160 to generate the database that was used in training the network. The proposed approach estimates principal curvatures values directly as opposed to Gaussian or mean curvature, as these were found to improve performance more during training.

A Euclidean loss criterion was employed 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 we use the following loss function

$$L(C, C^*) = \sum_{i=1}^n \left(\frac{(\kappa_{1i} - \kappa_{1i}^*)^2 + (\kappa_{2i} - \kappa_{2i}^*)^2}{(1 + D_i)^{-2}} \right), \quad (7.3)$$

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 represents the depth in meters for the i^{th} pixel.

7.6 Training The Network

This section is included to provide details of the training procedure

7.6.1 Data Generation

Training data was randomly augmented using flips, translations, rotations, and multiplicative variations of the colour channels. Any sort of motion based augmentation (i.e. not colour changes) were applied jointly to the RGB input, ground truth depth, surface curvature and surface normals in order to obtain consistent training data. Unlike some

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

notable previous approaches [34], the raw depth was used directly from the dataset provided without any post processing to fill holes or smooth surfaces. Additionally the raw depths were used to calculate the surface normals and surface curvature, which was intended to provide a stronger link between our three ground truth sources.

As described in Section 7.5.3 the method in [24] was used to produce training data for surface normals and surface curvature. This approach to curvature and normal estimation is specifically targeted for noisy data such as that from a Microsoft Kinect, and this has been shown in Chapter 5 to produce good estimates for both surface normals and principal curvatures. In practice scaling the ground truth curvature values by a factor of 0.1, to produce a similar range of values to the input depth, improved both qualitative and quantitative results of curvature estimation during training. An inverse scaling is applied to the final prediction for both principal curvatures κ_1 and κ_2 .

7.6.2 Hyperparameters and Weight Initialisation

The accelerated gradient [237] by Nesterov, was used as the optimizer with a base learning rate of 0.1 and a momentum of 0.95 and trained for 50 epochs using a single NVIDIA GeForce GTX 1080, taking approximately 4 days. Weights of the convolutional layers corresponding to feature extraction were initialized using VGG pretrained on ILSVRC [238] image data. This initialisation was found to be more qualitatively and quantitatively beneficial to performance over initialising the feature extraction layers with the VGG weights of [34], although these converged faster. All other convolutional layers, which correspond to depth, normals and curvature estimation and the fully connected layers, were randomly initialized using MSRA weight initialization scheme [37]. This converged much faster compared to initialising the filters from a Gaussian distribution with zero mean and 0.01 standard deviation. During training, each time the training loss plateaued (approximately every 10 epochs) the learning rate was halved and training resumed. This was based on the assumption that the network was essentially circling the solution, but moving too fast to converge. The learning framework used in this work was Caffe [28], and all the experiments were carried out using a mini batch size of 16. Batch-normalisation was not used in this work as it was not an element of the architecture used as a basis for this system, but the network may have been able to benefit from it in hindsight. A change in architecture was explored following this work and is discussed in Section 7.8.5.

7.6.3 Training Separate Models With Equal Model Capacity

Several models were trained with equivalent model capacity to estimate quantities both separately and jointly. This was done to demonstrate that the improved estimates for normals and depths are not the result of increased model capacity, but the result

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

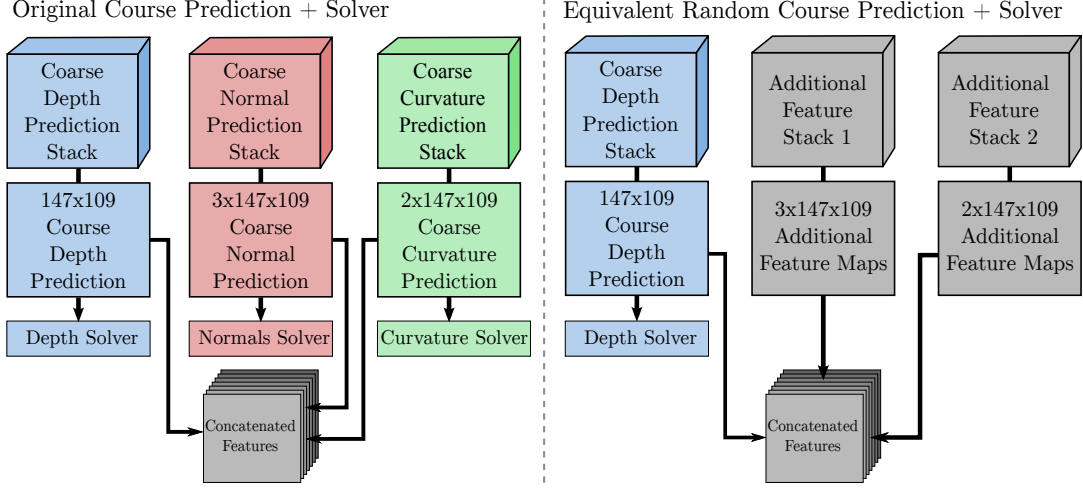


Figure 7.3: **Left:** When all three tasks are trained jointly, there is a solver at the end of the course scale for all three tasks and the coarse feature maps are passed on to the fine scale after being concatenated together. **Right:** When only a single task is trained (in this case depth) there is a single solver at the end of the course scale and the other two stacks now provide additional feature maps which can be trained by the fine scale solver (not shown in the figure).

of including derived features as related tasks for the network. Explicitly 4 models were trained, depth only, normals only, depth and normals, and depth, normals and curvature. Each model maintained a constant model capacity for all tasks. More concretely, when a single quantity network is trained (depths only or normals only) the coarse level convolutional layers corresponding to the other tasks in place are left in place. The key difference is the removal of the intermediate solvers on these feature maps, as only a single solver is attached at the end of the course scale based on the training task. This is demonstrated more clearly in Figure 7.3, in which the course scale prediction section of the network is shown. When training all three tasks jointly (Figure 7.3 Left), there is a solver attached at the end of each prediction stack which is passed to the finer scale as feature maps and refined further. In the scenario where there is only one training task (Figure 7.3 Right), the solver corresponding to the training task is kept intact while the other solvers are removed and the feature maps freed up to learn additional weights in an unguided manner, trained using only the finer scale solver. This preserves the model capacity by keeping the number of feature maps a constant regardless of the task/tasks that is being trained while greatly influencing what is being learnt by the feature maps through the use of additional tasks.

7.7 Evaluation Metrics

The metrics used to evaluate depth, normal and curvature performance are the same as those proposed in [35]. The depth errors are given by

$$\text{Rel}_{abs} = \frac{1}{N} \sum_{i=1}^N (|D_i - D_i^*| / D_i^*), \quad (7.4)$$

$$\text{RMS}_{lin} = \sqrt{\frac{1}{N} \sum_{i=1}^N |D_i - D_i^*|^2}, \quad (7.5)$$

$$\text{RMS}_{log} = \sqrt{\frac{1}{N} \sum_{i=1}^N |\log(D_i) - \log(D_i^*)|^2}, \quad (7.6)$$

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

where N is the number of points in the entire test set, and D_i and D_i^* are the i^{th} predicted and ground truth depth respectively. The normal errors are given by

$$\text{Mean} = \frac{1}{N} \sum_{i=1}^N (|\arccos(\hat{N}_i \cdot \hat{N}_i^*)|) \quad (7.7)$$

$$\text{Median} = \bar{A}\left(\frac{N}{2}\right), \quad (7.8)$$

$$(7.9)$$

where \hat{N}_i and \hat{N}_i^* are the i^{th} predicted and ground truth unit normal vectors respectively, and \bar{A} is the sorted vector of all measured angular differences in the test set. The curvature error is given by

$$\text{RMS}_{\kappa_j} = \sqrt{\frac{1}{N} \sum_{i=1}^N |\kappa_{ij} - \kappa_{ij}^*|^2} \quad (7.10)$$

where κ_j is the j^{th} principal curvature, and κ_{ij} and κ_{ij}^* is the j^{th} predicted and ground truth principal curvature value of the i^{th} test point.

7.8 Network Performance Evaluation

In this section the resulting network is evaluated qualitatively and quantitatively across the three tasks. A practical segmentation example is included in this work to showcase

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

how this work could be applied in a real life scenario. Additionally some discussion of possible architectural improvements and qualitative examples of networks trained on the same data are shown in Section 7.8.5.

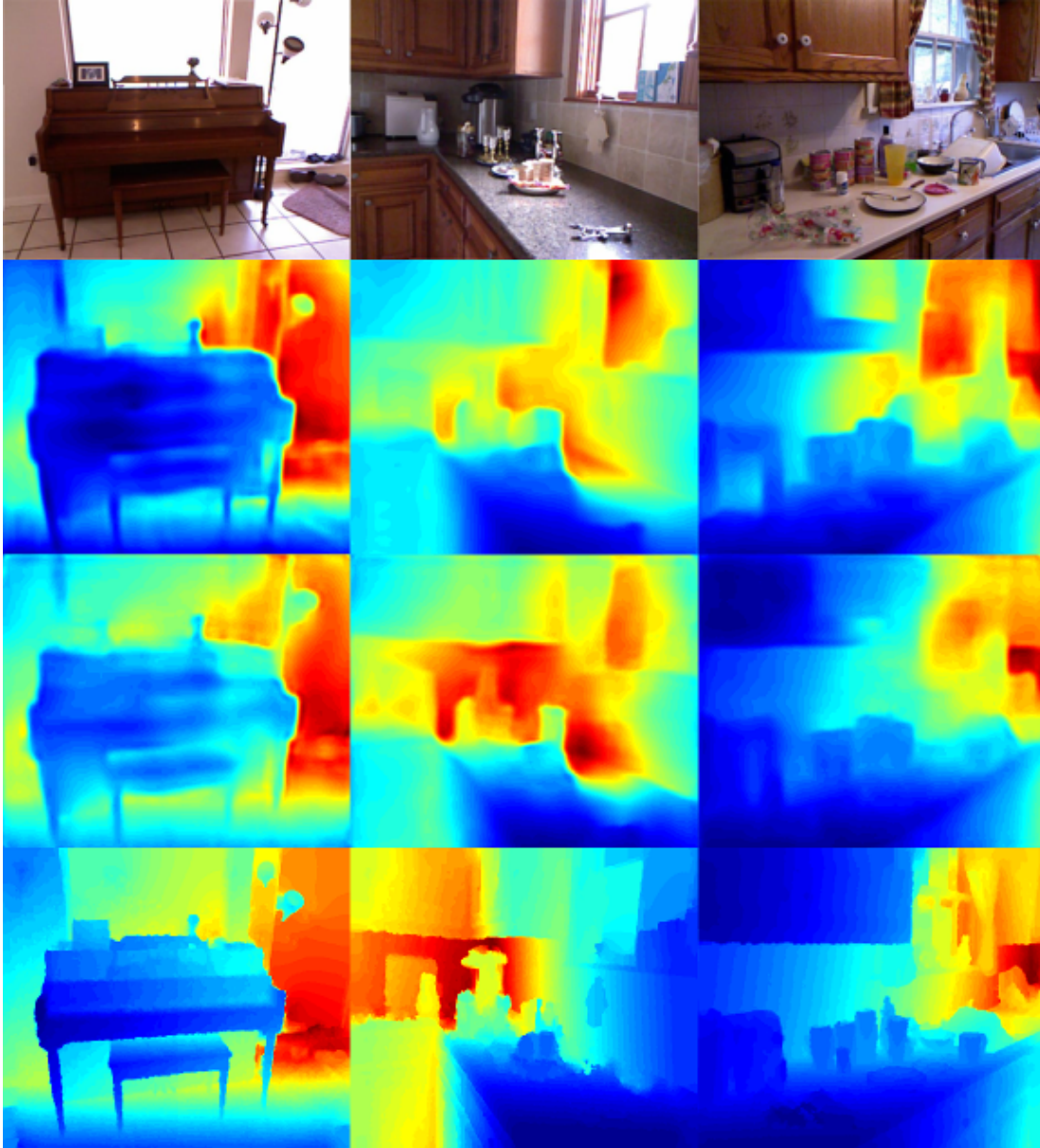


Figure 7.4: Demonstrates the qualitative improvement of our approach for depth estimation. **Top:** RGB image **1st row:** Eigen's Prediction **2nd row:** Our Prediction **Bottom:** Ground Truth

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

Depth Prediction							
Type	Method	<i>lower better</i>			<i>higher better</i>		
		Rel_{abs}	RMS_{lin}	RMS_{log}	δ	δ^2	δ^3
single	Liu[157]	0.230	0.824	-	0.614	0.883	0.972
	Eigen[35]	0.214	0.877	0.283	0.614	0.888	0.972
	Ours(Depth)	0.156	0.646	0.216	0.765	0.949	0.987
	Laina[158]	0.127	0.573	0.195	0.811	0.953	0.988
joint	Eigen(Alex)[34]	0.198	0.753	0.255	0.697	0.912	0.977
	Ours(D+N)	0.156	0.642	0.215	0.766	0.949	0.988
	Eigen(VGG)[34]	0.158	0.641	0.214	0.769	0.950	0.988
	Ours(Full)	0.156	0.624	0.212	0.776	0.953	0.989

Table 7.1: 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).

7.8.1 Depth

Depth predictions were evaluated in the same manner as outlined in previous work [158],[34] and the results are tabulated in Table 7.1. 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 terms of relative performance improvements were primarily in terms of RMS_{lin} , with similar performance for Rel_{abs} and RMS_{log} amongst different network configurations. These quantities are more related to the ratio of predicted and ground truth depths. Methods that estimate depth alone as a single task are included for completeness, although we outperform all the methods except that of Laina *et al.* [158], which uses a much more powerful ResNet [37] architecture. Based on the results of the *joint* task training scheme there is a strong indication that the performance of [158] could still be improved had it been trained simultaneously with normals and surface curvature. These architectural considerations are discussed and explored further in Section 7.8.5.

Adding additional tasks based on related quantities increases performance gains for each addition related task. Also the contribution from curvature is much more significant (reduction of RMSE by 0.02m) compared to the contribution of normals (reduction of RMSE by 0.004m). This is consistent with the contribution of semantic labels (Eigen

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

Surface Normal Predictions : Compared to [230]							Surface Normal Predictions : Compared to [24]						
Type	Method	Angular Error		Within t°			Type	Method	Angular Error		Within t°		
		Mean	Median	$\leq 11.25^\circ$	$\leq 22.5^\circ$	$\leq 30^\circ$			Mean	Median	$\leq 11.25^\circ$	$\leq 22.5^\circ$	$\leq 30^\circ$
single	Wang [228]	26.9	14.8	42.0%	61.2%	68.2%	single	Wang [228]	36.4	26.2	27.2%	45.6%	53.9%
	Ours (Norms)	21.1	13.5	43.6%	66.6%	75.4%		Ours (Norms)	27.7	20.2	31.8%	53.7%	63.8%
	Bansal et al [160]	19.8	12.0	47.9 %	70.0 %	77.8 %		Bansal[160]	27.1	19.0	32.8%	55.8%	65.7%
joint	Eigen(Alex)[34]	23.7	15.5	39.2 %	62.0 %	71.1%	joint	Eigen(Alexnet)[34]	29.7	21.8	30.0%	51.0%	61.0%
	Ours (D+N)	21.1	13.6	43.6%	66.5%	75.4%		Ours (D+N)	27.7	20.2	31.7%	53.6%	63.7%
	Eigen(VGG)[34]	20.9	13.2	44.4%	67.2%	75.9%		Eigen(VGG)[34]	27.3	19.6	32.3%	54.7%	64.6%
	Ours (Full)	20.6	13.0	44.9%	67.7%	76.3%		Ours (Full)	27.2	19.6	32.9%	54.7%	64.7%

Table 7.2: 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 two different methods [230, 24].

VGG [34]) shown in Table 7.1 as it helps to increase the performance. However, curvature provides the largest improvement potentially because it is more tightly coupled to depth.

7.8.2 Surface Normals

Surface normals were compared in the same way as [34, 228, 229]. Although ground truth normal data does not exist, approaches were compared against two different methods of estimating normals from the raw depth data as a proxy. The methods used included the normals as shown in [230] and the method used to compute the input data for training this approach taken from [24]. Qualitatively [230] takes a more aggressive approach to noise and produces overly smoothed out estimates, while the method in [24] produces smooth normals but still provides sharp edges. 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. The results of testing are summarised in Table 7.2 and demonstrate improvements for each normal estimation method over previous approaches. Quantitatively we approach the performance metrics of Bansal et al. [160] who used a skip architecture with a larger model capacity compared to ours, although arguably qualitatively both [34] and the approach proposed in this chapter outperform the predictions from [160] as shown in Figure 7.5.

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.

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION



Figure 7.5: Demonstrates the qualitative improvement of this approach for normal estimation. **Top:** RGB image **2nd row:** Bansal[160], **3rd row:** Eigen[34], **4th row:** Prediction from this approach **Bottom:** Ground Truth [24]. The missing areas in the ground truth normals coincide with those in the raw depth images.

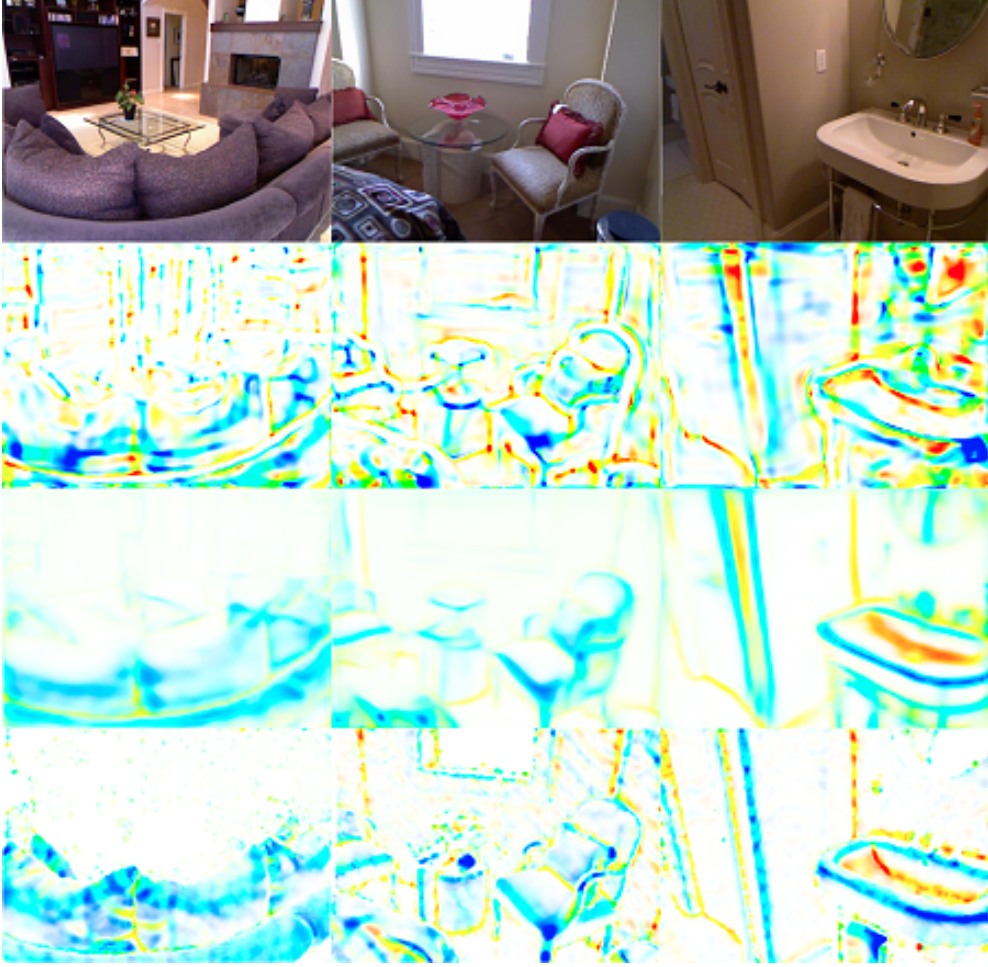


Figure 7.6: Demonstrates the qualitative improvement of our approach for surface curvature estimation. **Top:** RGB image **2nd row:** Computed surface curvature based on Eigen's[34] depth prediction **3rd row:** Prediction of this approach **Bottom:** Ground Truth computed from raw depth data

7.8.3 Surface Curvature

Again the dataset NYUv2 [40] lacks the ground truth surface curvature data, as a method to evaluate the accuracy of estimating surface curvature the performance of this approach is compared against the computed values using the method of [24] on the raw depth data of the test set. The predictions from the proposed network were evaluated against the estimated curvature values computed from the predicted depths produced by the proposed network and the network from [34]. The RMS curvature error of each of the principal curvatures (κ_1, κ_2) is compared against the computed ground truth and the median error of the *mean curvature* $0.5 * (\kappa_1 + \kappa_2)$ across two

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

categories, planar and non-planar. Planar surfaces are defined to be those 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 compared to the model which only learnt surface curvature.

Principal Curvature Predictions

Method [24]	RMS (m^{-1})		Median (m^{-1})		Within σ_t		
	κ_1	κ_2	planar	non-planar	σ_1	σ_2	σ_3
Eigen(Depth) [34]	5.56	7.50	3.86	1.44	25.7%	33.9%	43.5%
Ours (Depth)	6.03	6.50	4.23	1.38	26.9%	34.9%	44.2%
Ours (Curvatures)	3.41	5.17	1.984	0.184	52.6%	63.2%	73.2%
Ours (joint)	2.81	4.47	1.634	0.085	63.1%	72.7%	80.3%

Table 7.3: 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.

Figure 7.6 is included as a reference to show how the metrics in Table 7.3 translate into visual appearance.

7.8.4 Segmentation Using Predicted Quantities

As a purely qualitative demonstration of this approach, a simple example of a scene segmentation was performed that combines information from the colour, depth and curvature on selected scenes. A simple segmentation generated by combining the gradients of colour and depth, and curvature values through the use of a border function. This border function $b(u, v)$ can be expressed as

$$b(u, v) = w_I \cdot \nabla I(u, v) + w_d \cdot \nabla D(u, v) + w_c \cdot C(u, v), \quad (7.11)$$

where $\nabla I(u, v)$ is 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

$$\nabla I(u, v) = \sqrt{\frac{\partial I(u, v)^2}{\partial u} + \frac{\partial I(u, v)^2}{\partial v}} \text{ and } \nabla D(u, v) = \sqrt{\frac{\partial D(u, v)^2}{\partial u} + \frac{\partial D(u, v)^2}{\partial v}}. \quad (7.12)$$

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

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

$$\mathbf{B}(u, v) = \begin{cases} 1 & \text{if } b(u, v) \geq \delta_{thresh} \\ 0 & \text{otherwise} \end{cases}. \quad (7.13)$$

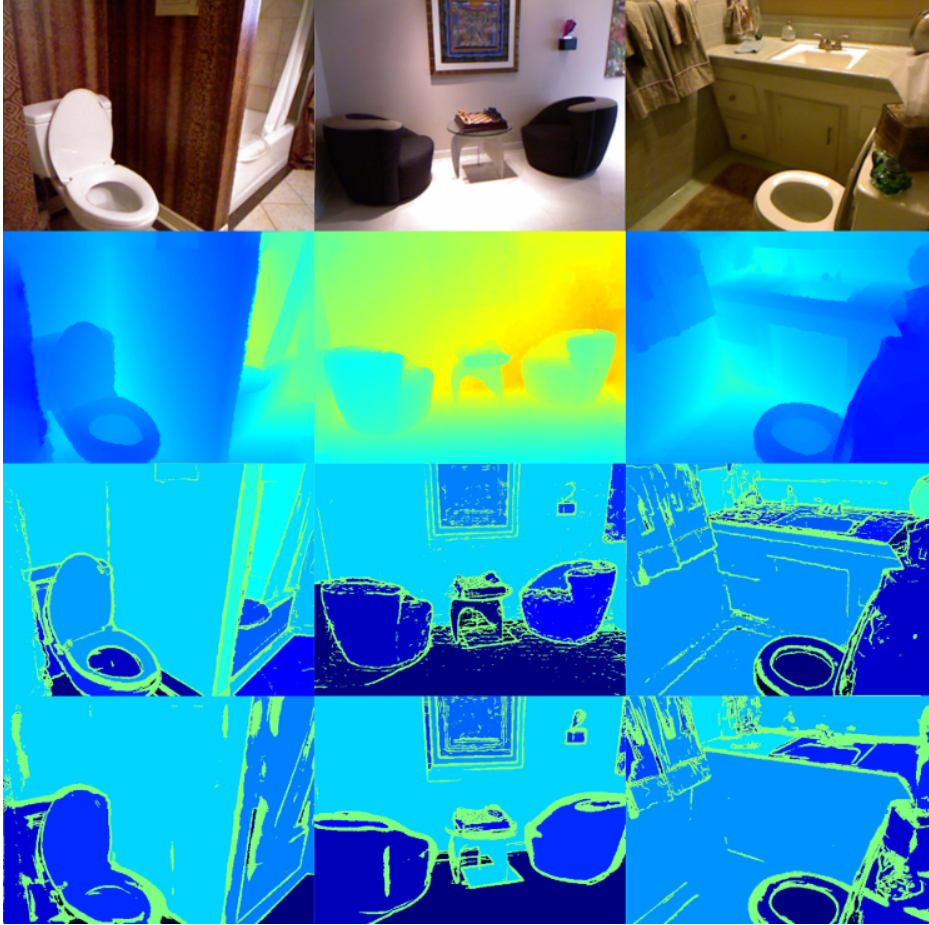


Figure 7.7: Demonstrates a basic segmentation algorithm, that uses colour, depth and curvature to generate a border function. **Top:** Input colour image **2nd row:** Input ground truth depth **3rd row:** Segmentation From GT data **Bottom:** Segmentation from Predicted Data. The key contribution of the depth and curvature to the segmentations, are on the depth boundaries and wall edges that can be difficult to differentiate from colour alone.

The performance was compared for this segmentation method using the ground truth quantities and the predictions (depths and curvature) generated by our network. The results of this are summaries in Figure 7.7. These results are not intended to be treated as state of the art segmentations although this approach is inspired by Mishra *et al.* in [142]. They are included to demonstrate a possible future extension of this work and

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

also to illustrate that the information from the network can be used to perform similar tasks.

7.8.5 Architectural Considerations

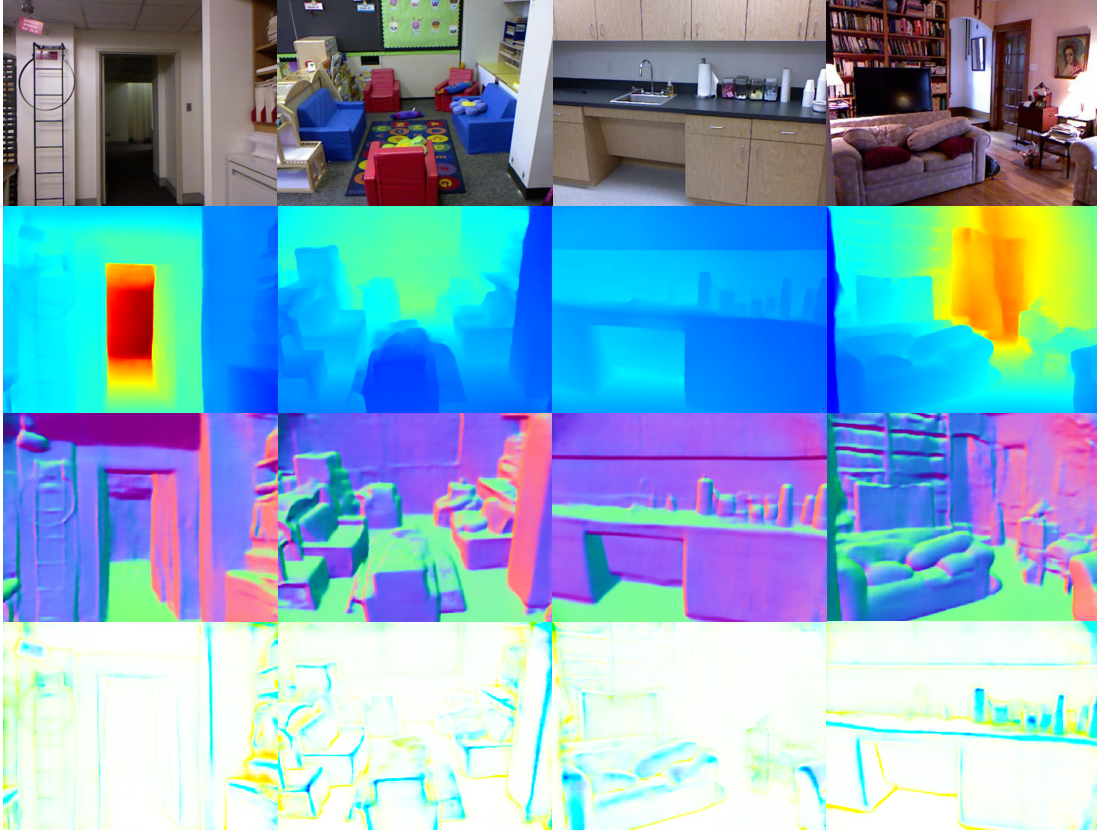


Figure 7.8: **Top:** Input colour image **2nd row:** Predicted depth **3rd row:** Predicted normals **Bottom:** Predicted curvature. Demonstrates a significant qualitative improvement in performance over the approach in this chapter by changing to an architecture similar to that used in Chapter 8, adapted for depth, normal and curvature estimation.

After the completion of this work, investigation into the contribution of architectural choice was performed. Qualitative results generated from the NYUv2 testset are shown in Figure 7.8, which demonstrate a clear improvement in depth and normal estimation over the approach detailed in this chapter. The network used for this is essentially identical to the depth estimation network described in Chapter 8, but similar to the approach described here, a separate convolutional layer is used per quantity. The results of this testing clearly demonstrated that architectural considerations are the most relevant to overall network performance, summarised in Table 7.4. The obviously poor suitability of VGG16 and the network proposed in this work is highlighted by the massive quantitative improvements of an approach like Laina *et al.* in [158] which uses a ResNet style

7 MACHINE LEARNING APPROACHES FOR GEOMETRIC QUANTITY ESTIMATION

approach. Both approaches, VGG16 and ResNet were state of the art architectural designs for the task of semantic image labelling, demonstrating that the features that are important to depth estimation are also important to semantics. However, in [158] there are no fully-connected layers indicating that the depth estimation network may not require whole image context and that this adding computational complexity is unnecessary or even potentially harmful to performance. This intuitively makes sense as the features that determine depth in general have a relatively local context, compared to those that are important for semantics. Such as a cow being in a meadow, where the meadow is a rather nebulous concept in image terms. The network used in Figure 7.8 also has no fully-connected layers, and instead relies on a fully convolutional architecture. This has become a standard approach to dense labelling tasks such as semantic segmentation and depth estimation.

Depth Prediction						
Method	<i>lower better</i>			<i>higher better</i>		
	Rel_{abs}	RMS_{lin}	RMS_{ln}	δ	δ^2	δ^3
Eigen(VGG)[34]	0.158	0.641	0.214	0.769	0.950	0.988
Ours _{VGG} (full)	0.156	0.624	0.212	0.776	0.953	0.989
Laina[158]	0.127	0.573	0.195	0.811	0.953	0.988
Ours _{Dense} (depth)	0.119	0.555	0.176	0.855	0.970	0.992
Ours _{Dense} (full)	0.118	0.513	0.171	0.864	0.972	0.992

Table 7.4: Demonstrates the improved performance of this change in architecture, while also demonstrating the same trend that estimating depth, normals and curvature (Ours_{Dense}(full)) performs better than just depth alone (Ours_{Dense}(depth)).

The architecture of this network is similar to the network presented in Chapter 8, however it has a slightly smaller model capacity and contains auxiliary loss terms for depth, normals and curvature at the previous scale level, similar to the VGG style network presented in this chapter. Again the performance of this new network is compared to an identical network trained only for depth estimation, and demonstrates a significant improvement with the combining loss functions. This network produces near state-of-the-art results on depth estimation. This demonstrates the importance of architectural choice to network performance, also indicating the network improvements designed for semantic labelling are still able to improve performance at depth estimation. Can architectural changes be incorporated that specifically target the improvement of depth estimation? This remains an open question.

7.9 Conclusions and Future Work

In this work presented in this chapter is a unified multi task learning platform which is capable of predicting depths, surface normals and surface curvatures using a single RGB image. The analysis demonstrates that carefully chosen hand crafted feature representations can outperform the machine learnt features provided they are closely related to the prediction task. This indicates that network guidance is an important consideration to network design and should not be ignored when training neural networks. As an extension of this work, an application is provided in Section 7.8.4, and an alternative architecture is explored in Section 7.8.5.

Using Pose Priors to Improve Depth Estimation

This chapter is largely drawn from work performed in collaboration with Thanuja Dhar-masiri, which was formed into the submission [171] and is likely to appear in a similar form in his thesis. This work extends upon the initial work of [159] presented in Chapter 7, to explore the use of a different more complex architecture with an informed method of combining predicted data from multiple networks. This further demonstrates the principal that network guidance through the use of related tasks in training is helpful, even when these tasks are trained across separate networks. The work also advocates for the *appropriate* use of machine learning to the task of relative pose prediction, to understand what a network finds more or less challenging and play to the strengths of the algorithm.

The relative contribution of this work was in fact equal in essentially all relevant areas.

8.1 Motivation

The importance of navigation and mapping to the fields of robotics and computer vision has only increased since its inception. Vision based navigation in particular is an extremely interesting field of research due to its discernible resemblance to human navigation and the wealth of information an image contains. Although creating a machine that understands structure and motion purely from RGB images is challenging, the computer vision community has developed a plethora of algorithms to replicate useful aspects of human vision with a computer. Tracking and mapping remains an unsolved problem, with many popular approaches. Traditional photometric based techniques rely on establishing correspondences across different viewpoints of the same scene and the matching points are then used to perform triangulation, acknowledging the strong geometric priors that exist in the real-world. The field of tracking and mapping can be coarsely divided into dense [14], semi-dense [10] and sparse[50] approaches, each comes with advantages and disadvantages as discussed in Sections 1.1 and 2.2.2.

Applying machine learning techniques to solve vision problems has been another popular area of research. Great advances have been made in the fields of image classification [26, 37, 155] and semantic segmentation [146, 239] and this has led geometry based machine learning researchers to attempt to replicate this success. The massive growth in

neural network driven research has largely been facilitated by the increased availability of low-cost high performance GPUs as well as the relative accessibility of machine learning frameworks such as Tensorflow and Caffe.

In this work, both machine learning approaches (discussed in Section 1.4) as well as SfM techniques (discussed in Section 1.1) are drawn from 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. The network framework was constructed 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 relative motion vector corresponding to a member of the Lie-algebra (\mathfrak{se}_3) of the Special Euclidean Transformation $\mathbb{SE}(3)$, which describes the relative camera motion between the frames as discussed in Section 3.6. As with the work in Chapter 7 the connection between related tasks is tightly coupled inside a network.

8.2 Contributions

The contributions made in this work are summarised as follows:

- An extension of the work detailed in Chapter 7, that explores a change in architecture as well as the use of alternative related error functions.
- A network that achieves state-of-the-art results for single image depth prediction on both NYUv2 (indoor) and KITTI (outdoor) datasets. [Section 8.11: Table 8.1 and Table 8.2]
- A network that outperform previous camera motion prediction frameworks on both TUM and KITTI datasets. [Section 8.11: Table 8.5 and Table 8.4]
- An approach that actively combats scale drift using the knowledge of metric depths and subsequently produce more accurate visual odometry estimates. [Section 8.11: Figure 8.9]

8.3 Related Work

Estimating motion and structure from two or more views is a well studied vision problem. In order to reconstruct the world and estimate camera motion, sparse feature based systems [111, 50] compute correspondences through feature matching while the denser approaches[10, 14] rely on brightness constancy across multiple viewpoints. In this work, we leverage CNNs to solve the aforementioned tasks and we summarize the

existing works in the literature that are related to the ideas presented in this paper.

8.3.1 Single Image Depth Prediction

Predicting depth from a single RGB image using learning based approaches has been explored even prior to the resurgence of CNNs. In [187], Saxena *et al.* employed a Markov Random Field (MRF) to combine global and local image features. Similar to our approach Eigen *et al.* [35] introduced a common CNN architecture capable of predicting depth maps for both indoor and outdoor environments. This concept was later extended to a multi-stage coarse to fine network by Eigen *et al.* in [34]. Advances were made in the form of combining graphical models with CNNs [157] to further improve the accuracy of depth maps, through the use of related geometric tasks [159] and by making architectural improvements specifically designed for depth prediction [158]. Kendall *et al.* demonstrated that predicting depths and uncertainties improve the overall accuracy in [141]. While most of these methods demonstrated impressive results, explicit notion of geometry was not used during any stage of the pipeline which opened the way for geometry based depth prediction approaches.

In one of the earliest works to predict depth using 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 [42]. Left-right consistency was explicitly enforced in the unsupervised framework of Goddard *et al.* [43] as well as in the semi-supervised framework of Kuznetsov *et al.* [44], which is a technique was also found to be beneficial during training on sparse ground truth data.

8.3.2 Optical Flow Prediction

An early work in optical flow prediction using CNNs was [240]. This was later extended by Ilg *et al.* to FlowNet 2.0 [241] which included stacked FlowNets [240] as well as warping layers. Ranjan and Black proposed a spatial pyramid based optical flow prediction network [242]. 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 [243]. 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 [244] showed the usefulness of estimating flow and uncertainty.

8.3.3 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 [245, 246]. Yi *et al.*[117] showed a full feature detection framework can be implemented using deep neural networks. Rad and Lepetit in BB8[247] 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 [248].

However, each 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.* [46] and Ummenhofer *et al.* [45] 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, the work discussed in this chapter is able produce far more accurate visual odometry and combat against scale drift. Architectural improvements combined with the loss functions employed, also yielded better depth prediction results (see Section 8.11).

8.4 Network Architecture

The overall architecture consists of 3 main subsystems in the form of a depth, flow and camera pose network. A large percentage of the model capacity is invested in to 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. To provide an overall understanding of the data flow a high level diagram of the network is shown in Figure 8.1.

8.5 Depth Prediction

The depth prediction network consists of an encoder and a decoder module. The encoder network is largely based on the DenseNet161 architecture described in [155]. In particular we use the variant pre-trained on ImageNet [238] and slightly increase the receptive field of the pooling layers. It takes a global mean subtracted RGB image as

Although there are separate models for indoor and outdoor scenes the underlying architecture is common.

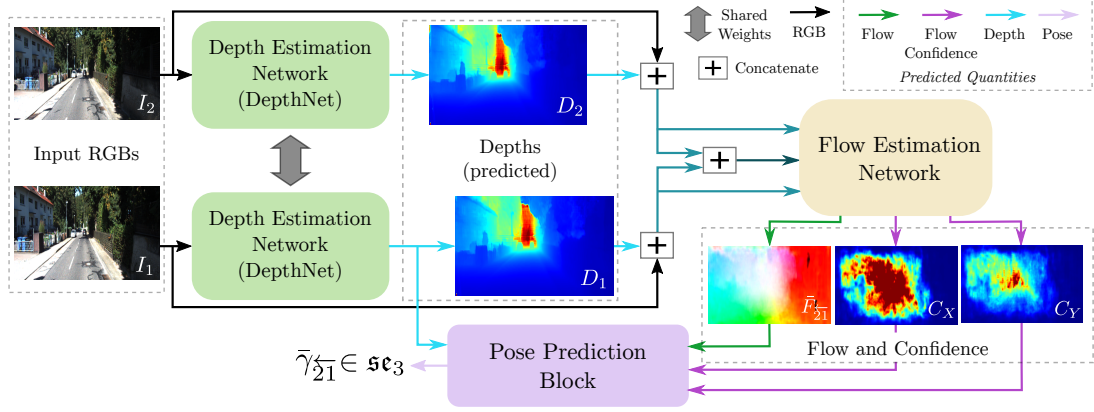
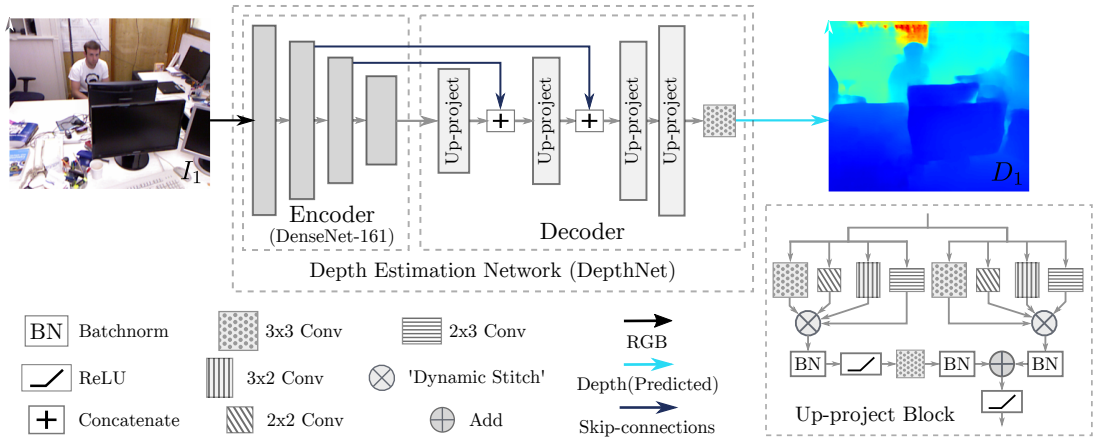


Figure 8.1: Overview of the system's full pipeline.

an input, during the feature encoding stage the resolution of the activations are reduced by a factor of 16 in 4 stages. The first down-sampling operation is performed using a strided convolutional layer, the next with a max-pooling layer and the final two with average-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. Skip connections are used in order to re-introduce the finer details lost during pooling, similar to [140]. 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 this network, which provide better depth maps compared to de-convolutional layers as shown in [158]. A visual summary of the resulting network is shown in Figure 8.2.


 Figure 8.2: The Depth Prediction Network. We include a summary of all operations (*bottom-left*) as well as description of the up-project blocks used in the decoder (*bottom-right*)

Due to the availability of dense ground truth data for indoor datasets (e.g NYUv2 [40], RGB-D[51]) this network can be directly utilised to perform supervised learning. The ground truth data for the outdoor datasets (KITTI) is much sparser and led to the incorporation a semi-supervised learning approach to provide a strong training signal. Therefore, during training on KITTI, a Siamese version of the depth network with complete weight-sharing is used and photometric consistency is enforced between the left-right image pairs through an additional loss function. This is similar to the previous semi-supervised approaches [42, 44] and is only required during the training stage, during inference only a single input image is required to perform depth estimation using this network.

8.6 Flow Prediction

The flow network provides 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 the flow predictions of [241] were integrated with the depths predicted by the network in Section 8.5. This was intended to examine the performance of the produced system in a configuration that uses a state of the art off-the-shelf approach to investigate how important the creation of a novel flow network was. The main limitation of using the flows from [241], was the lack of a mechanism to filter out the dynamic objects which are abundant in outdoor environments and greatly effect the resulting pose estimates. This was solved by estimating confidence, specifically the information matrix in addition to the optical flow. More concretely, for each pixel the flow network created for this work predicts 5 quantities, the optical flow $\bar{F} = [\Delta u, \Delta v]^T$ in the u and v normalised pixel directions (as described in Section 3.2), and the quantities $\hat{\alpha}$, $\hat{\gamma}$ and $\hat{\beta}$. These final three quantities are required to compute the information matrix (\mathcal{I}) or the inverse of the covariance matrix of the predicted flow given by

$$\mathcal{I} = \begin{bmatrix} \alpha & \beta \\ \beta & \gamma \end{bmatrix}, \quad \alpha = e^{\hat{\alpha}}, \gamma = e^{\hat{\gamma}}, \beta = e^{\frac{\hat{\gamma} + \hat{\alpha}}{2}} \tanh(\hat{\beta}). \quad (8.1)$$

This parameterisation guarantees \mathcal{I} is positive-definite and can be used to parameterise any 2×2 information matrix. This made the gradients 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 $\hat{\beta} \rightarrow \pm \infty$. By formulating the uncertainty of the flow explicitly through the information matrix in this way the system can use this information to aid in the computation of the relative pose, by providing an estimated weighting of each point.

With respect to the architecture, elements were borrowed from FlowNet [240] as well as FlowNet 2.0 [241]. As mentioned in [241], FlowNet 2.0 was unable to reliably estimate

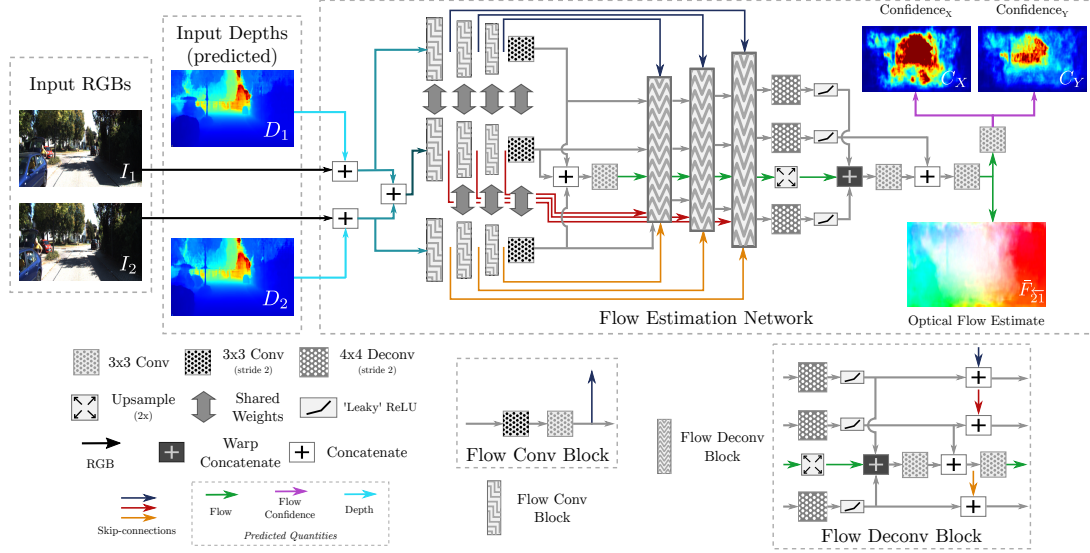


Figure 8.3: The optical flow prediction network. A summary of all operations (*bottom-left*) as well as description of the *flow-conv* and *flow-deconv* blocks is included *bottom-right*

small motions, this was addressed with two key changes. Firstly, the flow network takes the predicted depth map as an input, allowing the network to learn the relationship between depth and flow explicitly. That is, closer objects appear to move more compared to the objects that are further away from the camera. Secondly, “warp-concatenation” is used, where coarse flow estimates are used to warp the CNN activation tensors during the decoder stage before concatenating the resulting warped activations. This appears to resolve small motions more effectively particularly on the TUM [51] dataset as shown in Section 8.11.

8.7 Pose Estimation

Two approaches were taken to pose estimation, shown in Figure 8.4, an iterative and a fully-connected(FC). This was an attempt to contrast 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. FC layers were used 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.

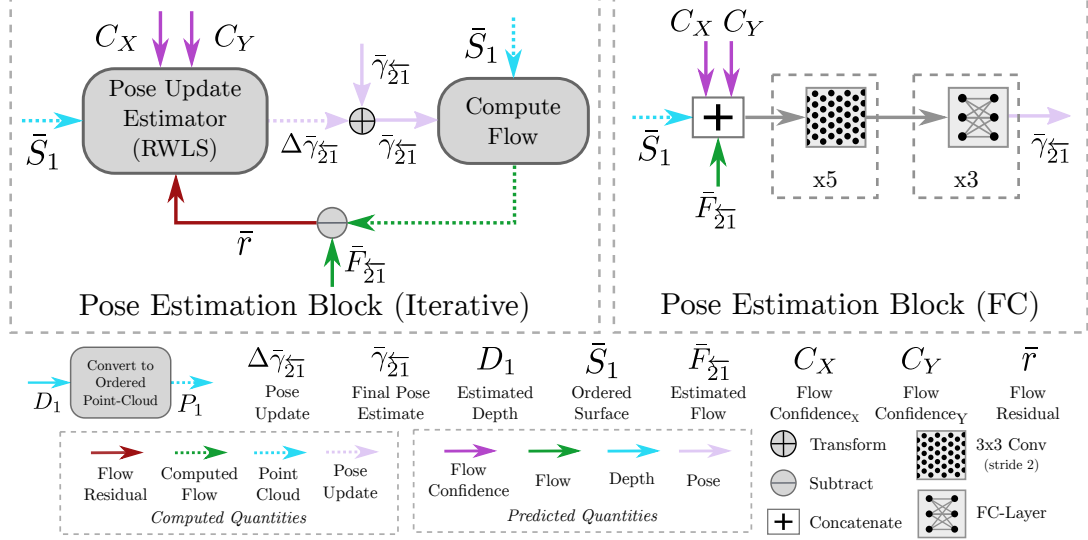


Figure 8.4: Details the two approaches used to estimate the relative pose alignment between adjacent frames (best viewed in colour). *Top-Left* The iterative approach taken, that incorporates a re-weighted least-squares solver (RWLS) into a pose estimation loop. Details in Section 8.7.1. *Top-Right* The fully-connected (FC) approach, which incorporates a succession of 3×3 strided convolutions, followed by several FC layers. Detailed in Section 8.7.2. *Bottom* Summary of the information in this figure.

8.7.1 Iterative

This approach uses a more conventional method for computing relative pose estimates. A standard re-weighted least squares solver (similar to the approaches throughout the previous chapters of this thesis) based on the residual flow, given an estimate of the relative transformation. This approach attempts to minimise the following error function with respect to the relative transformation

$$\bar{\epsilon}_{21} = \sum_{i=1}^N ((p_i - \mathbf{E}_{21} q_i) - \bar{F}_{21}(u_i)) = \sum_{i=1}^N (\bar{F}_{21}^+(u_i) - \bar{F}_{21}(u_i)), \quad (8.2)$$

where $\bar{\epsilon}_{21} \in \mathbb{R}^2$ is the total residual flow vector in normalised camera coordinates, $p_i \in \bar{S}_1$ is the i^{th} homogeneous inverse depth coordinate $p_i = \begin{bmatrix} u & v & 1 & q \end{bmatrix}^T$ of a surface (\bar{S}_1), $\bar{F}_{21}(u_i)$ and $\bar{F}_{21}^+(u_i)$ are the i^{th} predicted flow and estimated flow respectively, and $u_i = \begin{bmatrix} X & Y \end{bmatrix}_i^T$ is the i^{th} pixel coordinate. For simplicity, the values flow values are expressed in terms of normalised camera coordinates. The estimated flow \bar{F}_{21}^+ is computed from the normalised camera coordinate and the current estimated transformation $\mathbf{E}_{21} \in \mathbf{SE}(3)$ as shown in Equation 8.2. As previously shown in Section 3.6 the transformation matrices can be expressed using a matrix exponential as

$\mathbf{E}_{\bar{\gamma}_{21}} = e^{\sum_{j=1}^6 \gamma_j \mathbf{G}_j}$, where $\gamma_j \in \bar{\gamma}_{21}$ is the j^{th} component of the motion vector $\bar{\gamma}_{21} \in \mathbb{R}^6$, 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. The residual function was differentiated with respect to the motion parameters to generate the following Jacobian

$$\mathbf{J}_i = \begin{bmatrix} q & 0 & -uq & -uv & u^2 + 1 & -v \\ 0 & q & -vq & -v^2 - 1 & uv & u \end{bmatrix}, \quad (8.3)$$

where \mathbf{J}_i is the i^{th} Jacobian, which is concatenated to form a larger Jacobian matrix $\mathbf{J} = [\mathbf{J}_1^T, \mathbf{J}_2^T, \dots, \mathbf{J}_N^T]^T$. Additionally the residual vectors can be concatenated to form the final residual vector $\bar{r} = [\bar{r}_1^T, \bar{r}_2^T, \dots, \bar{r}_N^T]^T$. Using a standard Gauss-Newton approach (as described in Section 3.5) the loss function \mathbf{e} was reduced iteratively through successive updates to the pose vector calculated using

$$\Delta \bar{\gamma}_{21} = (\mathbf{J}^T \mathbf{W} \mathbf{J})^{-1} \mathbf{J}^T \mathbf{W} \bar{r}, \quad (8.4)$$

where $\Delta \bar{\gamma}_{21}$ is the additive update to the motion parameters $\bar{\gamma}_{21}$, and \mathbf{W} is a diagonal weight matrix $\mathbf{W} = \text{diag}(\{\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_N\})$. $\mathbf{W}_i = \mathbf{W}(\mathbf{u}_i)$ is the i^{th} weight matrix defined by

$$\mathbf{W}_i = \begin{bmatrix} (C_{Xi}m^2)/(m^2 + \bar{r}_{Xi}^2) & 0 \\ 0 & (C_{Yi}m^2)/(m^2 + \bar{r}_{Yi}^2) \end{bmatrix}, \quad (8.5)$$

where C_{Xi} is the i^{th} confidence value in the x-direction, m is a constant that is computed from the residual \bar{r}_i (Equation 8.2), to be the mean residual magnitude of a single image, and \bar{r}_{Xi} is the i^{th} residual in the x-direction. This pipeline is implemented in Tensorflow [206] and allows us to train the network end to end.

8.7.2 Fully-Connected

Similar to Zhou *et al.* [46] and Ummenhofer *et al.* [45] a fully connected layer based pose estimation network was constructed. This network utilised 3 stacked fully connected layers and was provided with the same inputs as the iterative approach, as shown in Figure 8.4. This method ultimately did outperform the pose estimation benchmarks of [46] and [45] using fully-connected network. However, the iterative network is the approach that should be used. This preference for an iterative approach is beyond the improved performance, and is discussed further in Section 8.11.

8.8 Loss Function Design

8.8.1 Depth Losses

For supervised training on indoor and outdoor datasets a reverse Huber loss function [158] was used, defined by

$$\mathcal{L}_B(D_i, D_i^*) = \begin{cases} |D_i - D_i^*| & |D_i - D_i^*| < c, \\ ((D_i - D_i^*)^2 + c^2)/(2c) & |D_i - D_i^*| > c, \end{cases} \quad (8.6)$$

where $c = \frac{1}{5}\max(D_i - D_i^*)$, and $D_i = D(u_i)$ and $D_i^* = D^*(u_i)$ represent the i^{th} predicted and the ground truth depth respectively. For the KITTI dataset an additional photometric loss was employed during training as the ground truth is highly sparse, as the LIDAR data used to generate the ground truth depth only samples the bottom half of the image in this dataset. This unsupervised loss term enforces left-right consistency between stereo pairs, defined by

$$\begin{aligned} \mathcal{L}_C = & \frac{1}{n} \sum_{i=1}^n |I_L(u_i) - I_R(\pi(\mathbf{K}\mathbf{E}_{LR}^{-1}\pi^{-1}(D_i^L, u_i)))| \\ & + \frac{1}{n} \sum_{i=1}^n |I_R(u_i) - I_L(\pi(\mathbf{K}\mathbf{E}_{RL}^{-1}\pi^{-1}(D_i^R, u_i)))|, \end{aligned} \quad (8.7)$$

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, $\pi(x) = ((x_0/x_2), (x_1/x_2))^T$ is a normalisation function where $x \in \mathbb{R}^3$ is a 3D point, \mathbf{K} is the camera intrinsic matrix as described in Section 3.2. The function $\pi^{-1}(u_i, D) = D\mathbf{K}^{-1}(u_i)$ is the transformation from pixel to camera coordinates, and $\mathbf{E}_{RL}^{-1} \in \mathbf{SE}(3)$ and $\mathbf{E}_{LR}^{-1} \in \mathbf{SE}(3)$ 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, a smoothness term is used, which is defined by

$$\mathcal{L}_S = \frac{1}{n} \sum_{i=1}^n (|\nabla_x D_i| + |\nabla_y D_i|), \quad (8.8)$$

where ∇_x and ∇_y are the horizontal and vertical gradients of the predicted depth. This was found to provide qualitatively better depths as well as a faster convergence rate during training. The final loss function used to train KITTI depths is given by

$$\mathcal{L}_{ss} = \lambda_1 \mathcal{L}_B + \lambda_2 \mathcal{L}_C + \lambda_3 \mathcal{L}_S, \quad \lambda_1 = 2, \lambda_2 = 1, \lambda_3 = e^{-4}, \quad (8.9)$$

where \mathcal{L}_B and \mathcal{L}_S are computed on both left and right images separately.

8.8.2 Flow Loss

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

$$p(x|\mu, \mathcal{I}) = ((|\mathcal{I}|^{\frac{1}{2}})/(2\pi))e^{-\frac{1}{2}(x-\bar{\mu})^T \mathcal{I}(x-\bar{\mu})}, \quad (8.10)$$

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

$$\mathcal{L}_F = \frac{1}{2}((\bar{F}_{21} - \bar{F}_{21}^*)^T \mathcal{I}(\bar{F}_{21} - \bar{F}_{21}^*) - \log |\mathcal{I}|), \quad (8.11)$$

where \bar{F}_{21} is the predicted flow, and \bar{F}_{21}^* is the ground truth flow. This optimises by maximising the log-likelihood of the probability distribution over the residual flow error. This is an important aspect of the approach as it feeds into the re-weighting function of the iterative pose estimation network.

8.8.3 Pose Loss

Given two input images I_1, I_2 , the predicted depth map D_1 of I_1 and the predicted relative pose $\bar{\gamma}_{21} \in \mathbb{R}^6$ the unsupervised loss \mathcal{L}_U and pose loss \mathcal{L}_P can be defined as

$$\mathcal{L}_U = \frac{1}{n} \sum_{i=1}^n |I_1(u_i) - I_2(\pi(\mathbf{K}\mathbf{E}_{21}\pi^{-1}(D_1(u_i), u_i)))|, \text{ and} \quad (8.12)$$

$$\mathcal{L}_P = \|\bar{\gamma}_{21} - \log_e(\mathbf{E}_{21}^*)\|_2 = \|\bar{\gamma}_{21} - \bar{\gamma}_{21}^*\|_2, \quad (8.13)$$

where $\log_e(\mathbf{E})$ maps a transformation \mathbf{E} from the Lie-group $\mathbf{SE}(3)$ to the Lie-algebra \mathfrak{se}_3 as described in Section 3.6. Finally, the ground truth relative pose parameters are given by $\bar{\gamma}_{21}^*$.

8.9 Network Training Regime

The network is trained end-to-end on the NYUv2 [40], TUM[51] and KITTI[41] datasets. The standard test/train split for NYUv2 and KITTI was used, and a new scene split defined for TUM as this is not a traditional dataset used for this purpose. The amount of training data used by the proposed approach in this chapter was radically reduced compared to [46] and [45]. More concretely, for NYUv2 $\approx 3\%$ of the full dataset was used and for KITTI $\approx 25\%$ was used. The reason for the reduction was not an investigation in small training examples, although it shows that the network doesn't actually require a huge amount of training data to perform exceptionally well. One of the key motivations of the reduced training size, particularly in the case of the KITTI dataset, was to avoid

overlap with testsets used for other tasks. In the KITTI dataset, many of the training datasets overlap with other testing datasets, so to avoid these collisions potentially marring the results, all overlaps were removed. This was something, both [45] and [46] seemed to ignore, and as such both include test images in their training set. For all network training the Adam optimiser [249] was used with an initial learning rate of 1×10^{-4} and Tensorflow [206] was the chosen learning framework. The network training was performed on a an NVIDIA-DGX1, and split across each of the GPUs to reduce the overall training time significantly.

8.9.1 Depth Training

All of the DenseNet-161 layers [155] of the depth nets are initialised using Imagenet[238] pretrained weights. Remainder of the layers are intialised using MSRC[37] initialisation. NYUv2[40] and TUM[51] models are trained purely using the supervised loss term, as the ground truth data is essentially dense, and no ground-truth poses are available from the dataset NYUv2. The network is regularized using a weight decay of 1×10^{-4} through out training and the learning rate schedule is shown below:

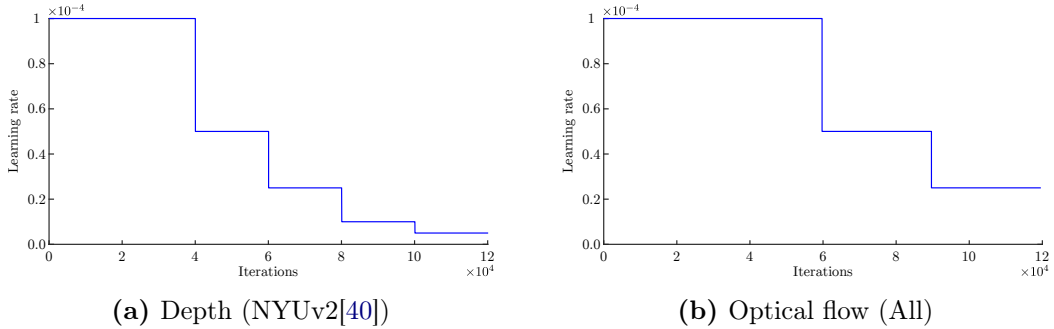


Figure 8.5: Learning rate schedules for network training

Out of $\approx 400,000$ images in the NYU dataset, only 12,000 image were used during training. We perform data augmentation in four ways to each image increasing the total training set to 48000 images, using color shifts, random crops and left-right flips. The input data was augmented offline to reduce the overall training time. The train set and the corresponding ground truth are downsampled by a factor of two, in order to fit a reasonable sized batch on an individual GPU. Hence, the resolution of each training example becomes 320×240 , which due to the network’s fully convolutional approach is the same as the output resolution. Each training batch contains 8 images and we use between 4 and 8 GPUs, resulting in a overall batch size of 32 to 64. In terms of training speed we observe on average 19.3 examples/sec or 0.415 sec/batch using the NVidia P100 GPUs present in the DGX1.

For the KITTI dataset 10,000 training images were used. This train set is extracted out

of the proposed train set by [35], where any images that are part of the odometry test set were excluded to ensure a proper separation between all networks. As the training set is significantly smaller, the learning rate schedule used to train on NYUv2 (as shown in Figure 8.5a) was doubled to avoid over fitting (as described in Section 3.10.4.3).

8.9.2 Optical Flow Training

In order to compute the ground truth optical flow image, for the NYUv2 [40] dataset the relative camera pose was computed using the Iterative Closest Point (ICP) algorithm (as described in Section 3.7) and these poses were used with the ground truth depth map to compute optical flow. This is done by transforming the valid depths and projecting to the image plane. Points with high photometric inconsistency and those that project outside the image are removed from the training data. The ground truth poses are provided for both the TUM[51] and KITTI[41] dataset which allowed the use of these poses to generate the ground truth flows, and they were cleaned up in the same way as for NYUv2. The network was then trained using the optical flow loss criterion from Equation 8.11. All the layers of the flow network are initialised using the MSRC[37] initialisation and the learning rate schedule is shown in Figure 8.5b. This also demonstrates the overall training duration is much smaller compared to the depth network training. The reduced training time is used as the primary objective at this stage is to obtain a crude representation for both optical flow and the information matrix as a complete end-to-end fine tuning happens when the network is trained using the pose loss criterion from Equation 8.13.

8.9.3 Pose Training

The full network was optimised end-to-end using the pose loss. The full network (constructed from both depth networks, the flow network and the iterative or fully-connected pose network) was trained for 20,000 iterations with an initial learning rate of 1×10^{-5} which is halved at the half-way point. Due to the size and complexity of the overall network, a lower initial learning rate was used (1×10^{-5}) and this was halved, this was found to stabilise the overall resulting network. An important consideration in training, particularly when combining multiple loss functions as was done in this research, is the relative weighting of these losses. This process requires significant trial and error and a more rigorous approach to weighting should be a further point of investigation, possible making these learning meta-parameters but this requires careful consideration. This training regime was able to demonstrate that the state-of-the art depths can be further improved using the knowledge of pose predicted by the network. This is an extension of the ideas conveyed in Chapter 7, which showed that related loss functions aid in network performance. Clearly depth, flow and pose are highly related to, as having any two of these quantities allows the computation of the third.

8.10 Evaluation Metrics

The depth is quantitatively evaluated in the same manner as Chapter 7, using the error metrics outlined in Section 7.7. These pose metrics used were drawn from [51]. Pose is quantitatively evaluated using the absolute trajectory error (ATE) in the same way as for Chapter 6. Additionally relative pose error (RPE) is also evaluated, which is calculated using

$$\text{RPE}_{jk} = \left(\mathbf{E}_{ji}^{*-1} \mathbf{E}_{ki}^* \right)^{-1} \left(\mathbf{E}_{ji}^{-1} \mathbf{E}_{ki} \right) \quad (8.14)$$

where \mathbf{E}_{ji}^{*-1} and \mathbf{E}_{ji}^{-1} are the ground truth and predicted transformations from frame i to frame j . As described in [51], the RPE provides a measure of the system’s tendency to drift while the ATE gives an overall error of a resulting trajectory. The latter of these metrics favours systems that use information from the entire trajectory, such as those that close loops and perform bundle-adjustment, as these techniques redistribute the error across poses. In the configuration used by this approach, the pose is estimated in an open-loop visual odometry tracking approach, which makes the drift very influential to tracking accuracy.

8.11 Network Performance Evaluation

In this section the single-image depth prediction and relative pose estimation performance of the proposed system is summarised for several popular machine learning and SLAM datasets. Additionally the effect of using alternative optical flow estimates from [241] and [250] in this network’s pose estimation pipeline was investigated as an ablation study, discussed in Section 8.12.

8.11.1 Depth Estimation

The results of evaluating the single-image depth estimation on the datasets NYUv2[40], RGB-D[51] and KITTI[41] in is shown in Tables 8.1, 8.2 and 8.3 respectively. This quantitatively compares the current approach against previous state-of-the-art approaches using the standard metrics proposed in [35].

Significant improvement across all datasets is demonstrated using the proposed baseline approach (*Ours(baseline)*), which was developed for this work. This effectively validated the choice of architecture used in the depth estimation network. Importantly an additional consistent improvement across all datasets is demonstrated when inferring using the fully end-to-end trained network (*Ours(full)*). This is most noticeable in Tables 8.2 and 8.3, for which ground truth pose data was available for training. This seems to indicate the quality of the *ground-truth* poses has an effect on the improvement, as

8 USING POSE PRIORS TO IMPROVE DEPTH ESTIMATION

the results in Table 8.1 are less significant after training which used the computed ICP poses. However, this still validates the proposed method for improving single image depth estimation performance, and demonstrates a depth estimation network can be improved by enforcing more geometric priors on the loss functions.

Method	<i>lower better</i>			<i>higher better</i>		
	RMS_{lin}	RMS_{ln}	Rel_{abs}	δ	δ^2	δ^3
Eigen _{vgg} [34]	0.641	0.214	0.16	76.9%	95.0%	98.8%
Laina <i>et al.</i> [158]	0.573	0.195	0.13	81.1%	95.3%	98.8%
Kendall <i>et al.</i> [141]	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 8.1: The performance of several approaches evaluated on single-image depth estimation using the standard testset of NYUv2[40] proposed in [34]

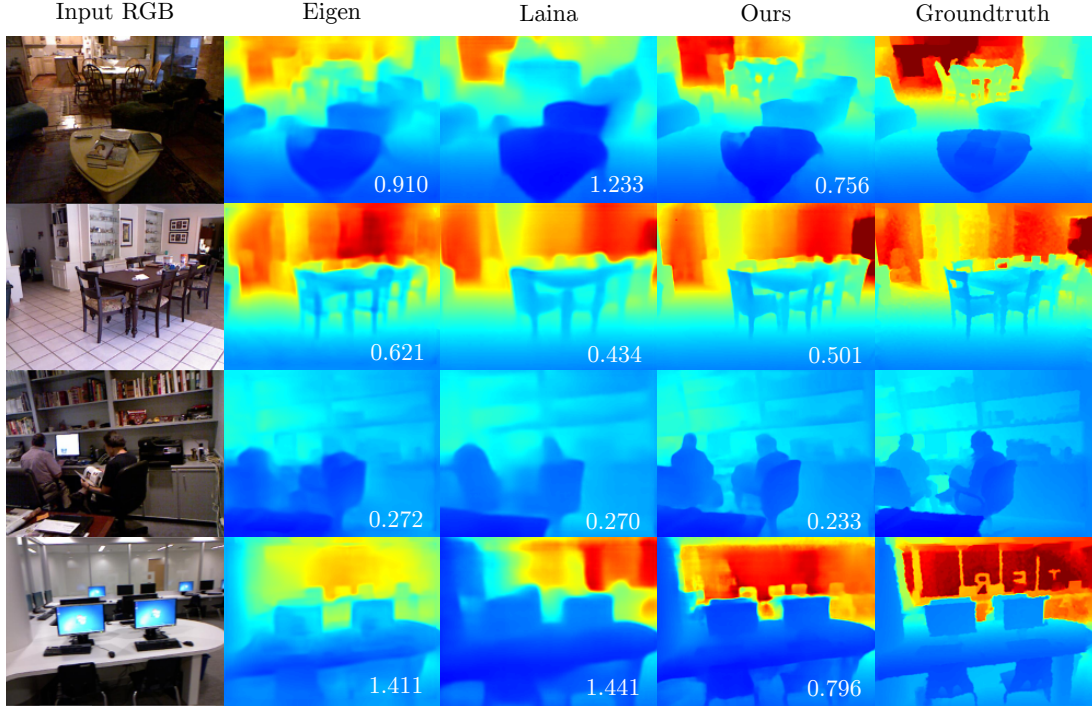


Figure 8.6: Resulting single image depth estimation for several approaches and ours against the ground truth on the dataset NYUv2[40]. The RMSE for each prediction is included

Qualitative results are shown for NYUv2[40] and KITTI[41] in Figure 8.6 and 8.7 respectively. Each of which illustrates a noticeable qualitative improvement over previous methods. This also demonstrates that the contribution of this approach is beyond the numbers, as the proposed approach generates more convincing depths even when the

8 USING POSE PRIORS TO IMPROVE DEPTH ESTIMATION

Cap	Method	<i>lower better</i>			<i>higher better</i>		
		RMS_{lin}	RMS_{ln}	Rel_{abs}	δ	δ^2	δ^3
0-80m	SfM-learner[46]	6.856	0.283	0.208	67.8%	88.5%	95.7%
	Godard <i>et al.</i> [43]	4.935	0.206	0.141	86.1%	94.9%	97.6%
	Kuznietsov <i>et al.</i> [44]	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[46]	5.181	0.264	0.201	69.6%	90.0%	96.6%
	Garg <i>et al.</i> [42]	5.104	0.273	0.169	74.0%	90.4%	96.2%
	Godard <i>et al.</i> [43]	3.729	0.194	0.108	87.3%	95.4%	97.9%
	Kuznietsov <i>et al.</i> [44]	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%

Table 8.2: The performance of previous state-of-the-art approaches evaluated on the standard testset of the KITTI dataset [41]

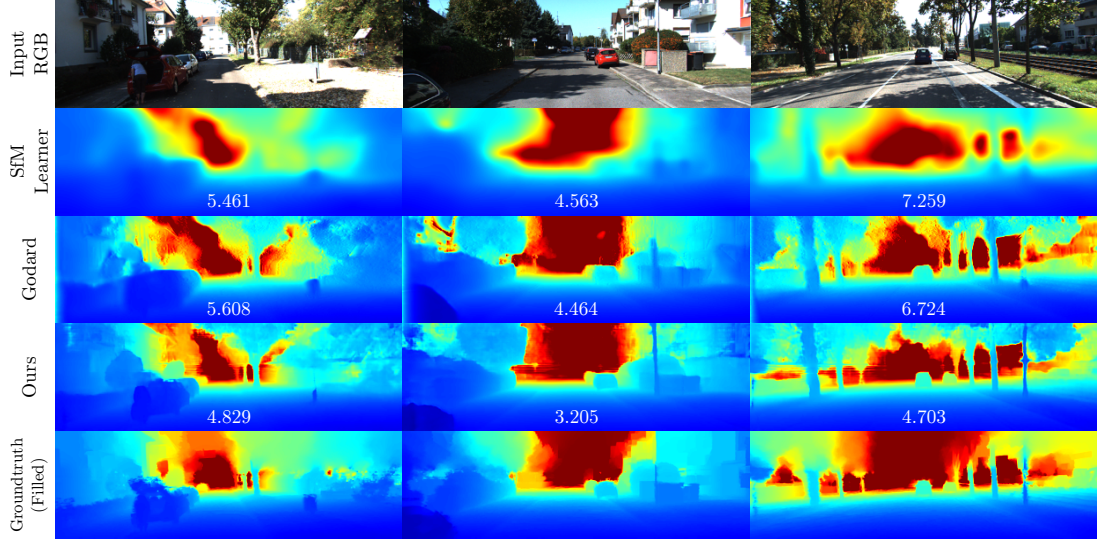


Figure 8.7: The resulting single image depth estimation for several approaches including SfM-Learner[46], Godard[43] and Ours against a ground truth filled using [251] on the testset of the KITTI dataset [41]. We include the RMSE values for each methods prediction. Filled depths are included for visualisation purposes during evaluation the predictions are evaluated against the sparse velodyne ground truth data.

RMS_{lin} depth error may be higher. This is demonstrated by the second row of Figure 8.6, where [158] computes a lower RMS_{lin} . More impressive still are the results in Figure 8.7, where the proposed approach is compared against previous approaches (that are trained on much larger training sets than the one used in this work) and demon-

8 USING POSE PRIORS TO IMPROVE DEPTH ESTIMATION

Method	<i>lower better</i>				<i>higher better</i>		
	RMS_{lin}	RMS_{log}	Rel_{abs}	Rel_{sqr}	δ	δ^2	δ^3
Laina <i>et al.</i> [158]	1.275	0.481	0.189	0.371	75.3%	89.1%	91.8%
DeMoN(est)[45]	2.980	0.910	1.413	5.109	21.0%	36.6%	48.9%
DeMoN(gt)[45]	1.584	0.555	0.301	0.581	52.7%	70.7%	80.7%
Ours(baseline)	1.068	0.353	0.128	0.236	86.9%	92.2%	93.5%
Ours(full)	0.996	0.329	0.108	0.194	90.3%	93.6%	94.5%

Table 8.3: The performance of previous state-of-the-art approaches on a randomly selected subset of the frames from the RGB-D dataset [51]. Entries are separated for DeMoN(est) and DeMoN(gt), the former is scaled by the estimated scale of their system while the latter is scaled by the median ground-truth depth

strates noticeable qualitative improvements. It should be stressed that all comparisons are performed on single-image depth estimation, that is at testing the network only has access to a single image. This validates that the improvement is genuinely in terms of the system’s ability to estimate depth from image features alone.

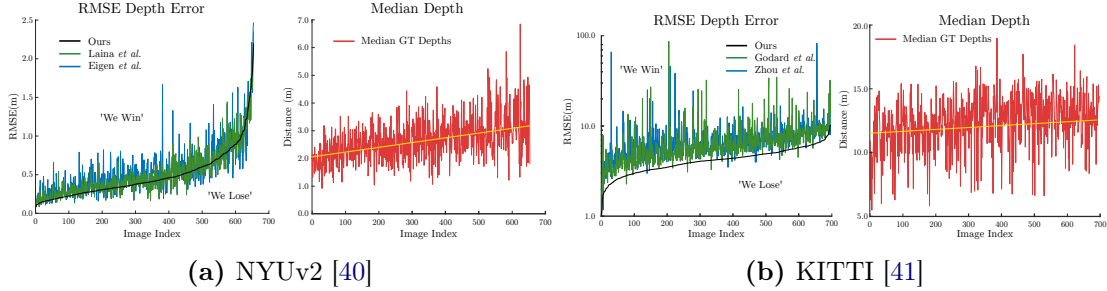


Figure 8.8: **Left:** The RMS_{lin} error (RMSE) on each image of the test set, sorted by our performance on the dataset. The two competing approaches are included, as well as marking which side of the line indicates where the proposed approach is better (‘We Win’) and where it is worse (‘We Lose’). **Right:** The median ground-truth depth of each image in the test set also sorted by the proposed approach’s RMSE performance. Additionally an approximate trend-line to show the relationship between median sensor depth and RMSE

In order to shed some light on how useful the numbers are in evaluating the performance of competing methods a more detailed analysis was performed on the test sets of NYUv2 [40] and KITTI [41]. This reveals that not surprisingly, when a depth estimation network is trained to minimise the squared difference between the predicted and ground-truth linear depth, a relationship forms between the median depth and the error. This is a result of mathematics of this metric, that is to have a large error there needs to be large depths in the original image for the estimator to get wrong. The improvement in KITTI is potentially the most significant contribution of this work, not only does the graph in Figure 8.8b Left demonstrate that the proposed approaches is quantitatively

better on almost every single image of the test set, but this was achieved with a much smaller training set. In the case of Godard *et al.*, an addition refinement network was also required in order to improve the output. Unfortunately this analysis does reveal an issue with the KITTI dataset as an evaluation tool, highlighted by the disparity between the qualitative and quantitative performance of Zhou *et al.* The resulting predictions in Figure 8.7 2nd row, reveal a network that fits to the modes of the dataset. This is not a failure of the approach, rather the dataset only contains sparsely sampled ground-truth data even for the test set, and the dataset contains almost entirely scenes with a visible horizon line and a road moving away at the same distance from the camera. It could be expected that a network that simply predicted the average depth for every image would perform competitively with some previous approaches.

8.11.2 Pose Estimation

To demonstrate the ability of the proposed approach to perform accurate relative pose estimation, it was compared on several unseen sequences from 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 [51]. To mitigate the effect of scale-drift on these quantities all poses were scaled to match the magnitude of the ground-truth associated poses during evaluation. As mentioned in Section 8.10, using both metrics provides an estimate of the consistency of each pose estimation approach. The results of this quantitative analysis are summarised for KITTI[41] in Table 8.4 and for RGB-D[51] in Table 8.5. The results of other state-of-the-art pose estimation networks, namely SfM-Learner[46] and DeMoN[45], are included for comparison. Additionally the proposed approach was compared against state-of-the-art SLAM systems, namely ORB-SLAM2[50] and LSD-SLAM[10], in order to provide context for the current state of machine learning approaches to visual odometry.

Sequence	09			10		
Method	ATE(m)	RPE(m)	RPE(°)	ATE(m)	RPE(m)	RPE(°)
ORB-SLAM(no-loop)[50]	57.57	0.040	0.103	8.090	0.033	0.105
ORB-SLAM(full)[50]	9.104	0.056	0.084	7.349	0.031	0.100
SfM-learner(5)[46]	58.31	0.077	0.803	31.75	0.069	1.242
SfM-learner(1)[46]	81.09	0.050	0.976	75.89	0.045	1.599
Ours(fc)	41.50	0.087	0.387	29.29	0.081	0.486
Ours(iterative)	16.55	0.047	0.128	9.846	0.039	0.138

Table 8.4: Performance of several approaches evaluated on two sequences of the KITTI dataset [41]. 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

8 USING POSE PRIORS TO IMPROVE DEPTH ESTIMATION

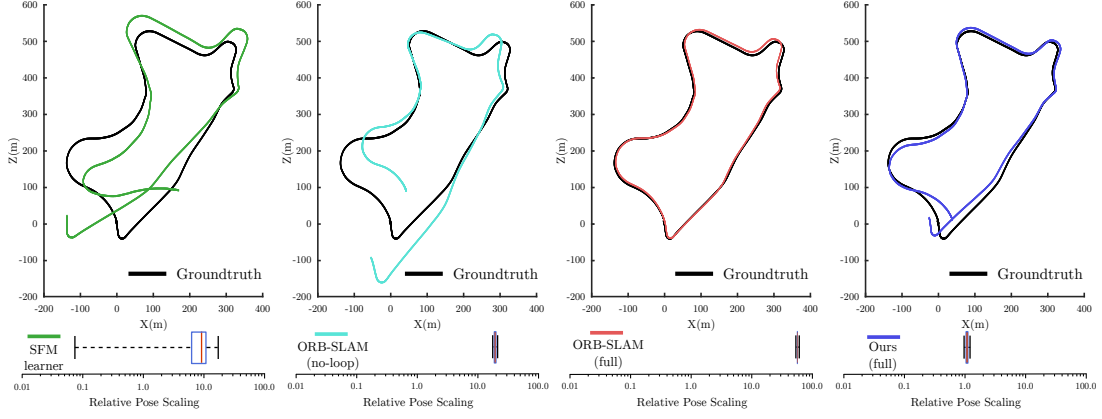


Figure 8.9: *Top* the scaled and aligned trajectories for SFM-Learner [46], ORB-SLAM2 [50] (with and without loop-closure) and Ours respectively. *Bottom* box-plots of the relative pose scaling required to bring the predicted translation to the same magnitude as the ground-truth pose

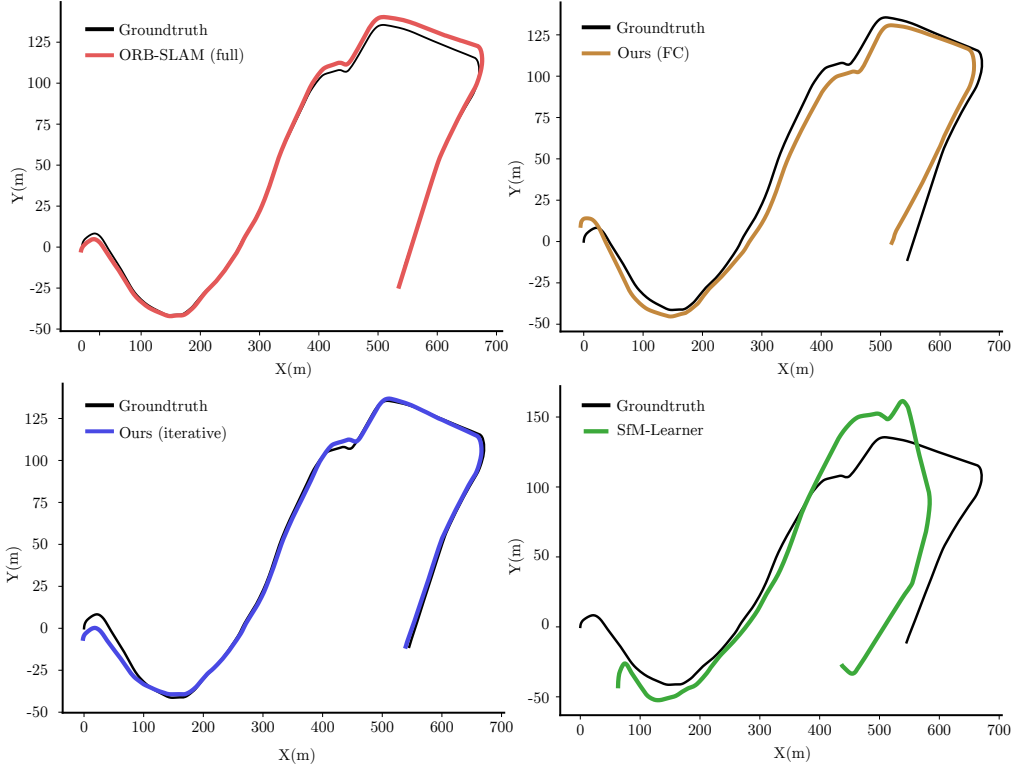


Figure 8.10: Trajectories of this approach in both configurations (*iterative* and *fully-connected*), as well as the resulting trajectories of ORB-SLAM(full) [50] and SfM-Learner [46]. This demonstrate comparative quality of this approach to ORB-SLAM, while significantly outperforming SfM-Learner

8 USING POSE PRIORS TO IMPROVE DEPTH ESTIMATION

Table 8.4 demonstrates the most comparable performance to state-of-the-art SLAM systems against the proposed approach. However, these state-of-the-art SLAM systems do consistently out perform the machine learning approach. This is due to a number of reasons, the primary of which is that SLAM systems use salient information from a large number of frames simultaneously to perform bundle-adjustment. Another key reason is that these approaches enforce geometric constraints more explicitly, something that must be more carefully considered in future iterations of machine learning approaches.

There is a noticable improvement over SfM-Learner on both sequences across all metrics. SfM-Learner was evaluated on its frame-to-frame tracking performance for adjacent frames (*SfM-Learner(1)*) and for separations of 5 frames (*SfM-Learner(5)*). The reason these two configurations were chosen is because they train their approach to estimate this size frame gap[84]. Using a larger frame separation for poses is analogous to using a keyframe which should result in a significant reduction in accumulation error as there are far fewer poses to accumulate error. Even with this significant reduction in accumulation error demonstrated in reduced ATE, the proposed system still produces more accurate pose estimates and trajectories.

The resulting scaled trajectories of *sequence 09* are shown in Figure 8.9, 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 two-full log scales, while ORB-SLAM’s and the proposed approach’s are barely visible at this scale. A benefit of the proposed approach is that scale is centered around 1.0, as it estimates scale directly through the metric depth network. This seems to significantly reduce scale-drift and making the system more usable in practice.

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[10]	0.090	-	-	-	-	-	0.124	-	-
ORB-SLAM[50]	0.009	0.007	0.645	-	-	-	0.012	0.013	0.694
DeMoN(10)[45]	0.178	0.021	1.193	0.601	0.035	2.243	0.265	0.049	1.447
DeMoN(1)[45]	0.183	0.037	3.612	0.669	0.032	3.233	0.279	0.040	3.174
Ours(fc)	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

Table 8.5: Performance of pose estimation on several sequences from the RGB-D dataset [51]. DeMoN(10) and DeMoN(1) indicates the trajectories were constructed with a frame gap of 1 and 10 respectively. Both [50] and [10] fail to track on *fr2-360-hs*. The results are separated by SLAM and machine learning approaches. We highlight the strongest results in bold

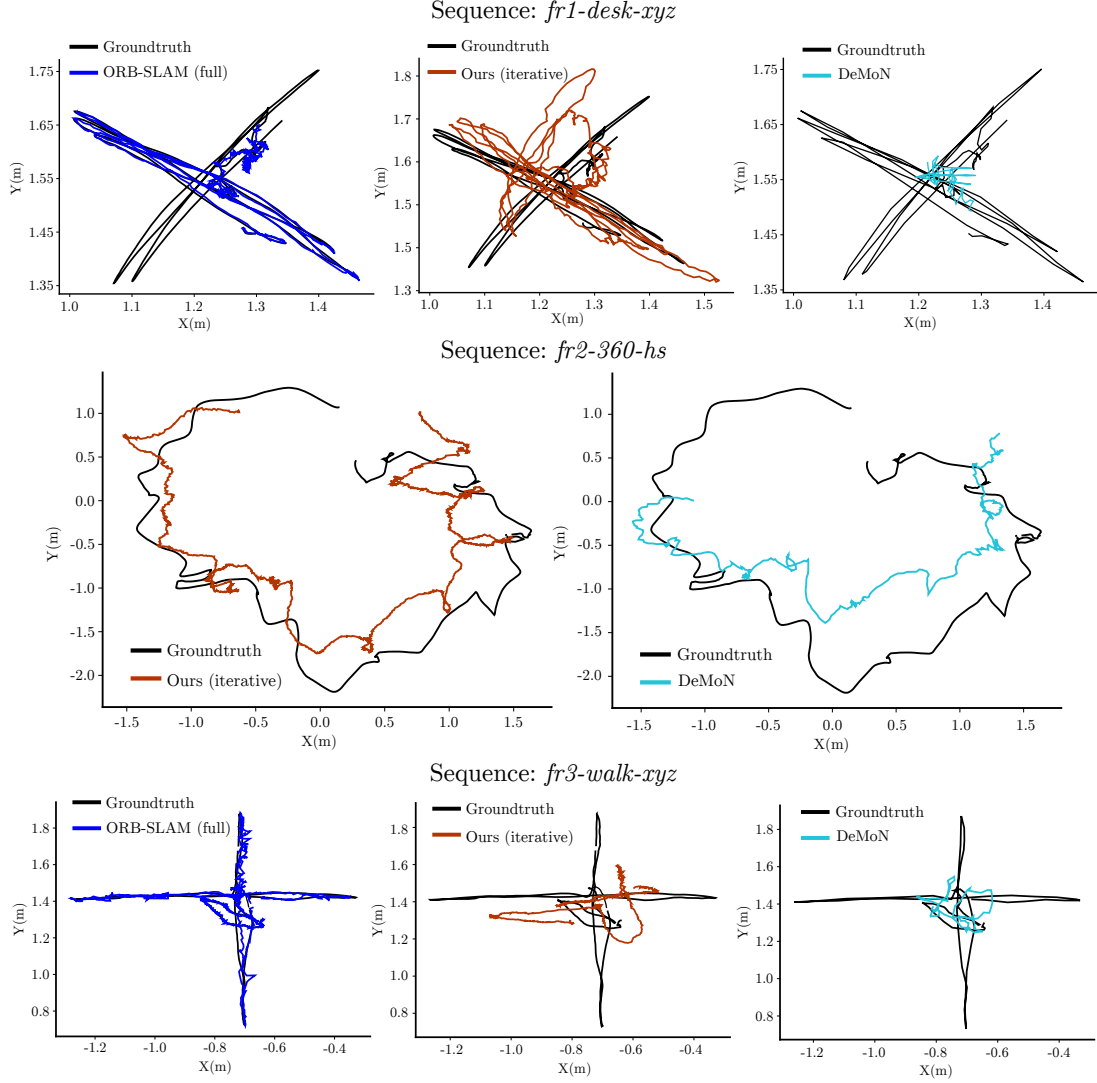


Figure 8.11: Trajectories of Our method against ORB-SLAM [50] and DeMoN(10) [45], for the evaluated sequences from the RGB-D dataset [51]. We demonstrate a marked improvement upon DeMoN which, although being given a slight advantage in some respects by widening the baseline and reducing accumulated pose 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.

Table 8.5 demonstrates a significant improvement in performance against existing machine learning approaches across several sequences from the RGB-D dataset [51]. The approach DeMoN from [45] was evaluated in two ways, frame-to-frame (*DeMoN(1)*) as a visual odometry system, and in an attempt to provide the same advantage to DeMoN as SfM-Learner by using wider baselines. This is a configuration which they claim improves their depth estimations [45], and a frame gap of 10 (*DeMoN(10)*) was chosen as

it experimentally improved their results the most. It was observed that even with the significant reduction in accumulation error over a frame-to-frame approach, the proposed approach still manages to significantly out-perform their approach in ATE, even surpassing LSD-SLAM 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. An example of a frame from the sequence *fr3-walk-xyz* is shown in Figure 8.12, which shows this scene is not static. The proposed system has the ability to deal with non-static scene objects through the flow confidence estimates, which is discussed further in Section 8.12. Additionally, the resulting trajectories for the scenes are shown in Figure 8.11, this truly demonstrates the difficulty both machine learning approaches have under these conditions. The resulting trajectories are far from usable at this stage and indicate that machine learning approaches have a significant gap to catch up to pure geometry based approaches.

8.12 Ablation Experiments

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[241]	29.64	0.349	0.838	51.90	0.222	0.954
Flownet2.0(iterative)[241]	24.61	0.185	0.400	22.61	0.142	0.484
EpicFlow[250]	119.0	0.566	0.931	20.98	0.199	0.853
EpicFlow(iterative)[250]	59.79	0.379	0.459	14.80	0.154	0.581
Ours(single)	31.20	0.089	0.324	24.10	0.095	0.389
Ours(iterative)	16.55	0.047	0.128	9.846	0.039	0.138

Table 8.6: Results of pose estimation on KITTI[41] *sequence 09* with various components of the network removed or replaced. We highlight the strongest results in bold

In order to examine the contribution of using each component of the full network the pose estimates were evaluated under various configurations on sequences 09 and 10 of the KITTI odometry dataset[41]. The results of this evaluation are summarised in Table 8.6. This demonstrates the relative improvement of iterating the pose estimate until convergence (*ours(iterative)*), against a single weighted-least-squares iteration (*ours(single)*). To show the importance of using the predicted flows of the jointly trained network, the performance was contrasted against a system where they were replaced with flow estimates from two other state-of-the-art flow estimation methods from [241] and [250]. This consistently demonstrated that the approach taken provided an improvement over previous approaches to flow estimation. In retrospect, the previous approaches should probably have been integrated into the resulting system and trained jointly and con-

fidence added to one of those pipelines separately. This may result in even further improvement to pose estimation performance. The result of optimising with and without confidences is also shown, demonstrating quantitatively how important they are to pose estimation accuracy, with significant reductions across all metrics.



Figure 8.12: For a frame pair (I_i and I_j) from the sequence *fr3-walk-xyz*, \bar{F}_{ji}^L 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

An example flow is shown in Figure 8.12, which clearly indicates the system is able to determine the presence of dynamic objects in the scene. Additionally, the system appears to highlight points that should have projected outside the image but have not given the predicted flow. Both these qualities significantly improved the systems overall pose accuracy and robustness in practical tracking examples, but there is still clearly further improvement to be made given the resulting trajectories in Figure 8.11. The system also reinforces the justification of traditional methods, demonstrating higher confidences on edge pixels, helping the system focus on salient information during optimisation in an approach similar to [10].

8.13 Conclusion and Further Work

In this work a unified framework was introduced which was trained end-to-end and estimates depths and the motion parameters governing the camera pose given an image pair. This achieved state-of-the-art performances on single image depth prediction for both NYUv2 [40] and KITTI [41] datasets. Two models were trained, one each for indoor and outdoor scenes with a common underlying architecture, and this was the first work to achieve that result in that manner. Additionally the network demonstrated both qualitatively and quantitatively that it was capable of producing better visual odometry than previous machine learning approaches, considerably reducing scale-drift by predicting metric depths.

The result of the research presented in this chapter indicates a step forward in the evolution of machine based approaches, as they can be applied to the problem of visual odometry. The main message is that the machine learning certainly appears to be stronger at some aspects, such as reducing scale drift as shown in *sequence 09* in Figure 8.9. This shows that approaches like this can be used to help improve the overall performance of monocular approaches. An additional point is that machine

learning approaches should be leveraged to their strengths, its clear that machine learning approaches are very good at estimating per pixel quantities such as depth, normals, curvature, semantic labels, and optical flow, but find tasks such as computing a relative pose from two images very challenging. This appears to be down to the nature of the neural networks, computing a relative pose given a convolution of an image pair is a very challenging task, this can be made simpler by embedding the images in some more descriptive space before attempting the alignment but still ignores the fact that there are geometric constraints, and its very challenging to introduce those to a network formulation. Using the iterative pose estimation approach in this work, demonstrates a system that attempts to acknowledge the strengths and weaknesses of neural networks. The pose is still predicted entirely by a neural network but is enforced using traditional geometric priors, this is also being explored in other recent works [252, 253]. In the context of this thesis this work demonstrates neural networks have the potential to supplement data from depth sensors, particularly in the case of KITTI, where the depth sensor provides a sparse sampling, this approach to depth estimation could be integrating as a method of providing dense depths. One issue is still the computational requirements of this approach, which are currently quite demanding, the model in fact only fits on cards in the highest current performance level ($>10\text{GB}$). This performance issue is addressed in the follow-up work presented in Chapter 9.

8.14 Code/Datasets

The code and training details, as well as the original submission with supplementary material is available at the following url: <https://github.com/tfb0/ENGnet>.

The NYUv2 [40] dataset is available for download at: https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html.

The KITTI [41] dataset is available at: <http://www.cvlibs.net/datasets/kitti/>.

The RGB-D [51] dataset is available at: <https://vision.in.tum.de/data/datasets/rgbd-dataset/download>.

Real-time Depth Estimation via Model Compression

The following chapter is largely drawn from work performed in collaboration with Thanuja Dharmasiri, which was formed into the submission [172] and is likely to appear in a similar form in his thesis. This is an extension of the depth estimation work in Chapters 7 and 8, towards a practical application in robotics. The goal was to train a network that could perform reasonably well at the task of depth estimation and operate in real-time on a mobile device. This was a challenging task at the time of writing, and the use of the accelerated machine learning inference engines made it possible.

A list of elements and the relevant percentages both first authors have agreed in terms of contribution to the production of this work is provided:

- Conception of Idea (80%)
- Network architecture design (50%)
- Loss/Objective function design (50%)
- Coding the network (50%)
- Training data creation (50%)
- Testing and evaluation (50%)

9.1 Motivation

Roboticians endeavour to build systems which are real-time capable for a vast array of applications including autonomous vehicle navigation, visual servoing, and object detection. The majority of the aforementioned tasks require a robot 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 such 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.

9 REAL-TIME DEPTH ESTIMATION VIA MODEL COMPRESSION

Another 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.

The research conducted in this chapter, takes a step towards combining real-time robotics and machine learning on a resource constrained mobile platform. This is 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.

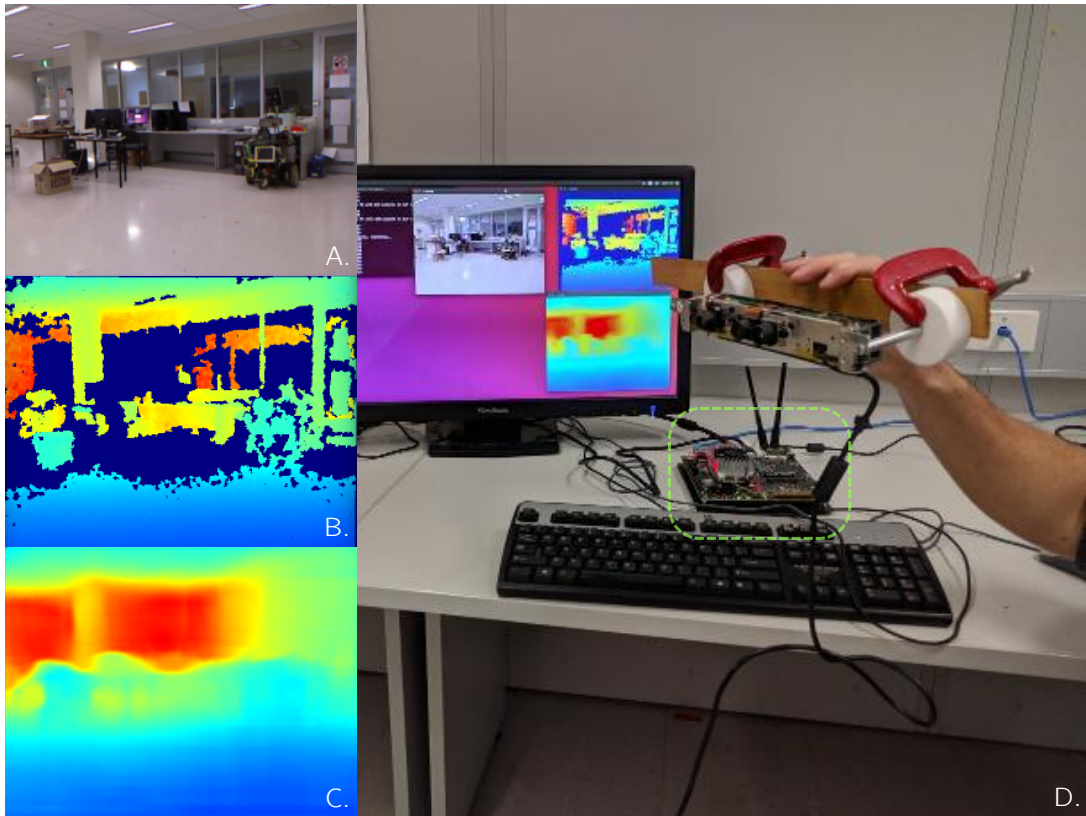


Figure 9.1: Demonstrates the model running on the NVIDIA-TX2 development board (shown in green square **D.**) in real-time. **A.** The colour image, **B.** The jet coloured groundtruth depth image from a the Kinect **C.** The predicted depth from the colour input image. Using the condensed model the demonstration runs at 30fps on the NVIDIA-TX2. Additionally the model demonstrated has only been trained on the NYUv2 [40] dataset, but is still able to predict relatively convincing depths on novel scenes such as the one pictured.

Learning deeper (more layers) and wider (more channels per layer) models generally leads to better results [29, 155] for most vision based tasks. However, the fundamental

limitation of such approaches is the inability to deploy on resource constrained devices. The depth prediction framework presented here not only runs at frame rate on an NVIDIA-TX2 but also outperforms denser architectures such as [35] 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 the network to achieve superior performance over randomly initialised models of equivalent size, while training both models to convergence.

This is an extension of the work presented in Chapter 8, with a focus on targeting depth prediction as basis for improving mapping and navigation systems. The resulting system is targeted towards situations with low levels of parallax while also eliminating the requirement for expensive hardware such as LIDAR, in outdoor environments. The networks created perform at frame rate for both indoor and outdoor scenarios. Additionally, a practical application of this work to robotics is demonstrated, by coupling the real-time depth prediction with an *off-the-shelf* SLAM system ORB-SLAM2 [7].

9.2 Contributions

The following bullet points provide a summary of the contributions made in this paper 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 architecture with model architectures which has more than 30 times the number of parameters than the proposed network. (Video demonstration included in supplementary materials)
- 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 integration of the predicted depths with ORB-SLAM2[7] in order to perform tracking and mapping on mobile platforms while significantly reducing scale-drift.

9.3 Related Work

Inferring higher order quantities (semantic labels, structural information) from only colour image data allows researchers to tackle a range of problems. Convolutional Neural Networks perform remarkably well at extracting key pieces of information and ignoring noise in image data. Although a lot of image driven machine learning frameworks aim to solve classification and semantic segmentation problems [26, 146, 37, 155], most of the techniques introduced can be readily applied to predict geometric quantities such as depths [34, 35, 158, 42], normals [34, 160] and curvature [159].

An added benefit of predicted structural information is it allows the use of conventional geometry based techniques in concert with machine learning systems. Garg et al. [42] constructed an unsupervised depth prediction framework by enforcing an image reconstruction loss. Left-right consistency of stereo images was leveraged in [43] and the relationship of depths, normals and curvatures was exploited in [159] to improve the accuracy of all three quantities. Structure prediction systems have also been combined with SLAM systems [254], however the neural network employed in their approach [158] does not run at frame rate even on a conventional gpu thus making it impossible to deploy on a mobile platform. Contrary to this the proposed framework in this chapter runs in real-time on an NVIDIA-TX2.

Due to the attractive qualities such as low power consumption and high mobility, researchers have been keen to examine the possibility of building smaller architectures. MobileNets [48], ICNet [188], ERFnet [189], have shown reasonable accuracy and real-time performance on modern GPUs. However, none of these methods achieve inference at frame rate on an NVIDIA-TX2. Many of these approaches focus on reducing the overall model size, while attempting to maintain a comparable performance to larger systems. This work focuses more on the latent space transfer aspect in which a larger supervisor network is employed to aid in training the condensed network.

The machine learning community has investigated the problem of model compression or emulating the performance of a larger network. Hinton et al. in [49] 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 work, a tensor loss was introduced in this work, where the aim was 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 [49] Bucila et al. [191]

showed that it is possible to replicate the performance of an ensemble of classifiers using a single model. 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 [192] for image classification.

9.4 Proposed Framework

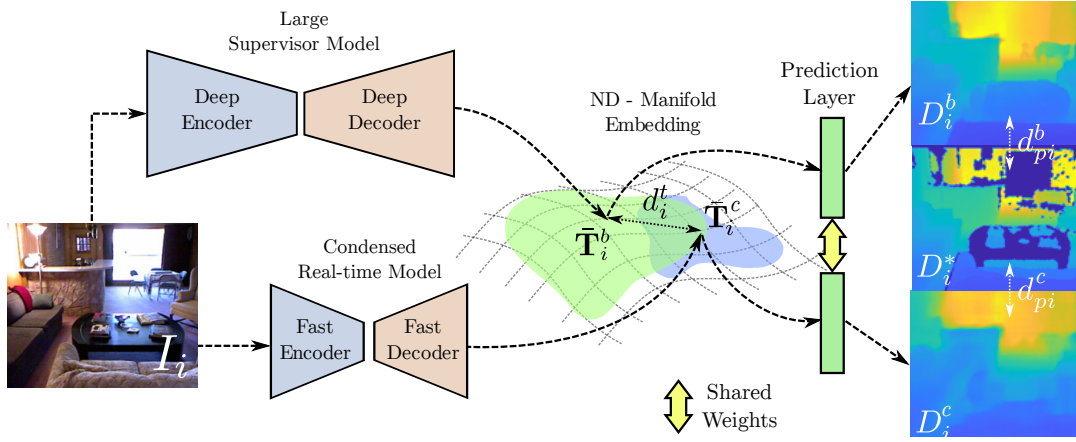


Figure 9.2: This demonstrates the concept behind our training regimes we use to perform model compression through knowledge transfer. The initial strategy was to minimise the difference (d_i^t) between the intermediate activations (also referred to as the tensor loss) produced as input to the prediction layer by the network. The other approach involves transplanting of the prediction layer from the large network onto the condensed network, and an examination of 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.

This section aims to provide a step by step breakdown of the proposed network, training process and datasets used. Initially the implemented model architecture is presented, followed by the loss functions use during training. Finally the datasets are introduced, that were used during training. An additional analysis of the training regime and how this varies for each of the tested knowledge transfer training approaches.

9.4.1 Model Architecture

The model design was inspired by ENet [190] and ERFNet [255], which have demonstrated a decent trade-off between performance and speed for the task of semantic segmentation. They show in [190] the ability to run at near real-time ($>10\text{fps}$) performing a dense semantic segmentation task on the targeted hardware, the NVIDIA-TX1.

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

Table 9.1: A summary of the architecture implemented given an input resolution of 320×240 , which is used for both [40] and [51]. The left column refers to the broad section of the network as shown in Figure 9.3, **E**: Encoder, **D**: Decoder and **P**: Predictor, where the predictor layer is the layer that can be transplanted from the supervisor network.

However, the target was to produce a system with an even higher frame rate, to allow for every frame to have a depth estimate in real-time. The targeted hardware platform was the NVIDIA-TX2, which has $\approx 30\%$ more compute power over the previous NVIDIA-TX1. The NVIDIA supported TensorRT framework [256] was used in order to accelerate inference of our models at runtime. This framework was developed by NVIDIA to target a real-time inference on low-powered hardware, such as the TX1/TX2. However, this limited the available layers to those supported by the framework, which at the time of this research did not support dilated convolutions [257], despite purporting to do so. Taking these factors into consideration and after a number of attempts the architecture was decided upon, which is defined in Table 9.1. This architecture was felt to provide a good compromise between runtime and accuracy, but a more thorough investigation of architecture is required to fully develop this technique in the future.

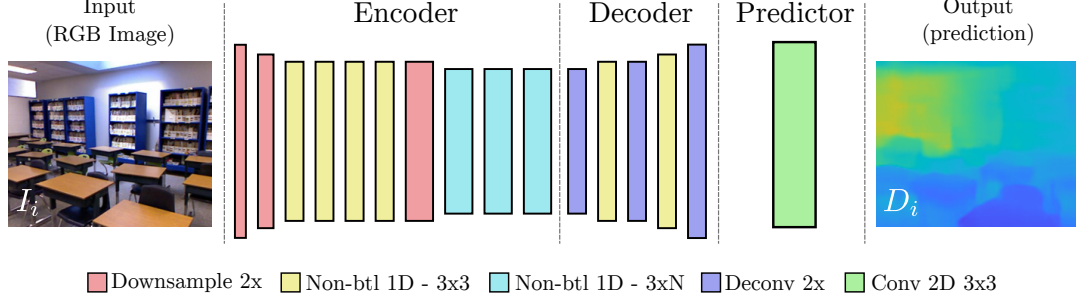


Figure 9.3: The model architecture for our real-time depth estimation network. This network is constructed from mostly Non-bottleneck blocks (Non-btl in figure), which are a series or residual type blocks shown in Figure 9.4. Downsample, Conv 2D and Deconv are all standard operations.

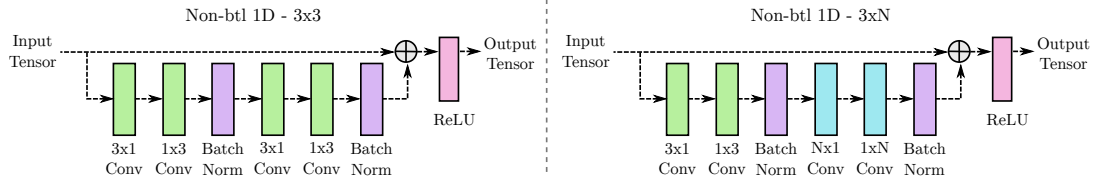


Figure 9.4: The submodules of the Non-bottleneck 1D blocks. "Non-bottleneck" refers to the channel count which remains unchanged when passing through this layer. The $N \times 1$, $1 \times N$, 3×1 and 1×3 Conv operations are standard asymmetrical convolutions where N is chosen. In practice we used two Non-btl ND - 3×5 blocks followed by one Non-btl ND - 3×7 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 function of all layers used is described further in Section 3.10.

9.4.2 Loss Functions and the Knowledge Transfer Process

The first choice of loss function when performing regression is the L_2 distance between the prediction and the ground truth as shown in Equation 9.1. This is the same loss used in Chapter 8, which produced state-of-the-art depths. This was chosen as a starting point to train the condensed networks defined in Table 9.1. The random models trained using this formulation are referred to as **R** models (or by the superscript r) throughout the remainder of the chapter. Note that since the network architecture was defined from scratch in this case, there were no pre-trained weights to initialise from, as such all the weights were initialized using MSRA initialization [37]. However, in the case of the transplanted networks denoted by **TR** (or the superscript tr) in this chapter, the final layer is drawn directly from the pre-trained supervisor network. The depth loss

function is given by

$$L_d = \frac{1}{N} \sum_{i=0}^N \|D_i - D_i^*\|_2 = \frac{1}{N} \sum_{i=0}^N \|d_{pi}\|_2, \quad (9.1)$$

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. The distance between the i th predicted and ground-truth depth was defined as d_{pi} , shown in Figure 9.2, where the super script c and b are used to denote the error from the condensed and big network predictions respectively. By training the randomly initialized model using the Euclidean loss defined in Equation 9.1, the provided a baseline performance for the given architecture to measure the contributions of this work against.

The model size is fixed for the comparison, providing a system that predicted at 30fps. Additional loss functions were used (similar to [44]) in order to improve the network’s performance. Although the question became what additional losses could be incorporated? Since the completion of the work in Chapter 7, a very large, very accurate depth estimation network was available. The goal became to investigate how the knowledge contained in this slow, state-of-the-art network could be used to help train the a faster model.

Inspired by the work of [49], the basic motivation is to take the power of a state of the art depth estimation network [171] and attempt to transfer the useful knowledge to the condensed network defined in this chapter. This does immediately create a soft cap on the possible performance, as a smaller network is unlikely to out-perform the larger network. This trade off between speed and performance, is in general acceptable for applications where the real-time performance is more important such as robotics applications. The hope was to improve the performance enough over random to be usable in robotics, and validate this approach as a valid method of training. The condensed network was designed around the idea that the final layer of the large depth estimator could be transplanted onto our condensed network, shown in Figure 9.2.

The approach in [49] was able to transfer knowledge through the logit distribution, as their problem was classification. In this case the network is attempting to estimate a continuous value, meaning there are no distributions to learn, so how can the knowledge be transferred? As the logit distribution is passed to the soft-max solver in the case of classification, one could learn based on the activations that are input to the depth *solver* layer. The so-called *tensor loss* as depicted in Equation 9.2 aims to mimic the activations of the penultimate layer of the supervisor (strong deep) model. This is a supervised loss where we attempt to enforce the tensor of large and the condensed model

to match in value.

$$L_t = \frac{1}{N} \sum_{i=0}^N \|\bar{\mathbf{T}}_i - \bar{\mathbf{T}}_i^*\|_2 = \frac{1}{N} \sum_{i=0}^N \|d_i^t\|_2 \quad (9.2)$$

$\bar{\mathbf{T}}_i$ represents a tensor corresponding to the activations of the penultimate layer of the condensed network and $\bar{\mathbf{T}}_i^*$ represents that of the deeper network. After training till convergence using the tensor loss and a fixed solver that is randomly initialised, the final layer is freed and the network is fine tuned using the depth loss. This model is denoted as \mathbf{T} in the results section. The thought is that potentially these activations contain extra knowledge about the depth, that is not accessible through the random initialisation training.

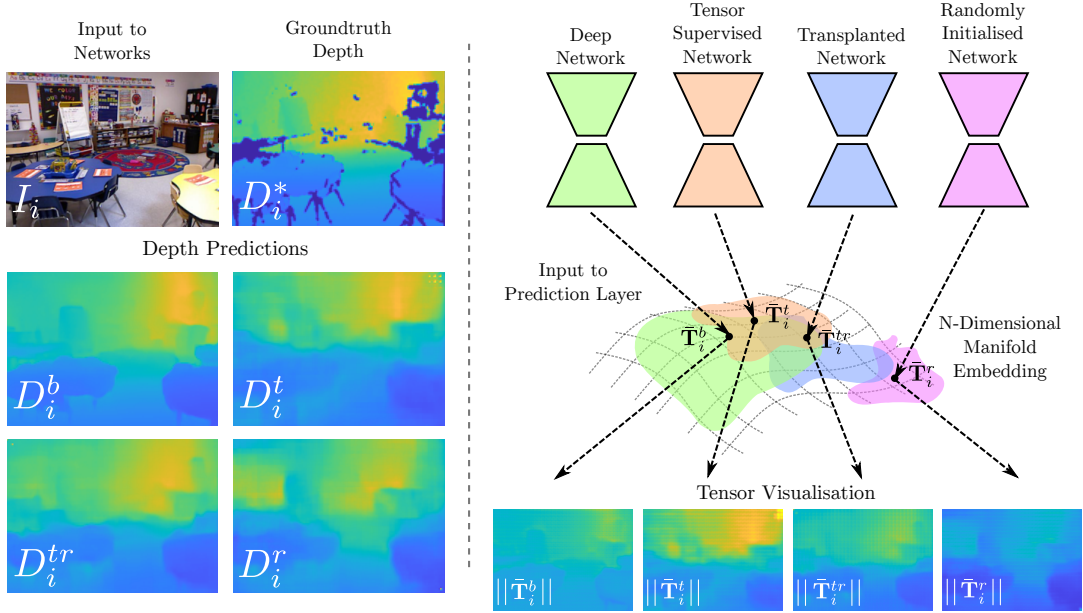


Figure 9.5: Demonstrates how the different network training regimes will ultimately determine the manifold embedding that the resulting networks form. The range of possible outputs of the network are shown in the colour of the network, and is intended to show that some of the embeddings are expected to very similar (overlapping) while others will generate potentially completely unrelated embeddings. A visualization is included of the embedded tensors as a magnitude image, which correlates strongly to the resulting depth magnitude.

As an alternative approach, the penultimate layers were also trained in an unsupervised manner to mimic the output of the larger network. This is done by transplanting the final/solver layer of the larger model onto a randomly initialized condensed model, the network is trained using Equation 9.1 with a fixed solver. Once the network has initially converged, the final solver to fine-tuned to improve the overall performance. This strategy was designated the transplanted model (**TR**) training strategy.

The final approach attempted was the $\mathbf{T}+\mathbf{TR}$ model, which uses a combination of the tensor loss and the transplanted solver. In this case, the network was trained for roughly 20 epochs using the tensor loss followed by introduction of the transplanted layer and further fine tuning using the depth loss for another 5 epochs.

In an attempt to visualise the resulting embeddings created by this training process, a simplified diagram of the rationale is shown in Figure 9.5. This demonstrates, that one would expect to have the tensorloss embedding to most closely relate to the large network, while the transplanted network will find some point nearby, and the random to be largely uncorrelated as the space of possible networks would be incredibly complex with many local minima. It is very difficult to evaluate these claims, but a more detailed discussion as well as the findings made in this research is presented in Section 9.6.

9.4.3 Datasets

The same datasets as were used in Chapter 8 (*NYUv2* [40], *RGB-D*[51] and *KITTI* [41]) as these each had a corresponding state-of-the-art trained depth network. The first two of these are indoor datasets, filmed using the Microsoft Kinect style sensor. *NYUv2* provides a large number of varied indoor sequences, where 249 scenes are used during training, and 215 for testing. The official test set was created by drawing 654 images from the test scenes. *RGB-D* provides substantially less variation in a dataset than *NYUv2* but has enough to provide a challenging set of indoor scenes to predict on. Finally *KITTI* is a large and varied outdoor dataset with over 20,000 training images. The standard train-test split was followed when creating the training set and evaluate on the official test images. In addition to being outdoors and providing a much larger range of depths to estimate, this dataset also provides ground-truth odometry for a subset of sequences.

9.4.4 Hyperparameters

Each network was trained with a batch size of 12 on a cluster of GTX1080Ti GPUs. Once again the Adam optimizer [249] was used, with a initial learning rate of 1e-4. Similar to the training regime of the original supervisor network, the learning rate was halved every 2 epochs after the 4th. The Tensorflow [206] framework was used during training allowing for rapid prototyping and later integrated with TensorRT for deployment on the NVIDIA-TX2 device.

9.5 Evaluation Metrics

The same evaluation metrics as were used in Chapters 6, 7, and 8. Please refer to those chapters for the equations of the metrics.

9.6 Overall Evaluation

The output of the proposed networks was evaluated using the datasets described in Section 9.4.3. This section summaries the results across several qualitative and quantitative comparisons to existing approaches, and demonstrates the usefulness of our approach.

NYUv2 [40]						
Method	<i>lower better</i>			<i>higher better</i>		
	Rel_{abs}	RMS_{lin}	RMS_{log}	δ	δ^2	δ^3
Liu [157]	0.230	0.824	-	61.4%	88.3%	97.2%
Eigen _{alex} [34]	0.198	0.753	0.255	69.7%	91.2%	97.7%
Eigen _{vgg} [34]	0.158	0.641	0.214	76.9%	95.0%	98.8%
Laina [158]	0.127	0.573	0.195	81.1%	95.3%	98.8%
Baseline [171]	0.111	0.480	0.161	87.2%	97.8%	99.5%
Real-time Networks						
Ours (R)	0.216	0.765	0.277	64.4%	89.3%	97.1%
Ours (T)	0.204	0.713	0.261	68.5%	90.9%	97.5%
Ours (T+TR)	0.205	0.715	0.262	68.3%	90.8%	97.5%
Ours (TR)	0.190	0.687	0.251	70.4%	91.7%	97.7%

Table 9.2: The metrics are explained in Subsection 9.6.1. 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.

9.6.1 Depth Evaluation

Both NYUv2 and KITTI datasets were evaluated using the following standard depth prediction metrics, defined in Section 7.7.

The results for the NYUv2 dataset are summarised in Table 9.2 and for KITTI in Table 9.3. Additionally some qualitative results from NYUv2, RGB-D and KITTI are included

9 REAL-TIME DEPTH ESTIMATION VIA MODEL COMPRESSION

0-50m KITTI [41]

Method	<i>lower better</i>			<i>higher better</i>		
	Rel_{abs}	RMS_{lin}	RMS_{log}	δ	δ^2	δ^3
Zhou [46]	0.201	5.181	0.264	69.6%	90.0%	96.6%
Garg [42]	0.169	5.104	0.273	74.0%	90.4%	96.2%
Goddard [43]	0.140	4.471	0.232	81.8%	93.1%	96.9%
Kuznietsov [44]	0.108	3.518	0.179	87.5%	96.4%	98.8%
Baseline [171]	0.092	3.359	0.168	90.5%	97.0%	98.8%
Real-time Networks						
Ours(R)	0.147	4.530	0.234	80.3%	93.3%	97.3%
Ours(T)	0.139	4.434	0.228	81.7%	93.7%	97.5%
Ours(T+TR)	0.140	4.426	0.225	81.7%	93.8%	97.6%
Ours(TR)	0.156	4.363	0.224	81.8%	94.0%	97.7%

Table 9.3: Results of evaluating KITTI dataset, using the same metrics as defined in Subsection 9.6.1 and Table 9.2

in Figure 9.7 and 9.8. From the numerical results in both tables a consistent behaviour was observed 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 especially when it comes to condensed networks.

Taking a more in depth look at the contributions of the tensor loss model (**T**) and the transplanted model (**TR**), it can be observed in Figure 9.6 the tensor angles highly correlate with that of the supervisor network when trained using the tensor loss, however the magnitude of the activations correlate less strongly. This is because the magnitude of the output is incredibly difficult to estimate in this way. Overall this appears to negatively effect the quality of the reconstruction as shown in Figure 9.6 **Top-Right**. However, the angle does negatively correlate to the depth error, that is the most aligned tensors in the **T** network, have the lowest error in this case.

This seems to indicate the presence of a sort of *uncanny valley*, where the small network begins to improve emulating the penultimate activations of the supervisor, also improving their predictions to a point. Then at some stage the network can't quite perfectly

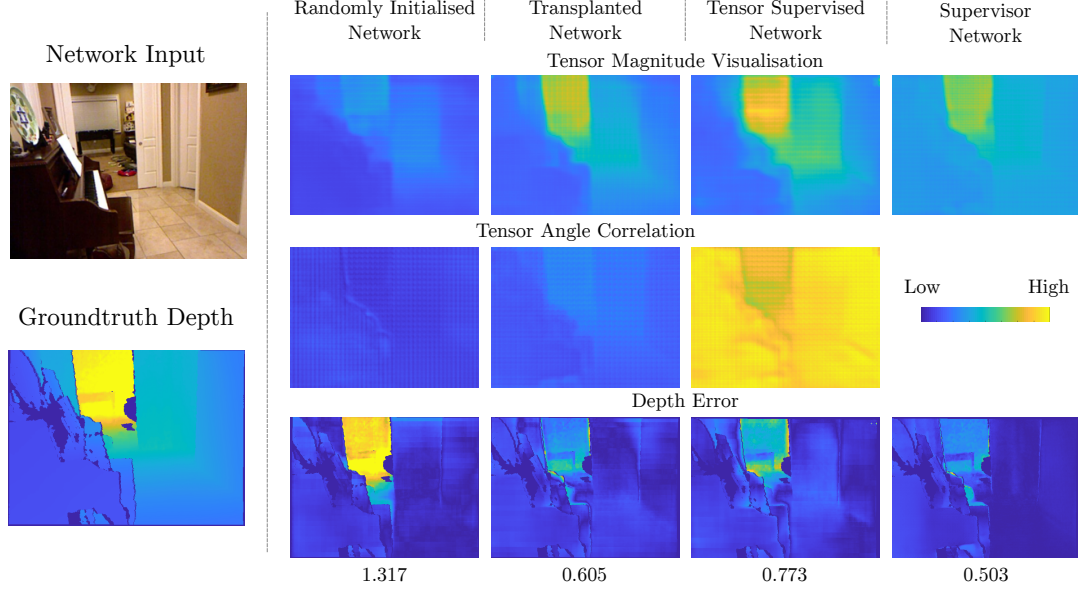


Figure 9.6: Demonstrates the relationship between the tensors produced with the three different training approaches. **Left** The RGB input, the ground-truth depth **Top-Right:** The tensor magnitude images for each network, which are a visualisation of the norm of the tensor value. **Right 2nd row:** The angle correlation, which is the degree to which the direction of the tensors agree. **Bottom-Right:** The magnitude of the depth error between the prediction and ground truth. The RMS error in meters is included for each of the predictions below their respective columns.

reproduce the tensors and gets stuck in a suboptimal minima, while 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. This valley seems to be created by the vastly reduced model capacity, as given model of sufficient size the output of the supervisor should be perfectly replicated. This is something that requires further investigation in future.

9.6.2 Pose Estimation

As an practical application of this work, the tracking ability of an off-the-shelf SLAM system using the depths inferred by our real-time network was evaluated. This was compared against the ground-truth pose data available on a select number of KITTI datasets. These results are summarised in Table 9.4 and examine the comparative performance of the original SLAM system [7] in the Mono, Stereo and finally the RGB-D configuration which used the predicted depths as the depth channel. The standard Absolute Trajectory Error (ATE) as proposed in [51] was used to provide a quantitative method of comparison for a number of novel test sequences.

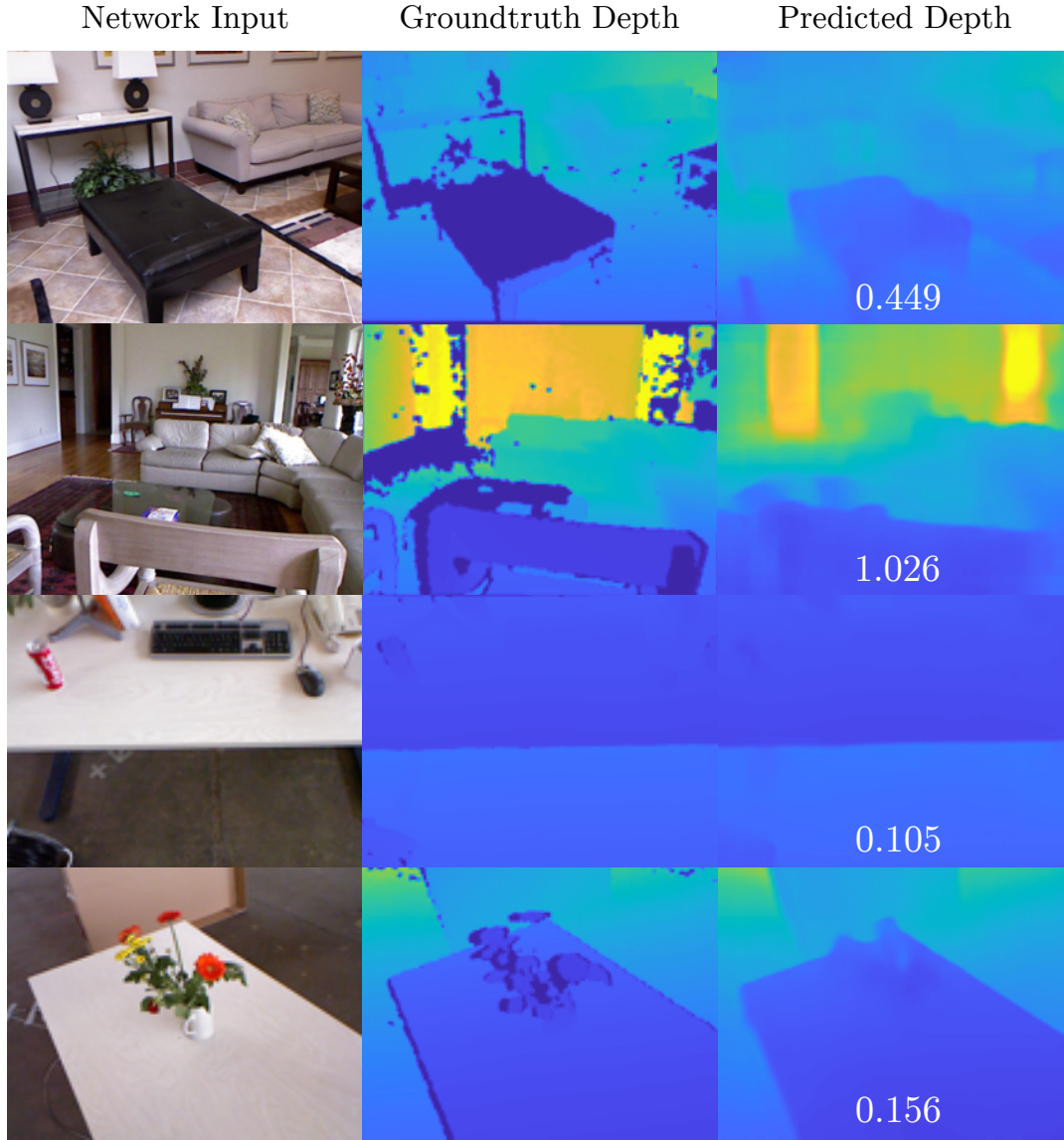


Figure 9.7: The performance of our network at estimating depths on the datasets NYUv2 [40] and RGB-D [51]. The first two rows are from [40] while the last two rows are from [51]. All the images are from the test sets, and are not present in the training data. Included for each prediction is the RME depth error for that frame.

Additionally, Figure 9.9 shows a qualitative comparison of trajectory accuracy using the predicted depths (from the **TR** network) compared to using purely monocular data against the ground-truth trajectory. Again these trajectories were computed using the popular ORB-SLAM2 system [7]. This demonstrated that by using the predicted depths from a real-time depth estimation network such as this, the system significantly out-

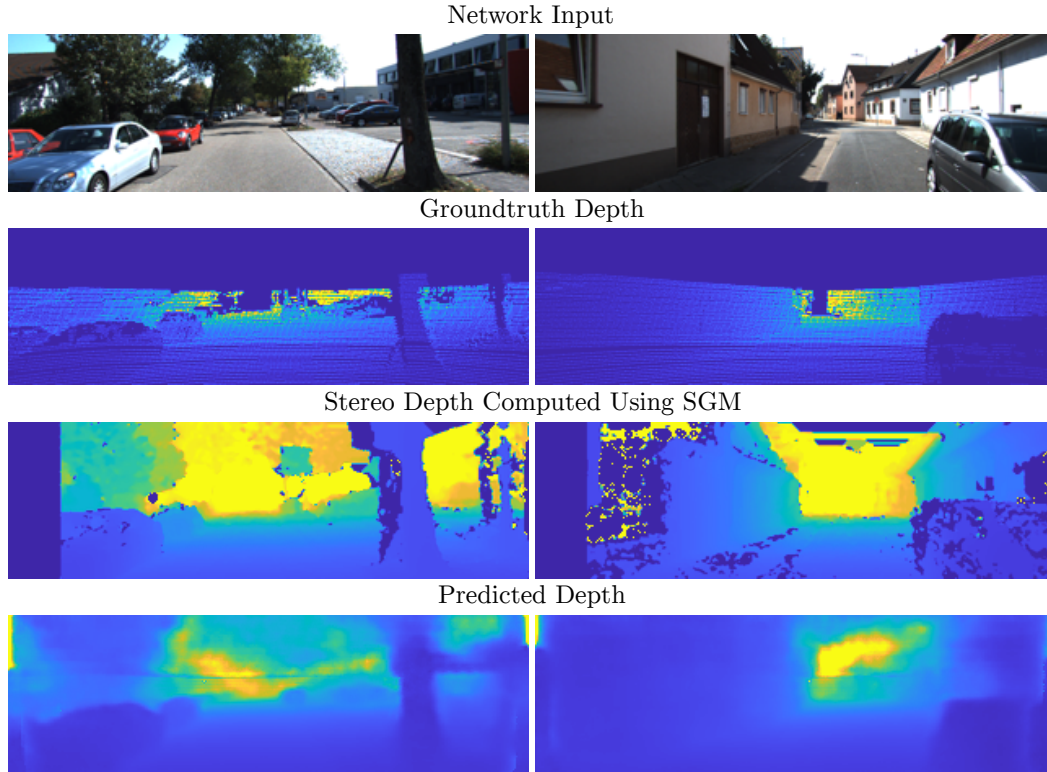


Figure 9.8: Demonstrates impressive qualitative performance on the KITTI dataset [41], with the predicted depths closely aligning to the ground-truth. The stereo reconstruction is included to densify the sparse Velodyne points, using the method SGM from [59].

KITTI Odometry Absolute Trajectory Error (m)

Sequence	<i>Ours</i>	<i>Mono</i>	<i>Stereo</i>
	<i>Predicted Depths</i>	<i>ORB-SLAM [50]</i>	<i>ORB-SLAM [7]</i>
Seq00	4.23	6.62	1.3
Seq05	2.01	8.23	0.8
Seq07	1.15	3.36	0.5

Table 9.4: Pose estimation evaluation on KITTI sequences, measuring the ATE as defined in [51].

performs the monocular only approach. This method is far from optimal, and could be further improved by attempting to estimate quantities such as uncertainty or non-static scene elements as discussed in Chapter 8.

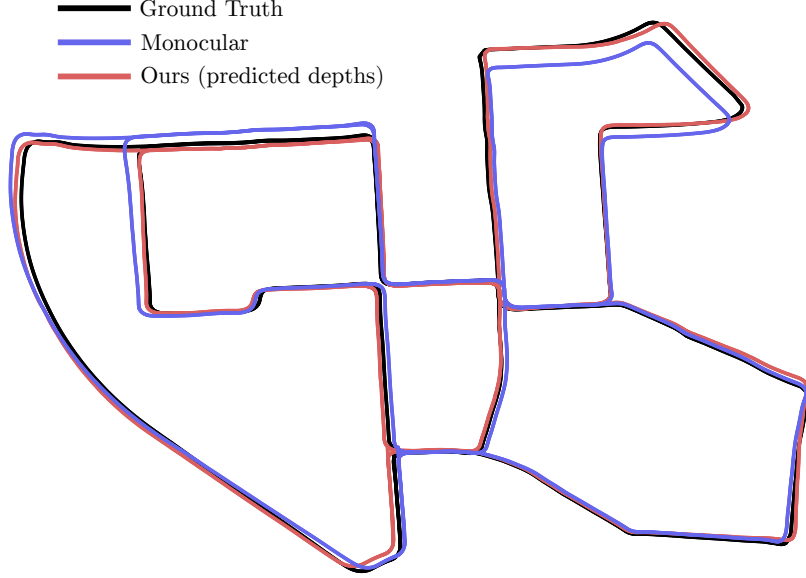


Figure 9.9: Demonstrates the improvement to the trajectory produced using our predicted depths from colour vs using colour images alone as input to the off the shelf SLAM system [7] for seq00 of KITTI-odometry [41].

In an attempt to show a concrete example of what this system can contribute to an off-the-shelf SLAM approach the effect of scale-drift was examined. Figure 9.10 clearly demonstrates the reduction in scale-drift given the same SLAM configuration, using the predicted depths vs using the monocular data alone. This establishes a practical application of this approach, given that the network can also infer at over 70FPS for this sequence, as shown in Table 9.5.

Average FPS over 50 runs (Min, Max)		
Resolution	<i>GTX1080Ti</i>	<i>TX2</i>
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 9.5: Pose estimation - 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.

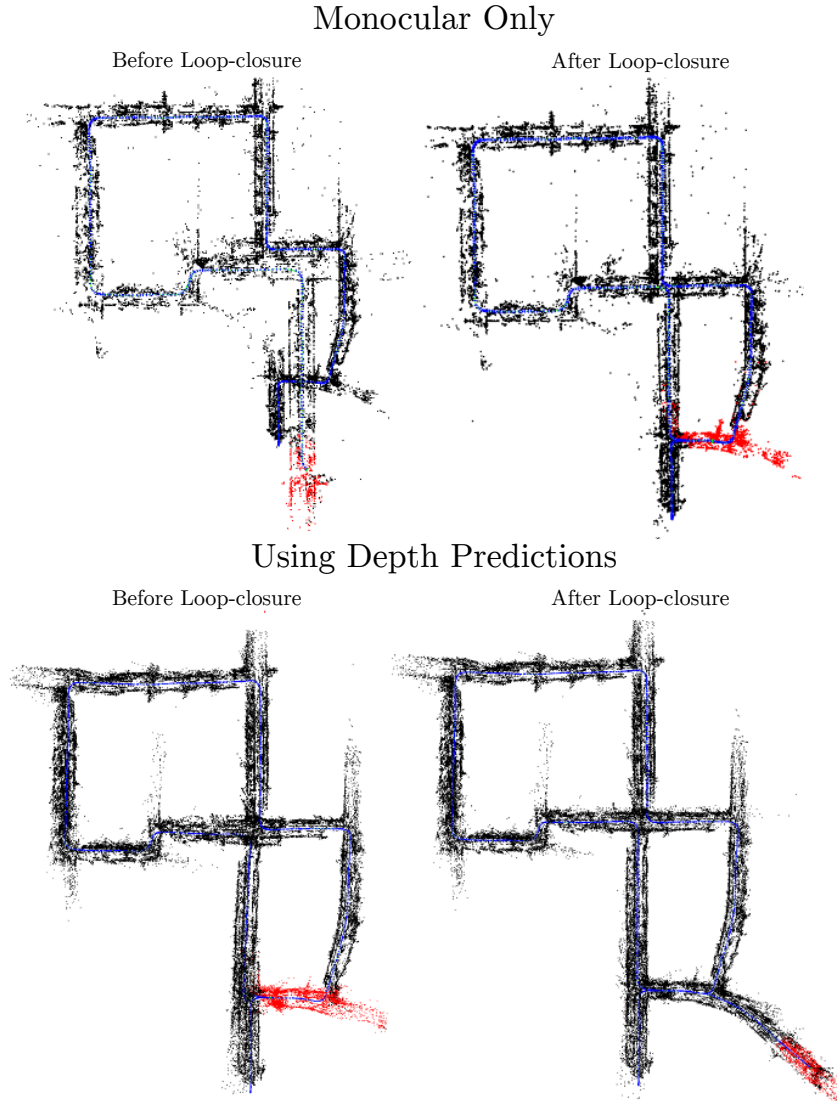


Figure 9.10: Demonstrates the amount to which the scale-drift can be reduced using the proposed approach in an off-the-shelf SLAM system. The first row shows the performance of standard monocular ORB-SLAM2 [7] on sequence 00 of KITTI-odometry [41]. Before loop-closure a very pronounced level of scale-drift is present. In contrast when estimated depths are provided, the scale drift is almost completely removed, and the difference before and after loop-closure is barely visible.

9.6.3 Speed and Computation Performance

The timing information for each of the approaches is summarised in Table 9.5. This shows that at the typical operating resolutions we can infer at real-time on the NVIDIA-

TX2, particularly on KITTI which was inferred at (320×96) for all evaluations. This demonstrates this approach is sufficient to dramatically improve the accuracy and reduce the scale-drift in tracking from monocular cameras.

9.7 Conclusions and Further Work

This work demonstrated a practical implementation of a novel approach to network knowledge transfer, and showed this can be used to directly improve the performance of a SLAM tracking system using only the information contained in the colour imagery. This strongly suggests the compatibility of using machine learning depth approaches in collaboration with an off-the-shelf depth sensor to provide improved performance in a variety of situations. This was shown in the work of [254], where they focus on situations where low-cost depth sensors fail and also integrate into an existing semi-dense SLAM approach [10] to further optimise the predicted depths. The work presented in this chapter is an attempt to move towards a greater level of collaboration in real-time on a low-power mobile platform like the TX2. This work also demonstrates a general approach to training fast models that produces significantly improved performance over random initialisation. There is a strong suggestion from this work that transplanting further layers of the supervisor network, or even other networks that have been trained on the same task will produce even better performance. This is the same idea as partial weight-transfer to an existing model, as the weights (regardless of their position in the network) appear to contain important information for solving the problem.

An issue with the approach presented, in terms of training the tensorloss models, was estimating the values of tensors is not a well conditioned problem, as the network is attempting to replicate a multi-dimensional vector of varying magnitudes. The results in Figure 9.6 clearly demonstrate that part of the network has encoded the depth in the magnitude of this tensor vector. However, this makes the replication of this tensor very challenging. A way to improve the ability of this approach may be to train the original supervisor with a normalisation layer between the penultimate layer and the *solver*. This would effectively force the network to encode the depth information in the activations independent of the tensor magnitude. This may be easier to emulate, but could reduce the effective model capacity as the network would no longer be able to encode information using magnitude. This is something that requires further investigation.

Conclusion

This thesis presents a number of research outcomes, that improve alignment accuracy, robustness of mapping approaches, and depth computation and estimation. Previous related approaches have been compared, and new techniques thoroughly discussed. The primarily contributions of this thesis are:

- A novel approach to low-cost depth sensor calibration, that's applicable to structured light type sensors (see Chapter 4). This method is provided as open-source and with few external library requirements, aiming to improve its utility. The resulting calibration improved upon previous approaches and didn't require the use of external calibration patterns or large planar regions. The solution is based on a general mapping based approach, where the calibration is solved jointly along with the mapping parameters, resulting in an improved calibration that's simpler to obtain.
- As an effort to combat the wide-baseline problem present in dense alignment approaches, a method to compute accurate surface curvature estimates in real-time was developed. The presented approach is more accurate than previous approaches, and shown to generate more robust measurements than previous approaches (see Chapter 5). This method is provided open-source and is implemented in CUDA for GPU based programming. Additionally, this work demonstrated that regions of similar surface curvature, can be used as a discriminative region level segmentation, that can be used to compute an initial estimate for a dense alignment approach.
- The approach for real-time curvature estimation was extended in Chapter 6 to jointly optimise for the relative pose alignment (in addition to the curvature) across multiple frames. This work is a novel joint optimisation approach, that demonstrated qualitatively and quantitatively that a more general surface representation is not only a more accurate method of aligning overlapping point clouds, but fitting the general surfaces to noisy data becomes much more robust for curvature estimation.
- The approach in Chapter 7 is the first of its kind to generate an estimate of surface curvature from a colour image, additionally producing near state-of-the-art normals and depths. This work demonstrated that learning related tasks is a valid approach to improving accuracy in the primary task, learning depth for example. This forces a network to learn more general features that direct it towards a better position in the network embedding space, and was shown

to be applicable across different architectures. This implies a general strategy for improved depth estimation performance is through the use of related and auxiliary tasks for better network conditioning.

- In a joint effort to improve visual odometry and depth estimation performance a method the jointly optimises for both was also developed (see Chapter 8). This approach produces state-of-the-art depths as a baseline, across indoor and outdoor datasets, and then demonstrated a further improvement by forcing the network to obey geometric constraints using relative pose estimates. This approach also demonstrated that more traditional optimisation approaches can be used to enhance the performance of networks, by enforcing geometric constraints that might be challenging for a network to learn. All these approaches indicate that using the most available information will generate the strongest solutions, where supervised approaches seem to contribute more significantly than unsupervised.
- Finally a method of network compression/distillation is shown that is a step towards either replacing or enhancing low-cost depth sensors in real-time for low-cost low-power robotic applications (see Chapter 9). This approach demonstrated that knowledge can be efficiently transferred from a large teacher network to a student network by simply transplanting layers. This work may even imply that transplanting any layers from a trained state-of-the-art network will improve performance on the given task.

10.0.1 Discussion and Future Work

Given the contributions outlined in this thesis, this brings up several points of discussion as well as many possibilities for future research that have been touched upon throughout this thesis.

- The approach presented in Chapter 5 and extended in Chapter 6, to incorporate the pose into the optimisation of a quadric, demonstrated a number of points. This work reaffirmed the fact that dense approaches are generally more accurate [17, 13], and a more general surface alignment is more robust and more accurate [258, 203]. In this work significant effort was put into increasing the speed of the multi-frame approach. As the speed of computation increases, and with adjustments to resolution or image pyramiding, a future implementation of the frame-to-frame approach could be realistically incorporated into a real-time odometry approach. Additionally, image luminance could also be included in the optimisation in a similar way to the works [15, 133, 17], to further improve the performance and robustness.
- In Chapter 7, a general approach to improving network performance through the use of related loss functions was presented. The performance of the networks that incorporate related loss functions reliably improved, regardless of the network architecture. This indicates the importance of the choice of loss functions in

optimisation, and further quantification of this contribution should be performed. This work was extended in Chapter 8, to focus on pose and flow in the optimisation, two quantities that are related to depth but less coupled. The analysis of this work also showed an improvement in network performance, based on the inclusion of the related loss terms. This result raises a number of questions; Is it just that a network that attempts to learn the quantity and a derivative of that, is going to learn better features in order to estimate the quantity? How much network guidance is appropriate, and can a network learn what loss functions are relevant to a primary goal?

- The work in Chapter 8 also explored the notion of direct pose estimation using a neural network vs a more computationally grounded approach, that focused on using quantities estimated by networks to iterate to a solution. In this case the solution that iterates on the estimated quantities proved to be a much stronger approach than previous machine learning approaches. This indicates, that perhaps the path forward with neural networks in SLAM systems may be to incorporate them for specific tasks that are well suited to a CNN, including optical flow estimation [240, 241], and depth estimation [171], instead of direct pose estimation. The inclusion of certainty estimation is extremely helpful for robust optimisation techniques, and proved to supply a massive improvement in this case. While this doesn't mean a machine learning approach won't learn a better alignment using only the information contained in two images, passed through a network, this may indicate a simpler way, that's motivated by geometry and could prove to be a very powerful solution, with works like [259, 260] possibly paving the way forward. Another possible extension of this could bring in the work of Chapter 6, as a possible loss function for surface curvature and pose alignment.
- The work presented in Chapter 9 demonstrates one of the desired directions for geometry based estimation networks, towards real-time inference. This work again demonstrated another method of increasing network performance through student-teacher training. This technique is becoming more popular [49, 261] as a method of reducing the time-taken by a significantly bulkier approach by attempting to transfer knowledge between them. The original intuition of this work was that it should show that a tensor-supervised approach will improve the network performance the most, however as shown in the results it was the transplanted network. The tensor-supervised network did improve the performance but to a lesser extent. The intuition as to why this could be, comes down to a combination of insufficient model capacity and unstable training regime. A future continuation of this work may seek to train a new supervisor/teacher network, that normalising the output of the penultimate layer, forcing the network to encode all the information for depth in the vector, without using the magnitude as a piece of information. This would be interesting from depth estimation point of view regardless. The model capacity can also be further explored through novel architectures and inference optimisation in future accelerated libraries like TensorRT.
- The works of Chapters 7, 8 and 9, show a steady improvement of single-image

10 CONCLUSION

depth estimation networks. This could signal a future where a monocular camera can be used as a depth sensor in a more general approach. In the short-term the work presented here can easily be used as a method to enhance the ability of existing low-cost depth sensors, possible through the estimation of super-resolution depths, or in-painting on incomplete depth images. The work in Chapter 9 in particular indicates a path towards real-time performance in these domains, perhaps complementing the work of [254].

Bibliography

- [1] H. Durrant-whyte, D. Rye, and E. Nebot, “Localization of Autonomous Guided Vehicles,” *Robotics Research*, pp. 613–625, 1996. [3](#), [28](#), [29](#)
- [2] D. Lowe, “Object Recognition From Local Scale-Invariant Features,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 1150–1157 vol.2, IEEE, 1999. [3](#), [7](#), [29](#)
- [3] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB : An Efficient Alternative to SIFT or SURF,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 2564–2571, 2011. [3](#), [7](#), [29](#), [30](#)
- [4] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “BRIEF : Binary Robust Independent Elementary Features,” in *European Conference on Computer Vision (ECCV)*, pp. 778–792, 2010. [3](#), [30](#)
- [5] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, “Monoslam: Real-time single camera slam,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 29, no. 6, pp. 1052–1067, 2007. [4](#)
- [6] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” *European Conference on Computer Vision (ECCV)*, pp. 430–443, 2006. [4](#)
- [7] R. Mur-Artal and J. D. Tardós, “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras,” *IEEE Transactions on Robotics (ToR)*, vol. 33, no. 5, pp. 1255–1262, 2017. [4](#), [18](#), [28](#), [29](#), [30](#), [43](#), [63](#), [225](#), [235](#), [236](#), [237](#), [238](#), [239](#)
- [8] M. Pizzoli, C. Forster, and D. Scaramuzza, “Remode: Probabilistic, monocular dense reconstruction in real time,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2609–2616, IEEE, 2014. [4](#), [43](#), [44](#), [45](#)

BIBLIOGRAPHY

- [9] C. Forster, M. Pizzoli, and D. Scaramuzza, “Svo: Fast semi-direct monocular visual odometry,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15–22, IEEE, 2014. 4
- [10] J. Engel, T. Schöps, and D. Cremers, “Lsd-slam: Large-scale direct monocular slam,” in *European Conference on Computer Vision (ECCV)*, pp. 834–849, Springer, 2014. 4, 28, 43, 199, 200, 216, 218, 221, 240
- [11] J. Engel, V. Koltun, and D. Cremers, “Direct sparse odometry,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017. 4, 28, 43
- [12] P. J. Besl and N. D. McKay, “A Method for Registration of 3-D Shapes,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 14, no. 2, pp. 239–256, 1992. 5, 31, 115, 155, 156
- [13] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, “KinectFusion : Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera,” in *ACM Symposium on User Interface Software and Technology (UIST)*, 2011. 5, 28, 31, 63, 112, 115, 156, 181, 242
- [14] R. a. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, “KinectFusion: Real-Time Dense Surface Mapping and Tracking,” in *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 127–136, IEEE, Oct. 2011. 5, 6, 19, 28, 31, 32, 34, 63, 91, 94, 102, 112, 155, 156, 178, 181, 199, 200
- [15] R. a. Newcombe, S. J. Lovegrove, and A. J. Davison, “DTAM: Dense Tracking and Mapping in Real-Time,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 2320–2327, IEEE, Nov. 2011. 5, 6, 28, 43, 44, 45, 115, 156, 181, 242
- [16] 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*, (Sydney, Australia), Jul 2012. 5, 6, 19, 28, 31, 156
- [17] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, and A. Davison, “Elasticfusion: Dense slam without a pose graph,” in *Robotics: Science and Systems*

BIBLIOGRAPHY

- (*RSS*), Robotics: Science and Systems, 2015. 5, 6, 19, 28, 31, 33, 34, 63, 91, 94, 102, 115, 242
- [18] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. B. McDonald, “Robust Real-Time Visual Odometry for Dense RGB-D Mapping,” in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2013. 5, 6, 19, 28, 31, 152
- [19] P. Besl and R. Jain, “Segmentation Through Variable-Order Surface Fitting,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 10, pp. 167–192, Mar. 1988. 6, 25, 132, 142, 154, 157, 179, 180, 181
- [20] D. Scharstein and R. Szeliski, “High-accuracy stereo depth maps using structured light,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. I–I, IEEE, 2003. 6, 21, 61, 62
- [21] T. Weise, B. Leibe, and L. Van Gool, “Fast 3d scanning with automatic motion compensation,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, IEEE, 2007. 6, 19, 21, 22, 62
- [22] K. Khoshelham, “Accuracy Analysis of Kinect Depth Data,” *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXVIII-5, no. August, pp. 133–138, 2012. 6
- [23] MobilEYE, “Mobileye-the evolution of eyeq,” 2018. 6, 20
- [24] 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. 7, 102, 154, 155, 157, 158, 171, 175, 176, 177, 179, 181, 185, 186, 191, 192, 193, 194
- [25] A. Spek and T. Drummond, “Joint pose and principal curvature refinement using quadrics,” in *International Conference on Robotics and Automation (ICRA)*, pp. 3968–3975, IEEE, 2017. 7, 28
- [26] A. Krizhevsky and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Neural Information Processing Systems (NIPS)*, pp. 1–9, 2012. 11, 44, 181, 199, 226

BIBLIOGRAPHY

- [27] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *International Conference on Learning Representations (ICLR)*, 2015. 11, 35, 38, 181, 182
- [28] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in *International Conference on Multimedia (ICM)*, pp. 675–678, ACM, 2014. 11, 102, 186
- [29] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, IEEE, 2016. 11, 38, 109, 224
- [30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, *et al.*, “Going deeper with convolutions,” in *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2015. 11
- [31] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, p. 436, 2015. 11
- [32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. 11, 12
- [33] 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 (JMLR)*, vol. 15, no. 1, pp. 1929–1958, 2014. 11, 109
- [34] D. Eigen and R. Fergus, “Predicting Depth, Surface Normals and Semantic Labels With a Common Multi-Scale Convolutional Architecture,” *IEEE International Conference on Computer Vision (ICCV)*. 12, 19, 35, 38, 39, 40, 44, 63, 179, 180, 181, 184, 186, 190, 191, 192, 193, 194, 197, 201, 213, 226, 233
- [35] D. Eigen, C. Puhrsch, and R. Fergus, “Depth Map Prediction From a Single Image Using a Multi-Scale Deep Network,” *Neural Information Processing Systems (NIPS)*, pp. 1–9, 2014. 12, 13, 19, 38, 39, 40, 42, 44, 107, 181, 188, 190, 201, 211, 212, 225, 226

BIBLIOGRAPHY

- [36] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” in *European Conference on Computer Vision (ECCV)*, pp. 346–361, Springer, 2014. 12, 38
- [37] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, IEEE, 2015. 12, 44, 181, 186, 190, 199, 210, 211, 226, 229
- [38] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, “Facial landmark detection by deep multi-task learning,” in *European Conference on Computer Vision (ECCV)*, pp. 94–108, Springer, 2014. 13
- [39] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, “Netvlad: Cnn architecture for weakly supervised place recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5297–5307, IEEE, 2016. 13
- [40] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, “Indoor Segmentation and Support Inference from RGBD Images,” in *European Conference Computer Vision (ECCV)*, pp. 746–760, 2012. 13, 19, 36, 38, 39, 63, 113, 180, 181, 184, 185, 193, 204, 209, 210, 211, 212, 213, 215, 221, 222, 224, 228, 232, 233, 236
- [41] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The KITTI dataset,” *International Journal of Robotics Research (IJRR)*, 2013. 13, 45, 209, 211, 212, 213, 214, 215, 216, 220, 221, 222, 232, 234, 237, 238, 239
- [42] 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)*, pp. 740–756, 2016. 13, 45, 201, 204, 214, 226, 234
- [43] C. Godard, O. Mac Aodha, and G. J. Brostow, “Unsupervised monocular depth estimation with left-right consistency,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017. 13, 45, 201, 214, 226, 234
- [44] Y. Kuznetsov, J. Stückler, and B. Leibe, “Semi-supervised deep learning for monocular depth map prediction,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017. 13, 45, 201, 204, 214, 230, 234

BIBLIOGRAPHY

- [45] 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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 14, 45, 202, 207, 209, 210, 215, 216, 218, 219
- [46] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, “Unsupervised learning of depth and ego-motion from video,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017. 14, 45, 202, 207, 209, 210, 214, 216, 217, 234
- [47] A. Crivellaro, M. Rad, Y. Verdie, K. M. Yi, P. Fua, and V. Lepetit, “Robust 3d object tracking from monocular images using stable parts,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017. 14
- [48] 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. 14, 46, 226
- [49] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015. 15, 46, 226, 230, 243
- [50] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, “Orb-slam: A versatile and accurate monocular slam system,” *IEEE Transactions on Robotics (ToR)*, vol. 31, no. 5, pp. 1147–1163, 2015. 16, 18, 28, 29, 30, 43, 199, 200, 216, 217, 218, 219, 237
- [51] 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 Robot Systems (IROS)*, IEEE, 2012. 19, 39, 170, 204, 205, 209, 210, 211, 212, 215, 216, 218, 219, 222, 228, 232, 235, 236, 237
- [52] T. Weise, T. Wismer, B. Leibe, and L. Van Gool, “In-Hand Scanning With Online Loop Closure,” in *IEEE International Conference on Computer Vision Workshops (ICCV)*, pp. 1630–1637, IEEE, Sept. 2009. 19
- [53] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, “Close-Range Scene Segmentation and Reconstruction of 3D Point Cloud Maps for Mobile Manipulation in Domestic Environments,” in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–6, IEEE, Oct. 2009. 19, 24, 27, 36, 37

BIBLIOGRAPHY

- [54] R. B. Rusu, A. Holzbach, M. Beetz, M. Garching, G. Bradski, W. Garage, W. Road, and M. Park, “Detecting and Segmenting Objects for Mobile Manipulation,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 47–54, IEEE, 2009. 19, 24, 27, 34, 37, 39, 41
- [55] R. B. Rusu, *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, 2010. 19, 25, 154, 157
- [56] M. J. Hannah, “Computer matching of areas in stereo images,” tech. rep., Stanford University Dept. of Computer Science, 1974. 19
- [57] B. D. Lucas, T. Kanade, *et al.*, “An iterative image registration technique with an application to stereo vision,” 1981. 19
- [58] R. Mandelbaum, G. Kamberova, and M. Mintz, “Stereo depth estimation: A confidence interval approach,” in *International Conference on Computer Vision (1998)*, pp. 503–509, IEEE, 1998. 19, 22
- [59] H. Hirschmuller, “Stereo processing by semiglobal matching and mutual information,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 30, no. 2, pp. 328–341, 2008. 19, 59, 237
- [60] E. Tola, V. Lepetit, and P. Fua, “Daisy: An efficient dense descriptor applied to wide-baseline stereo,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 32, no. 5, pp. 815–830, 2010. 19, 29, 43
- [61] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, “Real-time stereo vision system using semi-global matching disparity estimation: Architecture and fpga-implementation,” in *International Conference on Embedded Computer Systems (SAMOS)*, pp. 93–101, IEEE, 2010. 19, 20, 59
- [62] S. K. Gehrig, F. Eberli, and T. Meyer, “A real-time low-power stereo vision engine using semi-global matching,” in *International Conference on Computer Vision Systems (ICCVS)*, pp. 134–143, Springer, 2009. 20
- [63] G. Van Der Wal, M. Hansen, and M. Piacentino, “The acadia vision processor,” in *IEEE International Workshop on Computer Architectures for Machine Perception (IWCAMP)*, pp. 31–40, IEEE, 2000. 20

BIBLIOGRAPHY

- [64] S. Foix, G. Alenya, and C. Torras, “Lock-in time-of-flight (tof) cameras: A survey,” *Sensors*, vol. 11, no. 9, pp. 1917–1926, 2011. 20, 58
- [65] StereoLabs, “Stereolabs - zed stereo camera,” 2018. 20
- [66] P. Grey, “Point grey - bumblebee2-firewire stereo vision camera systems,” 2018. 20
- [67] S. B. Gokturk, H. Yalcin, and C. Bamji, “A time-of-flight depth sensor-system description, issues and solutions,” in *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, pp. 35–35, IEEE, 2004. 20
- [68] Y. Cui, S. Schuon, D. Chan, S. Thrun, and C. Theobalt, “3d shape scanning with a time-of-flight camera,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1173–1180, IEEE, 2010. 20
- [69] J. Zhu, L. Wang, R. Yang, and J. Davis, “Fusion of time-of-flight depth and stereo for high accuracy depth maps,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, IEEE, 2008. 20, 21
- [70] J. Zhu, L. Wang, R. Yang, J. E. Davis, *et al.*, “Reliability fusion of time-of-flight depth and stereo geometry for high quality depth maps,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 33, no. 7, pp. 1400–1414, 2011. 20, 21
- [71] C. Niclass, M. Soga, H. Matsubara, M. Ogawa, and M. Kagami, “A $0.18\mu m$ cmos soc for a 100-m-range 10-frame/s 200×96 -pixel time-of-flight depth sensor,” *IEEE Journal of Solid-State Circuits*, vol. 49, no. 1, pp. 315–330, 2014. 20
- [72] C. Niclass, M. Soga, H. Matsubara, S. Kato, and M. Kagami, “A 100-m range 10-frame/s 340×96 -pixel time-of-flight depth sensor in $0.18\text{-}\mu m$ cmos,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 2, pp. 559–572, 2013. 20
- [73] R. J. Walker, J. A. Richardson, and R. K. Henderson, “A 128×96 pixel event-driven phase-domain $\delta\sigma$ -based fully digital 3d camera in $0.13\text{ }\mu m$ cmos imaging technology,” in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 410–412, IEEE, 2011. 20

BIBLIOGRAPHY

- [74] J.-H. Cho, S.-Y. Kim, Y.-S. Ho, and K. H. Lee, "Dynamic 3d human actor generation method using a time-of-flight depth camera," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 4, 2008. [20](#)
- [75] M. Hansard, S. Lee, O. Choi, and R. P. Horaud, *Time-of-Flight Cameras: Principles, Methods and Applications*. Springer, 2012. [21](#), [57](#), [58](#)
- [76] K. Khoshelham and S. O. Elberink, "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications.," *Sensors*, vol. 12, pp. 1437–54, Jan. 2012. [21](#), [22](#), [25](#), [112](#), [132](#), [154](#), [185](#)
- [77] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, *et al.*, "The digital michelangelo project: 3d scanning of large statues," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 131–144, ACM Press/Addison-Wesley Publishing Co., 2000. [21](#)
- [78] O. Kinect, "Open kinect - libfreenect getting started," 2018. [22](#)
- [79] C. Daniel Herrera, J. Kannala, and J. Heikkil, "Joint Depth and Color Camera Calibration With Distortion Correction," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 34, no. 10, pp. 2058–2064, 2012. [22](#), [112](#), [113](#)
- [80] C. Raposo, J. P. Barreto, and U. Nunes, "Fast and accurate calibration of a kinect sensor," in *International Conference on 3D Vision (3DV)*, pp. 342–349, 2013. [22](#), [112](#), [113](#)
- [81] B. Jin, H. Lei, and W. Geng, "Accurate Intrinsic Calibration of Depth Camera With Cuboids," in *European Conference Computer Vision (ECCV)*, vol. 8693 LNCS, pp. 788–803, 2014. [23](#), [113](#)
- [82] M. Di Cicco, L. Iocchi, and G. Grisetti, "Non-Parametric Calibration for Depth Sensors," *Robotics and Autonomous Systems (RAS)*, vol. 74, pp. 309–317, 2015. [23](#), [112](#), [113](#)
- [83] A. Teichman, S. Miller, and S. Thrun, "Unsupervised Intrinsic Calibration of Depth Sensors via SLAM.," *Robotics Science and Systems (RSS)*, 2013. [23](#), [113](#), [124](#), [126](#), [127](#)

BIBLIOGRAPHY

- [84] Q.-y. Zhou and V. Koltun, “Simultaneous Localization and Calibration : Self-Calibration of Consumer Depth Cameras,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2014. [23](#), [112](#), [113](#), [218](#)
- [85] R. Yagel, D. Cohen, and A. Kaufman, “Normal estimation in 3d discrete space,” *The Visual Computer*, vol. 8, no. 5-6, pp. 278–291, 1992. [24](#)
- [86] T. K. Dey, G. Li, and J. Sun, “Normal estimation for point clouds: A comparison study for a voronoi based method,” in *Eurographics/IEEE VGTC symposium Point-Based Graphics*, pp. 39–46, IEEE, 2005. [24](#), [25](#)
- [87] N. Amenta, M. Bern, and M. Kamvysselis, “A new voronoi-based surface reconstruction algorithm,” in *Conference on Computer Graphics and Interactive Techniques (CGIT)*, pp. 415–421, ACM, 1998. [24](#), [25](#)
- [88] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, “Comparison of surface normal estimation methods for range sensing applications,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3206–3211, IEEE, 2009. [24](#)
- [89] J. H. Lambert, *Photometria sive de mensura et gradibus luminis, colorum et umbrae*. Klett, 1760. [24](#)
- [90] N. N. Abdelmalek, “Algebraic Error Analysis for Surface Curvatures and Segmentation of 3-D Range Images,” *Pattern Recognition*, pp. 807–817, 1990. [25](#), [132](#)
- [91] T. Gatzke and C. Grimm, “Estimating Curvature on Triangular Meshes,” *International Journal of Shape Modeling (IJSM)*, vol. 12, no. 1, pp. 1–28, 2006. [25](#)
- [92] 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 (ToVCG)*, vol. 18, pp. 1603–13, Oct. 2012. [25](#), [26](#), [102](#), [132](#), [154](#), [181](#)
- [93] W. Griffin, Y. Wang, D. Berrios, and M. Olano, “GPU Curvature Estimation on Deformable Meshes,” in *Symposium on Interactive 3D Graphics and Games*, pp. 159–166, 2011. [25](#), [132](#), [157](#)

BIBLIOGRAPHY

- [94] S. Rusinkiewicz, “Estimating Curvatures and Their Derivatives on Triangle Meshes,” in *Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, no. 2, pp. 486–493, IEEE, 2010. [25](#), [132](#), [154](#), [157](#), [181](#)
- [95] P. J. Flynn and A. K. Jain, “On Reliable Curvature Estimation,” pp. 110–116, IEEE, 1989. [25](#)
- [96] 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. [25](#), [179](#), [180](#), [181](#)
- [97] L. Guillaume, D. Florent, and B. Atilla, “Curvature Tensor Based Triangle Mesh Segmentation With Boundary Rectification,” in *IEEE International Conference on Computer Graphics (ICCG)*, pp. 10–25, IEEE, 2004. [25](#), [132](#), [154](#), [157](#)
- [98] I. Douros and B. Buxton, “Three-Dimensional Surface Curvature Estimation Using Quadric Surface Patches,” *Scanning*, vol. 44, no. 0, 2002. [25](#), [132](#), [142](#), [154](#), [171](#), [175](#), [179](#), [180](#), [181](#)
- [99] R. B. Rusu and S. Cousins, “3D Is Here: Point Cloud Library (PCL),” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–4, IEEE, May 2011. [25](#), [26](#), [132](#), [133](#)
- [100] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, “Learning Informative Point Classes for the Acquisition of Object Model Maps,” in *IEEE International Conference on Control, Automation, Robotics and Vision (ICCARV)*, no. December, pp. 17–20, IEEE, 2008. [26](#), [27](#), [34](#), [36](#), [39](#), [41](#)
- [101] R. B. Rusu, N. Blodow, and M. Beetz, “Fast Point Feature Histograms (FPFH) for 3D registration,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3212–3217, IEEE, May 2009. [26](#), [27](#), [36](#), [37](#), [41](#), [132](#)
- [102] P. Scovanner, S. Ali, and M. Shah, “A 3-Dimensional Sift Descriptor and Its Application to Action Recognition,” *International Conference on Multimedia*, no. c, p. 357, 2007. [26](#), [27](#), [28](#)

BIBLIOGRAPHY

- [103] B. Drost, M. Ulrich, N. Navab, and S. Ilic, “Model Globally, Match Locally: Efficient and Robust 3D Object Recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 998–1005, IEEE, June 2010. 27, 32
- [104] A. E. Johnson, *Spin-Images : A Representation for 3-D Surface Matching*. PhD thesis, 1997. 27
- [105] D. Ni, Y. Qul, X. Yang, Y. P. Chui, T.-T. Wong, S. S. M. Ho, and P. A. Heng, “Volumetric Ultrasound Panorama Based on 3D SIFT,” *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, vol. 11, pp. 52–60, Jan. 2008. 27, 28
- [106] G. Flitton, T. Breckon, and N. Megherbi Bouallagu, “Object Recognition using 3D SIFT in Complex CT Volumes,” *British Machine Vision Conference (BMVC)*, pp. 11.1–11.12, 2010. 27, 28
- [107] A. Klaser, M. Marszalek, and C. Schmid, “A Spatio-Temporal Descriptor Based on 3D Gradients,” in *British Machine Vision Conference (BMVC)*, British Machine Vision Association, 2008. 27
- [108] B. Steder and K. Konolige, “NARF : 3D Range Image Features for Object Recognition,” in *IEEE International Conference on Robots and Systems (IROS)*, IEEE, 2010. 28
- [109] H. Durrant-whyte and T. Bailey, “Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms,” *Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110. 28, 29
- [110] T. Bailey and H. Durrant-whyte, “Simultaneous Localization and Mapping (SLAM): Part II,” *IEEE Robotics & Automation Magazine*, no. September, pp. 108–117, 2006. 28, 29
- [111] G. Klein and D. Murray, “Parallel Tracking and Mapping for Small AR Workspaces,” in *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 1–10, IEEE, Nov. 2007. 28, 43, 181, 200
- [112] C. Engels, H. Stewenius, and D. Nister, “Bundle Adjustment Rules,” *Photogrammetric Computer Vision*, 2006. 28

BIBLIOGRAPHY

- [113] M. Meilland, A. I. Comport, and P. Rives, “Dense Visual Mapping of Large Scale Environments for Real-Time Localisation,” in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 4242–4248, IEEE, Sept. 2011. 28, 31, 33
- [114] M. Meilland, P. Rives, and A. I. Comport, “Dense RGB-D Mapping of Large Scale Environments for Real-Time Localisation and Autonomous Navigation,” in *Intelligent Vehicle Workshop on Navigation, Perception, Accurate Positioning and Mapping for Intelligent Vehicles*, 2012. 28, 31, 33
- [115] H. Bay, T. Tuytelaars, and L. V. Gool, “SURF : Speeded Up Robust Features,” in *European Conference on Computer Vision (ECCV)*, pp. 404–417, 2006. 29, 30
- [116] E. Rosten and T. Drummond, “Machine Learning for High-Speed Corner Detection,” in *European Conference Computer Vision (ECCV)*, pp. 430–443, 2006. 29, 30
- [117] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua, “Lift: Learned invariant feature transform,” in *European Conference on Computer Vision (ECCV)*, pp. 467–483, Springer, 2016. 29, 30, 202
- [118] S. Mourato, P. Fernandez, L. Pereira, and M. Moreira, “Improving a dsm obtained by unmanned aerial vehicles for flood modelling,” in *IOP Conference Series: Earth and Environmental Science*, vol. 95, p. 022014, IOP Publishing, 2017. 29
- [119] Y. Chen and G. Medioni, “Object Modeling by Registration of Multiple Range Images,” in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 1991. 31, 155, 156
- [120] S. Rusinkiewicz and M. Levoy, “Efficient Variants of the ICP Algorithm,” in *International Conference on 3D Digital Imaging and Modeling (3DIM)*, pp. 145–152, IEEE, 2001. 31, 90, 115, 132, 156
- [121] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy, “Real-Time 3D Model Acquisition,” *ACM Transactions on Graphics (TOG)*, vol. 21, July 2002. 31
- [122] W. L. D. Lui, T. J. J. Tang, T. Drummond, and W. H. Li, “Robust Egomotion Estimation Using ICP in Inverse Depth Coordinates,” *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1671–1678, May 2012. 31

BIBLIOGRAPHY

- [123] T. Jost, H. Hügli, and C. Neuchâtel, “A Multi-Resolution Scheme ICP Algorithm for Fast Shape Registration,” in *3D Data Processing Visualization and Transmission (3DPVT)*, pp. 2000–2003, 2002. 31
- [124] T. Jost and H. Hügli, “Fast ICP Algorithms for Shape Registration,” in *DAGM Symposium*, pp. 91–99, 2002. 31
- [125] 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)*, vol. 31, pp. 647–663, Feb. 2012. 31, 33
- [126] P. Henry, D. Fox, A. Bhowmik, and R. Mongia, “Patch Volumes: Segmentation-Based Consistent Mapping with RGB-D Cameras,” in *International Conference on 3D Vision (3DV)*, pp. 398–405, IEEE, June 2013. 31, 33
- [127] C. Audras and A. Comport, “Real-Time Dense Appearance-Based SLAM for RGB-D Sensors,” in *Australasian Conf. on Robotics and Automation (ACRA)*, ARAA, 2011. 31, 33
- [128] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, “Real-Time 3D Reconstruction in Dynamic Scenes Using Point-Based Fusion,” in *International Conference on 3DTV-Conference*, pp. 1–8, IEEE, June 2013. 31
- [129] R. F. Salas-Moreno, R. a. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison, “SLAM++: Simultaneous Localisation and Mapping at the Level of Objects,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1352–1359, June 2013. 31, 32, 155, 156
- [130] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann, “Robust Global Registration,” in *Symposium on Geometry Processing*, pp. 197–206, 2005. 32
- [131] E. Ataer-Cansizoglu, Y. Taguchi, S. Ramalingam, and T. Garaas, “Tracking an RGB-D Camera Using Points and Planes,” in *IEEE Workshop on Consumer Depth Cameras for Computer Vision*, 2013. 33
- [132] T. Tykkala, C. Audras, and A. I. Comport, “Direct Iterative Closest Point for Real-time Visual Odometry,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 2050–2056, IEEE, 2011. 33

BIBLIOGRAPHY

- [133] M. Meilland, A. Comport, and P. Rives, “Real-time dense visual tracking under large lighting variations,” in *British Machine Vision Conference (BMVC)*, British Machine Vision Association. 33, 45, 242
- [134] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Neural Information Processing Systems (NIPS)*, pp. 91–99, 2015. 34, 38
- [135] S. Gould, T. Gao, and D. Koller, “Region-Based Segmentation and Object Detection,” in *Neural Information Processing Systems (NIPS)*, vol. 1, p. 2, 2009. 34, 35, 39
- [136] V. Prisacariu, O. Adrian, D. Kahler, W. Murray, and Ian D. Reid., “Simultaneous 3D Tracking and Reconstruction on a Mobile Phone,” in *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, no. October, pp. 89–98, 2013. 34, 35
- [137] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, “Learning Hierarchical Features for Scene Labeling,” *IEEE Transaction on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 35, no. 8, pp. 1915–1929, 2013. 34, 35
- [138] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 39, no. 12, pp. 2481–2495, 2017. 34, 35, 36
- [139] 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,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 40, no. 4, pp. 834–848, 2018. 35, 36
- [140] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention (ICMICCAI)*, pp. 234–241, Springer, 2015. 36, 203
- [141] A. Kendall and Y. Gal, “What uncertainties do we need in bayesian deep learning for computer vision?,” in *Neural Information Processing Systems (NIPS)*, pp. 5580–5590, 2017. 36, 42, 45, 201, 213

BIBLIOGRAPHY

- [142] A. K. Mishra, A. Shrivastava, and Y. Aloimonos, "Segmenting "Simple" Objects Using RGB-D," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4406–4413, IEEE, May 2012. [37](#), [195](#)
- [143] B.-s. Kim, A. Arbor, and S. Savarese, "Accurate Localization of 3D Objects from RGB-D Data using Segmentation Hypotheses," in *Computer Vision and Pattern Recognition (CVPR)*, pp. 3182–3189, 2013. [37](#), [39](#)
- [144] J. Carreira and C. Sminchisescu, "Constrained Parametric Min-Cuts for Automatic Object Segmentation.," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, Nov. 2011. [37](#)
- [145] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning rich features from rgb-d images for object detection and segmentation," in *European Conference on Computer Vision (ECCV)*, pp. 345–360, Springer, 2014. [38](#)
- [146] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," [38](#), [44](#), [178](#), [181](#), [199](#), [226](#)
- [147] C. J. Holder, T. P. Breckon, and X. Wei, "Depth not needed-an evaluation of rgb-d feature encodings for off-road scene understanding by convolutional neural network," *arXiv preprint arXiv:1801.01235*, 2018. [38](#)
- [148] M. Ranzato, V. Mnih, J. M. Susskind, and G. E. Hinton, "Modeling Natural Images Using Gated MRFs.," *IEEE Transaction on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 35, pp. 2206–22, Sept. 2013. [38](#)
- [149] K. Lai, L. Bo, X. Ren, and D. Fox, "Detection-Based Object Labeling in 3D Scenes," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1330–1337, IEEE, May 2012. [38](#), [39](#), [42](#)
- [150] L. Bo, X. Ren, and D. Fox, "Kernel Descriptors for Visual Recognition," *Neural Information Processing Systems (NIPS)*, vol. 1, no. 2, p. 3, 2010. [38](#), [39](#)
- [151] H. Azizpour and I. Laptev, "Object Detection Using Strongly-Supervised Deformable Part Models," *European Conference on Computer Vision (ECCV)*, pp. 836–849, 2012. [38](#)

BIBLIOGRAPHY

- [152] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, Ieee, June 2009. 38, 40
- [153] K. Lai, L. Bo, X. Ren, and D. Fox, “A Large-Scale Hierarchical Multi-View RGB-D Object Dataset,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1817–1824, IEEE, May 2011. 38, 39
- [154] L. Bo, X. Ren, and D. Fox, “Unsupervised Feature Learning for RGB-D Based Object Recognition,” *Experimental Robotics*, pp. 387–402, 2013. 38, 39
- [155] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, “Densely connected convolutional networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, p. 3, 2017. 38, 42, 199, 202, 210, 224, 226
- [156] M. C. Bernardes, B. V. Adorno, P. Poignet, and G. a. Borges, “Semi-Automatic Needle Steering System With Robotic Manipulator,” in *IEEE International Conference on Robotics and Automation (IEEE)*, pp. 1595–1600, IEEE, May 2012. 38
- [157] F. Liu, C. Shen, and G. Lin, “Deep Convolutional Neural Fields for Depth Estimation from a Single Image,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 38, 45, 179, 181, 190, 201, 233
- [158] I. Laina, C. Rupprecht, and F. Tombari, “Deeper Depth Prediction with Fully Convolutional Residual Networks,” 2016. 38, 45, 179, 181, 190, 196, 197, 201, 203, 208, 213, 214, 215, 226, 233
- [159] T. Dharmasiri, A. Spek, and T. Drummond, “Joint prediction of depths, normals and surface curvature from rgb images using cnns,” *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2017. 38, 39, 40, 45, 178, 199, 201, 226
- [160] A. Bansal, B. Russell, and A. Gupta, “Marr revisited: 2d-3d alignment via surface normal prediction,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5965–5974, IEEE, 2016. 38, 181, 182, 191, 192, 226
- [161] N. Corporation, “Cuda 2.0 release notes,” 2009. 38, 39

BIBLIOGRAPHY

- [162] O. Chapelle, P. Haffner, and V. N. Vapnik, “Support Vector Machines for Histogram-Based Image Classification.,” *IEEE Transactions on Neural Networks*, vol. 10, pp. 1055–64, Jan. 38, 39, 41
- [163] J. J. DiCarlo, D. Zoccolan, and N. C. Rust, “How Does the Brain Solve Visual Object Recognition?,” *Neuron*, vol. 73, pp. 415–34, Feb. 2012. 38, 39
- [164] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object Detection With Discriminatively Trained Part-Based Models.,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pp. 1627–45, Sept. 39
- [165] D. Lin and R. Urtasun, “Holistic Scene Understanding for 3D Object Detection with RGBD cameras,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 1417–1424, 2013. 39, 42
- [166] Z.-C. Marton, D. Pangercic, N. Blodow, and M. Beetz, “Combined 2D-3D Categorization and Classification for Multimodal Perception Systems,” *The International Journal of Robotics Research (IJRR)*, vol. 30, pp. 1378–1402, Aug. 2011. 39, 132
- [167] X. Ren, L. Bo, and D. Fox, “RGB-(D) Scene Labeling: Features and Algorithms,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2759–2766, IEEE, June 2012. 39
- [168] L. Bo and D. Fox, “Hierarchical Matching Pursuit for Image Classification : Architecture and Fast Algorithms,” *Neural Information Processing Systems (NIPS)*, vol. 1, no. 2, p. 6, 2011. 39
- [169] J. Xiao, A. Owens, and A. Torralba, “SUN3D : A Database of Big Spaces Reconstructed Using SfM and Object Labels,” *Frontiers in Psychology*, vol. 4, 2013. 39
- [170] N. Silberman and R. Fergus, “Indoor scene segmentation using a structured light sensor,” in *Proceedings of the International Conference on Computer Vision - Workshop on 3D Representation and Recognition*, 2011. 39
- [171] T. Dharmasiri, A. Spek, and T. Drummond, “Engnet: End-to-end neural geometry,” 2018. 39, 42, 46, 199, 230, 233, 234, 243

BIBLIOGRAPHY

- [172] A. Spek, T. Dharmasiri, and T. Drummond, “Cream: Condensed real-time models for depth prediction using cnns.” 2018. 39, 223
- [173] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A Fast Learning Algorithm for Deep Belief Nets,” *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006. 40
- [174] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations,” *International Conference on Machine Learning (ICML)*, pp. 1–8, 2009. 41
- [175] A. Coates, A. Karpathy, and A. Y. Ng, “Emergence of Object-Selective Features in Unsupervised Feature Learning,” *Neural Information Processing Systems (NIPS)*, vol. 25, pp. 2690–2698, 2012. 41
- [176] A. Coates, H. Lee, and A. Y. Ng, “An Analysis of Single-Layer Networks in Unsupervised Feature Learning,” in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 215–223, 2011. 41
- [177] C. Russell, P. Kohli, P. H. Torr, *et al.*, “Associative hierarchical crfs for object class image segmentation,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 739–746, IEEE, 2009. 42
- [178] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr, “Conditional random fields as recurrent neural networks,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 1529–1537, IEEE, 2015. 42
- [179] J. Engel, J. Sturm, and D. Cremers, “Semi-Dense Visual Odometry for a Monocular Camera,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 1449–1456, IEEE, Dec. 2013. 43
- [180] C. Strecha and L. Van Gool, “Motion-stereo integration for depth estimation,” in *European Conference on Computer Vision (ECCV)*, pp. 170–185, Springer, 2002. 43
- [181] C. Strecha, R. Fransens, and L. Van Gool, “Combined depth and outlier estimation in multi-view stereo,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 2394–2401, IEEE, 2006. 43

BIBLIOGRAPHY

- [182] E. Tola, C. Strecha, and P. Fua, “Efficient large-scale multi-view stereo for ultra high-resolution image sets,” *Machine Vision and Applications (MVA)*, vol. 23, no. 5, pp. 903–920, 2012. 44
- [183] S. Lovegrove and A. J. Davison, “Real-time spherical mosaicing using whole image alignment,” in *European Conference on Computer Vision (ECCV)*, pp. 73–86, Springer, 2010. 44
- [184] J. Redmon and A. Angelova, “Real-Time Grasp Detection Using Convolutional Neural Networks,” pp. 1316–1322, 2015. 44, 181
- [185] M. Gualtieri, K. Saenko, R. Platt, and I. Science, “High Precision Grasp Pose Detection in Dense Clutter,” *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2016. 44, 181
- [186] B. Rogers and M. Graham, “Similarities between motion parallax and stereopsis in human depth perception,” *Vision research*, vol. 22, no. 2, pp. 261–270, 1982. 44
- [187] A. Saxena, S. H. Chung, and A. Y. Ng, “Learning Depth from Single Monocular Images,” pp. 1161–1168, 2006. 44, 179, 181, 201
- [188] 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. 46, 226
- [189] L. Deng, M. Yang, H. Li, T. Li, B. Hu, and C. Wang, “Restricted deformable convolution based road scene semantic segmentation using surround view cameras,” *arXiv preprint arXiv:1801.00708*, 2018. 46, 226
- [190] 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. 46, 227
- [191] C. BuciluȚ, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in *International Conference on Knowledge Discovery and Data Mining (ICKDDM)*, pp. 535–541, ACM, 2006. 46, 226

BIBLIOGRAPHY

- [192] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *International Conference on Learning Representations (ICLR)*, 2015. 47, 227
- [193] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge university press, 2003. 52, 70, 84
- [194] D. C. Brown, “Decentering distortion of lenses,” *Photogrammetric Engineering and Remote Sensing*, 1966. 56
- [195] N. Einecke and J. Eggert, “Block-matching stereo with relaxed fronto-parallel assumption,” in *Intelligent Vehicles Symposium Proceedings (IVSP)*, pp. 700–705, IEEE, 2014. 59
- [196] J. P. Lewis, “Fast template matching,” in *Vision interface*, vol. 95, pp. 15–19, 1995. 59
- [197] I. Lysenkov, V. Eruhimov, and G. Bradski, “Recognition and pose estimation of rigid transparent objects with a kinect sensor,” *Robotics*, vol. 273, 2013. 63
- [198] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*, vol. 589. John wiley & sons, 1987. 70
- [199] G. F. Piepel, “Robust regression and outlier detection,” *Technometrics*, vol. 31, no. 2, pp. 260–261, 1989. 70
- [200] Z. Zhang, “Parameter estimation techniques: A tutorial with application to conic fitting,” *Image and vision Computing*, vol. 15, no. 1, pp. 59–76, 1997. 70
- [201] Z. Zhang, *Parameter Estimation Techniques: A Tutorial With Application to Conic Fitting*, vol. 15. Elsevier, January 1997. 79
- [202] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981. 91
- [203] N. Mitra, N. Gelfand, H. Pottmann, and L. Guibas *SIGGRAPH Symposium on Geometry Processing*. 96, 132, 154, 155, 157, 159, 171, 181, 242

BIBLIOGRAPHY

- [204] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing With GPUs*. Newnes, 2012. 98, 100
- [205] S. Hong and H. Kim, "An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness," in *ACM SIGARCH Computer Architecture News*, vol. 37, pp. 152–163, ACM, 2009. 101
- [206] 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. 102, 207, 210, 232
- [207] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016. 103
- [208] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *IEEE International Conference on Computer Vision (ICCV)*, pp. 1520–1528, IEEE, 2015. 103
- [209] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics (ICAIS)*, pp. 315–323, 2011. 106
- [210] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964. 107
- [211] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Neural Information Processing Systems (NIPS)*, pp. 950–957, 1992. 108
- [212] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 427–436, 2015. 109
- [213] J. Su, D. V. Vargas, and S. Kouichi, "One pixel attack for fooling deep neural networks," *arXiv preprint arXiv:1710.08864*, 2017. 109

BIBLIOGRAPHY

- [214] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *International Conference on Machine Learning (ICML)*, 2015. 110
- [215] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” in *Neural Networks: Tricks of the Trade*, pp. 9–50, Springer, 1998. 110
- [216] A. Spek and T. Drummond, “A compact parametric solution to depth sensor calibration,” 2017. 112
- [217] A. Belhedi, A. Bartoli, V. Gay-Bellile, S. Bourgeois, P. Sayd, and K. Hamrouni, “Depth correction for depth cameras from planarity,” *British Machine Vision Conference (BMVC)*, 2012. 112
- [218] M. Lindner, I. Schiller, A. Kolb, and R. Koch, “Time-of-flight sensor calibration for accurate range sensing,” *Computer Vision and Image Understanding*, vol. 114, no. 12, pp. 1318–1328, 2010. 112, 126, 127
- [219] J. Smisek, M. Jancosek, and T. Pajdla, “3D with Kinect,” *IEEE International Conference on Computer Vision (ICCV)*, no. November 2011, pp. 1154–1160, 2011. 112
- [220] S. Helgason, *Differential Geometry, Lie Groups, and Symmetric Spaces*, vol. 80. Academic press, 1979. 116
- [221] A. Spek and T. Drummond, “A Fast Method For Computing Principal Curvatures From Range Images,” in *Australasian Conference on Robotics and Automation (ACRA)*, 2015. 131
- [222] J. Lee, S. Kim, and S.-J. Kim, “Mesh Segmentation Based on Curvatures Using the GPU,” *Multimedia Tools and Applications*, June 2014. 132, 154, 179, 180, 181
- [223] M. Pauly, M. Gross, and L. P. Kobbelt, “Efficient Simplification of Point-Sampled Surfaces,” in *Conference on Visualization*, no. Section 4, pp. 163–170, 2002. 133
- [224] W. Cheney and D. Kincaid, *Linear Algebra: Theory and Applications*. Sudbury, 2009. 136

BIBLIOGRAPHY

- [225] M. Sanchez-Fibla, A. Duff, and P. F. Verschure, “A Sensorimotor Account of Visual and Tactile Integration for Object Categorization and Grasping,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 107–112, IEEE, May 2013. 154
- [226] N. Corporation, “Cuda programming guide: Warp shuffle functions.” 163
- [227] B. Espiau, F. Chaumette, and P. Rives, “A new approach to visual servoing in robotics,” *IEEE Transactions on Robotics and Automation (TRA)*, vol. 8, no. 3, pp. 313–326, 2002. 178
- [228] X. Wang, D. Fouhey, and A. Gupta, “Designing deep networks for surface normal estimation,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 539–547, IEEE, 2015. 181, 182, 191
- [229] D. F. Fouhey, A. Gupta, and M. Hebert, “Unfolding an Indoor Origami World,” *European Conference on Computer Vision (ECCV)*, 2014. 181, 191
- [230] B. Z. Lăăzubor Ladický and M. Pollefeys, “Discriminatively trained dense surface normal estimation,” pp. 468–484, Springer, 2014. 181, 191
- [231] R. Caruana, “Multitask learning,” in *Learning to Learn*, pp. 95–133, Springer, 1998. 182
- [232] X. Gibert, V. M. Patel, and R. Chellappa, “Deep multitask learning for railway track inspection,” *IEEE Transactions on Intelligent Transportation Systems (TITS)*, vol. 18, no. 1, pp. 153–164, 2017. 182
- [233] 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,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 38, no. 6, pp. 1070–1083, 2016. 182
- [234] O. Chapelle, P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng, “Boosted multi-task learning,” *Machine learning*, vol. 85, no. 1-2, pp. 149–173, 2011. 182

BIBLIOGRAPHY

- [235] X. Wang, C. Zhang, and Z. Zhang, “Boosted multi-task learning for face verification with applications to web image and video search,” in *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 142–149, IEEE, 2009. 182
- [236] Z. Li and D. Hoiem, “Learning without forgetting,” pp. 614–629, Springer, 2016. 182
- [237] Y. Nesterov, “A method of solving a convex programming problem with convergence rate $O(1/k^2)$,” in *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983. 186
- [238] 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)*, vol. 115, no. 3, pp. 211–252, 2015. 186, 202, 210
- [239] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network,” in *IEEE Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017. 199
- [240] 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,” *IEEE International Conference on Computer Vision (ICCV)*, 2015. 201, 204, 243
- [241] 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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017. 201, 204, 212, 220, 243
- [242] A. Ranjan and M. J. Black, “Optical flow estimation using a spatial pyramid network,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017. 201
- [243] 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. 201

BIBLIOGRAPHY

- [244] A. S. Wannenwetsch, M. Keuper, and S. Roth, “Probflow: Joint optical flow and uncertainty estimation,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 1182–1191, IEEE, 2017. 201
- [245] P. Agrawal, J. Carreira, and J. Malik, “Learning to see by moving,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 37–45, IEEE, 2015. 202
- [246] D. Jayaraman and K. Grauman, “Learning image representations tied to ego-motion,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 1413–1421, IEEE, 2015. 202
- [247] 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 *IEEE International Conference on Computer Vision (ICCV)*, 2017. 202
- [248] A. Kendall and R. Cipolla, “Modelling uncertainty in deep learning for camera relocalization,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4762–4769, IEEE, 2016. 202
- [249] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014. 210, 232
- [250] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, “Epicflow: Edge-preserving interpolation of correspondences for optical flow,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1164–1172, IEEE, 2015. 212, 220
- [251] A. Levin, D. Lischinski, and Y. Weiss, “Colorization using optimization,” in *ACM Transactions on Graphics (ToG)*, vol. 23, pp. 689–694, 2004. 214
- [252] G. Elbaz, T. Avraham, and A. Fischer, “3d point cloud registration for localization using a deep neural network auto-encoder,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2472–2481, IEEE, 2017. 222
- [253] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, “Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes,” *arXiv preprint arXiv:1711.00199*, 2017. 222

BIBLIOGRAPHY

- [254] K. Tateno, F. Tombari, I. Laina, and N. Navab, “Cnn-slam: Real-time dense monocular slam with learned depth prediction,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6243–6252, 2017. 226, 240, 244
- [255] 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 (TITS)*, vol. 19, no. 1, pp. 263–272, 2018. 227
- [256] NVIDIA, “Nvidia tensorrt - programmable inference accelerator,” 2018. 228
- [257] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” *arXiv preprint arXiv:1511.07122*, 2015. 228
- [258] N. Gelfand, L. Ikemoto, S. Rusinkiewicz, and M. Levoy, “Geometrically stable sampling for the ICP algorithm,” *International Conference on 3-D Digital Imaging and Modeling (3DIM)*, vol. 2003-January, pp. 260–267, 2003. 242
- [259] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison, “Codeslam-learning a compact, optimisable representation for dense visual slam,” *arXiv preprint arXiv:1804.00874*, 2018. 243
- [260] “Learning to align semantic segmentation and 2.5 d maps for geolocalization,” 243
- [261] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, “Deep mutual learning,” *arXiv preprint arXiv:1706.00384*, 2017. 243