

Robust Lifelong Visual Navigation and Mapping



Geoffrey Pascoe
Worcester College
University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity Term, 2017

Yellow and Black

Acknowledgements

Personal

I'd like to begin by thanking the Richmond Football Club for the inspiration they have provided in the final run towards the finish line of this thesis. Sometimes fairytales do come true.

Thanks must go to my supervisors Paul Newman and Will Maddern for their support, guidance, and wisdom throughout the course of my DPhil. This work would not have happened without them.

My thanks are also due to all the friends I have made in Oxford: Ben Hunn, Amanda Rojek, Katie Stone, Pat Bateman, Dave Sherwood, Matt Gadd, Corina Gurau, Jeff Hawke, Jack Valmadre, Ollie Bartlett, Freya Shearer, Justin Ball, Sarah Noyce, Sarah Barber, Katie Light, and too many others to name. Also to the many members of the Oxford University Australian Rules Football Club — a wonderful group of people with whom I've had some great times.

To Christie Noble, thank you for putting up with me these last two years. Oxford wouldn't have been the same if I hadn't met you.

Finally, I simply can't adequately express my gratitude to my incredible parents, Ron and Nina Pascoe. Having a child move to the opposite side of the planet can not be easy, but as has been the case throughout my life, they have never wanted anything but the best for me. The amount of love and support they have always given me is truly superhuman, and I will be forever in their debt.

Institutional

My presence in Oxford was due to the support provided by the Rhodes Trust — I can't thank them enough for the opportunity, and the experiences I've had here.

Thank you also to all the wonderful people at Worcester College, from admin staff, to academics, to fellow students. Worcester really made Oxford feel like home, and I will always remember it fondly.

Abstract

Robust Lifelong Visual Navigation and Mapping

Geoffrey Pascoe

Doctor of Philosophy

Worcester College

Trinity Term, 2017

The ability to precisely determine one's location in within the world (localisation) is a key requirement for any robot wishing to navigate through the world. For long-term operation, such a localisation system must be robust to changes in the environment, both short term (eg. traffic, weather) and long term (eg. seasons).

This thesis presents two methods for performing such localisation using cameras — small, cheap, lightweight sensors that are universally available. Whilst many image-based localisation systems have been proposed in the past, they generally rely on either feature matching, which fails under many degradations such as motion blur, or on photometric consistency, which fails under changing illumination.

The methods we propose here directly align images with a dense prior map. The first method uses maps synthesised from a combination of LIDAR scanners to generate geometry and cameras to generate appearance, whilst the second uses vision for both mapping and localisation. Both make use of an information-theoretic metric, Normalised Information Distance (NID), for image alignment, relaxing the appearance constancy assumption inherent in photometric methods. Our methods require significant computational resources, but through the use of commodity GPUs, we are able to run them at a rate of 8-10Hz. Our GPU implementations make use of low level OpenGL, enabling compatibility across almost any GPU hardware.

We also present a method for calibrating multi-sensor systems, enabling the joint use of cameras and LIDAR for mapping.

Through experiments on both synthetic data and real-world data from over 100km of driving outdoors, we demonstrate the robustness of our localisation system to large variations in appearance. Comparisons with state-of-the-art feature-based and direct methods show that ours is significantly more robust, whilst maintaining similar precision.

Contents

List of Figures	viii
List of Tables	x
List of Abbreviations	xi
1 Introduction	1
1.1 Motivation	1
1.2 Contributions	4
1.3 Structure	4
1.4 Publications	5
2 Background	6
2.1 Introduction	6
2.2 Related Work	8
2.2.1 LIDAR-based Localisation	9
2.2.2 Feature-based Localisation	10
2.2.3 Direct Tracking and Localisation	12
2.2.4 CNN-based Localisation	14
2.2.5 Sequence-based Localisation	15
2.2.6 Visual Servoing	15
2.2.7 Mutual Information	16
2.3 OpenGL for Computation	16
2.3.1 OpenGL	17
2.3.2 Alternative GPGPU Libraries	18
2.3.3 The OpenGL Pipeline	18
2.3.4 Computation with OpenGL	21
2.3.5 Conclusion	23
2.4 Transformations in $\mathbb{SE}(3)$	23
2.4.1 Matrix Representation	24
2.4.2 Lie Algebra	24

3	LIDAR Mapping	26
3.1	Introduction	26
3.2	Sensor Configuration	27
3.3	Geometry	29
3.3.1	Local Geometry	30
3.3.2	Mesh Generation	32
3.4	Texturing	34
3.4.1	Mesh Reprojection	34
3.4.2	Texture Selection	36
3.5	Conclusion	41
4	Robust, Dense, Visual Localisation	42
4.1	Introduction	43
4.2	Problem Formulation	45
4.3	Metrics for Image Comparison	46
4.3.1	Photoconsistency	47
4.3.2	Normalised Information Distance	49
4.3.3	Void Filling	60
4.3.4	Mesh Distortion	62
4.3.5	Image Blurring	64
4.4	Optimisation	64
4.4.1	Gradient Descent	66
4.4.2	Newton's Method	67
4.4.3	Gauss-Newton Method	67
4.4.4	Levenberg-Marquardt Method	68
4.4.5	BFGS Method	68
4.5	Evaluation	71
4.6	Localisation	72
4.6.1	Outlier Rejection	73
4.6.2	Implementation Details	74
4.6.3	Evaluation	76
4.6.4	Synthetic Data	77
4.6.5	Real Data	81
4.6.6	Computation Time	88
4.7	Conclusion	89

5	Extrinsic Calibration of Sensor Sets	91
5.1	Introduction	91
5.2	NID-based Extrinsic Calibration for Sensor Sets	93
5.3	Chequerboard-based Calibration	94
5.3.1	Target Detection	95
5.3.2	Extrinsic Calibration	102
5.4	Evaluation	107
5.5	Conclusion	113
6	Vision-only Mapping and Localisation	114
6.1	Introduction	114
6.2	System Overview	116
6.3	NID Tracking	116
6.4	Variations on NID	119
6.4.1	Spatial vs Appearance Weight	119
6.4.2	Depth Uncertainty	121
6.4.3	Histopyramids	122
6.4.4	Overlap Barrier	126
6.5	Depth Estimation	128
6.5.1	Photometric Depth Estimation	129
6.5.2	NID Depth Estimation	130
6.5.3	Initialisation	132
6.6	Evaluation	133
6.6.1	Synthetic Data	133
6.6.2	Real Data	139
6.7	Conclusion	147
7	Conclusion	151
7.1	Summary	151
7.2	Contributions	153
7.3	Future Work	155
7.4	Concluding Remark	156
	References	157

List of Figures

3.1	RobotCar sensors	28
3.2	Example images from cameras	29
3.3	Example pointcloud	32
3.4	Mesh stitching from pushbroom pointcloud	33
3.5	Components of OpenGL MVP matrix	35
3.6	Centroid weighting function for triangle selection	37
3.7	Textures for a slice of mesh	38
3.8	Textured Mesh	40
4.1	Examples of appearance variation seen in an outdoor environment .	48
4.2	Building joint histograms	50
4.3	Robustness of NID to changing appearance	52
4.4	Robustness of NID to image degradations	53
4.5	Histograms as a continuous function	56
4.6	Histogram derivatives	58
4.7	Histogram Jacobian	59
4.8	Derivative images generated during rendering	61
4.9	Filling of gaps in prior	62
4.10	Image with wide-angle lens	63
4.11	Effect of distorting projected mesh	65
4.12	Effect of blurring on derivatives	66
4.13	Example image and render from the Handa Living Room dataset .	78
4.14	Route used for outdoor localisation evaluation	82
4.15	Camera images from each of the 12 traversals used	83
4.16	Error histograms	85
4.17	Localisation failures due to poor camera exposure	86
4.18	Localisation failures due to dynamic obstacles	87
4.19	Night time image	88
4.20	Computation time for localisation	89
5.1	NID calibration cost function	94
5.2	Camera image of chequerboard target used for calibration.	96

5.3	Chequerboard detection in a camera image	97
5.4	Examples of LIDAR calibration target detections	100
5.5	Ratios for target detection	101
5.6	Kernel used for 2D centroid cost function	106
5.7	Huber kernel loss	107
5.8	Vicon setup	109
5.9	Sensor locations on Land Rover	110
5.10	Pointcloud projected into image	112
6.1	NID-SLAM pipeline	117
6.2	Image warping	118
6.3	Spatial vs Appearance Weighting	120
6.4	Spatial Weighting at Edges	122
6.5	Multi-resolution histogram representation	124
6.6	NID vs offset for image pyramids and histopyramids	127
6.7	Overlap barrier	128
6.8	Example images from the New Tsukuba dataset	134
6.9	Sample indoor localisations	137
6.10	Example outdoor images	141
6.11	Sample outdoor localisations	144
6.12	Example images for convergence experiment	148
6.13	Success rates for different starting offsets	149

List of Tables

4.1	RMS errors and success probabilities for different optimisers	72
4.2	Mean computation time for histograms	76
4.3	RMS errors from an 882-image trajectory in a synthetic dataset . .	79
4.4	RMS errors from open loop tracking	81
4.5	Traversals from Oxford RobotCar dataset	84
4.6	RMS errors for outdoor datasets	84
5.1	Sensors used for calibration	110
5.2	Calibration errors	111
6.1	Tracking results on New Tsukuba dataset	136
6.2	Depth map refinement on the New Tsukuba dataset	138
6.3	Traversals from RobotCar Dataset for outdoor comparison	140
6.4	Outdoor tracking success rates	142
6.5	NID-SLAM outdoor evaluation	145

List of Abbreviations

1D, 2D, 3D	One- or two-dimensional, referring in this thesis to spatial dimensions in an image.
BFGS	Broyden-Fletcher-Goldfarb-Shanno
CAD	Computer Aided Design
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
DOF	Degrees of Freedom
GLSL	OpenGL Shader Language
GPGPU	General-Purpose computation on GPU
GPU	Graphics Processing Unit
LM	Levenberg-Marquardt
LSD-SLAM	Large Scale Direct SLAM
OpenCL	Open Compute Library
OpenGL	Open Graphics Library
MI	Mutual Information
MVP	Model View Projection
NID	Normalised Information Distance
RANSAC	RANdom SAmple Consensus
$\mathbf{sim}(3)$	3-dimensional similarity transform
$\mathbf{SE}(2)$	Special Euclidean Group (2)
$\mathbf{SE}(3)$	Special Euclidean Group (3)
SfM	Structure from Motion
$\mathbf{SO}(2)$	Special Orthogonal Group (2)
$\mathbf{SO}(3)$	Special Orthogonal Group (3)
SLAM	Simultaneous Localisation and Mapping
VO	Visual Odometry

1

Introduction

Contents

1.1	Motivation	1
1.2	Contributions	4
1.3	Structure	4
1.4	Publications	5

1.1 Motivation

This thesis deals with the problem of localisation using camera images in changing outdoor environments. The key ability to precisely determine one's position unlocks further technologies in a wide variety of application domains, including augmented or mixed reality [1], mobile robotics [2], micro aerial vehicles [3], and autonomous driving [4].

Our primary application of interest is in navigation for autonomous vehicles. In this field in particular, many existing localisation systems make use of 3D light detection and ranging (LIDAR) scanners [4–7], as these provide precise 3D information about the surrounding environment, enabling reliable localisation. Whilst such sensors have recently seen significant reductions in price, they still add

a non-negligible cost burden when compared to cameras. Furthermore, for other platforms with size or weight constraints, the addition of a large 3D LIDAR scanner will often be impractical, and cameras will in many cases be necessary regardless. We are therefore interested in developing a robust camera-based localisation system. Our primary application domain of interest is that of autonomous driving, however the problems and innovations discussed in this thesis are generally applicable to many other contexts.

A key challenge when using vision-based localisation systems in dynamic outdoor environments is the variability in scene appearance. Even without dynamic objects such as traffic, variation in lighting, weather, and seasons means that the appearance of the ‘static’ background structure can change significantly over time. Any system that is to be used in such an environment thus needs to be robust to these sorts of variations.

Visual localisation is a widely studied problem, and many image-based systems have been developed [8–12]. Most of these systems make use of *feature-based* or *indirect* methods [12]. These methods rely upon the detection and matching of salient feature points within images, whilst discarding the rest of the image. Robust features can provide some invariance to appearance change, but matching is not robust to large changes in lighting, weather conditions and longer term seasonal changes seen outdoors [13, 14].

More recently, a number of *direct* tracking and localisation systems have been developed [15–18]. These methods don’t attempt to match feature points across different images, instead optimising a cost function (typically photoconsistency) across the whole image. Whilst this provides greater robustness to viewpoint change and blur [15], it does introduce a much stronger appearance constancy requirement — when localising by directly comparing pixel values, it is vital that the same points always have exactly the same appearance. As a result, to date direct methods have

mostly been confined to tracking over a single session or operation in an indoor environment with constant artificial lighting.

A key advantage of feature-based methods is that, by using only a sparse set of points in each image, the computational requirements for localisation can be quite low. Even direct photometric methods, whilst more computationally costly than feature-based methods, are relatively efficient and easily parallelisable. The computational constraints on many platforms have been greatly loosened in recent years, however, with the sharp increase in power reductions in price of graphics processing units (GPUs). This has enabled a proliferation of general purpose computation on GPUs (GPGPU). When used judiciously, GPGPU brings unprecedented power to desktop-sized machines, allowing more complex computation than was previously possible for many tasks.

In this thesis we develop and explore a system for performing localisation using a direct image-based method. We present a monocular method, first for pure localisation against LIDAR-based maps, then develop it into a SLAM system capable of performing mapping as well. We make use of an information theoretic metric, known as normalised information distance (NID), to relax the appearance constancy assumption, and enable operation across varying conditions outdoors. Implementation on a GPU enables efficient, fast operation despite the high computational requirements. We use the low-level but universally available OpenGL framework for our GPU implementation for compatibility across the widest possible range of hardware.

We evaluate the localisation performance using a combination of synthetically generated and real data. Through this evaluation we show high precision in localisation, whilst obtaining significant robustness to both artificial and real variation in scene appearance.

1.2 Contributions

The primary contributions of this thesis are:

- A dense, direct method for performing $\mathbb{SE}(3)$ camera localisation under varying appearance conditions.
- A method for building 3D maps using 2D LIDAR and camera images.
- An extension of NID-based localisation into a SLAM framework, in which not only localisation, but also map maintenance can be performed using cameras across long time scales.
- GPU implementations of the above, to enable operation in the context of an autonomous vehicle.
- A system for obtaining extrinsic calibration of a multi-sensor platform.

1.3 Structure

The remainder of the thesis is structured as follows.

Chapter 2 presents a review of existing literature, introducing important concepts and exploring previous work on localisation generally, and visual localisation more specifically. Also presented is some background information necessary for comprehending the remainder of the thesis.

Chapter 3 presents a method for producing 3D maps from a push-broom LIDAR scanner and cameras. These are maps designed to be produced by specialist survey vehicles outfitted with LIDAR, but nevertheless using 2D LIDAR scanners that are still much lower cost than their 3D cousins.

Chapter 4 presents a robust direct localisation system using NID to align camera images with the maps from Chapter 3. We evaluate the performance of the system on a synthetic indoor dataset, with artificial image degradation introduced to test robustness. We also demonstrate the performance of the system on over 100km of driving in Oxford over a period of 6 months.

Chapter 5 presents a method for multi-sensor extrinsic calibration. This calibration system enables cooperative use of multiple sensors, for applications such as the map building from Chapter 3. We evaluate the calibrations through the use of a commercial motion capture system.

Chapter 6 presents a visual SLAM method, based on NID, which enables long term use and maintenance of maps using cameras. This removes the need for survey vehicles carrying LIDAR scanners, allowing every vehicle equipped with cameras to perform both localisation and map updates.

1.4 Publications

The work detailed in Chapter 3 was first presented at ICRA 2015 [19]. Parts of this mapping pipeline were also used for the system described in [6]. Chapter 4 includes material from publications at BMVC 2015 [20] and ICCV 2015 [21]. The work in Chapter 6 was published at CVPR 2017 [22]. A small amount of the content in Chapter 5 is present in [21], but the vast majority of the calibration material is as yet unpublished. The outdoors data used was published as part of the Oxford RobotCar Dataset [23].

2

Background

Contents

2.1	Introduction	6
2.2	Related Work	8
2.2.1	LIDAR-based Localisation	9
2.2.2	Feature-based Localisation	10
2.2.3	Direct Tracking and Localisation	12
2.2.4	CNN-based Localisation	14
2.2.5	Sequence-based Localisation	15
2.2.6	Visual Servoing	15
2.2.7	Mutual Information	16
2.3	OpenGL for Computation	16
2.3.1	OpenGL	17
2.3.2	Alternative GPGPU Libraries	18
2.3.3	The OpenGL Pipeline	18
2.3.4	Computation with OpenGL	21
2.3.5	Conclusion	23
2.4	Transformations in $\mathbb{SE}(3)$	23
2.4.1	Matrix Representation	24
2.4.2	Lie Algebra	24

2.1 Introduction

The core problem addressed in this thesis is that of allowing a robot to precisely determine its position and orientation (pose) during operation, known as localisation.

This is a core competency in a range of applications, from mobile robotics [2, 3] to autonomous driving [4] or virtual and augmented reality [1].

Whilst the Global Positioning System (GPS) and other Global Navigation Satellite Systems (GNSS) aim to provide accurate localisation on a global scale, they fall short of the capabilities required for many mobile robotics applications [24]. Particularly in urban environments, multipath propagation of GNSS signals significantly reduces the available precision — GNSS is usable for an approximate location estimate, but not for feeding into a control or perception system.

Many early localisation systems for autonomous operation made use of LIDAR sensors [4]. These sensors provide highly precise information about the surrounding environment, however they are relatively high in both cost and weight, making them problematic for use in many applications. We are therefore interested in exploring localisation using camera sensors — cheap, small, and universally available sensors that nevertheless provide relatively high-resolution information.

In recent years, an explosion in computing power, particularly in graphics processing units (GPUs), has enabled methods that were previously infeasible (eg. [16]). The methods presented in this thesis make heavy use of this computing power, in order to perform whole-image localisation at a rate that is usable for robotics applications.

This chapter presents prior work on localisation and mapping, with both LIDAR and image sensors, introducing the issues which are addressed in this thesis. We also introduce the OpenGL toolchain, which we make use of later in the thesis, in order to bring the power of GPU computation to bear on the localisation problem. Finally, we introduce the reader to $\text{SE}(3)$ transforms, used to represent sensor, vehicle, and object poses throughout this thesis.

2.2 Related Work

The work in this thesis is related primarily to the fields of visual odometry (VO), localisation, simultaneous localisation and mapping (SLAM) and structure from motion (SfM). The core competencies in these four areas are similar, with the primary differences being the requirements of the application, and the prior information available [25]:

SLAM is the task of determining one’s position whilst also building a map of one’s surroundings — in a SLAM application, no prior map of the environment is available.

Visual Odometry or **tracking** refers to the estimation of one’s trajectory over time. No attempt is made to obtain consistent poses in a global frame — odometry systems can be considered accurate only in a *local* frame, and can drift over time.

SfM refers to offline batch processing of data (visual or otherwise) to determine both a map and sensor poses. Unlike SLAM, SfM methods generally have all data available simultaneously.

Localisation refers to determining one’s position within a map that already exists. That map can come from a SLAM system, an SfM system, or any other method.

In common to all of these systems is the requirement to determine the pose of a sensor using a map, whether that map is fixed or under construction.

One of the challenges when using localisation for long-term autonomy is the significant variation in appearance observed outdoors. High frequency changes such as traffic, daylight and weather, along with longer-term variation such as seasons and construction works can cause the same place to vary significantly in appearance over time [23].

2.2.1 LIDAR-based Localisation

Several state-of-the-art autonomous vehicle systems make use of LIDAR for precise localisation. An early such system was presented in [4], and further refined in [5]. In this system, a map is first built using 3D Velodyne LIDAR [26]. A GraphSLAM-based approach [27] is used with inertial, LIDAR, and GPS data to perform an offline optimisation which places the LIDAR scans in a single, consistent global frame. The final map consists of a probabilistic 2D grid, in which the ground plane is discretised, and each cell contains a mean and variance of reflectivity. At localisation time, a GPS/IMU system is used for high-frequency pose updates, whilst global corrections are applied by computing the joint probability of an observed 3D LIDAR scan with the map. The 2D nature of the map means that the system is limited to $\text{SE}(2)$ localisation.

A similar system is presented by [7], except the maps used are more flexible than a grid on the ground plane. As well as representing reflectivity, the maps also contain information about the height of points in each cell. To allow for structures such as overhanging trees or buildings, the height is represented by a multi-modal Gaussian distribution. The use of height allows localisation based on structure, which is significantly more robust to seasonal changes such as snow. Localisation is performed by maximising the likelihood of a given scan being produced from the underlying map. A 2D grid search with intelligent pruning is used to find the maximum, and maps are rasterised using a GPU to allow real-time operation. Similar to [5], only $\text{SE}(2)$ localisation is performed.

The system presented by [6] uses only a 2D LIDAR, in conjunction with vehicle odometry, in order to build a 3D model of the environment. Similar to [7], this system makes use of both height and reflectance information in a 2D grid. Unlike [5] and [7], however, in this case no globally optimised map is produced. Instead, localisation is performed by comparing ‘swathes’ of LIDAR data in a local frame.

These swathes are produced at run time for just the area around the vehicle. Once again, a 2D grid search is performed, limiting the system to $\mathbb{SE}(2)$ localisation, and a GPU is used to rasterise the swathes.

2.2.2 Feature-based Localisation

Most image-based SLAM or localisation systems make use of *indirect* or *feature-based* methods [8–11, 28–32]. These methods rely on the detection and matching of feature points in the images, and localise by minimising the reprojection error of those features.

The core requirement of indirect methods is being able to identify and match the same points across different images. One of the most widely used features is the scale invariant feature transform (SIFT) [33]. The speeded up robust features (SURF) descriptor [34] was designed to reduce the high computational requirements of SIFT, whilst the binary robust independent elementary features (BRIEF) [35] descriptor uses binary strings for both efficiency and robustness. More recently, ORB features [36] adds rotational invariance to BRIEF, producing robustness similar to SIFT, whilst being faster to compute than SIFT or SURF, and similar to BRIEF.

In order to ignore erroneous matches between features representing different points, indirect methods often make use of a random sample consensus (RANSAC) step [37]. When applied to the localisation problem, RANSAC involves making several localisation estimates with random subsets of all matched feature points, and choosing the subset which produces the lowest reprojection error [8]. In this way, subsets that include erroneous matches will be discarded.

By using sparse feature points, indirect methods lower the computational requirements for localisation, as opposed to minimising a function across all pixels in the image. In MonoSLAM [9], each new frame is tracked using sparse feature points, and past poses are marginalised using a Kalman filter. In contrast, the parallel tracking and mapping (PTAM) method [30] doesn't marginalise previous

poses — instead, it performs continuous bundle adjustments on keyframes, in order to improve global consistency. This optimisation-based approach has been shown to provide greater accuracy than filtering [38]. To enable real-time operation, PTAM runs the tracking front end on one thread, whilst the global bundle adjustment runs in the background on a separate thread.

Many approaches such as [39] also integrate an inertial measurement unit (IMU) with the vision system, in order to provide accurate, robust pose estimates. In [39], this is combined with a bounded window of keyframes used in a bundle-adjustment, providing increased global accuracy with bounded computation time.

Large-scale localisation against a map produced by SfM is demonstrated in [8]. A subset of the feature points from the map, determined by the previous computed pose, is used, and matched to the live image. The reprojection error between these matched feature points is then iteratively minimised. To enable global real-time localisation against a large prior map on a mobile device, [11] compress both the map (by removing superfluous features) and the feature descriptors themselves. They combine a high-framerate visual-inertial tracker with a slower global localisation to provide real-time global pose estimates.

The recent ORB-SLAM system [28] makes use of ORB features in a PTAM-like framework, and demonstrated state-of-the-art accuracy for monocular SLAM. The ORB features give the system some invariance to illumination change, though cross-seasonal performance is not demonstrated.

Whilst local image features can provide some robustness to appearance change, they lose efficacy over large periods of time [13], particularly with seasonal change [14]. Furthermore, local features tend not to be robust to large changes in viewpoint [40].

In order to enable localisation under significant appearance variation, [41] proposed an ‘experience-based’ paradigm. Under this framework, rather than attempting to develop features that are invariant to all environmental change,

several versions of the map (*experiences*) are maintained. An agent needs only to be able to match against one of those experiences to localise. Whilst this does provide much-increased robustness, it also significantly increases both storage and processing requirements. An alternative is proposed by [42], in which the appearance of a scene is first predicted under different conditions, before the prediction is matched to a database or map created under those conditions. This is shown to improve the recall for topological place recognition, although it is not demonstrated with metric localisation.

2.2.3 Direct Tracking and Localisation

In contrast to feature-based or indirect tracking methods, in recent years, a number of *direct* tracking methods have been developed [15–18]. They differ from feature-based methods in that they do not attempt to minimise reprojection error between feature points matched across images. Instead, they directly minimise a cost function (often photometric error) on the pixels [43].

Direct methods have been shown to provide higher accuracy, as unlike feature-based methods they don’t discard most of the image [44], and they are more robust to viewpoint changes and blurred images (for example, due to motion blur) [15]. Furthermore, direct methods perform better in environments in which there are few distinctive features, such as textureless areas or high-frequency repeating textures [45].

The system presented by Wolcott et. al [16] localises via direct alignment of a camera image with a LIDAR-borne map. The map takes the form of a mesh produced via a Velodyne 3D LIDAR scanner, and coloured with the infrared reflectance of each range measurement. To align a camera image with the map, virtual camera images of the mesh are rendered at a grid of hypothesis locations, and compared to the live image. Only the ground plane is used for localisation, as the authors found that including other structure didn’t improve localisation in their environment,

whilst impacting computation time. A GPU is used for rendering, in order to enable real-time operation of the system. As the two images come from different modalities (infrared reflectance vs. RGB colour), the comparison is performed using a normalised mutual information measure. The grid search approach used is too computationally expensive to localise in the six-dimensional parameter space of $\mathbb{SE}(3)$ — this system is only capable of $\mathbb{SE}(2)$ localisation, assuming a flat road surface.

Whilst many direct localisation or tracking approaches are fully dense (using every pixel) [15, 16], both semi-dense [17] and sparse [44] methods have also been developed. In a semi-dense approach, only salient regions, with information useful for the localisation task, are used. In [17], pixels are selected based on the gradients in the local neighbourhood. Only those pixels with a significant gradient in their neighbourhood are used — intuitively this means that large areas of constant texture, which give little information for image alignment, are not used. Additionally, pixels for which a reliable depth estimate is not available (determined by the variance in the inverse depth estimate) are also discarded.

The direct sparse odometry approach presented in [44] only uses points from areas of sufficient gradient, similar to [17], but further subsamples those points, so that only a small number of pixels end up being used. This enables a sparse system, thus reducing the computation required for residual computation, without requiring the detection or matching of feature points. It does, however, strengthen the photoconsistency assumption inherent in direct photometric methods, and requires a precise photometric calibration of the camera being used.

One of the biggest challenges with long-term operation of direct methods is the underlying assumption in most systems of consistent lighting conditions. This is due to the fact that most direct methods align images by minimising photometric error — this requires pixel intensities (or colours) to always be the same from any viewpoint and at any time. The LAPS method [18] relaxes this assumption by instead minimising an information-theoretic cost between a camera image and a

sparse, LIDAR-derived pointcloud. Similarly, in [46], a mutual information cost function was used to relax the brightness constancy assumption in a direct visual odometry system. A different approach is taken by [47], in which binary feature descriptors are matched directly — descriptors are computed for every pixel, and the Hamming distance minimised, with no feature-matching step. This provides increased robustness to local lighting variation in a visual odometry framework, but does not address global changes in scene appearance over long periods of time.

2.2.4 CNN-based Localisation

In recent years, the proliferation of high-powered GPUs has enabled convolutional neural networks (CNNs) to be developed at a much larger scale than was previously feasible. These CNNs have proven to be highly capable at many computer vision tasks [48–50].

CNNs have been applied to both pure localisation [51] and SLAM [52]. In PoseNet [51], a neural network is trained to regress a global $\mathbb{SE}(3)$ camera pose directly, using camera poses from a structure from motion (SfM) model as training data. The model was able to regress camera poses successfully across large changes in scene appearance, however the accuracy (within approximately 2m and 3°) was not high enough for many applications. This was improved upon by using reprojection error of the SfM model as the loss function during training, rather than the pose error [53]. Whilst this improved localisation accuracy, it still produces errors on the order of 1m and 1° in small environments.

In addition to pure localisation, CNNs have also been applied to SLAM. In [52], a CNN is used to predict depth values for monocular images. These depth values are then used and refined in a direct photometric matching scheme based on [17]. This produces denser pointclouds than the temporal stereo approach from [17], as depth can be estimated for more pixels, as well as enabling scale-aware SLAM with only a monocular camera.

2.2.5 Sequence-based Localisation

The methods discussed thus far all use a single image at a time for localisation. On a robotic platform, however, one has access to a video stream, not just individual images. SeqSLAM performs matches across sequences of images, and has been shown to significantly increase robustness to appearance change and image degradation [54]. Specifically, successful localisation was achieved across different days, with highly blurred images, or even with image resolutions of just one pixel — all cases that are extremely difficult for single-image approaches. SeqSLAM, however, is very dependent on viewpoint, making it unsuitable for tasks other than following the exact same trajectory. Furthermore, it provides only topological place recognition, not metric localisation.

In [55], a recurrent neural network (RNN) is used in a framework similar to PoseNet [51] to regress pose from a video stream. The use of a recurrent network allows the system to take advantage of a sequence of images, rather than just a single image at a time, and localisation error is significantly reduced. Whilst still generally producing errors on the order of metres, and so not appropriate for many navigation tasks, the sequence does produce more accurate pose estimates and smoother trajectories whilst having little effect on runtime.

2.2.6 Visual Servoing

An alternative to localisation in some scenarios, which nevertheless faces some of the same issues, is visual servoing. Visual servoing uses visual information directly to control a robot, without explicitly performing localisation [56]. Instead, the current image observed by the robot is compared to an image of the robot’s desired location, and a control law built to move the robot towards that view. In this case, the ‘map’ consists simply of a series of images taken along the desired route of the robot.

Similar to localisation and SLAM, both feature-based [57] and direct [58] methods have been developed. In [56, 59], the authors compare images using mutual

information, to handle the possibility of illumination change between capturing the ‘target’ images, and following the path. A controller guides the robot in the direction that will increase the mutual information between the target and observed image. This is similar in concept to localisation by optimising a virtual camera to align with an image as in [16, 19], but instead of a virtual camera, the camera being moved is attached to a real robot [56].

2.2.7 Mutual Information

As noted previously, several visual servoing and localisation methods have made use of mutual information based measures to perform robust image comparison [16, 18, 56]. We will formally define mutual information when we make use of it later in the thesis.

Mutual information was first defined by Shannon [60], and provides a measure of the common information carried by two signals. Its first use in vision was in the field of medical imaging, where it was used to align images from different sensing modalities [61, 62]. Whilst mutual information has since been used in many clinical tasks [63], it carries a high computational cost, and thus has not been suitable for real-time robotics applications.

Recently, however, with the advent of powerful GPUs, mutual information based methods have made their way into the robotics literature, due to their robustness to changes in appearance. As noted previously, they have been used for both visual servoing [56, 59] and localisation [16, 18] tasks. These methods have demonstrated the efficacy of mutual information across appearance change, and even across different modalities.

2.3 OpenGL for Computation

The well known Moore’s law states that the number of transistors that can be manufactured in an integrated circuit doubles approximately every two years [64].

For many years, this led to faster and faster processors, however in recent years, the increase in raw computational throughput has slowed or stopped completely. Instead, recent increases in computation have tended to move towards greater parallelism. This has led to a proliferation of multi-core processors.

Bringing even more parallelism than multi-core central processing units (CPUs), however, are graphics processing units (GPUs). These devices have seen significant increases in power in recent years, with the ability to run thousands of parallel threads, whilst also reducing in cost, thus bringing processing power previously only available to supercomputers to commodity desktop computers. The threads on a GPU, however, are not as flexible as those on a CPU, and require careful design to fully exploit the parallelism available.

Many of the algorithms in this thesis make use of the parallelism available in GPUs to bring the runtime of large computations down to a point at which they can be useful in real-world applications. Whilst increasing computing power is likely to reduce the magnitude of this challenge in the future, exploiting large-scale parallelism will remain vital for obtaining acceptable performance. The specific hardware and software choices made are not necessary for successful implementation of our system, however we present our design here both as an efficient method for running our algorithms on commodity hardware, and as a guide for using similar abstractions available on a wide range of hardware platforms.

2.3.1 OpenGL

The Open Graphics Library (OpenGL) [65] is a cross-platform library for generating computer graphics through a GPU. Whilst originally designed for the generation of 3D graphics via a *fixed-function* pipeline, with version 3.0, OpenGL gained further flexibility through the use of *shaders*. The pipeline is still primarily aimed at 3D graphics, however it can also be used as a fully programmable general-purpose computing platform, providing cross-platform GPU acceleration.

2.3.2 Alternative GPGPU Libraries

At the time of writing, there are two major libraries designed explicitly for general purpose computation on GPUs (GPGPU): NVIDIA’s Compute Unified Device Architecture (CUDA) [66] and Khronos group’s Open Compute Library (OpenCL) [67]. Unlike OpenGL, these tools are designed from the ground up for unconstrained, general purpose computation. Both, however, have problems with cross-platform support, which makes their use in robotics applications problematic.

CUDA is strongly tied to NVIDIA devices, which places a constraint on the compute hardware that can be deployed. Conversely, OpenCL lacks support on some NVIDIA devices, such as the Jetson [68], and we found the implementations on devices that do support OpenCL to vary significantly in completeness, performance, and correctness. In contrast to CUDA and OpenCL, OpenGL is supported on virtually every GPU available. Furthermore, with many years of development behind it, it is a solid, reliable standard.

It should be noted that modern implementations of OpenGL have made available a *compute shader* — an extra shader stage designed for general purpose computation. This is a relatively new addition to the standard, however, and there are few drivers available that support it. Hence, the remainder of this section assumes no access to compute shaders for computation.

2.3.3 The OpenGL Pipeline

The OpenGL pipeline consists of a number of stages that successively transform input data into, in general, an image that is drawn to screen (or an off-screen buffer). In modern OpenGL, each of these stages is specified through the use of a *shader* — a small program written in a domain-specific language known as OpenGL Shader Language (GLSL) [69]. GLSL is a C-like language, with built-in primitives for types such as textures, matrices, and vectors. Because it is Turing-complete, GLSL can be used to implement any computation.

Inputs

The OpenGL pipeline takes inputs in three major forms:

Vertex Arrays are general arrays, with a fixed number of values per vertex. These can, for example, be a 3D vector per vertex representing Euclidean coordinates, a 2D vector per vertex representing texture coordinates, or a single value per vertex representing reflectance. Each vertex shader (see below) has access to the values from a single vertex.

Textures are generalised representations of images, in 1, 2, or 3 dimensions. They can also have multiple levels for mipmapping (see below). Unlike with vertex arrays, every stage of the shader pipeline has access to the whole texture, not just a single point.

Uniforms are constant values available to every stage in the pipeline. They are set once at the start of a render and stay constant over the course of that render.

Vertex Shader

The vertex shader is the first stage of the OpenGL pipeline [69]. Its intended purpose is for transforming geometry from *world coordinates* into *normalised device coordinates*. Normalised device coordinates specify a location on the screen (or in a texture), with the centre of the screen being at (0,0), and the top right of the screen at (1,1). The output of the vertex shader has two extra dimensions: a normalised depth value, used for depth testing during rasterisation (see below), and a homogeneous divisor w . All values are divided by w after the vertex or geometry stages are complete — this allows projection using a single matrix multiplication.

Each invocation of the vertex shader takes as input a single vertex, and generates as output a single vertex. Both input and output vertices can contain both geometric and non-geometric information — for example a vertex could have transparency or colour values in addition to its coordinates.

Geometry Shader

The geometry shader performs a very similar role to the vertex shader, being designed for doing further geometric processing after the vertex stage [69]. The primary differences between the geometry shader and the vertex shader are:

- Each invocation of the geometry shader processes a single *primitive*, rather than a single *vertex*. For example, if rendering triangles, there is one vertex shader invocation per vertex, but one geometry shader invocation per *triangle*.
- A geometry shader invocation can generate new geometry — whilst each invocation gets a single primitive, it can output multiple primitives.

These features allow the geometry shader to be used, for example, to generate multiple versions of given geometry — this was used in [6] to render multiple orientations of a LIDAR-based cost function in parallel.

Clipping

Once the geometric processing is complete, the generated geometry passes through a clipping stage. This removes any geometry whose normalised device coordinates, after division by the homogeneous divisor w are outside the range $[-1, 1]$. The planes along which the coordinates are equal to 1 are known as *clipping planes*. Clipping is also carried out in the depth dimension, except the limits for the depth clipping are $[0, 1]$, where 0 is the *near clipping plane* and 1 is the *far clipping plane*.

Rasterisation

Rasterisation is the process of converting the continuous geometry generated by the vertex and geometry shaders, and converting to *fragments*, mapping to pixels on a display or texture [69]. This stage of the pipeline is fixed — the user is not in control of the rasterisation process, and no shader code is run.

Fragment Shader

The fragment shader is the final stage of the pipeline, and is required unless transform feedback (see below) is enable. The fragment shader is responsible for generating the actual values (colours) written to each pixel in the bound framebuffer [69]. Each invocation of the fragment shader writes to exactly one fragment; a fragment corresponds to exactly one pixel, but each pixel can have many fragments from multiple primitives being drawn at a given pixel location. The final pixel value is generated from the fragment values depending on depth testing and alpha-blending (transparency) [70].

The inputs to the fragment shader come from the output of the vertex or geometry shaders, interpolated for position within the primitive being rendered. For example, if rendering individual points, the fragment shader will receive the exact values generated by the vertex/geometry shader. If rendering a triangle, however, the fragments corresponding to the interior of the triangle will receive values interpolated between the values for the three vertices of the triangle.

Importantly for many of the applications in this thesis, the fragment shader also has access to a depth buffer, containing depth values for each fragment.

Transform Feedback

OpenGL has a feature known as *transform feedback*, which captures the geometry generated by the initial shader stages, before rasterisation. This allows a variable-sized output to be captured, rather than having to render to a fixed-sized texture [70].

2.3.4 Computation with OpenGL

Whilst the OpenGL API was designed expressly for the generation of 3D graphics, it can also be used for general-purpose computation [71]. For computations involving images, or more generally, large 2D arrays, the graphics idiom lends itself well to efficient formulations. For some operations, however, there is not an obvious

mapping from the computation required to features present in the OpenGL system. Many of the features required, however, can be found in the API, by repurposing functionality designed for specific areas of 3D graphics. By using these built-in features, one gains access to the many thousands of engineering hours placed into the hardware and drivers for the GPUs. Two important examples are described below.

Alpha Blending

A core part of many of the algorithms described in this thesis involves generating joint histograms for pairs of images. In comparison to some other direct methods, such as LSD-SLAM [17] or DTAM [15], this is not entirely parallelisable, as multiple pixels contribute to any given bin, requiring atomic operations.

Atomic operations can be extremely slow on a GPU, as they are inherently not parallel. To perform atomic operations naively, all threads wishing to write to a particular location would need to pause whilst a value is being written. As GPUs work on a single-instruction-multiple-data model, this would effectively mean all threads pausing whilst a write takes place. This could be expected to be very slow.

In OpenGL, the atomic addition required for parallel histogram generation can be achieved using the built-in mechanism of alpha blending. This mechanism is designed for rendering graphics with transparent or semi-transparent elements. It essentially provides fast, weighted, atomic addition of values written to a pixel, for the purposes of blending the appearance of transparent elements with the background. When writing a value to a fragment, instead of completely overwriting a previous pixel value, or choosing one fragment value for the pixel based on a depth test, alpha blending allows one to perform a weighted sum of the previous pixel value and the new fragment.

Mipmapping

Another common operation in many algorithms herein is that of reduction — essentially summing large arrays of numbers. Once again, a naive implementation would require slow atomic operations. A better method is to take a ‘divide and conquer’ approach — to sum subsets of the array into a successively smaller array, until only one value is left. OpenGL, however, has a builtin mechanism that allows this to be done in an efficient manner: mipmapping.

Mipmapping is designed for computing downscaled versions of textures for rendering at different resolutions, and on most graphics cards is hardware accelerated. The mipmap operation computes ‘levels’ of textures, each downscaled by a factor of two in each dimension from the level above, where each pixel consists of the mean of 2^D pixels from the level above (where D is the dimensionality of the texture). Thus mipmaps provide both a natural storage structure for ‘pyramids’ of images or textures, as well as a fast method for summing large arrays of numbers.

2.3.5 Conclusion

Through judicious use of its graphics-oriented functions, OpenGL can be used for many general-purpose computing tasks. This brings the massive parallelism available in modern GPUs to computational tasks in a portable manner.

2.4 Transformations in $\mathbb{SE}(3)$

Throughout this thesis, we make regular use of $\mathbb{SE}(3)$ transformations to represent the relative poses of sensors, platforms, and objects in the world. The Special Euclidean group $\mathbb{SE}(n)$ is the set of all possible rigid-body transformations in n -dimensional space [72].

We use two primary representations of such transformations in this thesis: 4×4 homogeneous transformation matrices and the Lie algebra.

2.4.1 Matrix Representation

The matrix representation is used for performing transformations between coordinate frames. A point ${}_A\mathbf{p}$ in frame A can be transformed into frame B via a single matrix multiplication:

$$\begin{bmatrix} {}_B\mathbf{p} \\ 1 \end{bmatrix} = {}^A_B G \begin{bmatrix} {}_A\mathbf{p} \\ 1 \end{bmatrix} \quad (2.1)$$

where ${}^A_B G$ is the homogeneous matrix representation of the transform from frame A to frame B . The matrix form may also be used for compositing multiple transforms:

$${}^A_C G = {}^B_C G {}^A_B G \quad (2.2)$$

The matrix G is composed of a rotation matrix R and translation vector t [72]:

$$G = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \quad (2.3)$$

Here $R \in \text{SO}(3)$, where $\text{SO}(n)$ is the special orthogonal group containing rotation transforms in n -dimensional space, and $\mathbf{t} \in \mathbb{R}^3$.

2.4.2 Lie Algebra

The matrix representation is useful for composing and applying transforms, but is not suitable for use in an optimisation. This is due to the fact that it is over-parameterised — the matrix has 12 variable elements, but represents a space with only 6 degrees of freedom. Furthermore, not all combinations of values for those 12 elements will yield a valid $\text{SE}(3)$ transform.

For use in optimisation, we make use of the Lie algebra associated with $\text{SE}(3)$. In the Lie algebra, an $\text{SE}(3)$ transform G is represented by a vector $\mathbf{g} \in \mathbb{R}^6$. This is a minimal representation (6 parameters for a 6DOF space) which is continuously differentiable [72].

We make use of an exponential map to transform between the Lie algebra and matrix representations of a transform:

$$G = \exp_{\mathbb{SE}(3)}(\mathbf{g}) = \exp(\hat{\mathbf{g}}) \quad (2.4)$$

Here we use $\exp_{\mathbb{SE}(3)}$ to denote the exponential map for $\mathbb{SE}(3)$. It is composed of two steps as shown above: the $\hat{\bullet}$ operator $\hat{\bullet} : \mathbb{R}^6 \mapsto \mathbb{R}^{4 \times 4}$, and the matrix exponential [73].

The $\hat{\bullet}$ operator for $\mathbb{SE}(3)$ is defined by [73]:

$$\hat{\mathbf{g}} = \begin{bmatrix} \hat{\nu} \\ \omega \end{bmatrix} = \begin{bmatrix} [\omega]_{\times} & \nu \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{SE}(3) \quad (2.5)$$

where $\nu, \omega \in \mathbb{R}^3$, and $[\omega]_{\times}$ is the skew-symmetric operator defined by:

$$[\omega]_{\times} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \quad (2.6)$$

Similarly, we can use a matrix logarithm to map from the matrix representation to the Lie algebra:

$$\mathbf{g} = \log_{\mathbb{SE}(3)}(G) = \log^{\vee}(G) \in \mathbb{R}^6 \quad (2.7)$$

where $\log^{\vee} : \mathbb{R}^{4 \times 4} \mapsto \mathbb{R}^6$ is the inverse of the $\hat{\bullet}$ operator defined previously, and $\log$ is the matrix logarithm.

For further details on Lie groups, and their exponential maps, we refer the reader to [73].

3

LIDAR Mapping

Contents

3.1	Introduction	26
3.2	Sensor Configuration	27
3.3	Geometry	29
3.3.1	Local Geometry	30
3.3.2	Mesh Generation	32
3.4	Texturing	34
3.4.1	Mesh Reprojection	34
3.4.2	Texture Selection	36
3.5	Conclusion	41

3.1 Introduction

The thrust of this thesis is about using vision-based systems to precisely localise a road vehicle within a prior map. Existing road infrastructure — for example road markings, signs, or traffic lights — is designed primarily to be perceived by human eyes. Any vehicle will therefore require the ability to perceive the visual spectrum in order to drive without large infrastructure changes. The minimal, and lowest cost, feasible perception solution would therefore be a vision-only system.

Whilst the lowest-cost case is a vision-only system, in many cases having LIDAR

scanners on a small number of survey vehicles would be acceptable. Such a system would involve the survey vehicles producing high-quality maps, which then allow commodity vehicles to localise using cameras only. As the cost of the sensor suite on the survey vehicles could be amortised across many consumer vehicles, the price impact would be minimal. Furthermore, as opposed to previous LIDAR-based mapping methods that rely upon expensive 3D LIDAR scanners [4, 5, 7, 16], our method only requires a much cheaper single-beam 2D LIDAR, leading to further cost reductions. Later on in this thesis, (see Chapter 6), we explore the use of vision for mapping as well, however this chapter deals with the LIDAR-based mapping case.

The maps we create in this chapter consist of two primary types of data: *geometry* and *texture*. The geometry is represented by a triangular mesh, derived from LIDAR scans, as such sensors provide accurate range measurements, whilst the textures are represented by an image for each triangle in the mesh, taken from camera images of the scene. The remainder of this chapter describes the generation of these two elements.

3.2 Sensor Configuration

The platform used for mapping in this section is the Oxford RobotCar. This is a standard Nissan Leaf, with multiple sensors mounted, as displayed in Figure 3.1. The primary sensors used for mapping are the Bumblebee XB3 cameras (for visual odometry), the LMS-151 LIDARs (for geometry), and the Grasshopper2s cameras (for textures).

The PointGrey Bumblebee XB3 is a double stereo camera, with three fronto-parallel lenses spaced approximately 12cm apart. The camera is mounted on the roof of the car facing forwards, to provide a sensor for visual odometry. Each lense provides a 1280×960 Bayer-encoded colour image, at a rate of approximately 20Hz. Example images are shown in Figure 3.2.

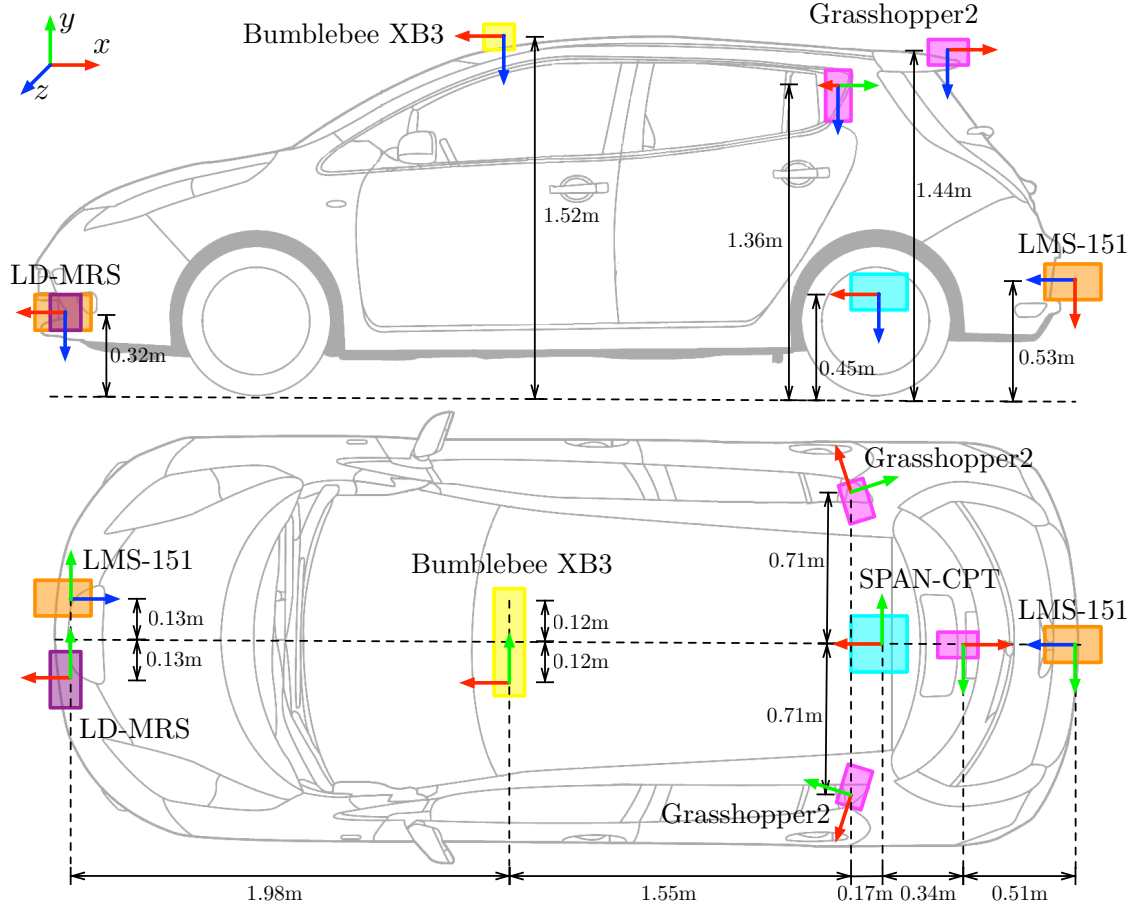


Figure 3.1: Sensor configuration on the Oxford RobotCar used for this section.

The PointGrey Grasshopper2s are monocular cameras which provide 1384×1086 Bayer-encoded colour images, at a rate of approximately 8Hz. There are three of these cameras mounted on the vehicle — two pointed to the side, and one backwards. They are mounted with fisheye lenses to give a large field of view of approximately 185° .

The SICK LMS-151 LIDARs are mounted at the front and rear of the vehicle, pointing down in a ‘pushbroom’ configuration. This means that their scans sweep through a plane perpendicular to both the road and the direction of travel of the vehicle. Each scan provides 541 range measurements, equally spaced across a 270° sweep.



(a)



(b)



(c)



(d)

Figure 3.2: Example images from the cameras mounted on the Oxford RobotCar. (a) The forward facing Bumblebee 3 has three synchronised lenses. (b) Left Grasshopper 2. (c) Rear Grasshopper 2. (d) Right Grasshopper 2. Between them, these sensors give a surround view of the vehicle.

3.3 Geometry

The geometry of our maps comes from LIDAR scanners. These sensors directly measure range via time-of-flight with infrared light. This gives highly precise geometric information.

We represent the geometric element of our maps as a triangular mesh. This is a standard method of representing 3D structure for OpenGL, and indeed in many other rendering environments for computer graphics. The basic form of the mesh is a series of triangles, with 3D coordinates for each vertex.

By using a mesh, rather than a pointcloud (as in [18]), we obtain a fully dense map of the environment¹. This dense map allows the use of more image data for

¹The model is not completely dense, as there are gaps in which there is no structure — eg. sky.

localisation (see Chapter 4) than would a sparse pointcloud. Furthermore, as a mesh represents the full scene geometry, occlusions are implicitly modelled — unlike with a pointcloud, background geometry cannot be ‘seen’ through occluding structure.

3.3.1 Local Geometry

The raw data from the LIDAR consists of a series of scans, each of which contains an array of points in the form $(range, reflectance)$. The angle of the laser beam in the scanner moves at a constant angular velocity throughout each scan, allowing a derivation of the beam angle for each point:

$$\theta = \frac{i}{n} \times \phi \quad (3.1)$$

where i is the sequence ID of a particular point within the scan, n is the total number of points in a scan, and ϕ is the angle through which the beam sweeps in each scan (here, $\phi = 270^\circ$).

With the beam angle and range, we can directly obtain 3D cartesian coordinates for each point, relative to the sensor. To obtain the actual 3D structure of the world, however, we require all these points to be put into a single coordinate frame. More formally, we require a transform ${}^{S(t_i)}_W G$ for each scan, which maps the sensor frame S at time t_i into the world frame W , and the set t_i contains the timestamps at which each LIDAR scan was captured.

Note that there is no requirement here for the sensor poses ${}^{S(t_i)}_W G$ to be in a *globally* consistent frame. The only requirement is *local* consistency. As everything the maps are to be used for is relative to the map, a single global coordinate frame is not necessary — we only need a map that accurately represents the local 3D structure. For this reason, we are able to use visual odometry (VO), from the

In structured regions, however, the map is fully dense, as opposed to the semi-dense maps used in Chapter 6 or the sparse pointclouds used in [18]

front-facing stereo camera, with no need for loop closures or bundle adjustment, to obtain the relative pose estimates. We make use of the VO system described by [74].

The VO pipeline produces a series of relative poses ${}^{C(t_j)}G$ between camera frames at times t_j and t_{j-1} , where the set t_j are the timestamps of the camera images captured by the stereo camera. To obtain relative poses at times t_i (LIDAR timestamps), we use a linear interpolation between neighbouring VO poses — ie. for time t_i , we interpolate between poses at times t_j and t_{j+1} , where $t_j \leq t_i \leq t_{j+1}$. The timestamps of the LIDAR and camera sensors are synchronised using TICSYN [75] to ensure consistency. We use linear interpolation in translation components, and spherical linear interpolation (Slerping [76]) in quaternions for orientation.

Whilst this linear interpolation scheme assumes constant velocity in the period (t_j, t_{j+1}) , this period is very short — the camera we use here has a frame rate of 20Hz — so the effect of acceleration is negligible.

The interpolation allows us to obtain the poses ${}^{C(t_i)}G$ — ie. the pose of the camera relative to the world frame W at the time of each laser scan. The world frame is arbitrary, and for simplicity is set as the first camera frame. To transform these poses to the LIDAR frame, we require the transform ${}^L_C G^2$, which is obtained through extrinsic calibration. For details on this point, we refer the reader to Chapter 5. Figure 3.3 shows an example of a section of a pointcloud constructed using this method.

The discussion so far in this section assumes that an entire LIDAR scan is obtained at a single point in time. Whilst that assumption does produce reasonable geometry, it can be improved upon by adding an extra interpolation step within each scan. We divide the period of the scan into equally spaced intervals, and interpolate poses for each of these intervals, identically to the per-scan interpolation discussed previously. The effect of this can be seen in Figure 3.3(b), in the slight

² L : LIDAR, C : Camera

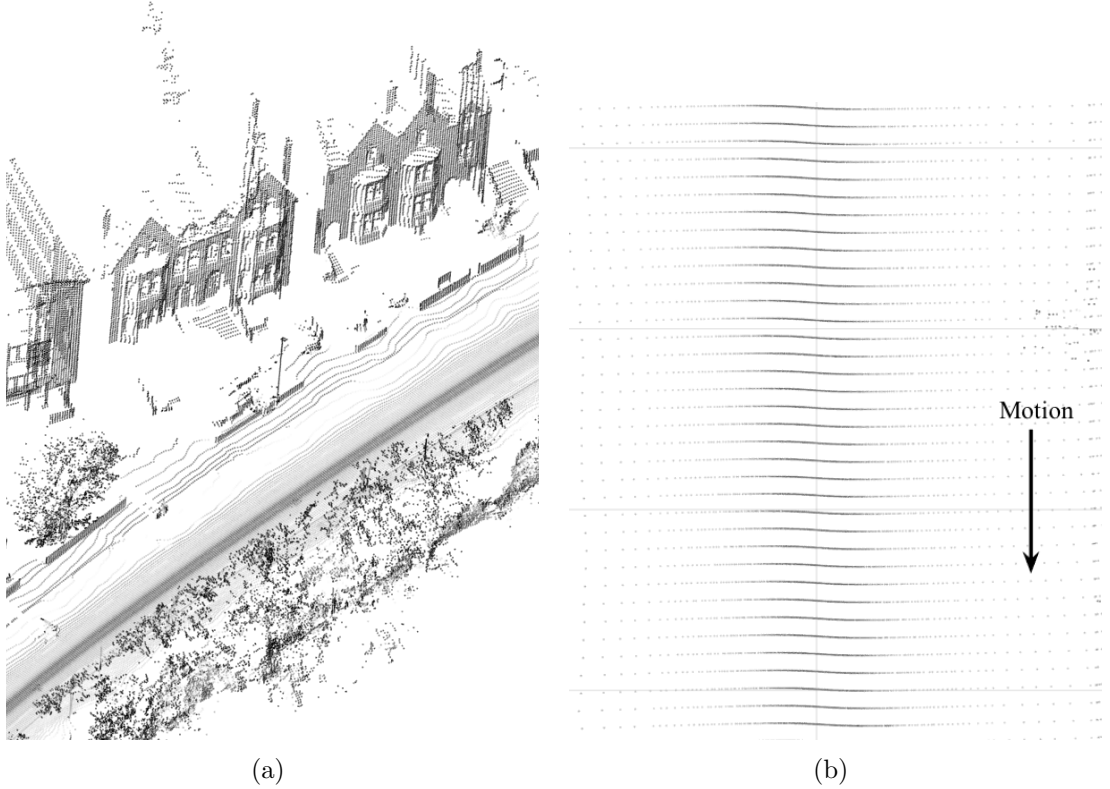


Figure 3.3: (a) Example section of a pointcloud constructed from a 2D LIDAR scanner and vehicle motion. (b) Detail of the road in the pointcloud, showing the lines of points across the path of the vehicle that come from the use of the pushbroom LIDAR configuration. A mesh is constructed by joining subsequent lines together.

curve in the scan lines — without these extra interpolations, these lines would be perfectly straight.

With a pose for the LIDAR sensor corresponding with each point, we obtain a pointcloud in a locally consistent frame by transforming each point in the LIDAR frame to the world frame.

3.3.2 Mesh Generation

Once we have a sparse pointcloud in a consistent frame, the next step is to generate a triangular mesh from that pointcloud. Whilst a number of approaches have been developed for meshing generalised pointclouds [77, 78], the unique structure of the pointclouds from our sensor setup makes the problem much simpler. This pointcloud structure, however, has no downstream impact on the use of the map

— once constructed, the map simply consists of triangles in space, the same as a mesh constructed via any other method.

Because the pointclouds we use here come from a pushbroom LIDAR setup, with the vehicle primarily moving in a single direction, they consist of approximately parallel lines of points, where the i th point in one scan is adjacent to the i th point in the next scan. As a result of this structure, a reasonable mesh can be generated by stitching together successive scans into a triangle strip, as is illustrated in Figure 3.4.

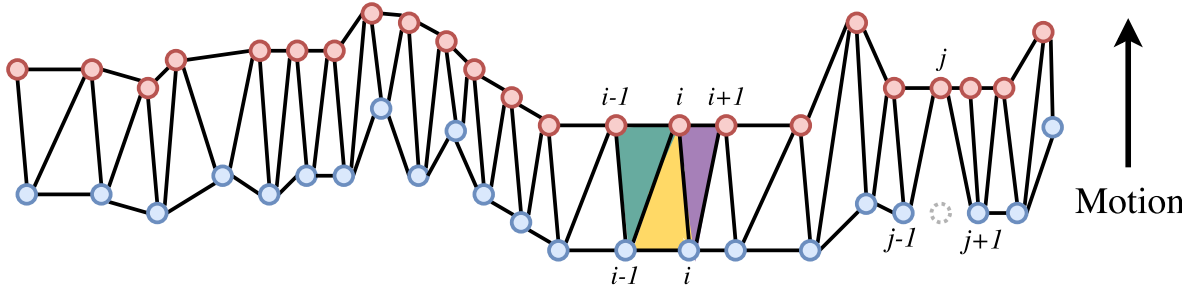


Figure 3.4: Mesh stitching from pushbroom pointcloud. Each point i in a scan is linked with points $i + 1$ and $i - 1$ in the same scan, and points i and $i - 1$ from the previous scan, resulting in three triangles. In some cases, a given index will not exist in a scan, due to gaps in the structure of the scene (eg. side streets parallel to the beam direction.) In this case, as with point j in the blue scan above, we make a diagonal link in the opposite direction — here red point j is linked with blue $j + 1$ instead of j .

The stitching procedure produces reasonable pointclouds that are acceptable for most parts of the environment, however near small objects or object boundaries, erroneous surfaces are often generated. For example, if an overhanging tree is captured by one scan, but not the next, a triangle may be generated with one vertex on a tree leaf and another on the ground, creating a surface in the mesh that does not exist in the real world.

To combat this problem, we use a heuristic limit on the maximum length of any side in a triangle. Experimentation with several heuristics, including limits on the area of triangles, on the ratio of longest to shortest axis, or difference in appearance of the vertices when projected into images, showed the maximum length

to be at least as effective and far more computationally efficient compared to the others. We discard any triangles with a side longer than this limit.

We store each ‘slice’, made up of the triangles obtained from stitching two adjacent scans, as a separate object. This enables efficient streaming of small sections of the mesh to and from the GPU and disk for localisation (Chapter 4) or texturing (Section 3.4).

3.4 Texturing

The previous section dealt with generation of the 3D geometry of a scene using a combination of 2D LIDAR and VO. The second requisite component of the map is appearance, required in order to be able to perform appearance-based localisation (see Chapter 4).

The texturing process is a two-pass procedure. The first pass is purely geometric, projecting the mesh into each available camera image and using geometric heuristics to determine from which image to obtain texture information. On the second pass, the actual textures are obtained from the relevant images.

3.4.1 Mesh Reprojection

The first step of the texturing process is reprojection of the mesh geometry into multiple camera images. As we use OpenGL for texturing, this requires projecting the 3D geometry in world space into 2D normalised device coordinates (see Section 2.3).

The projection of world-space geometry into normalised device coordinates is normally represented by a model-view-projection (MVP) matrix [70]. The three components of this matrix are:

- M: Model matrix: transforms an object to its desired location in the world frame

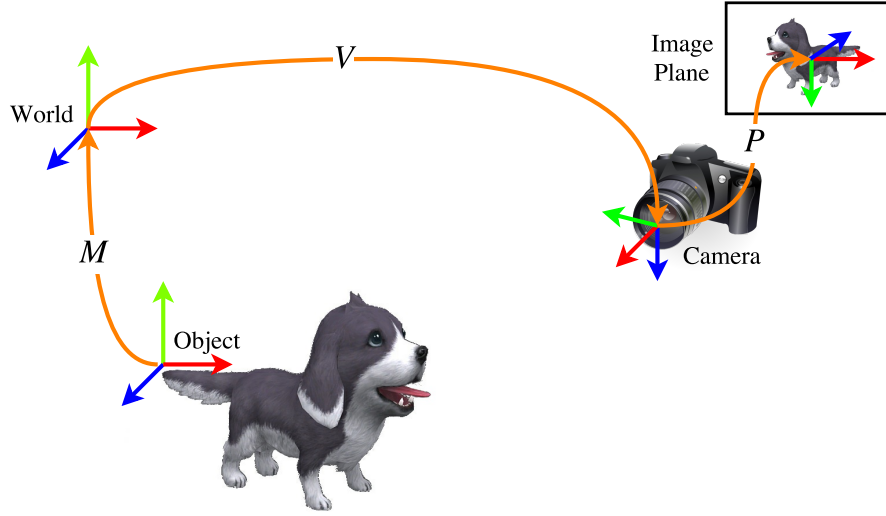


Figure 3.5: Components of OpenGL MVP matrix: the model matrix M transforms the object into world space, the view matrix V transforms from world space to camera coordinates, and the projection matrix P projects onto the image plane.

- V: View matrix: transforms the object in world frame to coordinates relative to the virtual camera
- P: Projection matrix: generally a pinhole projection model.

These component matrices are illustrated in Figure 3.5.

In the case of the mesh generated in the previous section, the coordinates of all the triangles are already in the world frame, so no model matrix is required.

The view matrix, for transforming world-frame points into a frame relative to the camera, is the pose of the camera for each image timestamp. This is the same transform ${}^{C(t_j)}_W G$, for image j , obtained from VO in an identical fashion to that described in Section 3.3.1.

Finally, for the projection matrix, we use a projection matrix P :

$$P = \begin{bmatrix} -\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & -\frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (3.2)$$

where n and f are the distances from the camera to the near and far clipping planes respectively, r and l are the left and right clipping planes respectively, t and b are the top and bottom clipping planes respectively, and f is the focal length of the camera. This is equivalent to a pinhole camera projection, with projected values scaled such that they are equal to ± 1 at each of the the clipping planes — ie. normalised device coordinates.

Thus, the projection of a homogeneous vertex $\mathbf{p}_k$ into camera image j can be represented as:

$$\hat{\mathbf{p}}_k = P^{C(t_j)}_W G \mathbf{p}_k \quad (3.3)$$

This produces 4-vector $\hat{\mathbf{p}}_k = (u, v, z, w)$, where (u, v) are the normalised device coordinates, z is the normalised depth, and w is the homogeneous divisor — the OpenGL pipeline automatically performs division by w prior to rasterisation.

3.4.2 Texture Selection

Most triangles in the mesh will reproject into multiple successive frames, and thus have multiple candidates for textures. We use two heuristics to choose which images to use for texturing each triangle:

- Triangle area (in normalised device coordinates) — a larger triangle means it is closer to the camera and provides a higher resolution texture.
- Proximity to image centre — regions of the image near the edges are more effected by distortion.

In the first pass through a mesh, we reproject each point into each nearby camera image according to Equation 3.3. We stream the mesh from disk, using only slices within 50m of the camera for each image. For each triangle, we then compute the area

of the triangle in the normalised device coordinates space, along with the location of the triangle's centroid. We then compute a score for each triangle according to:

$$\text{score} = Aw(c_u)w(c_v) \quad (3.4)$$

where A is the area of the triangle, c_u and c_v are the horizontal and vertical components of the triangle's centroid in the image, and $w(\bullet)$ is a weighting function defined by:

$$w(u) = \exp\left(-\frac{(u - 0.5)^2}{2 * \sigma}\right) \quad (3.5)$$

where we use $\sigma = 0.064$. This function is illustrated in Figure 3.6.

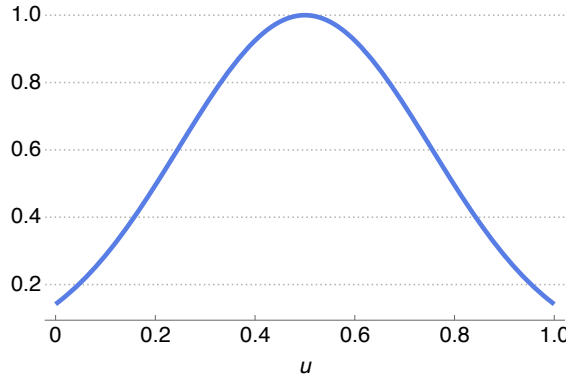


Figure 3.6: Centroid weighting function for triangle selection. The weighting prefers triangles towards the centre of the image.

In addition to the weighting, we also apply a depth test to each triangle projection. This is to account for the possibility for multiple overlapping triangles to project into the same image region — for example, a tree and the building behind it may project into the same pixels. We require all fragments in the reprojected triangle to have the minimal depth for their respective pixel for a texture to be used. This may be directly obtained from OpenGL's depth buffer.

At the end of the first pass through the mesh, each triangle is associated with the image with the highest score according to Equation 3.4, subject to the depth test. As stated in Section 3.3.1, we store each 'slice' of the mesh independently,



Figure 3.7: Example of an image used to store textures for a slice of mesh. The image consists of a regular triangular grid, storing the texture for each triangle sequentially across the slice.

in order to enable streaming of subsections of the mesh to/from GPU and CPU memory. For the same reason, in a second pass through the data, we store all the textures for the triangles in one slice of mesh in a single image. These images have a regular triangle pattern, with the texture for each triangle taking up the same area. An example of such an image is shown in Figure 3.7.

In Figure 3.8, we show an example of a mesh that has been built using this process. Figure 3.8(a) shows an example image from the dataset used to create the mesh, whilst Figure 3.8(b) shows the mesh projected into that camera image. The mesh is coloured by LIDAR reflectance for visualisation purposes — the intensity of each vertex is taken from the reflectance value of that LIDAR point. This projection is used for taking textures from camera images to be used for the mesh.

Figure 3.8(c) shows the same view of the mesh after texturing. Note that whilst this image is from the location of the camera image in (a), the textures used to colour the mesh come from multiple nearby camera images. Finally, Figure 3.8 shows a birds-eye view of the textured mesh.

Implementation Details

Whilst this mapping process can be done offline, the volume of data involved means that some care for efficiency is still necessary. Note that in the first pass, the actual images are not required, as only geometric information is needed. We load a small section of the mesh onto the GPU at a time, and compute the projected coordinates of each triangle vertex in a vertex shader. We then compute the area and centroid statistics and resulting weight according to Equation 3.4 in a geometry shader (see Section 2.3). If the resulting score is greater than the previous highest score for a triangle, and the triangle passes the depth test, we save the index of the image. At the end of the first pass, this gives us a list of the index of the image from which to obtain texture for each triangle, along with the coordinates of its vertices projected into that image. All of this is done using transform feedback, without any use of fragment shaders or textures.

For the second pass, we load each image required for a given slice into a 3D texture on the GPU. We can then obtain the textures for every triangle in parallel with a 3D texture lookup — two coordinates specify normalised image coordinates, and a third specifies which image (which layer in the 3D texture) to sample.

In order to minimise copying of images to the GPU, we use a buffer, which only removes images from the GPU after they have not been used for 10 successive slices. We keep track of which layers in the 3D texture are no longer required, and load new images into these. In practice, this means that most images are loaded to the GPU only once, whilst enabling texturing of arbitrary-sized meshes in linear time.

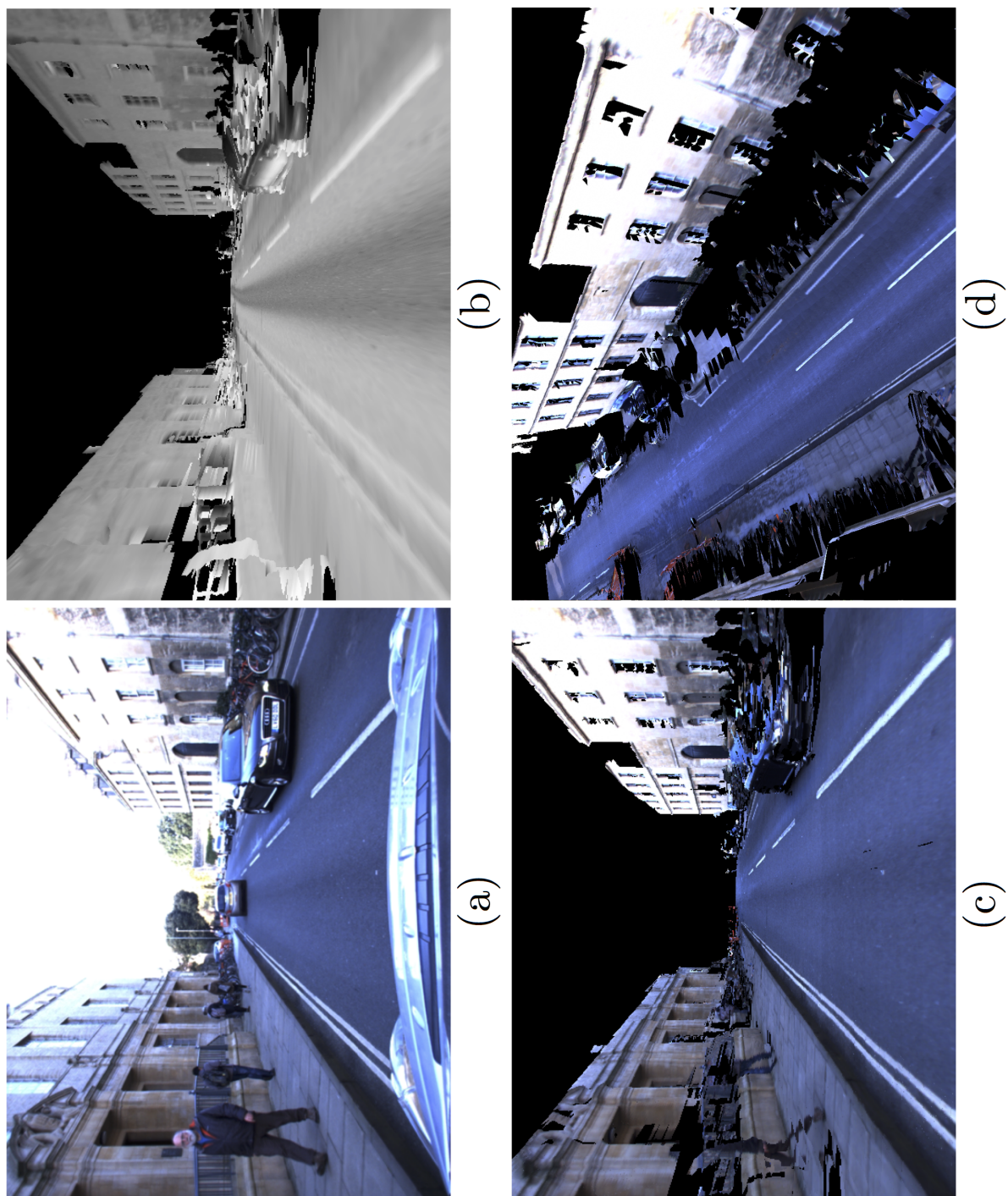


Figure 3.8: Example of a textured mesh. (a) Camera image used for texturing. (b) Untextured mesh, coloured with reflectance from LIDAR scanner. (c) Textured mesh from the same location as (a). The textures are taken from many images, not just that shown in (a). (d) Overhead view of the mesh.

3.5 Conclusion

In this chapter, we have presented a method for generating dense textured 3D maps of urban environments using only a camera and 2D LIDAR scanner. These sensors are an order of magnitude less expensive than the 3D scanners previously used to build maps.

Our maps are constructed using relative pose estimates from visual odometry only, and so they are not globally consistent. The 3D structure of the maps is, however, locally accurate, and thus sufficient for relative pose localisation.

The maps consist of both geometric and appearance information, and can be generated efficiently in linear time through the use of a GPU. Furthermore, they are stored as a series of small, distinct ‘slices’, allowing for efficient streaming of submaps to and from the GPU. This feature will be used in Chapter 4 when dealing with localisation.

4

Robust, Dense, Visual Localisation

Contents

4.1	Introduction	43
4.2	Problem Formulation	45
4.3	Metrics for Image Comparison	46
4.3.1	Photoconsistency	47
4.3.2	Normalised Information Distance	49
4.3.3	Void Filling	60
4.3.4	Mesh Distortion	62
4.3.5	Image Blurring	64
4.4	Optimisation	64
4.4.1	Gradient Descent	66
4.4.2	Newton's Method	67
4.4.3	Gauss-Newton Method	67
4.4.4	Levenberg-Marquardt Method	68
4.4.5	BFGS Method	68
4.5	Evaluation	71
4.6	Localisation	72
4.6.1	Outlier Rejection	73
4.6.2	Implementation Details	74
4.6.3	Evaluation	76
4.6.4	Synthetic Data	77
4.6.5	Real Data	81
4.6.6	Computation Time	88
4.7	Conclusion	89

4.1 Introduction

In the previous chapter, we showed how we can generate textured 3D maps of an environment using a 2D LIDAR scanner in pushbroom configuration and camera(s) mounted on a vehicle. In this chapter, we explore how we can precisely localise a camera within that map. The work in this thesis is primarily focused on autonomous driving as the application domain, however much of what we discuss here is more generally applicable.

For a localisation solution to be used as an input to a control system for a robotic platform such as an autonomous vehicle, topological place recognition such as that provided FAB-MAP [79] is not sufficient. Instead, precise, metric pose estimates are required. For autonomous driving in urban environments specifically, successful navigation has been demonstrated with localisation errors on the order of 30-40cm in urban space [4, 16]. For driving through very narrow pathways, [5] reports a stricter requirement, with a localiser accurate to within approximately 15cm. A similar accuracy was reported by [7], by applying a filter to a LIDAR-based localisation system with raw (unfiltered) accuracy of up to 50cm.

Many state-of-the-art systems for localisation make use of LIDAR-based sensors [5–7]. As was noted in Chapter 3, the range-sensing capability of LIDAR enables the production of a precise geometric model of the sensor’s immediate surroundings. This model can be used not just for mapping, but also for localisation against a previously generated map [5, 7].

Unfortunately, many LIDAR sensors are very expensive, and despite recent cost reductions remain out of the reach of many applications. Furthermore, being an actively emitting sensor, LIDAR may not be appropriate in some environments, or for deployment in large numbers. Conversely, cameras do not actively emit any signals into the surrounding environment, and are both cheap and universally

available. More importantly, with a road system with infrastructure that is designed to be perceived with human eyes, no vehicle will be able to navigate successfully without a visual perception system. A camera is thus a required sensor on an autonomous vehicle, and here we explore the minimal case of a vehicle with a camera as the only sensor.

Whilst the maps we use here are generated through the use of LIDAR, as described in Chapter 3, a camera-based localiser would remove the need for a LIDAR sensor to be present on every vehicle or robot. A small number of vehicles tasked with mapping the environment could be equipped with LIDAR, whilst the majority of vehicles would need visual sensors only. The cost of such mapping vehicles would thus be amortised across the many vehicles making use of their maps, obviating the cost and scale problems of universal LIDAR use.

Existing approaches to visual localisation generally use either matching of feature descriptors from distinctive image points [8–11, 28, 30, 32] or direct comparison of pixel values in images [15, 44, 45]. Both of these approaches, however, suffer from failures when the scene being viewed has undergone significant appearance changes since it was mapped. Feature descriptors lose their efficacy over long periods of time [13, 14], and often fail with motion blur or changes of viewpoint [40]. Direct photometric methods, on the other hand, handle image degradations such as blur well, but suffer from the assumption that the same point in space will always have the same appearance — as a result of this assumption, changes in illumination, or even a change of camera, cause these systems to fail.

Several methods for handling large appearance change in visual localisation have been proposed in the past. The most robust systems are based on SeqSLAM [54], which uses a sequence of images along a trajectory, rather than just a single image, in order to localise. This makes it invariant to large environmental changes, but it has limitations that make it insufficient for the autonomous driving task that is the primary focus of this thesis. Firstly, by comparing sequences of images, SeqSLAM

makes localisation very dependent on the viewpoint of the camera — even small deviations from a prior trajectory have been shown to cause failure [54]. In addition, SeqSLAM provides only topological localisation along a given route, not the metric pose estimates that are required for controlling a vehicle.

Another possible method for dealing with large appearance change is to predict the appearance of a scene under different conditions. This has been shown to allow place recognition across seasons, including conditions similar to those evaluated in this chapter [42]. Whilst the appearance predictions are sufficient on a large scale for place recognition, however, they do not preserve the fine-grained detail that is required for precise metric localisation.

Finally, recent attempts have been made to solve the visual localisation problem using CNNs [51]. These methods demonstrate robustness to large appearance and illumination changes, but the most recent improvements [53, 55] still produce RMS errors greater than 1m in translation. Furthermore, CNN-based localisation has only been demonstrated over a small coverage area — coverage of an entire city’s road network is not as yet feasible with these approaches. These methods are thus not sufficient for use in an autonomous driving system, which requires precision in the tens of centimetres and a wide area of coverage.

4.2 Problem Formulation

Given a prior map such as that described in Chapter 3, the goal of localisation is determine the $\mathbb{SE}(3)$ pose of a camera with respect to that prior. As noted in the previous chapter, that prior is not necessarily in a globally consistent frame — we do not require a global 6D pose. All functions requiring a localisation solution (planning, control, feature lookup in maps etc) only need to know the pose of the vehicle with respect to a *local* map.

Localisation of the camera can be posed as a minimisation problem, in which the aim is to find the camera pose $\hat{G}_C$ such that:

$$\hat{G}_C = \arg \min_{G_C} Cost(I_c, I_r(G_C, S)) \quad (4.1)$$

Here I_c is an image taken by the camera, G_C is an estimate for the camera pose, and S is the prior scene, consisting of 3D geometry and texture information, and I_r is an image produced by placing a virtual camera at the pose G_C in the scene S . $Cost$ is a function for quantifying the difference between two images — further discussion of appropriate cost functions follows in the next section.

Intuitively, this minimisation problem attempts to place a virtual camera in the scene S such that a rendering of the map best ‘matches’ the image captured by the real camera. In a changing outdoor environment, however, we cannot expect the scene to always be identical in appearance. We thus require a method for evaluating the similarity between two images that takes into account the sort of variation seen in real-world environments.

4.3 Metrics for Image Comparison

A measure to be used for comparing live camera images with renderings of the map needs to be invariant (or at least somewhat invariant) to the appearance changes that an outdoor scene may undergo over time. These changes include, but are not limited to:

- Short-term illumination changes over the course of a day
- Longer-term illumination variation due to the passing of seasons
- Illumination variation due to weather (clouds, sun)
- Surface changes due to weather (rain, snow)

- Structural changes due to dynamic obstacles (eg. vehicles)

Some examples of this variation are shown in Figure 4.1.

4.3.1 Photoconsistency

A frequently-used measure in the domain of dense tracking and reconstruction is the photoconsistency metric [80], which computes the sum of squared differences between pixel intensity values in a camera image I_c and a (rendered) reference image I_r as follows:

$$\text{SSD}(I_r, I_r(G_C, S)) = \sum_{\mathbf{x} \in I_c} \|I_c(\mathbf{x}) - I_r(\mathbf{x}, G_C, S)\|^2, \quad (4.2)$$

where $\mathbf{x} = (u, v)^T$ is a pixel location within the image.

The photoconsistency metric is widely used [15, 17, 44, 58], as it is cheap to evaluate, easily differentiable and trivially parallelisable, lending it to very fast evaluation and optimisation. The availability of a per-pixel cost value also enables robustness to localised changes, such as dynamic obstacles, through outlier rejection or the use of a robust kernel [17].

Although photoconsistency is efficient to compute and find derivatives for (in order to use in an optimisation framework), as mentioned in [15] it suffers from a number of limitations. Principally, it requires I_r provide a photorealistic rendering of the scene S from location G_C , such that the resulting synthetic image matches the camera image I_c on a pixel-by-pixel basis. A true match would only be possible if the synthetic scene S captured all relevant geometry and material properties of the real-world environment, and the synthetic image rendering function $I_r(G_C, S)$ accurately modelled scene illumination intensity, spectrum and direction in accordance with the true lighting, which both remain challenging to render for real-time applications. Alternately, the scene would need to be captured under identical conditions to those under which localisation is required. This limitation restricts photoconsistency



Figure 4.1: Examples of appearance variation seen in an outdoor environment. Localisation under such a wide variety of conditions is a challenge.

to applications involving frame-to-frame tracking, where the synthetic image I_r can be derived from a warping of the previous camera image I_c [45, 81, 82], or applications in small indoor environments with controlled illumination where the scene S does not change over time [15].

4.3.2 Normalised Information Distance

Rather than considering a cost on a pixel-by-pixel basis, it is possible to define a whole-image cost function. The function we explore here is known as Normalised Information Distance (NID) [83], and is defined by:

$$\text{NID}(I_r, I_c) = \frac{2H(I_r, I_c) - H(I_r) - H(I_c)}{H(I_r, I_c)} \quad (4.3)$$

where $H(I_r, I_c) \in \mathbb{R}^+$ is the joint entropy of the images I_r and I_c , and $H(I_r) \in \mathbb{R}^+$ and $H(I_c) \in \mathbb{R}^+$ are the marginal entropies, defined as follows:

$$H(I_c) = -\sum_{a=1}^n p_c(a) \log(p_c(a)) \quad (4.4)$$

$$H(I_r, I_c) = -\sum_{a=1}^n \sum_{b=1}^n p_{r,c}(a, b) \log(p_{r,c}(a, b)) \quad (4.5)$$

where $H(I_r)$ is defined similarly to Eq. 4.4. The marginal $p_c, p_r \in \mathbb{R}^n$ and joint $p_{r,c} \in \mathbb{R}^{n \times n}$ distributions are represented by n -bin histograms of intensity where a and b are individual bin indices. Since both p_r and p_c can be obtained from $p_{r,c}$ by marginalisation, the primary computation for NID is computing the joint distribution $p_{r,c}$ and its derivatives.

To build the histograms, we discretise the intensity range of the images into equal-width bins — here we use 16-bin histograms and 8-bit images, resulting in 16 of the 255 possible intensity values falling in each bin. Each pixel coordinate then receives one bin value from each image by placing its intensity in the appropriate bin, and the resulting bin combination in a 2D histogram is incremented. This process

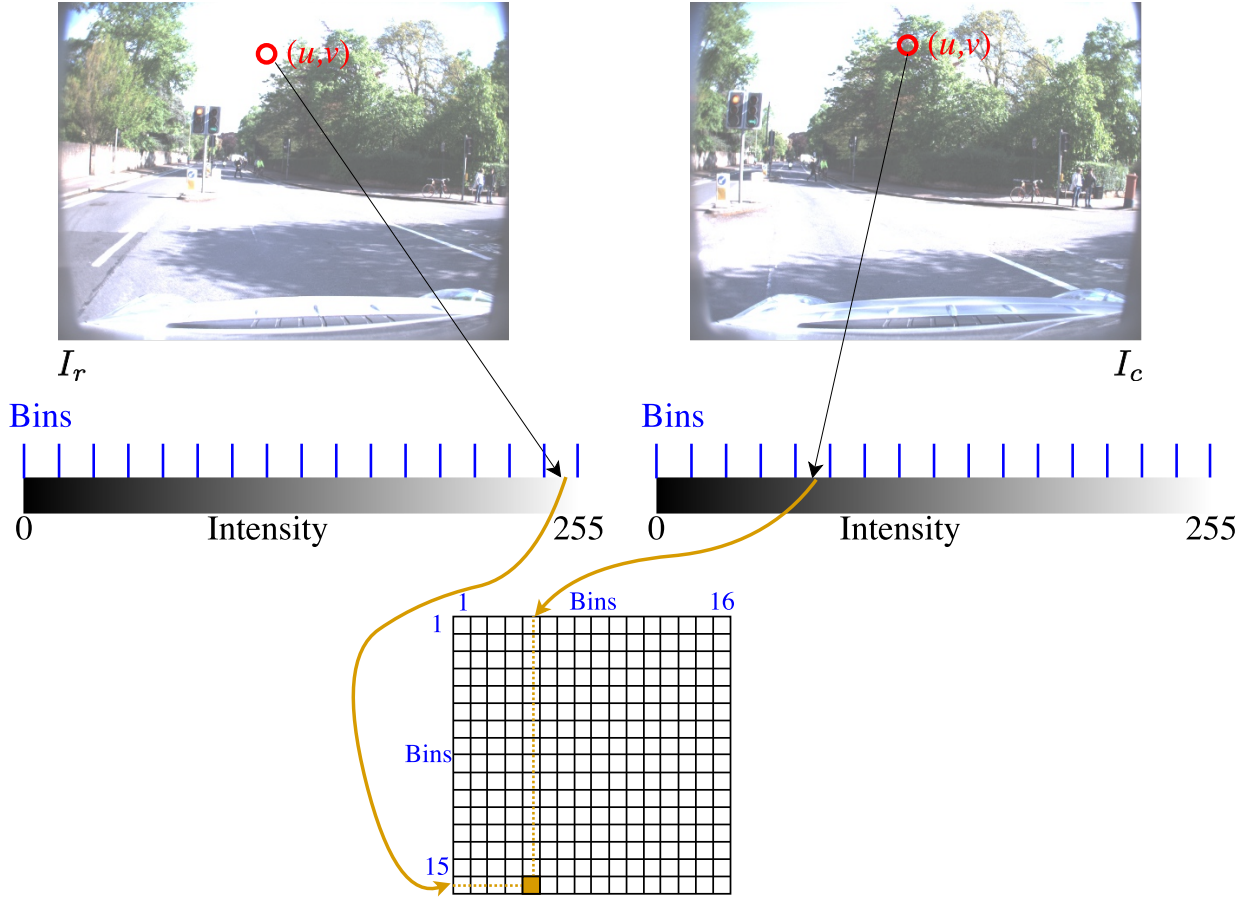


Figure 4.2: Building of joint histograms from two images. For each pixel coordinate (u, v) , the grayscale intensity is computed, and discretised into a small number of bins (in this example, there are 16 bins). The bins from the two images give a coordinate in a 2D histogram, shown here in orange, which is incremented. This is repeated for every pixel.

is demonstrated visually in Figure 4.2. NID is minimised for joint histograms in which there are a small number of bins with high values — this occurs when all the pixels in I_r that fall into a given range have a consistent value in I_r , indicating good alignment of single-colour regions.

Two identical images will give an NID of zero, as they both contain the same distribution — the value of $H(I_r, I_c)$ is the union of the information in the two images, so when they are identical we get $H(I_r) = H(I_c) = H(I_r, I_c)$. From Equation 4.3, the NID is equal to zero in this case. At the other extreme, with two entirely uncorrelated images, the information contained in each image is distinct, and so $H(I_r, I_c) \approx H(I_r) + H(I_c)$, and thus from Equation 4.3, we can see that

the NID will approach one.

Being an information-theoretic metric, NID obviates many of the problems inherent in using photoconsistency in a changing environment. Firstly, because NID depends only on the joint distribution of pixel values across the two images, it does not make the assumption that the same point in space will always have the same appearance — rather, it only assumes that regions with self-consistent appearance in one image will also have self-consistent appearance in another. For example, a green grass region in one image that has been covered in white snow in another will have different appearance, but will still contribute to just one histogram bin. This is equivalent to reordering the bins on one axis of the joint histogram, which does not affect the NID computation. Figure 4.3 gives a visual example of this effect. As the information contained in a histogram is invariant to bin ordering, this property makes NID invariant to many of the appearance changes seen outdoors.

In addition to pure appearance changes, NID also provides robustness to changes that are more structural in nature, such as dynamic obstacles. Because the appearance of a dynamic obstacle in an image is unlikely to correlate with the background, adding a dynamic obstacle to a scene only adds noise across the histogram. This will increase the NID value at the correct alignment, but as long as enough of the background is visible to obtain a signal, the noise doesn't affect the location of the minimum in the cost function.

Figure 4.4 shows the effect on image registration of applying various transforms to images from the synthetic Living Room dataset [84]. The NID cost correctly gives a distinct minimum at zero offset (the true alignment) for all degradations shown, whilst photoconsistency fails for all but the simplest of cases. We will discuss how to minimise the NID cost function across images in section 4.4.

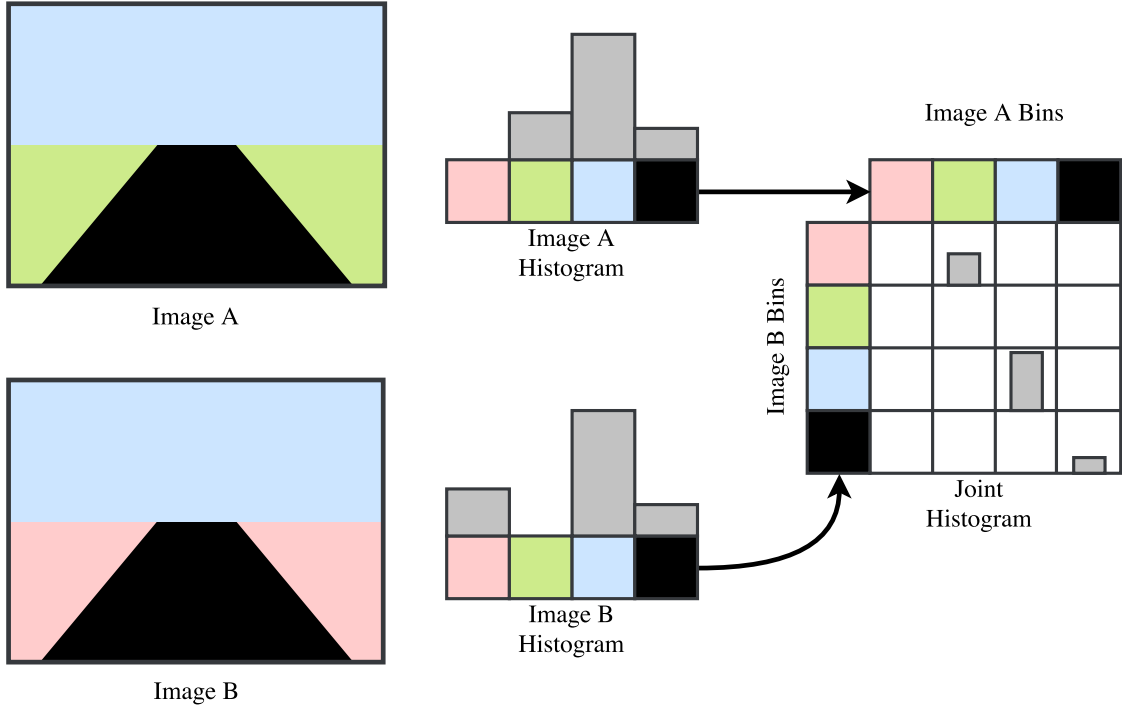


Figure 4.3: Robustness of NID to changing appearance. The difference in colour (green vs. red) in the two images is equivalent to relabeling the bins in the histogram. As the calculation of entropy takes no notice of the labels or order of the bins, this has no effect on the entropy, or consequently on NID. The two images here have an NID value of zero, despite having different appearance.

Alternative Information Measures

As may be deduced from the above definitions, NID is an information-theoretic metric, and is conceptually very similar to mutual information (MI), defined by:

$$\text{MI}(I_r, I_c) = H(I_r) + H(I_c) - H(I_r, I_c) \quad (4.6)$$

Mutual information is a widely used metric for image registration, particularly when registering images across modalities [63]. When compared to MI, however, NID has a couple of advantages for the task under consideration. Firstly, MI depends on the total amount of information (ie. total number of pixels) present in the computation. This introduces a bias towards greater overlap between the two images — given that the maps we generate have large gaps (eg. sky) with no information, this bias can cause problems [16]. NID, being a normalised metric, is

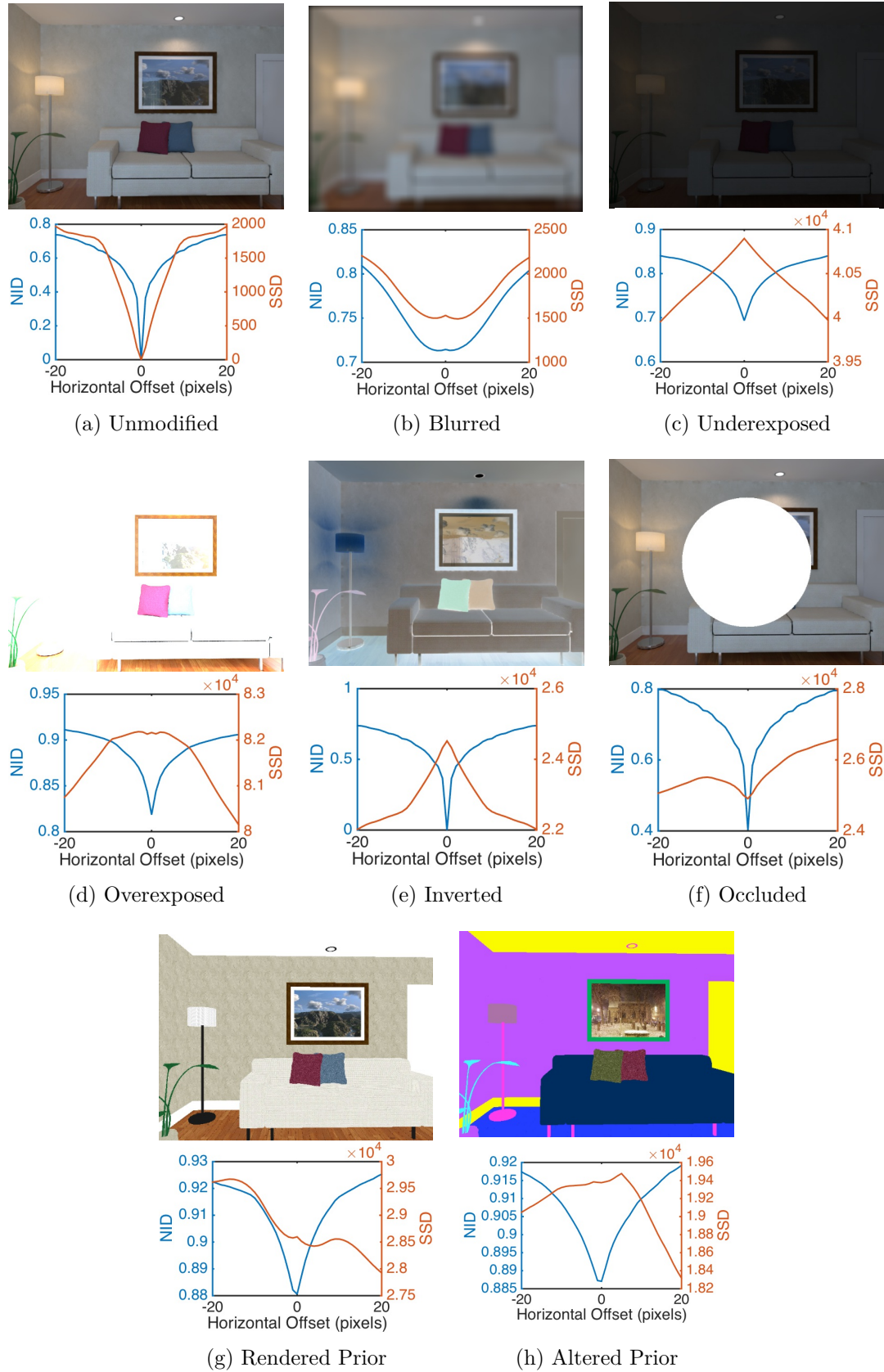


Figure 4.4: Effect of various image degradations on NID and photoconsistency (SSD) metric. Each graph shows the NID and photoconsistency compared to the original camera image in (a) with horizontal offsets applied. Whilst NID yields a distinct minimum at zero offset under all degradations, photoconsistency is only minimised at zero offset for the simplest of cases.

not subject to this bias. Furthermore, NID is a true metric [83]. This means that obeys the triangle inequality — $\text{NID}(A, C) \leq \text{NID}(A, B) + \text{NID}(B, C)$ — making it more suitable for use in an optimisation framework.

Another similar measure, known as normalised mutual information (NMI) was used in [16] to localise camera images in LIDAR reflectance maps. NMI is given by:

$$\text{NMI}(I_r, I_c) = \frac{H(I_r) + H(I_c)}{H(I_r, I_c)} \quad (4.7)$$

Combining Equations (4.3) and (4.7) yields a linear relationship between NID and NMI:

$$\text{NMI} = 2 - \text{NID} \quad (4.8)$$

Both NMI and NID are normalised quantities, and thus not dependent on the amount of overlap between the image and the map. The crucial difference between the two, however, is that NID is a metric (and thus obeys the triangle inequality), where as NMI is not. For the application in [16], in which pose was estimated in $\mathbb{SE}(2)$, with only three degrees of freedom, an exhaustive sampling approach within a defined region was used to find the pose with the maximum NMI. When estimating an $\mathbb{SE}(3)$ pose, however, the 6-dimensional search space is too large to be tractable for exhaustive sampling, and thus we make use of a gradient-based optimisation approach (see Section 4.4). In such a search framework, a metric is superior to a general cost or similarity function [85], thus leading to our choice of NID.

Differentiability

The NID metric presented here gives a useful metric for quantifying the similarity between two images, however, due to the use of histograms, it is not a continuous or differentiable function. Thus, in order to use NID in a gradient-based optimisation framework (see Section 4.4), the function needs to be modified to enable differentiation. We achieve this by making the contribution of each pixel

to the histogram a smooth continuous function across multiple bins, rather than an increment to a single bin per pixel.

To find the Jacobian of NID with respect to the six Lie algebra pose parameters, we make use of a chain rule. To begin with, equation 4.3 can be rearranged as follows:

$$\text{NID}(I_r, I_c) = \frac{H(I_r, I_c) - \text{MI}(I_r, I_c)}{H(I_r, I_c)} \quad (4.9)$$

where $\text{MI}(I_r, I_c) = H(I_r) + H(I_c) - H(I_r, I_c)$ is the mutual information between the two images. Applying the chain rule to equation 4.3 yields the following:

$$\frac{\partial \text{NID}(I_r, I_c)}{\partial \mathbf{g}_C} = \frac{\frac{\partial H(I_r, I_c)}{\partial \mathbf{g}_C} \text{MI}(I_r, I_c) - H(I_r, I_c) \frac{\partial \text{MI}(I_r, I_c)}{\partial \mathbf{g}_C}}{H(I_r, I_c)^2} \quad (4.10)$$

Here, $\mathbf{g}_C = \log_{\mathbb{SE}(3)}(G_C)$ is the Lie algebra representation of the transform G_C (see Section 2.4). We use this representation here, as we will be using these derivatives for optimisation — if we used the matrix representation directly, the optimiser would in general take steps that resulted in a matrix that did not represent a valid $\mathbb{SE}(3)$ transform.

Throughout an optimisation, a single live camera image is used, so the pixel values from I_c are constant. Thus, the entropy $H(I_c)$ is independent of $\mathbf{g}_C$, so we have $\frac{\partial H(I_c)}{\partial \mathbf{g}_C} = 0$. Therefore:

$$\frac{\partial \text{MI}(I_r, I_c)}{\partial \mathbf{g}_C} = \frac{\partial H(I_r)}{\partial \mathbf{g}_C} - \frac{\partial H(I_r, I_c)}{\partial \mathbf{g}_C} \quad (4.11)$$

Differentiating equations 4.4 and 4.5 then yields:

$$\frac{\partial H(I_r)}{\partial \mathbf{g}_C} = -\sum_{b=1}^n \frac{\partial p_r(b)}{\partial \mathbf{g}_C} (1 + \log p_r(b)) \quad (4.12)$$

$$\frac{\partial H(I_r, I_c)}{\partial \mathbf{g}_C} = -\sum_{a=1}^n \sum_{b=1}^n \frac{\partial p_{r,c}(a, b)}{\partial \mathbf{g}_C} (1 + \log p_{r,c}(a, b)) \quad (4.13)$$

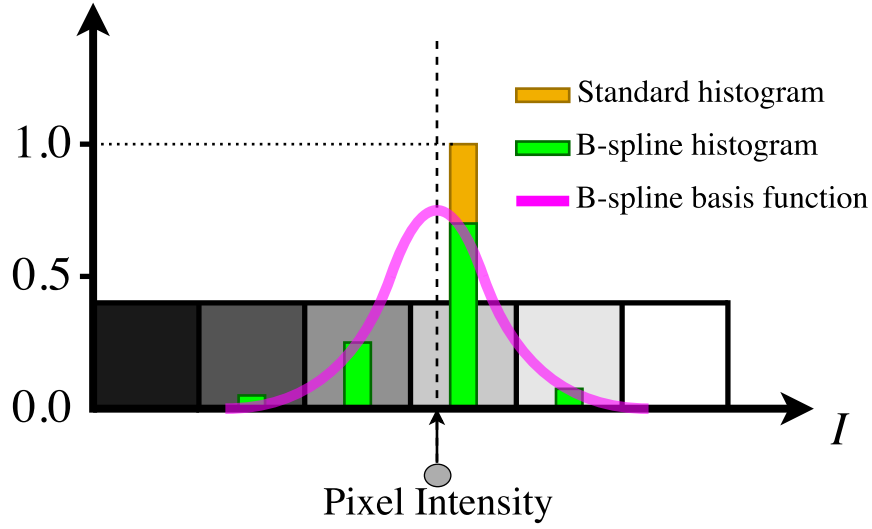


Figure 4.5: Histograms as a continuous function. With a standard image histogram, the pixel intensity is placed in a histogram bin, and that bin is incremented (the orange bar here). For a continuous B-spline weighted histogram, the increment is spread across adjacent bins in intensity space, with the amount added to each bin (green) determined by a cubic B-spline basis function (pink) centred on the actual pixel value. As a result, the histogram becomes a continuous function of the pixel values, with no discrete jumps.

This is where the differentiability of the function breaks down — the $p(\bullet)$ functions come from a joint histogram, where, for example:

$$p_r(b) = \frac{\text{count}_r(b)}{\sum_{b=1}^n \text{count}_r(b)} \quad (4.14)$$

where count indicates the count in a given bin in a histogram. As a result, the $p(\bullet)$ functions are discrete, and thus not differentiable.

The probability functions $p(\bullet)$ can be made continuous by altering the construction of the histogram, to produce bin values that are a continuous function of the image I_r . To do this, instead of each pixel contributing an increment of 1 to the bin into which its intensity falls, each pixel increments multiple bins, based on the exact intensity value. Conceptually, we construct a normalised cubic B-spline weighting function centred on the pixel intensity, and increment each bin by the value of that function at the bin centre. This is demonstrated in Figure 4.5.

More formally, in a standard histogram, a pixel $\mathbf{q} = \begin{bmatrix} u \\ v \end{bmatrix}$ produces the following

histogram update:

$$p_{r,c}(a, b) \leftarrow p_{r,c}(a, b) + 1 \quad \text{if } a = \mathcal{B}(I_r(\mathbf{q})), b = \mathcal{B}(I_c(\mathbf{q})) \quad (4.15)$$

where a and b are bin indices in the reference and camera histograms respectively, and $\mathcal{B}$ is a function that maps intensity values to bin indices. In our case, $\mathcal{B}$ simply discretises the intensity space into n bins. This results in exactly one bin being incremented per pixel.

In order to produce a continuous function, we instead update multiple bins in the histogram for each pixel as follows:

$$\forall a, b : p_{r,c}(a, b) \leftarrow p_{r,c}(a, b) + \beta(I_r(\mathbf{q}), I_c(\mathbf{q})) \quad (4.16)$$

Here, β is a weighting function that comes from the B-spline shown in Figure 4.5. We use a B-spline function here, as it is a simple polynomial, which is C^2 continuous, and uses only four support points. It is thus both differentiable and fast to compute.

Given that I_c is a constant throughout the optimisation, the histogram of I_c does not need to be differentiable, and thus no B-spline is required. We therefore only need to update a single bin per pixel in the camera image histogram. Expressing this as weights for Equation 4.16:

$$\beta(I_r(\mathbf{q}), I_c(\mathbf{q})) = \begin{cases} \alpha(I_r(\mathbf{q})) & \text{if } \mathcal{B}(I_c(\mathbf{q})) = b \\ 0 & \text{otherwise} \end{cases} \quad (4.17)$$

where α is a weight from the value of $I_r(\mathbf{q})$. The B-spline weighting is then applied to the reference image histogram:

$$\alpha(I_r(\mathbf{q})) = \frac{1}{6} \begin{cases} (1 - 3d + 3d^2 - d^3) & \text{if } a = \mathcal{B}(I_r(\mathbf{q})) - 2 \\ (4 - 6d^2 + 3d^3) & \text{if } a = \mathcal{B}(I_r(\mathbf{q})) - 1 \\ (1 + 3d + 3d^2 - 3d^3) & \text{if } a = \mathcal{B}(I_r(\mathbf{q})) \\ d^3 & \text{if } a = \mathcal{B}(I_r(\mathbf{q})) + 1 \end{cases} \quad (4.18)$$

$$d = I_r(\mathbf{q}) - C_{\mathcal{B}(I_r(\mathbf{q}))} \quad (4.19)$$

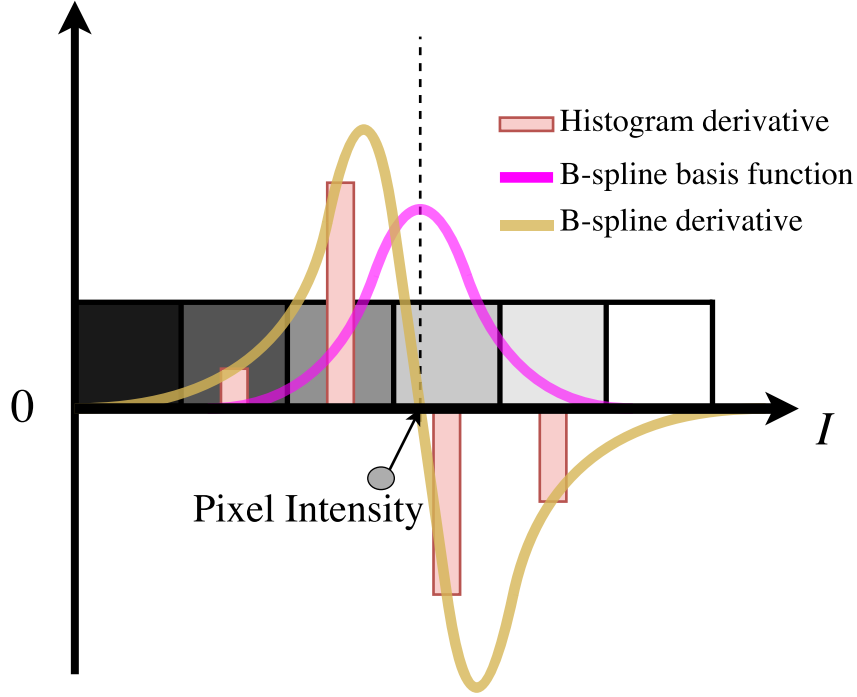


Figure 4.6: Derivative of histogram values with respect to pixel intensity. For each bin with a non-zero weight for a given pixel, we find the derivative of that bin with respect to the pixel intensity by differentiating the B-spline polynomial.

These are standard cardinal B-spline basis functions [86]. Here C_i is the center of the i th bin.

These functions are differentiable, allowing us to obtain an analytic value for the Jacobian of histogram values with respect to image appearance — ie. we now have $\frac{\partial p(\bullet)}{\partial I_r(\mathbf{q})}$ for each pixel $\mathbf{q} = \begin{bmatrix} u \\ v \end{bmatrix}$. These derivatives are shown graphically in Figure 4.6.

Applying the chain rule again to equations 4.12-4.13, we obtain:

$$\frac{\partial p(\bullet)}{\partial \mathbf{g}_C} = \frac{\partial p(\bullet)}{\partial I_r(\mathbf{q})} \frac{\partial I_r(\mathbf{q})}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \mathbf{g}_C} \quad (4.20)$$

This allows us to compute $\frac{\partial p(b)}{\partial \mathbf{g}_C}$ for each pixel in our input image. For each of the six Lie algebra pose parameters that make up $\mathbf{g}_C$, we construct a ‘histogram Jacobian’, which contains the sum of $\frac{\partial p(b)}{\partial g_i}$ over all pixels, in each histogram bin b . Thus for each bin in the joint histogram we have a probability value p , along

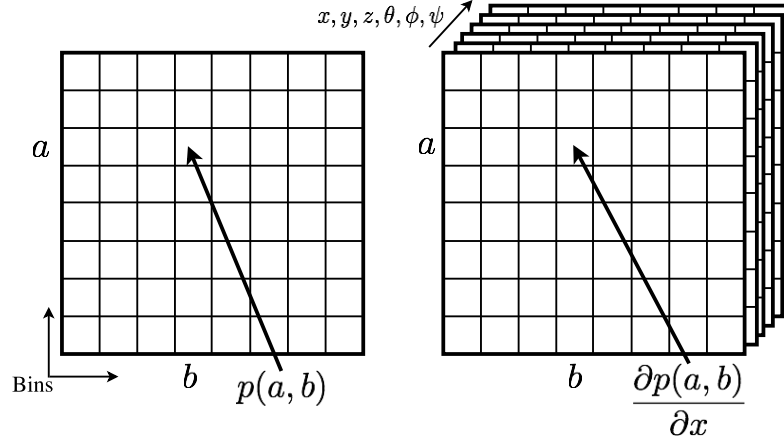


Figure 4.7: Jacobians of histograms. In addition to computing a joint histogram, which is the sum over all pixels of the B-spline weights for each bin combination (a, b) , we also compute six histogram Jacobians. These contain the sum, over all pixels, of the derivative of the B-spline weights for each bin combination, with respect to each of the six Lie algebra pose parameters, $x, y, z, \theta, \phi, \psi$.

with six derivatives that describe how p changes with variation in each camera pose parameter. This is shown in Figure 4.7.

The first term from Equation 4.20, $\frac{\partial p(\bullet)}{\partial I_r(\mathbf{q})}$, is obtained using the B-spline basis function derivatives as shown in Figure 4.6.

To find the Jacobian of pixel intensities with respect to screen coordinates $\left(\frac{\partial I_r(\mathbf{q})}{\partial \mathbf{q}}\right)$, we make use of a cubic B-spline interpolation to get a continuous, differentiable intensity surface. Firstly, we run a B-spline pre-filter over the synthetic image generated from the prior, as described in [87]. This prefilter produces an image $\hat{I}_r$, such that when B-spline interpolation is applied at pixel centres, the original values of the image I_r are obtained. Cubic B-spline interpolation can now be used on this image to smoothly interpolate intensities at sub-pixel locations. As it is a cubic B-spline, this interpolation surface is twice differentiable, allowing the Jacobian (and Hessian if required) of image intensity with respect to pixel coordinates to be evaluated, once again by using the derivatives of the polynomial B-spline basis functions.

The final term needed from equation 4.20 is that of image coordinates with

respect to $\mathbb{SE}(3)$ pose — ie. $\frac{\partial \mathbf{q}}{\partial \mathbf{g}_C}$. This is a purely geometric function, given by:

$$\frac{\partial \mathbf{q}}{\partial \mathbf{g}_C} = \begin{bmatrix} -1/z & 0 & u/z & uv & -(1+u^2) & v \\ 0 & -1/z & v/z & 1+v^2 & -uv & -u \end{bmatrix} \quad (4.21)$$

where the pose is expressed as a 6D vector $\mathbf{g}_C = \log_{\mathbb{SE}(3)}(G_C)$.

The Jacobian expressed in Equation 4.21 is purely geometric in nature, and can be computed independently for each pixel if we know its depth z . It can thus be easily computed at rendering time, alongside the synthetic image from the mesh map. To achieve this, in addition to generating a single ‘virtual’ image showing the expected appearance from a given pose, we also generate four additional three-channel (RGB) textures each containing the Jacobian for u or v with respect to a single degree of freedom in each channel. These twelve geometric derivative images are visualised in Figure 4.8.

4.3.3 Void Filling

The synthetic images generated from the map have large gaps in them, due to the fact that some areas (e.g. the sky) are not meshed. These regions are ignored in NID computations, but if we simply ran the B-spline prefilter over this raw image, the derivatives in the resultant interpolated surface would be dominated by the sharp changes at the edges of the non-meshed region.

To efficiently mitigate this effect, we downscale the image using OpenGL’s built in mipmapping operation, generating versions of the image at multiple resolutions. We then use values from the lower resolution images to fill the gaps in the higher resolution images. As the pixels in the lower resolution versions are taken from the mean of four or more pixels from the higher resolution images, this in effect blurs the edges of the non-textured regions. As a result, large derivatives at mesh boundaries are prevented.

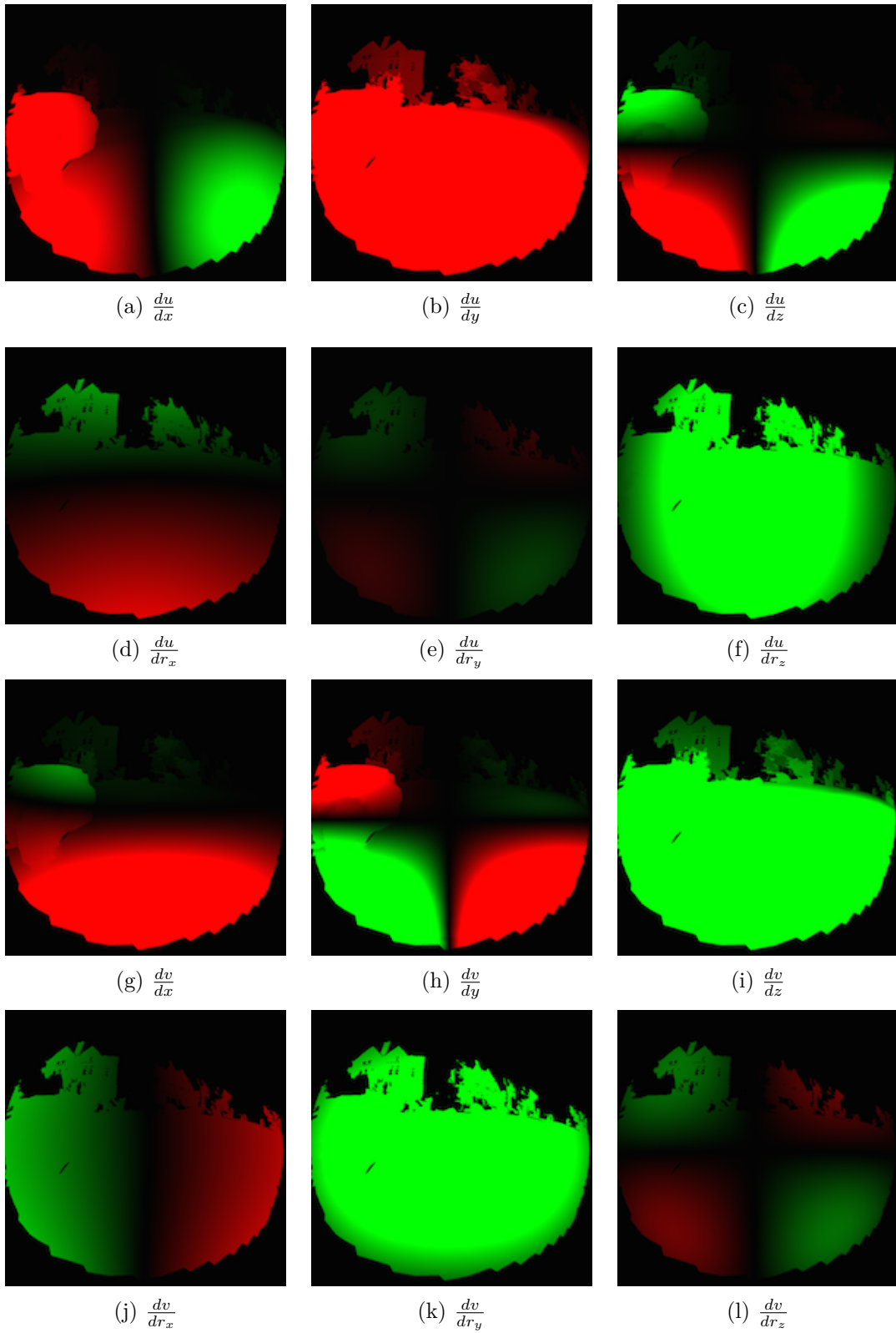


Figure 4.8: Derivative images generated during rendering. These show how the image coordinates of a point change with changes in camera pose. u and v are horizontal and vertical image coordinates respectively. Green indicates a positive derivative, red indicates negative. We use a frame with x into the screen, y to the right of screen, and z pointing down, and r_x , r_y and r_z represent rotations about the x , y and z axes respectively. Note that for visualisation purposes, the derivatives use two channels to display positive and negative values respectively — in the localisation pipeline only a single channel is required.

An example of this process is shown in Figure 4.9. Note that the gaps in the original (full resolution) image are still ignored in the NID computation — the smoothing is only used to prevent sharp derivatives at the boundaries.

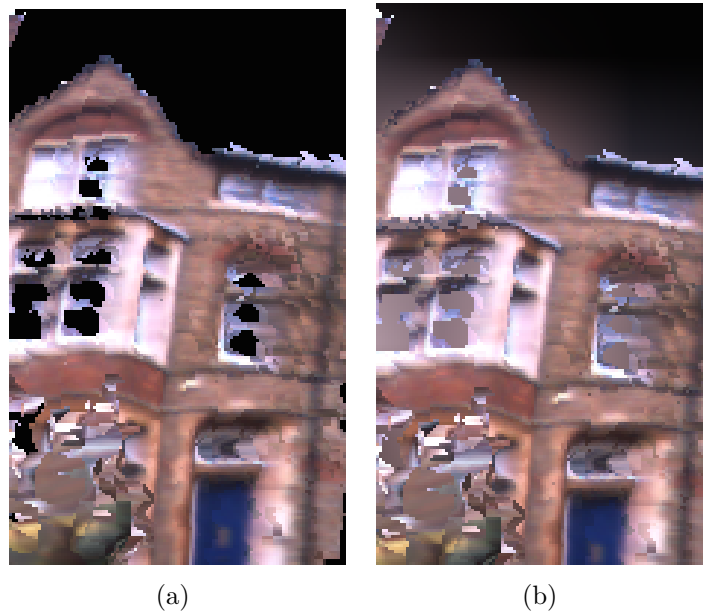


Figure 4.9: We render our synthetic images at two resolutions, and use the lower resolution version to fill in gaps in the higher resolution image. This prevents sharp changes at the edges of the mesh from dominating image derivatives. (a) A segment of the higher resolution image. (b) The same image filled in with a slightly lower resolution image, producing blurring around edges and filled-in holes in the windows.

4.3.4 Mesh Distortion

Some of the cameras we use for localisation have very wide-angle lenses. For example, Figure 4.10 shows an image from a PointGrey Grasshopper2 camera mounted with a fisheye lense, giving it a 185° field of view. Whilst such wide angle lenses allow observation of more of the scene, they do introduce some problems.

Attempting to undistort the images from these wide angle lenses results in large regions of the undistorted image being interpolated from a small number of pixels. This leads to image gradients that poorly reflect the actual appearance of the scene, as shown in Figure 4.11(a).



Figure 4.10: Image from a PointGrey Grasshopper2 camera showing the effect of a wide-angle fisheye lense.

Rather than undistorting live images, we thus chose to distort the synthetic image generated from the prior, to produce an image that should match the unprocessed image from an actual camera. As the mesh can be undistorted *before* textures are applied, we are able to obtain sharp textures in the distorted image. The distortion can be done on the GPU in the same vertex shader that carries out the original projection — it is simply an extra term in the projection transform. We thus never actually need to generate the undistorted projection of the mesh, and as a result, applying distortion adds almost no time to the rendering pipeline. Furthermore, building these images only requires a discrete lookup table obtainable by intrinsic calibration — no knowledge of the actual camera distortion function is needed. We obtained these models using the OCamCalib toolbox [88]. Figure 4.11(b) shows an example of an image produced from a distorted projection of a mesh.

To interpolate the discrete distortion look-up table, we again make use of cubic

B-spline interpolation with a pre-filter. As the look-up table is constant, this pre-filter can be computed offline. The distortion function is then added to the Jacobian expressed in equation 4.21 using the chain rule:

$$J_{\text{distorted}} = \frac{\partial \mathbf{q}_{\text{distorted}}}{\partial \mathbf{q}} J \quad (4.22)$$

where J is the Jacobian defined in 4.21, and $\frac{\partial \mathbf{q}_{\text{distorted}}}{\partial \mathbf{q}}$ is obtained from the derivatives of the cubic B-spline basis functions used for interpolation.

4.3.5 Image Blurring

One further transform is necessary to obtain usable, smooth derivatives for many datasets. Many textures have a repeating speckle-like pattern, such as that on the couch in Figure 4.4, which leads to large image derivatives between neighbouring pixels on the same surface. These hyper-local derivatives, however, do not actually convey any information about the structure of the scene. Nevertheless, they come to dominate the overall derivatives, producing a very small signal-to-noise ratio. To obviate this effect, we apply a 5x5 Gaussian blur to the rendered synthetic images. Figure 4.12 shows the effect of this blur on the NID derivatives.

4.4 Optimisation

Performing localisation via the cost function shown in Equation 4.1 is conceptually straightforward. In principle, it is a matter of finding the values for the 6 pose components that produce the minimum NID value. Unfortunately, however, the search space for the cost function is extremely large. Whilst a grid search approach has been shown to be feasible for localisation with similar cost functions in $\mathbb{SE}(2)$ [6, 16], the size of the search space increases exponentially with dimensionality. In $\mathbb{SE}(3)$ space, with six degrees of freedom, a grid search is infeasible.

To avoid a full grid search, with an infeasible number of cost function evaluations, We make use of a gradient-based optimisation method. Gradient-based optimisation



(a)



(b)

Figure 4.11: If we undistort camera images from a wide angle lense, as in (a), much of the resulting image comes from interpolating a small number of pixels. Instead, we distort the projected mesh before applying textures (b), and can match this distorted synthetic image directly with the real camera image.

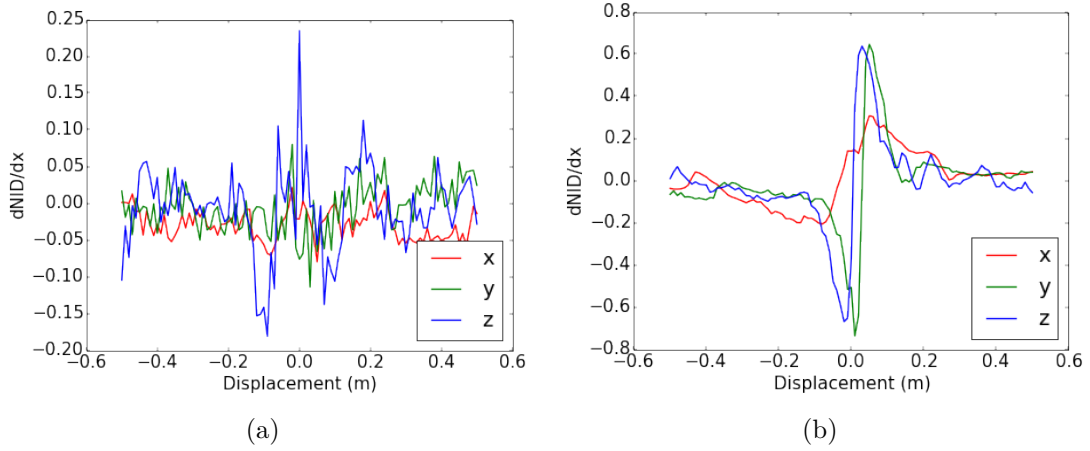


Figure 4.12: Effect of Gaussian blur on derivatives. These graphs show the derivative of NID with respect to translational offsets to the ground truth pose at an example point in the Handa living room dataset [84] (a) With no blur, the derivatives are dominated by the noisy textures. (b) A blur removes the local noise, smoothing the derivatives and giving a clear signal.

requires an initial estimate for a solution, which is then iteratively updated, guided by gradient information. Exactly how the estimate is updated varies, and a number of methods are discussed later in this section.

By sampling at locations selected by the local shape of the function (as described by derivatives), rather than at every point on a regular grid, a minimum may be found with far fewer samples of the cost function. This does, however, come at the expense of global optimality — any gradient-based method is susceptible to local minima.

The remainder of this section introduces and discusses several common gradient-based optimisation methods.

4.4.1 Gradient Descent

The simplest method for gradient-based optimisation is gradient descent. Intuitively, gradient descent updates an estimate by taking a step in the direction of lowest (most negative) gradient.

Given an estimate $\mathbf{x}_k$, gradient descent obtains the next estimate according to:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda \nabla f(\mathbf{x}_k) = \mathbf{x}_k - \lambda J \mathbf{x}_k \quad (4.23)$$

where f is the function being minimised, λ is a step size, and J is the Jacobian of f evaluated at $\mathbf{x}_k$ [85]. Various strategies exist for selecting the value of λ exist, but we won't go into any further details here, as gradient descent is not in any case suitable for the task at hand.

Due to the fact that it only uses first order derivatives, gradient descent will only choose a good update direction for very smooth, well-behaved functions. The NID between a real image and a map is anything but that. As a result, gradient descent converges very slowly, gets stuck in small local minima, or even diverges completely.

4.4.2 Newton's Method

Newton's method improves upon gradient descent by taking into account second order derivative information, rather than just the first-order gradient. The update equation here is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda H^{-1} \nabla f(\mathbf{x}_k) \quad (4.24)$$

where H is the Hessian matrix at $\mathbf{x}_k$.

Newton's converges faster (in terms of number of iterations) and more reliably than gradient descent, however the computation of the Hessian can be expensive. In performance sensitive applications, therefore, Newton's method is often infeasible.

4.4.3 Gauss-Newton Method

The Gauss-Newton method is similar to Newton's method, in that it makes use of second-order gradient information to improve convergence. Rather than computing the full Hessian, however, this method uses first-order derivatives to approximate the Hessian.

In effect, Gauss-Newton creates a local quadratic approximation of the function f at each iteration, and minimises that approximation. The update equation is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda \left(J^\top J \right)^{-1} \nabla f(\mathbf{x}_k) \quad (4.25)$$

Here, the Hessian matrix is being approximated by first-order term $J^\top J$. This approximation performs best when the number of residuals is greater than the number of variables.

4.4.4 Levenberg-Marquardt Method

The Levenberg-Marquardt (LM) method is essentially a blend of the gradient-descent and Gauss-Newton methods. The update equation for LM is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (J^\top J + \lambda_k I)^{-1} \nabla f(\mathbf{x}_k) \quad (4.26)$$

where λ_k is updated at each step depending on the success of the step in decreasing the value of f .

The idea of this method is that when in a region in which the local quadratic approximation from Gauss-Newton is not valid, λ is increased to behave more like gradient descent. Alternatively, where the step successfully decreases the function value, λ is increased to behave more like Gauss-Newton.

Unfortunately, as noted in the previous section, the approximation $H \approx J^\top J$ performs best when the number of residuals (the dimensionality of the output of f) is much greater than the number of variables. In fact, in cases for which there is only one residual — ie. f produces a scalar value — the LM method is equivalent to gradient descent [89]. This applies to our use case, where NID is a single scalar-valued function.

4.4.5 BFGS Method

As noted in the previous section, the LM method is not appropriate in cases with a small number of residuals relative to the degrees of freedom of the optimisation space. The case discussed in this chapter, of optimising a single-valued cost function, is an extreme example of such a case, and thus a different optimiser is called for.

The optimisation method known as Broyden-Fletcher-Goldfarb-Shanno (BFGS) [85] is one such alternative. Like LM, BFGS uses an approximation of the inverse Hessian, but unlike LM, it does not degenerate in cases with few (or only one) residuals.

The BFGS method iteratively updates its approximation of the Hessian, as well as the estimate of the minimum. The Hessian approximation B_k is updated according to the formula:

$$B_{k+1} = B_k - \frac{B_k \mathbf{s}_k \mathbf{s}_k^\top B_k}{\mathbf{s}_k^\top B_k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^\top}{\mathbf{y}_k^\top \mathbf{s}_k} \quad (4.27)$$

$$\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x} \quad (4.28)$$

$$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) \quad (4.29)$$

The initial estimate B_0 must be a positive definite matrix. In the absence of a better approximation, the identity matrix is often used as the initial estimate.

Given the Hessian estimate B_k , the update for the objective is updated in the same manner as Newton's method:

$$\mathbf{x}_{k+1} = \mathbf{x} - \lambda B_k^{-1} \nabla f(\mathbf{x}_k) \quad (4.30)$$

The step size λ is chosen at each iteration using a line search along the direction given by $-B_k^{-1} \nabla f(\mathbf{x}_k)$.

Wolfe Conditions for Line Search

The line search to select the value of λ in Equation 4.30 cannot in general exhaustively search along the selected update direction — this would be far too computationally expensive.

Rather than trying to find the *optimal* step size, one simply attempts to find a *good* step size. A good step is defined as one that meets the following conditions, known as Wolfe conditions [85]:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + c_1 \lambda \nabla f(\mathbf{x}_k)^\top \mathbf{s}_k \quad (4.31)$$

$$\left| \nabla f(\mathbf{x}_{k+1})^\top \mathbf{s}_k \right| \leq c_2 \left| \nabla f(\mathbf{x}_k)^\top \mathbf{s}_k \right| \quad (4.32)$$

where $0 < c_1 < c_2 < 1$.

Intuitively, the first condition, Equation 4.31, requires that the step taken must decrease the value of the function f . The larger the step taken, the larger the required decrease — this ensures that large steps are only taken when they lead to a correspondingly large improvement.

The second condition, Equation 4.32 requires that the magnitude of the *gradient* must also decrease. Intuitively, if the magnitude of the gradient were greater than the start point, it is likely to be further from a local minimum, where as a smaller gradient, combined with a smaller value, indicates proximity to a local minimum (where the gradient is zero).

Inverse Hessian Approximation

One further optimisation to the BFGS search is possible to avoid the computation of the matrix inverse B_k^{-1} in Equation 4.30. Instead of iteratively updating the Hessian approximation, we can rearrange Equation 4.27 to instead directly update the *inverse* Hessian:

$$B_{k+1}^{-1} = \left(I - \rho_k \mathbf{s}_k \mathbf{y}_k^\top \right) B_k^{-1} \left(I - \rho_k \mathbf{y}_k \mathbf{s}_k^\top \right) + \rho_k \mathbf{s}_k \mathbf{s}_k^\top \quad (4.33)$$

$$\rho_k = \frac{1}{\mathbf{y}_k^\top \mathbf{s}_k} \quad (4.34)$$

4.5 Evaluation

Many existing direct localisation or SLAM systems use an LM minimiser to optimise pose [15, 17]. For these cases, LM provides a simple solution that is fast to converge. These systems, however, have a residual per pixel, resulting in a much higher dimensionality of the residual space than the parameter space. As noted above, in the case of NID, with just a single cost value, LM is not expected to perform well. In this section, we evaluate the relative performance of LM and BFGS to validate the use of BFGS for NID-based localisation. As was noted previously, in the case of a single residual, LM is equivalent to gradient descent, so this comparison can also be considered to be a relative evaluation of gradient descent for this problem.

To evaluate optimiser performance, we made use of the Handa ‘living room’ dataset [84]. This is a synthetic dataset, with a full 3D model and photo-realistic renderings from virtual cameras along a trajectory through the room. This allows actual ground-truth poses to be used for evaluating localisation accuracy.

We used the implementations of BFGS and LM from the Ceres Solver package [90] to minimise the NID cost. For further details on the localisation framework, see section 4.6.

For each pose in the trajectory, we started the optimiser with an initial guess randomly offset from the ground truth pose. The offsets were uniformly distributed between 0.0m and 0.6m and between 0° and 10°. In addition to evaluating LM and BFGS, we also compared the results with BFGS using the built-in numerical differentiation in Ceres Solver, rather than the analytical derivatives developed in section 4.3.2. Table 4.1 presents the results of this comparison.

The results in table 4.1 confirm that, as expected, the best results come from using the BFGS minimiser with analytic derivatives. This can be explained by the quality of the update steps chosen by each optimiser. As noted above, with just a single residual, LM degrades to be equivalent to gradient descent. With a

Optimiser	RMSE (m)	p(error < 5cm)	RMSE (°)	p(error < 0.5°)
BFGS (analytic)	0.1692	0.7619	0.2497	0.9263
BFGS (numeric)	0.2553	0.1032	15.5739	0.4444
LM	6.0324	0.1247	38.6402	0.3515

Table 4.1: RMS errors and success probabilities for random initial offsets with different optimisers. The BFGS optimiser with analytic derivatives gives by far both the lowest errors and the highest success rates.

complex cost function like NID, the direction of steepest descent is not necessarily a good choice for updates, and leads to getting stuck in small local minima, or often diverging completely.

The BFGS optimiser with numeric differentiation performs better than LM, but is still far inferior to the analytic derivatives. This can be explained by the quality of the derivatives used — whilst the analytic derivatives are a true point value, the numeric derivatives are a coarse estimate from local sampling. It is thus not surprising that the analytic derivatives lead to better optimiser performance.

These results confirm that BFGS is a superior choice to the more commonly used LM optimiser for this specific problem formulation. Furthermore, the results indicate that the analytic derivatives developed in section 4.3.2 are appropriate for use in the optimisation.

4.6 Localisation

Now that we have a cost function and a method for optimising it, we can discuss the setup of our localisation framework. In essence, this involves generating a synthetic image from the mesh map, comparing it with the live image captured by the camera, and iteratively updating the camera pose until the two images align.

At each iteration of the optimiser, we render an image from a virtual camera placed at the current pose estimate in the scene. The geometry projection is identical to that described in Section 3.4.1, with textures being applied from the

images generated in Section 3.4.2. As mentioned in Section 4.3.2, we also generate textures representing the geometric derivatives for each pixel during this step.

To compute the NID between the synthetic image and the live camera image, we generate a joint histogram between the two images, along with six ‘derivative histograms’, representing the derivative of each histogram with respect to each pose parameter. From that histogram, we can compute both joint and marginal entropies and their derivatives, and thus NID and its derivatives. These values are then used to compute the BFGS update step.

4.6.1 Outlier Rejection

Whilst in most cases the optimiser is able to converge on a solution at or very close to the true pose, occasionally it will either fail to converge, or converge to an incorrect solution. The first of these cases is easily detectable from the optimiser. For the second case, we make use of odometry information to detect and reject such solutions.

Visual odometry provides poses in the form ${}^{C(t_{i+1})}_{C(t_i)}G$ — ie. relative poses between the camera at time t_{i+1} and time t_i . Whilst these poses can be expected to drift over time, and thus cannot be relied upon for long-term navigation, on a frame by frame basis they are highly accurate.

Given an estimated pose at time t_i , we can compute an estimate of the camera pose at time $t + 1$ by projecting forward the previous pose using odometry:

$${}^{C(t_{i+1})}_w\hat{G} = {}^{C(t_i)}_wG {}^{C(t_{i+1})}_{C(t_i)}G \quad (4.35)$$

We then use the final Hessian approximation B_k from the BFGS optimisation at time t_i , denoted B_i , and propagate it forward to estimate the uncertainty in ${}^{C(t_{i+1})}_w\hat{G}$:

$$\hat{\Sigma}_{i+1} = \frac{\partial {}^{C(t_{i+1})}_w\hat{G}}{\partial \mathbf{x}_i} B_i^{-1} \frac{\partial {}^{C(t_{i+1})}_w\hat{G}}{\partial \mathbf{x}_i}^\top \quad (4.36)$$

where $\mathbf{x}_i$ is a 6-vector $\mathbb{SE}(3)$ parameterisation of the camera pose at time t_i and $\hat{\Sigma}_{i+1}$ is the estimated covariance matrix for the camera pose at time t_{i+1} .

With this information, we can then compute the Mahalanobis distance between the pose estimated by the NID optimisation, and that obtained by projecting forward the previous pose with odometry:

$$\mathbf{e} = \log_{\mathbb{SE}(3)} \left({}^{C(t_{i+1})}_W \hat{G}^{-1} {}^{C(t_{i+1})}_W G \right) \quad (4.37)$$

$$s = \mathbf{e}^\top \left(\hat{\Sigma}_{i+1} + B_{i+1}^{-1} \right)^{-1} \mathbf{e} \quad (4.38)$$

where s is the Mahalanobis distance. By thresholding on the value of s , we're able to detect instances in which the localisation solution is inconsistent with local odometry, and thus reject outliers from the optimisation.

4.6.2 Implementation Details

Almost the entire localisation pipeline, as with the texturing system discussed in Section 3.4, is implemented in OpenGL. This allows access to the vast parallel computational resources available in a modern GPU, whilst simultaneously maintaining compatibility across almost any graphics hardware.

Note that the algorithms here are generally applicable beyond OpenGL, and indeed beyond the specific choice of GPUs as the processing platform. A field programmable gate array (FPGA), for example, could be used to implement the histogram building steps in hardware. Nevertheless, to enable the efficient implementation of our algorithms on commodity hardware, the specifics of the implementation are vital, and we present an overview here of how our reported performance was achieved. Regardless of the hardware choice, however, parallelisation of the algorithms is necessary for getting good runtime performance, and many of the design choices and abstractions outlined herein remain important.

For a small environment such as the Handa living room dataset, it is possible to load the whole map onto the GPU and keep it there for the duration of the session. For a large outdoor environment, however, memory limitations make this impossible. Instead, it is necessary to stream the map onto the GPU.

To stream the map onto (and off) the GPU without stopping computation, we create two separate OpenGL contexts on different threads, set up to share memory. As noted in Chapter 3, the maps we generate are represented as small self-contained ‘slices’ of a mesh. At startup, two chunks of memory are allocated on the GPU for map storage — an array of vertex arrays for mesh geometry and a 3-dimensional texture for slice textures. Throughout the session, a background thread determines which map slices are required based on the current pose estimate, and keeps track of where each slice currently loaded is stored in the vertex arrays and 3D texture. When new slice(s) are required, they are loaded into unused (or no-longer-required) vertex arrays and layers of the texture. Meanwhile, another thread performs the computation required for localisation using the other OpenGL context. As the localisation thread doesn’t perform any writes to the map memory, the threads do not interfere with each other.

To compute both the histograms and their derivatives, a large number of atomic operations are necessary. We achieve this using alpha-blending (see Section 2.3). Histograms are stored as a texture, with one pixel per bin. By ‘blending’ the increments from different image pixels into their respective bins, with a blending coefficient of 1, we effectively achieve atomic addition, whilst taking advantage of the optimised hardware and drivers designed for this task.

Table 4.2 shows a comparison of computation time for three different implementations of a joint histogram over 2 images — an OpenGL implementation using alpha-blending, an OpenCL implementation using naive atomic float operations, and a CPU implementation with a simple loop. These implementations all use float-valued histograms, as these are required for the B-spline weighted histograms,

Method	Mean Evaluation Time (ms)
OpenGL	4.323
OpenCL	59.198
CPU	15.501

Table 4.2: Mean computation time for 10 joint histograms, each of two random 640×640 RGB Images with an NVidia GeForce 650M GPU and 2.3GHz Intel Core i7 CPU. The OpenGL version takes advantage of built-in GPU features, and is by far the fastest.

but are purely producing a joint histogram with no extra computation per pixel (ie. no weighting, projection, derivatives etc.). Even in this simple case, the OpenGL implementation is the fastest — when extra computation is carried out per pixel, the problem gains more advantage from parallelisation, so the advantage of OpenGL over the CPU implementation can be expected to be greater. Unsurprisingly, the OpenCL implementation, which uses atomic floating point addition, is slow, as it causes every thread to pause whilst carrying out each write, as was noted in Section 2.3.4.

The derivative histograms, with six values per pixel (one for each degree of freedom), are stored as two RGB (three-channel) textures, as OpenGL does not support textures with more than four channels.

Once the histogram and derivatives have been computed, we transfer them back to the CPU for the actual entropy and NID computation. As the histograms are very small (we use 16×16 bins) and the operations needed very simple, the massive parallelism of the GPU doesn't provide much advantage. Any advantage there is from parallelism we found to be more than offset by the performance penalty of atomic operations, such that performing the computation on the CPU is faster.

4.6.3 Evaluation

To evaluate the performance of the localisation system described in this chapter, we ran it on two datasets:

- synthetic living room dataset by Handa et. al. [84]; and
- a subset of the RobotCar dataset by Maddern et. al. [23]

Between them, these two datasets provide the ability to evaluate the metric accuracy of localisation with ground truth poses and the robustness to both artificial and real-world degradation and alteration to the observed images.

4.6.4 Synthetic Data

We first evaluate the localisation system using the Living Room dataset by Handa et al [84]. This dataset contains a model of a living room approximately $10m \times 5.5m \times 3m$ in size, as well as photorealistic renders of a camera following a specified trajectory through the room. This provides ground truth for the camera pose at which each of the live images were taken. We preprocessed the data to produce a basic textured mesh of the room without any lighting information, in order to evaluate our system's ability to handle a low-fidelity map (e.g. from a 3D CAD model) with a different appearance to the camera images. Figure 4.13 shows an example of a virtual camera image and one of our renders, demonstrating the significant differences in appearance between the low-fidelity prior and the photorealistic live images.

We evaluate the robustness of the algorithm by applying various degradations to the camera images:

- Unmodified input images (Figure 4.4(a))
- Gaussian blur with a 15×15 and 25×25 pixel kernel (Figure 4.4(b))
- Underexposed image (Figure 4.4(c))
- Overexposed image (Figure 4.4(d))
- Inverting all colours in the image (Figure 4.4(e))
- Adding a randomly placed circular occlusion with radius 50, 100 and 150 pixels (Figure 4.4(f))
- Reduced bit depth of 2- and 4-bit colours



(a)



(b)

Figure 4.13: (a) Example image from the Handa Living Room dataset, and (b) a render from our map at the same camera pose.

Scenario	RMSE (m)	RMSE ($^{\circ}$)	p(error < 5cm)	p(error < 0.5°)
Unmodified	0.0410	0.5819	0.9725	0.9817
Gaussian blur (15 $\times$ 15)	0.0441	0.6274	0.9725	0.8073
Gaussian blur (25 $\times$ 25)	0.0589	0.6435	0.9541	0.5505
Underexposed	0.0407	0.5928	0.9462	0.9355
Overexposed	0.0450	0.6301	0.9541	0.9617
Inverted image	0.0400	0.5592	0.9633	0.9725
Circle occlusion (50px rad.)	0.0427	0.6146	0.9905	0.9905
Circle occlusion (100px rad.)	0.0450	0.6260	0.9558	0.9469
Circle occlusion (150px rad.)	0.0526	0.5634	0.9619	0.9429
2-bit colours	0.0395	0.5713	0.9725	0.9725
4-bit colours	0.0419	0.6118	0.9908	0.9908
Altered Prior	0.0397	0.5488	0.9633	0.8899

Table 4.3: RMS errors from an 882-image trajectory through the living room dataset. In all scenarios, the vast majority of poses converge to within 5cm and 0.5° .

In addition to the unmodified map (Figure 4.4(g)), we also test a map with radically altered colours (Figure 4.4(h)), to demonstrate the degree to which an NID-based system is invariant to the specific colours present in the scene.

For evaluation, we use a trajectory containing 882 poses. For each image, we start the localiser at the ground truth pose for the previous frame. Table 4.3 shows the RMS errors across this trajectory for each scenario.

As Table 4.3 shows, in all scenarios, the vast majority of poses converge to within 5cm and 0.5° of the ground truth. This demonstrates the ability of the NID-based localiser to handle significant appearance change between the map and the live camera images. Even the ‘unmodified’ map is far from being a photorealistic rendering of the scene.

The scenarios on which the localiser performs the worst are the Gaussian blurs and the largest occlusion. The Gaussian blurs have by far the highest probability of obtaining an orientation error greater than 0.5° , and the larger blur (25 $\times$ 25 kernel) has the largest RMS error in both orientation and translation. This is due to the blurring removing fine features (particularly edges), which widens the convergence basin and flattens the minimum, as shown in Figure 4.4(b). As the minimum is now spread across a large space, precise localisation is far more difficult.

Even with the largest Gaussian blur, however, the RMS errors are still reasonably close to the other scenarios, indicating that the blur has primarily caused a small universal reduction in precision, particularly in orientation, rather than causing localisation to completely fail for some frames.

The only other scenario with a translational RMS error greater than 5cm is that with the largest occlusion. This occlusion obscures more than 20% of the image, so it is not surprising that it leads to a reduction in precision. Unlike the blur effect, which affects the image universally, the occlusion is a localised degradation. Where the blur mostly affected orientation, the occlusion appears to have little effect on orientation, whilst mostly degrading translation performance. This can be explained by the fact that the occlusion is unlikely to cover all the distant features of the scene — ie. it is unlikely to obscure all the edges of the ceiling and walls. It will, however, in some cases obscure entire objects close to the camera. For orientation, points at a distance are equally useful for localisation, where as closer objects are required for precise translation estimation. Hence the large occlusion mostly affecting translation localisation.

Open Loop Localisation

The results in the previous section demonstrate the ability of our system to localise under large appearance changes and image degradations. We now evaluate the cumulative performance of our localiser over the course of a trajectory, rather than as independent localisation estimates. Here we make use of the same 882-image trajectory from the Handa Living Room dataset, initialising the camera pose at the ground truth for the very first image in the trajectory, and using the unmodified model as a map. For subsequent frames, we seed the localiser with the pose estimated for the previous frame, simulating open loop tracking. The results of this evaluation are presented in Table 4.4. The NID localiser is

RMSE (m)	RMSE ($^{\circ}$)	p(error < 5cm)	p(error < 0.5°)
0.0438	0.5802	0.9488	0.9671

Table 4.4: RMS errors open loop tracking on the Handa Living Room dataset. We are able to track across the whole trajectory, with the vast majority of pose estimates coming within 5cm of the ground truth values.

able to successfully track the whole sequence of images, with the vast majority of localisations within 5cm of the ground truth.

These results show marginally degraded performance compared to that reported in Table 4.3. The RMS error in translation has increased by a small amount, whilst the RMS orientation error has decreased slightly compared to the results from initialisation with the ground truth of the previous pose. Probabilities of errors below 5cm and 0.5° have decreased, but still constitute the vast majority of pose estimates. This indicates that the estimated poses from each frame are well within the optimiser’s convergence domain for the following frame, and that the final localisation solution is largely independent of the initialisation, as long as the initialisation lies within the convergence basin. As this is a pure localisation system, with the map as a static input, the errors do not compound over time as they may in a visual odometry or SLAM system.

4.6.5 Real Data

The evaluation presented in Section 4.6.4 indicates that the NID-based dense localisation system can successfully localise under significant changes in appearance between the map and live image. It doesn’t, however, test a scenario with real-world appearance changes on real data, or allow us to evaluate the magnitude of errors obtained in an autonomous driving application. We therefore evaluate the system here on a subset of the Oxford RobotCar Dataset [23].

The data we used contains pushbroom LIDAR and camera data from twelve traversals of a 9.3km route through Oxford over a period of six months. The data contains many different environmental conditions, including rain, snow, shadows,

sunshine, traffic and roadworks. The total distance is approximately 112km, with 450,000 images. Table 4.5 lists the specific datasets used, Figure 4.15 shows an image from each dataset, whilst Figure 4.14 shows the route taken.



Figure 4.14: Route used for outdoor localisation evaluation. The total route is approximately 9.3km. in length

A significant challenge when evaluating on a dataset of this size, with real-world data, is obtaining ‘ground truth’ poses with which to compare. Even the GPS-aided INS system present on the vehicle doesn’t solve this problem — the drift of the GPS constellation over such a long period introduces non-negligible translation offsets between different sessions.

To obtain a globally consistent ground truth, we performed a large-scale pose graph optimisation across all twelve datasets. Constraints in the optimisation came from visual odometry between adjacent camera frames, GPS/INS pose estimates and manually-corrected FAB-MAP [79] loop closures between datasets.

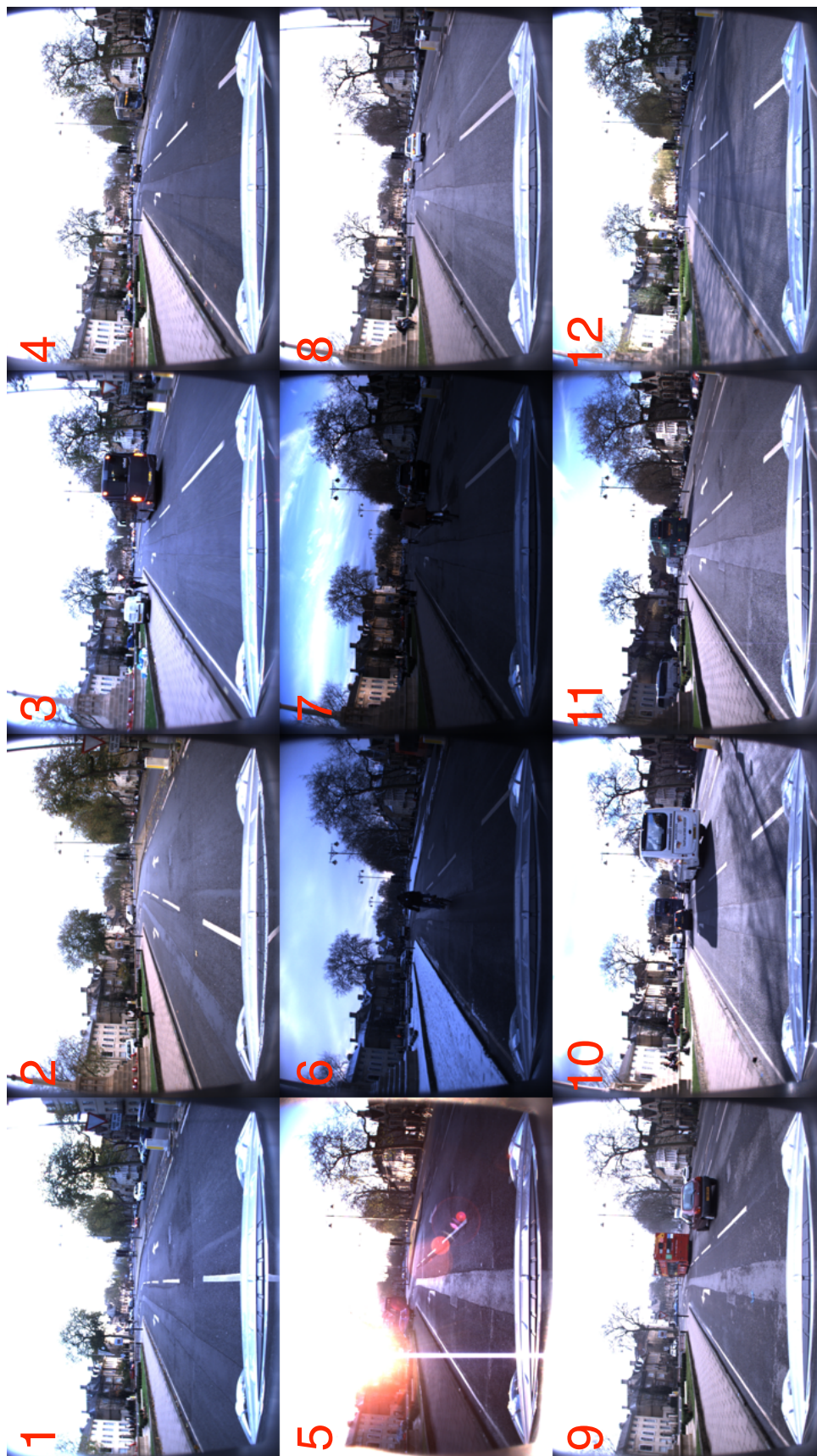


Figure 4.15: Camera images from approximately the same location in each of the 12 traversals used. As can be seen, these traversals capture a range of weather and illumination conditions.

Dataset	Date/Time
1	2014/11/18 13:20:12
2	2014/11/25 16:07:03
3	2014/12/02 15:30:08
4	2014/12/09 13:21:02
5	2014/12/16 09:14:09
6	2015/02/03 08:45:10
7	2015/02/20 16:34:06
8	2015/03/13 14:17:00
9	2015/03/17 11:08:44
10	2015/03/24 13:47:33
11	2015/04/17 09:26:25
12	2015/04/28 15:52:02

Table 4.5: Traversals from the Oxford RobotCar Dataset [23] used for evaluation in this chapter. Sample images from each traversal are shown in Figure 4.15. The datasets can be downloaded from <http://robotcar-dataset.robots.ox.ac.uk/datasets>.

Dataset	RMS Translation Error (m)	RMS Rotation Error ($^{\circ}$)	Outlier Rate (%)
2	0.2300	0.7391	3.46
3	0.2219	0.9912	2.76
4	0.2284	1.0600	8.06
5	0.1779	0.8422	2.76
6	0.3307	1.4037	7.44
7	0.2295	1.3751	5.86
8	0.2059	0.7620	1.72
9	0.2496	0.8301	9.58
10	0.2872	0.8480	4.82
11	0.2510	0.9568	5.79
12	0.2082	1.0451	4.06

Table 4.6: RMS errors in translation and rotation, for each dataset. Also shown is the percentage of rejected outliers, determined as described in Section 4.6.1. We achieve RMS errors less than 30cm and 1.5° on all datasets except number 6, in which the scene was covered in snow.

We built a textured map of the route using dataset 1, as described in Chapter 3. We then ran our localisation system for each image from the remaining 12 datasets. The results from this experiment are summarised in Table 4.6, whilst Figure 4.16 shows histograms of the error distributions in translation and orientation.

As Table 4.6 shows, all datasets except the one including snow (dataset 6) achieve RMS errors of less than 30cm and most had orientation errors of less than

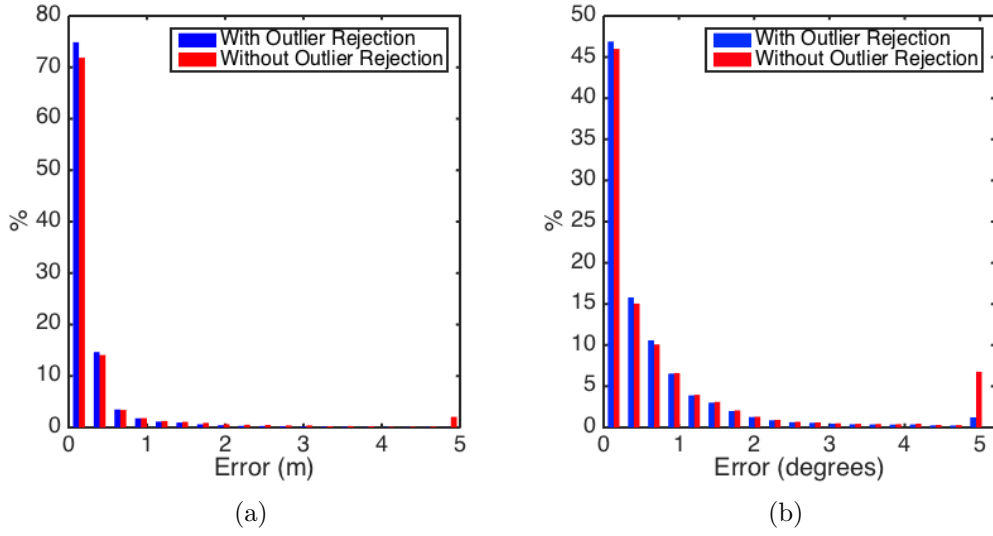


Figure 4.16: Error histograms for (a) translation and (b) orientation).

1°. It is not surprising that Dataset 6 has the largest RMS errors and a large outlier rate, as snow on the road can obscure many features, and give different areas of the scene the same appearance (eg. road, lane markings, and footpath may all appear white) in a way that other weather changes such as sunlight or rain does not. Nevertheless, the RMS errors are still below 35cm and 1.5°, and the outlier rate of 7.44% indicates that the system is still able to localise effectively for the vast majority of frames in this case.

Failure Cases

The NID-based localisation system evaluated here is able to localise successfully under large changes in appearance, however there are still a large number of failure cases, as indicated by the non-zero outlier rates in Table 4.6.

Most of these failures are due to either poor camera exposure or a large proportion of the image being obscured by dynamic obstacles. For example, Figure 4.17 shows two examples of images for which localisation has failed due to badly overexposed images. In these images there is not enough of the scene texture visible for localisation to succeed. This problem could possibly be solved by the use of

a higher dynamic range sensor, or by utilising a camera pointing in the opposite direction, and thus not directly into the sun.

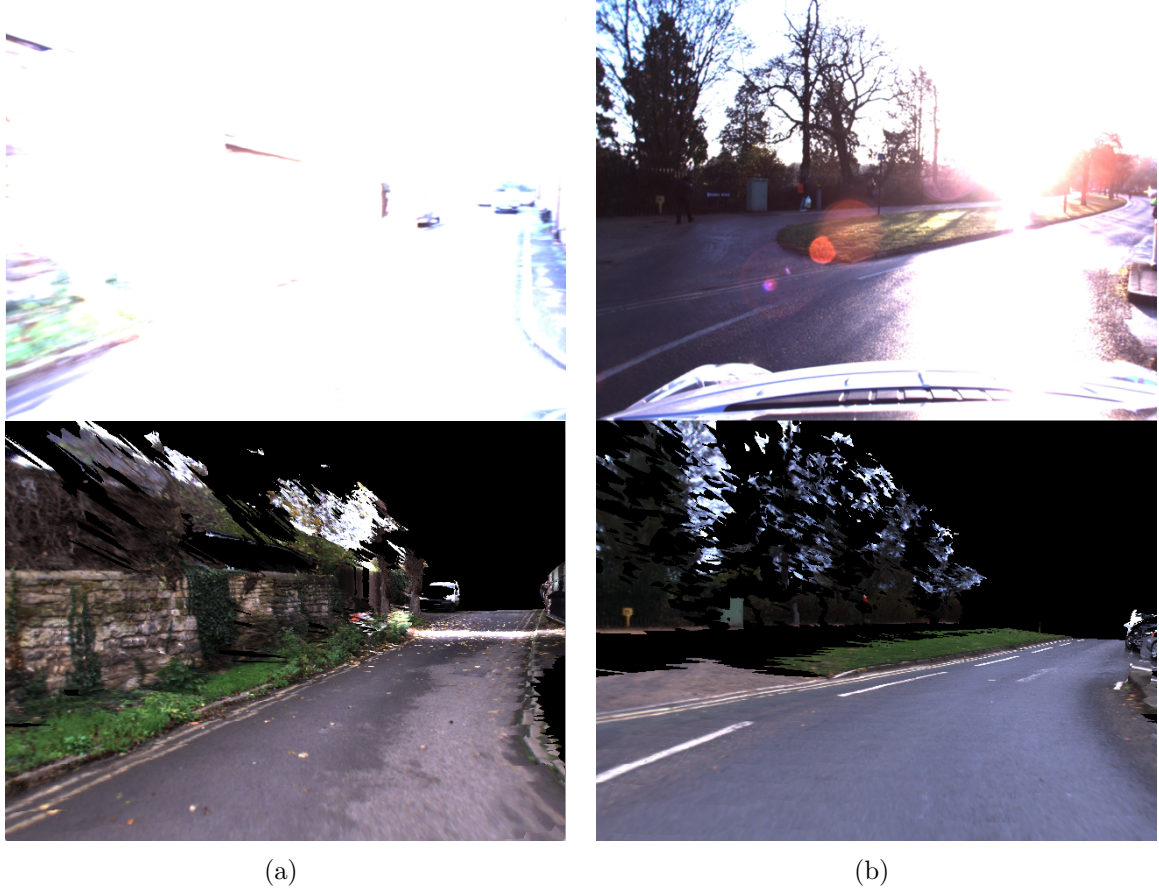


Figure 4.17: Localisation failures due to poor camera exposure. Top: camera image, Bottom: map. In these examples, overexposure of the camera image means that too few distinguishing regions in the image are discernable for localisation.

A second major failure case is when a large part of the scene is obscured by dynamic obstacles. This is exemplified by the two images in Figure 4.18. Whilst the localisation system can handle some dynamic obstacles, in both these cases, very little of the static structure of the scene is visible to the camera, resulting in very little localisation signal in the NID function. Both these images are taken from dataset 9, and it is mostly due to situations like this that the outlier rate in that dataset is high, as dataset 9 has the heaviest traffic of the traversals. Some such examples could be mitigated by classifying and removing dynamic obstacles from the

images, or by having a camera pointing in another direction. Fundamentally though, it will be very difficult to localise unless a significant amount of the background scene is visible — given these obstacles are only temporary, however, it would be possible to use odometry until a reliable localisation estimate can be achieved.



Figure 4.18: Localisation failures due to dynamic obstacles. Top: camera image, Bottom: map. Too little of the static scene is visible for the camera images to be matched against the map.

A final failure case is use of the system at night time. Figure 4.19 shows an example of a night time image for which localisation was not successful — localisation failed for the vast majority of images collected at night. The appearance of the scene at night is so different to that during the day, that the NID cost function is unable to compensate. This is due to the fact that the light sources are different to that present during the day, and in fact the primary light source

(the headlights) moves with the vehicle, making the distribution of intensity in the image very different to that at mapping time. This problem could possibly be solved by collecting a second map of the scene at night, at the expense of doubling storage requirements. Alternately, an entirely different localisation method could be used, by taking advantage of the very distinctive static light sources present at night — such a localisation method was demonstrated in [91].



Figure 4.19: Example of a night-time image for which localisation fails. Left: camera image, Right: map. The light distribution in the scene is very different to that during the day, partially due to a light source that moves with the vehicle.

4.6.6 Computation Time

We achieve localisation at a mean frequency of just over 6Hz with images 640×480 in size. This was evaluated on a PC with an AMD Radeon R295 GPU and Intel Core i7-4790 CPU with a clock frequency of 4GHz. The majority of the computation is performed on the GPU via OpenGL.

The localisation time consists of a mean of approximately 22 NID cost function evaluations per optimisation, with each cost function evaluation taking approximately 7.5ms. Figure 4.20 shows the make up of this evaluation time. The largest constituent parts are the rendering of synthetic images from the map, building of the joint histogram, and the application of the B-spline pre-filter.

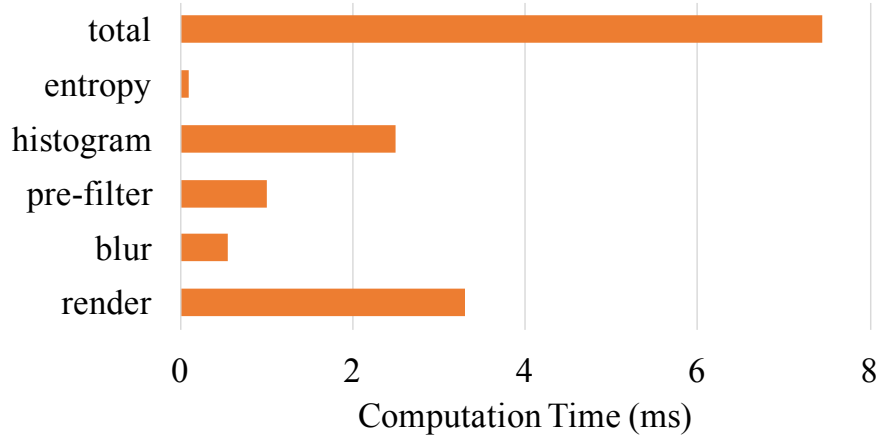


Figure 4.20: Time taken for the evaluation of the NID cost function on an AMD Radeon R295 GPU. Approximately 22 evaluations are required for the optimisation, giving a localisation rate of approximately 6Hz.

4.7 Conclusion

This chapter has presented a method for performing robust localisation in textured 3D maps. By using the information-theoretic NID metric, and optimisation with BFGS, we showed the system to be robust to many appearance changes, both artificial on synthetic data, and real changes in an outdoor environment over a period of six months. Whilst previous highly robust methods are not able to provide metric localisation, our system is able to provide a full $\text{SE}(3)$ pose estimate. On the outdoor localisation experiment, we obtained localisation errors of approximately 25-30cm, in a similar range to other visual localisation systems in use on autonomous vehicles.

The system developed here shows high reliability on all except night-time data. Through the use of this system, all that is needed for localisation is a single camera and a textured map. Whilst the map is derived from LIDAR data, only survey vehicles would require LIDAR sensors — other vehicles can localise using cameras only.

We also developed an outlier detection method based on consistency with local odometry, in order to improve the robustness of the localisation. If using visual odometry for outlier rejection, a stereo camera is needed, but if vehicle

odometry is used, only a monocular camera is required. Alternatively, a number of monocular visual odometry systems have been developed recently [43, 44], which could be used to enable a fully monocular system. This enables a very cheap localisation solution for road vehicles.

5

Extrinsic Calibration of Sensor Sets

Contents

5.1	Introduction	91
5.2	NID-based Extrinsic Calibration for Sensor Sets	93
5.3	Chequerboard-based Calibration	94
5.3.1	Target Detection	95
5.3.2	Extrinsic Calibration	102
5.4	Evaluation	107
5.5	Conclusion	113

5.1 Introduction

The previous chapters use a combination of different sensors — both LIDARs and cameras — to build and localise within maps. For this purpose, as well as many other multi-sensor applications, it is vital to have precise measurements of the relative position and orientation of the various sensors. Directly measuring the positions and orientations of sensors will not in general be feasible, as the origin of a sensor’s measurement frame will be internal to the sensor, and thus impossible to measure precisely. Having a system for obtaining these measurements is thus a prerequisite for any deployment of such methods.

Multiple systems for extrinsic calibration have been proposed, dealing with various different sensor combinations. These systems, however, have limitations that restrict their use in many applications. Many methods require human intervention to select correspondences between observations in different sensors [92, 93]. Accurate calibration requires many such correspondences, and as such this approach becomes intractable with a large number of sensors, or a large number of platforms to be calibrated. Furthermore, such an approach leads to a significant time burden in a research setting in which sensor configurations may be often changed.

Several automatic calibration approaches have been developed to ameliorate the need for manual data annotation. These systems, however, suffer from a lack of generality in which sensors they are able to calibrate. For example, [[94] proposes a system for camera to 3D LIDAR calibration, but requires dense 3D LIDAR data, precluding its use with newer, lower-density sensors such as the Velodyne VLP16 or with 2D LIDAR. More recently, a system for dealing with lower-resolution LIDAR data was proposed in [95], however this system requires stereo images — it does not work with monocular cameras. The OCamCalib toolbox [96] provides robust automatic calibration of both extrinsic and intrinsic parameters for cameras, but does not handle other sensor types.

The systems described above are all designed for finding the relative transforms between two sensors at a time. Whilst more sensors may be calibrated by composing relative transforms computed two sensors at a time, this will propagate errors from one calibration throughout the system.

This chapter presents a data-driven approach for obtaining extrinsic calibrations between sensors. Our system can handle data from both stereo and monocular cameras, including those with fisheye lenses. It is also designed to work with LIDAR data of varying density, from a Velodyne HDL64 providing 3D scans with approximately 300,000 points, to a 2D LIDAR scanner with fewer than 1000 points per scan. All sensors on a platform may be calibrated simultaneously, providing a

globally consistent estimate for each sensor, obtained using all available data. We demonstrate its accuracy on a vehicle with multiple sensors, with a comparison to a state-of-the-art motion capture system.

5.2 NID-based Extrinsic Calibration for Sensor Sets

We initially developed a calibration method that operated similarly to the NID-based localisation system presented in Chapter 4. As discussed in the previous chapter, NID is invariant to the specific colours that appear in a scene — this allows it to function across different imaging modalities. As a result, it can be used to localise a camera in a mesh coloured by the infrared reflectance from the LIDAR scanner. By ‘localising’ a camera within a LIDAR mesh, it is then possible to obtain an extrinsic calibration between the camera and LIDAR scanner. This system was presented in [21].

We found that the reflectance-coloured meshes we obtained were not enough to constrain the pose of the camera from a single image. The NID cost function was very noisy, and there was generally a shallow basin and no clear minimum. This made optimisation to the correct calibration difficult. By using several images separated from each other spatially, and linearly combining the NID from those different images, we found we were able to obtain a smoother, optimisable cost function with a clear minimum. This is demonstrated in Figure 5.1.

Unfortunately, the NID-based calibration system came with a lot of limitations, which made it unsuitable for many applications:

- The calibration was very sensitive to initial estimate, far more so than localisation. This was due to the reflectance providing less contrast or distinctive texture than image appearance, and due to the fact that poses from the cameras fed into the structure of the mesh — if the calibration

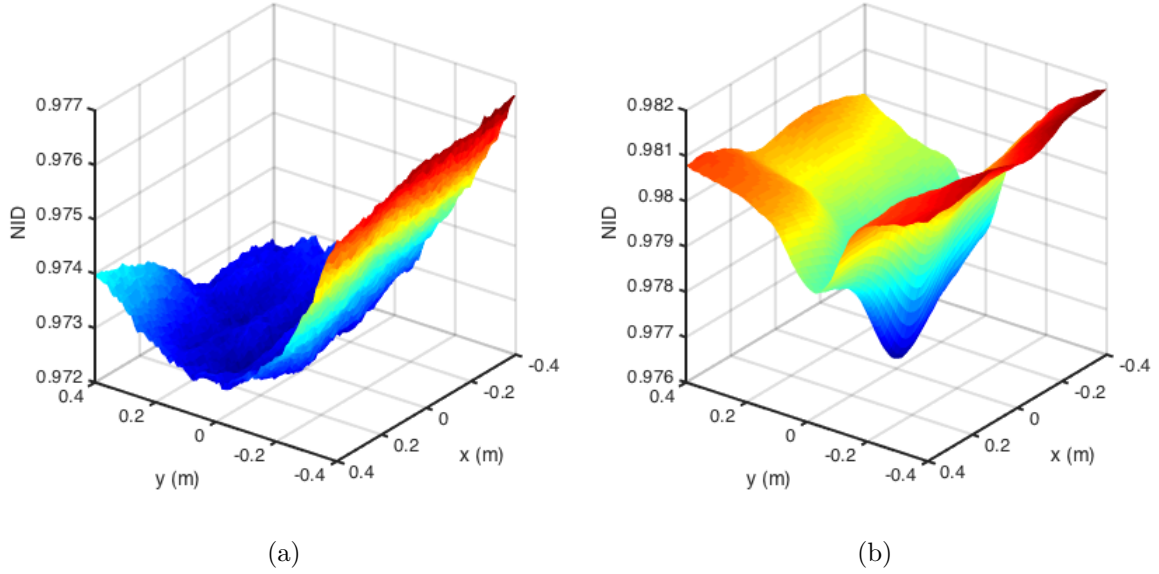


Figure 5.1: NID vs calibration offset in x and y translation with a mesh coloured by laser reflectance. (a) Evaluated using a single image. (b) Average NID value across 10 spatially separated images. Many images only weakly constrain the solution, but by averaging across multiple images, we get a clearer minimum, as shown in (b).

estimate was poor, then the structure of the mesh would be distorted, which would harm the optimisation.

- The optimisation was very sensitive to the specific images used and the structure in the scene.
- The system was limited to camera-LIDAR calibration — there was no way to perform camera-camera or LIDAR-LIDAR calibration

For these reasons, we moved towards a calibration system based on mutual observations of a specially designed target, rather than attempting calibration from general data collection.

5.3 Chequerboard-based Calibration

The system for calibration we've developed [is](#) relies on the detection of specially-designed targets in multiple sensors. With many detections of these targets, and

estimations of the target poses relative to the sensors, the relative poses of the sensors can be obtained via a cost function minimisation. This system depends on collecting data on a static vehicle, with a calibration target being waved around in view of the sensors.

This method consists of two distinct steps:

1. Target Detection
2. Extrinsic Optimisation

We now describe each of these steps in detail.

5.3.1 Target Detection

The first step in the calibration process is the detection of calibration targets in the various sensors. Because this calibration takes place offline, and requires a target to be manually moved around, the form of the target can be whatever suits the process best. For this purpose, we use a chequerboard pattern on a large, flat board as the calibration target. The chequerboard pattern is simple to generate, and the large contrast between white and black squares makes the corners straightforward to detect in camera images. Furthermore, a flat board moving in an environment that is otherwise mostly static can be detected in LIDAR data by fitting a plane to the non-static elements of the LIDAR scans.

Figure 5.2 shows an example camera image of a target used for calibration, taken from a PointGrey Blackfly camera.

Cameras

Detecting targets in camera images is quite simple thanks to the distinctive design of the chequerboard. As can be seen in Figure 5.2, the white and black squares have high contrast, and are distinctive from any other patterns present in the environment. We detect the target using the built-in `findChessboardCorners` function in the

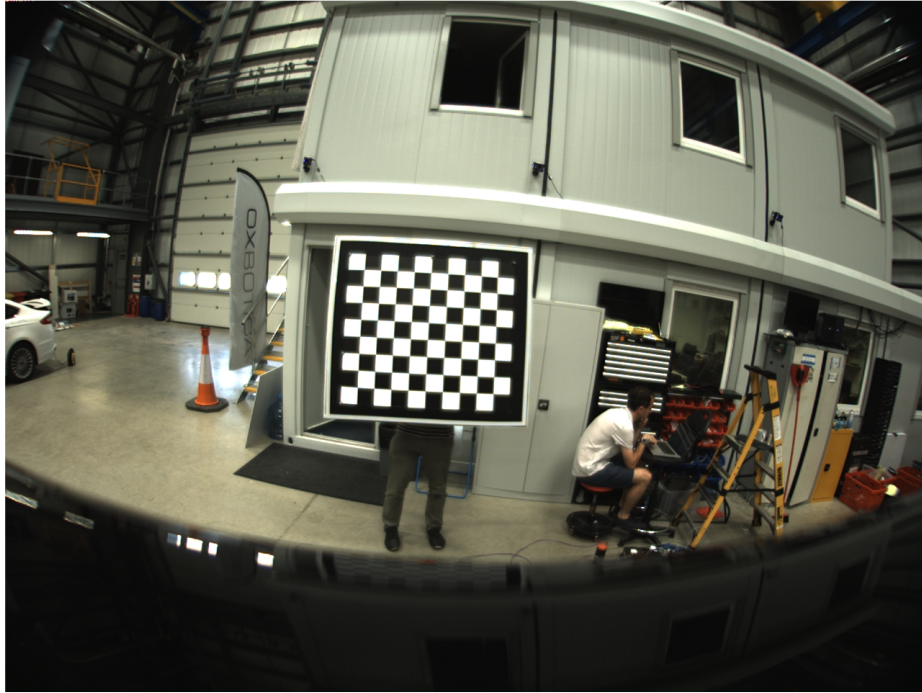


Figure 5.2: Camera image of chequerboard target used for calibration.

OpenCV library [97]. This works by creating thresholding and binarising the image, then using a dilation to separate the black squares. The contours on the boundaries of the black regions are then found. Contours are filtered by requiring both a local quad shape, and a regular global pattern, as expected for a chequerboard [97]. Figure 5.3 shows an example of a detected chequerboard in a camera image.

Lasers

Whilst the chequerboard pattern has a very distinctive appearance in camera images, the pattern is not quite so easily detectable in LIDAR data. Even with very high resolution 3D LIDAR, the point density on the surface of the target is not high enough to detect and localise the target using the pattern of reflectance on the chequerboard. Instead, we make use of the geometry of the target for detection.

Detection of targets in LIDAR data is a two step process — first we generate hypotheses for targets, then validate them using a number of heuristics.

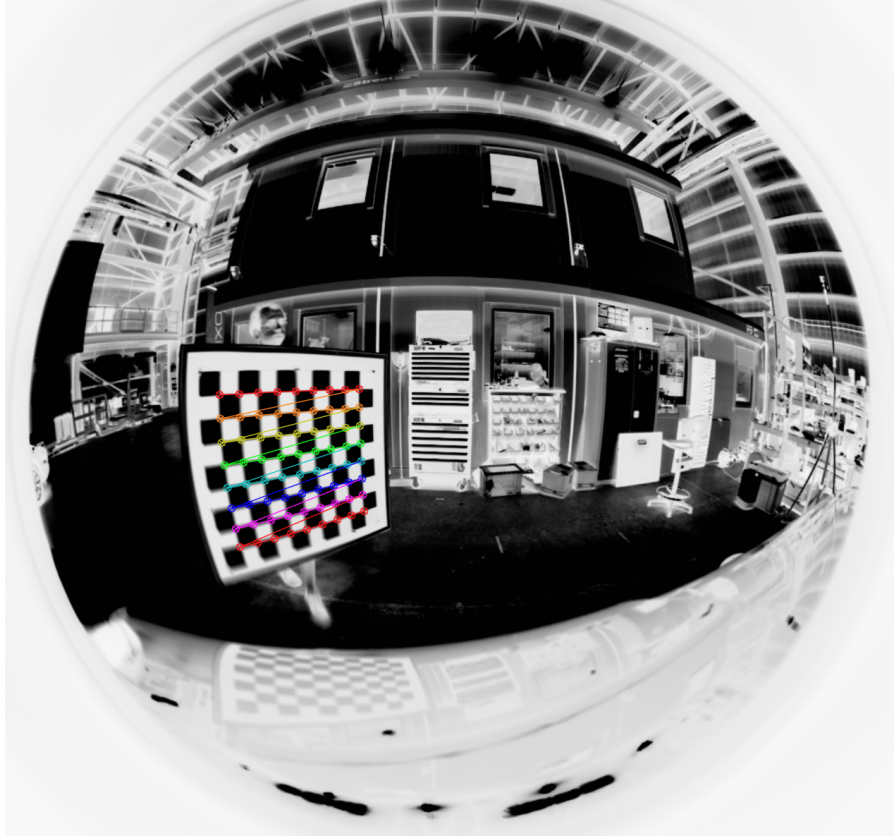


Figure 5.3: Chequerboard detection in a camera image. Note that the image is inverted, as the OpenCV function expects a white border around the pattern, where as this board had a black border.

To detect targets, we make use of the fact that most of the scene observed by the LIDAR during calibration will be static, whilst the target is moved around. We therefore ignore all points in the scene which are part of the static background, and attempt to detect the target only in the points we determine to be dynamic. To determine the set of dynamic points, we take a pair of scans separated by a minimum time threshold. We build a kd-tree [98] using all the points in the first scan. The kd-tree is a data structure with nodes representing points in k -dimensional space, where branching partitions the space in one dimension — this structure allows for fast nearest-neighbour lookup in a set of points.

With a kd-tree of points from the first of our pair of scans, we perform a nearest neighbour lookup for each point in the second scan. Those points whose distance to their nearest point from the first scan exceeds a threshold are considered to be

dynamic. The result of this is shown in Figure 5.4(a).

From the subset of dynamic points, we wish to detect the target by finding a plane (as the target is a flat board) that approximately matches the geometry of the target. We do this using a framework similar to Random Sample Consensus (RANSAC). The basic algorithm is shown in Algorithm 5.1.

Algorithm 5.1: LIDAR target detection

```

Input: Dynamic Points
Result: Target plane
max_inliers  $\leftarrow$  0;
for  $i \leftarrow 1$  to 50 do
    sample  $\leftarrow$  randomly sample 3 points from dynamic subset;
    plane_tmp  $\leftarrow$  FitPlane(sample);
    inliers  $\leftarrow$  FindInliers(plane_tmp);
    if Count(inliers) > max_inliers then
        if ValidateHeuristics(plane_tmp,inliers) then
            plane  $\leftarrow$  plane_tmp;
            max_inliers  $\leftarrow$  Count(inliers);
        end
    end
end
success  $\leftarrow$  max_inliers > 0;

```

The algorithm presented in Algorithm 5.1 is very similar to standard RANSAC plane fitting, except instead of simply choosing the plane with the highest inlier ratio, we additionally apply some heuristic tests to remove spurious detections (the `ValidateHeuristics` function). These spurious detections mostly fall into three categories, examples of which are shown in Figure 5.4:

- Background planes appearing dynamic because of moving occlusions. For example, if something (including the real target) moves in front of a wall, then parts of the wall that are not visible in one frame will be visible in the next. Those parts of the wall then appear dynamic, and fit very well to a plane, so can be spuriously detected as a target.

- Other dynamic objects in the scene. For example, if a vehicle — especially a bus — drives past during calibration, this may be picked up by the LIDAR, and detected as a planar region of dynamic points.
- People appearing planar. If a person is facing the LIDAR directly, and holding a target low to the ground or high up, the LIDAR may only detect the front surface of the person, which can appear planar.

We filter out these spurious target detections by thresholding on a number of heuristic values. Firstly, we place a threshold on the minimum number of inliers detected. This threshold is dependent on both the specific LIDAR being used (as different LIDAR models have different point density), the size of the board, and the distance from the sensor to the detected target — the density of points decays quadratically with distance from the sensor. This limit ensures that planar regions much smaller than the actual target are not accepted as target detections.

The second heuristic we use for filtering target detections is a limit on the distance between subsequent target detections. This essentially places a smoothness constraint on the trajectory of the target, based on the fact that there is only a single target with a continuous trajectory. If no targets are detected for more than ten seconds, this constraint is removed to allow for the movement of the target during this interval.

The final heuristics we use for filtering are based on the specific geometry of the target board, in order to reject detection of other flat surfaces like walls or the ground. We place a threshold on two ratios as shown in Figure 5.5. If a detected plane is the target, the rectangular shape of the target will cause most inliers to fall within the region A, with some in region B, and none in region C. If the detected surface is a wall, however, all three regions will have similar point density. Placing a threshold on these ratios, therefore, allows us to reject larger planes.

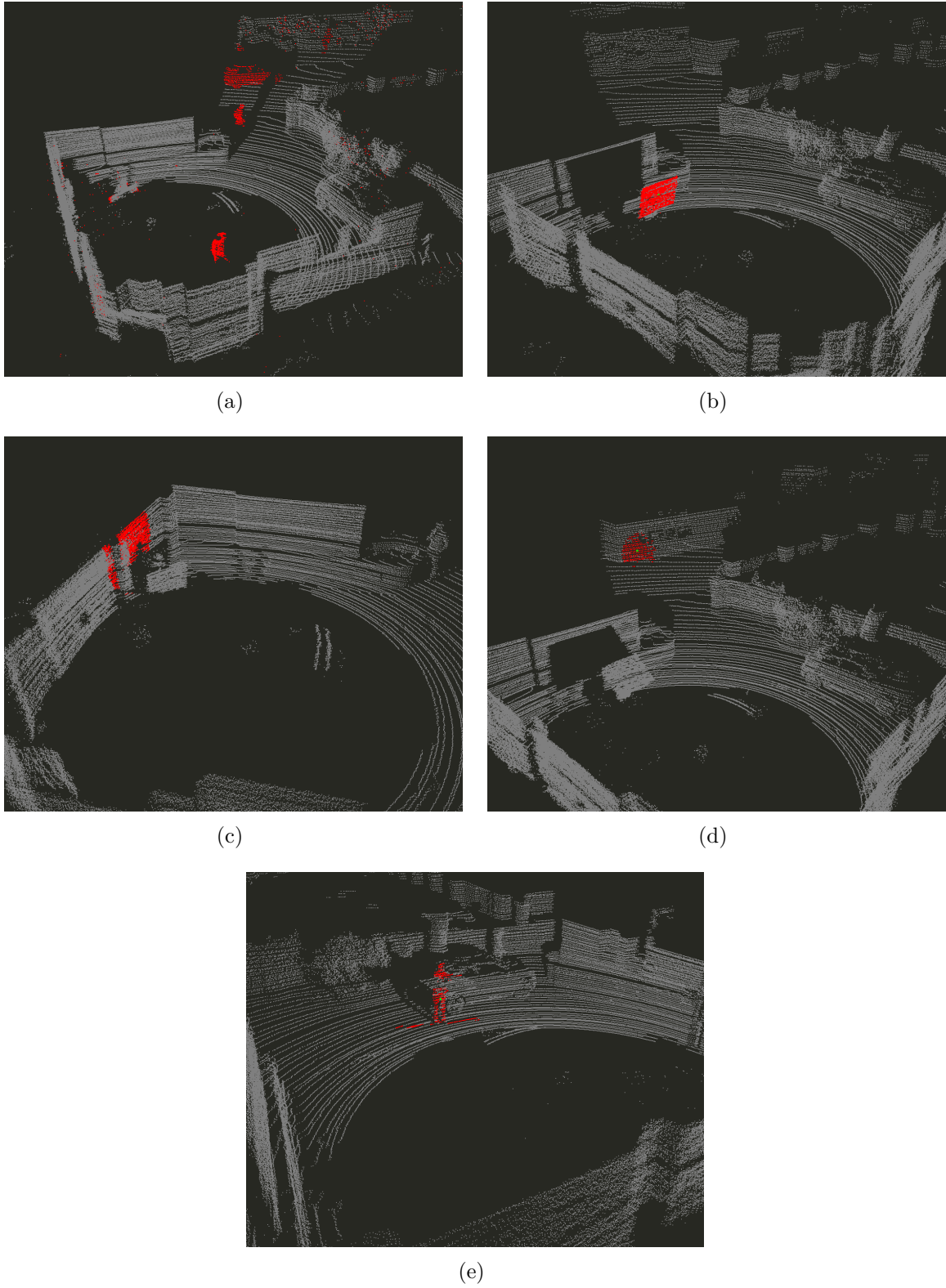


Figure 5.4: Examples of LIDAR calibration target detections and false positives. (a) Dynamic points highlighted in red. (b) Correct target detections (inlier points in red). (c) Part of the wall detected as a target. (d) A bus driving past appears similar to a target. (e) The front of a person appears to be planar.

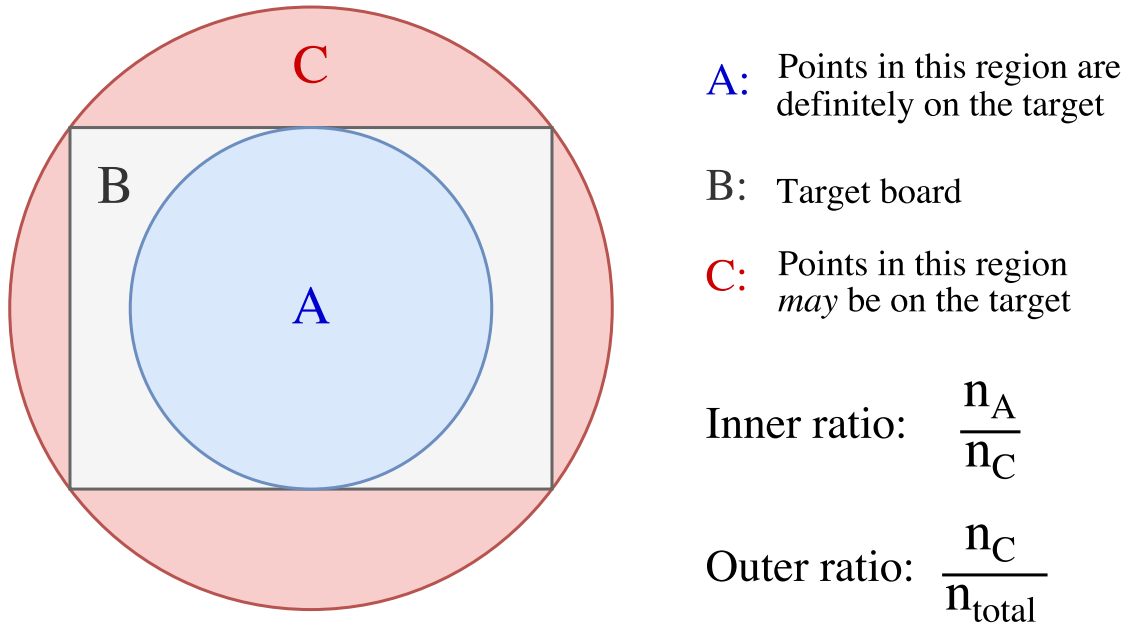


Figure 5.5: Ratios for target detection. If a fitted plane is indeed the target board, both these ratios would be expected to be low — there would be no points outside *C*, and the area of *C* outside *B* would be empty. If, conversely, a larger plane such as a wall or a bus were detected, the point density in all these regions would be approximately uniform, so both ratios would approach 1.

Note that these heuristics were determined to give good performance, but are not critical to the rest of the calibration pipeline. A more sophisticated target detector based on data-driven learning could be substituted in with no downstream changes.

If a target passes all of these heuristic tests, it then becomes a candidate within the RANSAC framework in Algorithm 5.1. After the RANSAC loop concludes, we are left with the best candidate plane for the target. At this point, we don't fit a final plane to all the inliers for the selected candidate, as the actual plane parameters don't particularly matter. All the target poses are optimised later in a joint optimisation along with the sensor poses — the plane fitting at this stage is mostly used to *detect* the targets in the LIDAR data, in order to associate measurements with the calibration target, not to localise the targets.

If the laser in use is a 2D LIDAR, rather than 3D, the detection method is almost identical. The only point of difference is that instead of attempting to fit

a plane, we fit a line to the dynamic points. The threshold values for each of the heuristics are also adjusted for the lower dimensionality and lower point density.

5.3.2 Extrinsic Calibration

As stated at the beginning of the chapter, the key information required out of extrinsic calibration is the relative $\mathbb{SE}(3)$ poses between the various sensors on a platform. This can be obtained approximately by either external measurements, or from computer aided design (CAD) models, but for precise applications these measurements are unlikely to be accurate enough, as the precise location and orientation of the origin of the measurement frame is difficult to measure externally, can change over time, and can be slightly different in each individual sensor. We therefore use the detections of the targets obtained in Section 5.3.1 in a joint optimisation to determine the relative poses of the sensors.

We formulate the extrinsic calibration problem as a cost minimisation over sensor poses and co-observed target poses. As input to this optimisation, we have a list of sensors:

$$\mathcal{S} = \{s_i\} \tag{5.1}$$

and a list of target observations:

$$\mathcal{O} = \{o_j\} \tag{5.2}$$

We combine the individual target observations o_j into co-observations by finding those observations from different sensors whose timestamps fall within a threshold τ of each other:

$$\mathcal{T} = \{t_k\} \tag{5.3}$$

$$z_k = \{\{o_a, o_b\} \text{ s.t } |\text{timestamp}(o_a) - \text{timestamp}(o_b)| < \tau\} \tag{5.4}$$

where the observations o_a and o_b can be from different sensor modalities. We set the value of τ to be half the period of the highest rate sensor. The set $\mathcal{Z}$ contains observations of the target at approximately the same time, but from different sensors.

We associate an $\mathbb{SE}(3)$ pose with each sensor $\xi_i \Leftrightarrow s_i$ and each co-observed target $\eta_k \Leftrightarrow z_k$. The calibration problem can now be posed as a minimisation over these parameters:

$$\arg \min_{\xi_i, \eta_k} \sum_{z \in \mathcal{Z}} C(z, \xi, \eta)^2 \Big|_{i \in |\mathcal{S}|, k \in |\mathcal{Z}|} \quad (5.5)$$

The cost function C is a measure of how well an estimate of sensor and target poses matches a target observation. It takes the form:

$$C(t) = c_a(o_a, \xi, \eta) + c_b(o_b, \xi, \eta) \quad (5.6)$$

where $o_a, o_b = t_k$ are the target observations in the co-observed target, and c is an observation cost function, the form of which is dependent on the type of sensor in which the target was observed.

Camera Cost Function

For targets observed in camera images, the cost function we use is based upon the reprojection error between points on a target with pose η_k and the corners detected in the image.

For each corner on the chequerboard $\mathbf{c}$, we compute its 3D position $\mathbf{p}$ by using the current estimated target pose η_k :

$$\mathbf{p} = \exp_{\mathbb{SE}(3)}(\eta_k) \mathbf{c} \quad (5.7)$$

We then transform each point into the camera frame, project using a camera model, and distort using a look up table produced via the OCamCalib toolbox [88]:

$$\hat{\mathbf{q}} = \gamma(K \exp_{\mathbb{SE}(3)}(\xi_j) \mathbf{p}) \quad (5.8)$$

where $\hat{\mathbf{q}}$ is the projected point in pixel coordinates, K is a pinhole projection matrix, and γ is the distortion look-up table.

The cost function for the camera targets is then simply a weighted sum of squared residuals between the projected pixel coordinates and the observed corners:

$$c_{cam}(o_k) = \sum_{c=0}^n \left(\frac{|\hat{\mathbf{q}}_c - \mathbf{q}_c|}{f\sqrt{n}} \right)^2 \quad (5.9)$$

where $\mathbf{q}_c$ is the observed location of corner c in the camera image, $\hat{\mathbf{q}}_c$ is the projected location of the corner c into the image, n is the number of corners on the chequerboard, and f is the focal length of the camera.

The focal length f in Equation 5.9 is used to make sure the costs from the camera are a similar order of magnitude as those from the LIDAR sensors, which measure in metres rather than pixels. Similarly, dividing by $\sqrt{n}$ ensures that observations carry the same weight in the optimisation regardless of the number of points observed. This is important, as the LIDAR sensors generally have several orders of magnitude more points on the target than the cameras do.

3D Laser Cost Function

The cost function for the LIDAR sensors takes place in geometry 3D space, rather than 2D image space. There are two components to the cost function we use: distance to target plane for each inlier point from the target detection, and distance from the centroid of the inlier points to the centre of the target.

It is worth noting here, that the target poses obtained from the calibration procedure are not actually required. They are purely a side-effect of the procedure from which sensor poses are obtained. It isn't strictly necessary, therefore, to obtain the precise location of the targets — simply finding the plane each target is on would be enough. This would reduce the number of parameters to four (a plane equation), rather than six (an $\mathbb{SE}(3)$ pose). By solving for a full $\mathbb{SE}(3)$ pose for each target, however, we are able to include the centroid constraint in

the LIDAR cost function, which constrains the system more tightly, and leads to faster and more reliable convergence.

The cost function used for targets observed with a 3D LIDAR sensor is:

$$\text{cost}_{3D}(o_k) = \sum_{\mathbf{p} \in \mathcal{I}} \left(\frac{d_p(\mathbf{p}, \eta_k) + d_c(\text{centroid}(\mathcal{I}), \eta_k)/|\mathcal{I}|}{\sqrt{|\mathcal{I}|}} \right)^2 \quad (5.10)$$

where $\mathcal{I}$ is the set of inlier points, $d_p(\mathbf{p}, \eta_k)$ is the distance from the point $\mathbf{p}$ to the plane of the target with pose η_k , and d_c is a standard distance function between the centroid of the inliers and the translation vector of the target pose.

2D Laser Cost Function

The cost function for targets observed with 2D LIDAR scanners is almost identical to that used for 3D LIDAR. The main difference is in the centroid portion of the cost function.

In Equation 5.10, we used a simple distance or L2 norm to measure the distance between centroids. In the 2D case, however, the sensor only measures a line across the target board, which in general will not pass through the centre of the board. Thus penalising pose estimates because the centroid of the inliers from the LIDAR does not coincide with the centre of the board does not make sense. Instead, the second term in equation 5.10 becomes:

$$\hat{d}_c(\text{centroid}(\mathcal{I}), \eta_k) = \begin{cases} d_c(\text{centroid}(\mathcal{I}), \eta_k) - w/2 & |d_c| > w/2 \\ 0 & |d_c| \leq w/2 \end{cases} \quad (5.11)$$

where w is the width of the target board along the longest axis and d_c is the L2 norm. This function is illustrated in Figure 5.6. This has the effect of treating all lines observed by the LIDAR that are consistent with a line somewhere on the board to be given zero cost for this term.

The value of $\hat{d}_c$ can be substituted into Equation 5.10, and the rest remains identical for 2D LIDAR.

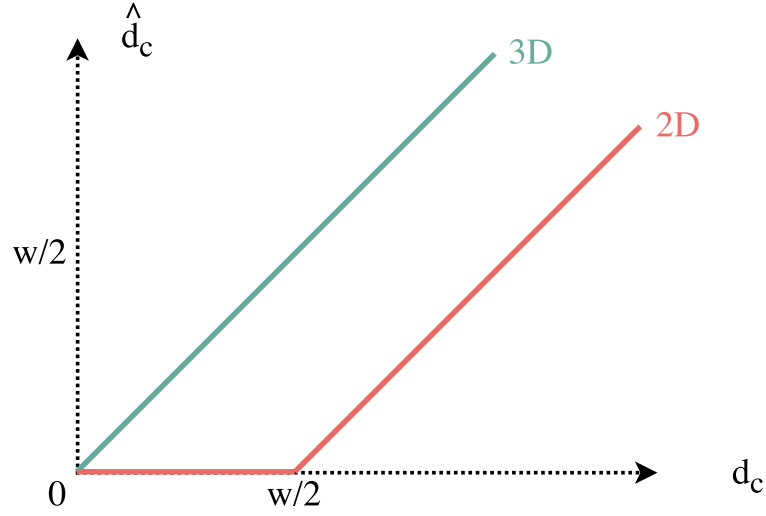


Figure 5.6: Kernel used for 2D centroid cost function. In 3D, we use an L2 norm, but in 2D, the cost is zero until it gets greater than half the width of the board.

Optimisation

Each cost for the various targets are summed up into a single cost for the whole system, as in Equation 5.5. The target detectors presented in Section 5.3.1, however, do generate some false positives — to prevent these erroneous target detections from dominating the cost value, we apply a Huber kernel [99] to the costs:

$$\arg \min_{\xi_i, \eta_k} \sum_{t \in \mathcal{T}} h \left(C(z, \xi, \eta)^2 \right) \Big|_{i \in |\mathcal{S}|, k \in |\mathcal{T}|} \quad (5.12)$$

where $h(\bullet)$ is a Huber kernel. The effect of this is shown in Figure 5.7.

Even with the Huber kernel in place, erroneous target detections can still be expected to have an effect on the precision obtained in the calibration. To deal with this, we run three rounds of optimisation. On the first one, we use all target observations. For subsequent iterations, we remove all targets with a cost value above a threshold — this removes targets that are inconsistent with the solution found, and thus are likely to be false positives. This threshold is reduced for each iteration. Rerunning the optimisation without these observations allows the calibration to be fine-tuned without being affected by outliers.

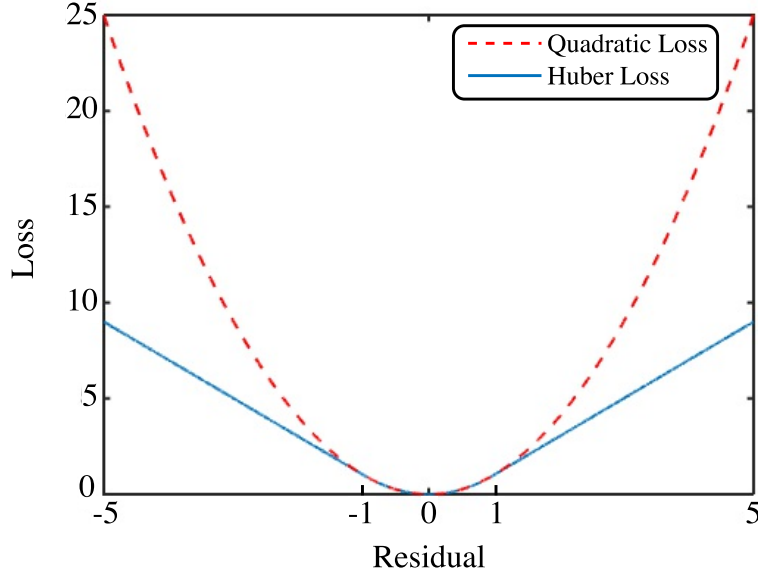


Figure 5.7: Effect of a Huber kernel in the loss function vs. standard quadratic loss. The Huber kernel is linear outside of a small range, and thus prevents the cost function from being completely dominated by outliers.

To optimise the cost function in Equation 5.12, we make use of the Levenberg-Marquardt (LM) method, implemented by Ceres Solver [90]. For a discussion of this optimisation method, and other related methods, we point the reader to Section 4.4. In contrast to the NID cost function presented in Chapter 4, the calibration cost function presented here has many residuals — one for each point on each detected target. Thus the residual space has far higher dimensionality than the parameter space (six parameters for each target and sensor), making the LM method suitable in this case, rather than the BFGS method as used earlier.

5.4 Evaluation

As with the localisation task presented Chapter 4, obtaining a ground truth for evaluation of the calibration system on real data is nigh on impossible. This difficulty is what motivated the development of this system in the first place. As a substitute, we used a Vicon Vero motion capture system [100] to provide accurate poses for each of the target observations — we then used these in the global sensor

pose optimisation to provide a calibration that is highly accurate. By comparing the calibration result with and without the Vicon system, we can evaluate the accuracy of the purely data-driven approach. The Vicon system is expensive, and requires a large workspace and significant set-up time, so a system that doesn't require the Vicon is far more universally applicable.

The Vicon system consists of a number of cameras (the set-up used here has 9 cameras), with specially designed reflective balls that are placed on the target being tracked. Figure 5.8 shows the test vehicle parked amongst the Vicon cameras. By tracking these balls in two or more cameras, they can be precisely localised in 3D space. For the purposes here, we placed the tracking balls around the edge of the chequerboard. In the optimisation, we treat the detected tracking balls exactly the same as points in a 3D LIDAR observation — they take the same form in that they are points in 3D space on a planar surface. The cost function for Vicon observations is the same as Equation 5.10. The only difference is that we do not use a Huber kernel for the Vicon observations — unlike our LIDAR detector, the Vicon does not produce false positives due to its specially designed tracking points. Thus, large residuals actually indicate a poor calibration, rather than indicating an erroneous target detection.

For the 'ground truth' obtained using the Vicon system, we only optimise using co-observations between a sensor and the Vicon system. No co-observations between different sensors (cameras or lasers) are used. This ensures that all observations include the highly accurate Vicon measurement, as well as making the 'ground truth' solution entirely distinct from the optimisation using only data captured by the sensors.

The platform we used for calibration evaluation is a Land Rover sport, with multiple cameras and 2D and 3D LIDARs mounted around the vehicle. The sensors are listed in Table 5.1, and a diagram of the sensor locations on the vehicle is shown in Figure 5.9.



Figure 5.8: Vicon setup used for validating calibration. The Vicon cameras are circled in red.

Name	Type	Model	Location	Resolution
HDL64	3D LIDAR	Velodyne HDL64E S3	Roof Centre	~200k points/scan
VLP16 Left	3D LIDAR	Velodyne VLP 16	Left Wing Mirror	~13k points/scan
VLP16 Right	3D LIDAR	Velodyne VLP 16	Right Wing Mirror	~13k points/scan
VLP16 Grille	3D LIDAR	Velodyne VLP16	Front Grille	~13k points/scan
Hokuyo Left	2D LIDAR	Hokuyo UTM30LX	Front Grille	~1k points/scan
Hokuyo Right	2D LIDAR	Hokuyo UTM30LX	Front Grille	~1k points/scan
Roof Colour	Camera	2×Point Grey Blackfly	Above Windshield	1300×960×2
Roof Grey	Camera	2×Point Grey Blackfly	Above Windshield	1300×960×2
Grille Colour	Camera	2×Point Grey Blackfly	Front Grille	1300×960×2
Grille Grey	Camera	2×Point Grey Blackfly	Front Grille	1300×960×2
Left Colour	Camera	2×Point Grey Blackfly	Roof Left Side	1300×960×2
Left Grey	Camera	2×Point Grey Blackfly	Roof Left Side	1300×960×2
Right Colour	Camera	2×Point Grey Blackfly	Roof Right Side	1300×960×2
Right Grey	Camera	2×Point Grey Blackfly	Roof Right Side	1300×960×2
Ladybug	Camera	Point Grey Ladybug 3	Roof Centre	1232×1616×6

Table 5.1: Sensors used for calibration evaluation. All Blackfly cameras were fitted with fisheye lenses.

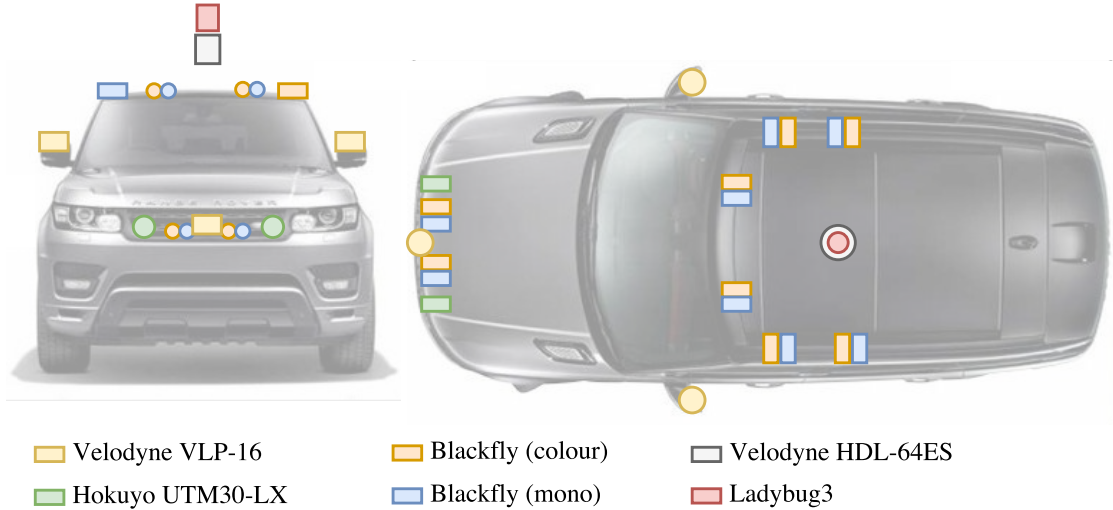


Figure 5.9: Sensor locations on vehicle used for calibration evaluation. The Blackfly cameras are set up as stereo pairs, each containing two cameras of the same type (mono or colour).

Sensor	Error (m)	Error (°)
VLP16 Left	0.0093	0.1315
VLP16 Right	0.0122	0.1190
VLP16 Grille	0.0183	0.1493
Hokuyo Left	0.0310	0.2641
Hokuyo Right	0.0347	0.2811
Roof Colour	0.0109	0.1636
Roof Grey	0.0153	0.1807
Grille Colour	0.0136	0.2495
Grille Grey	0.0012	0.0918
Left Colour	0.0295	0.2469
Left Grey	0.0221	0.2781
Right Colour	0.0174	0.2183
Right Grey	0.0137	0.0894
Ladybug	0.0125	0.0875

Table 5.2: Error between calibration obtained using Vicon motion capture system, and using a purely data driven approach.

Table 5.2 shows the translation and orientation differentials between the pose of each sensor obtained with and without the Vicon sensor. All poses are computed relative to the HDL64 sensor, so there is no relative difference for that sensor.

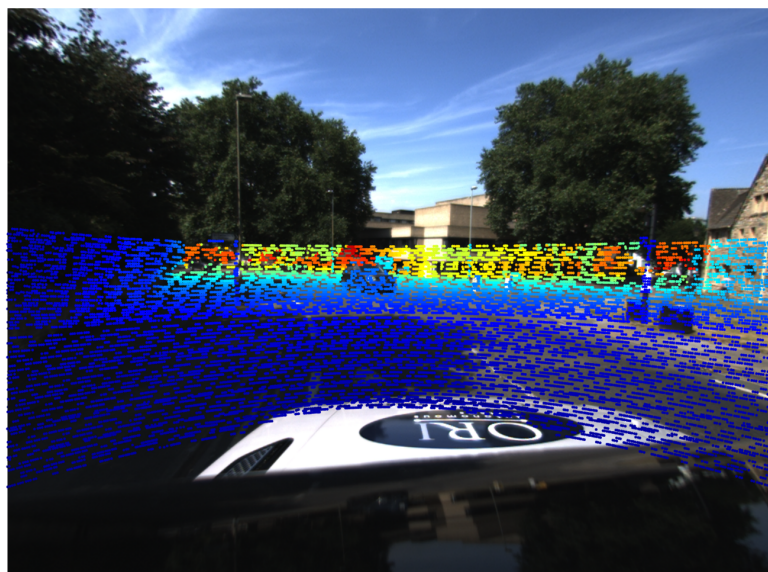
The results in Table 5.2 indicate that the purely data-driven calibration, without the use of the Vicon motion capture system, provides very accurate results, with errors of around 0.02m for cameras and 3D LIDARs. For 2D LIDAR, the errors are slightly larger, above 0.03m, but still in a usable range. It is unsurprising that the 2D LIDAR calibrations are the least accurate, as these are the most difficult to constrain — they only observe a line across the target board, rather than the whole board.

Whilst the calibration with the Vicon is not a ‘ground truth’ as such, it does provide extremely accurate poses for the target board, removing much of the variability in the system. The fact that the two calibrations match so closely gives us confidence that the calibrations obtained are accurate.

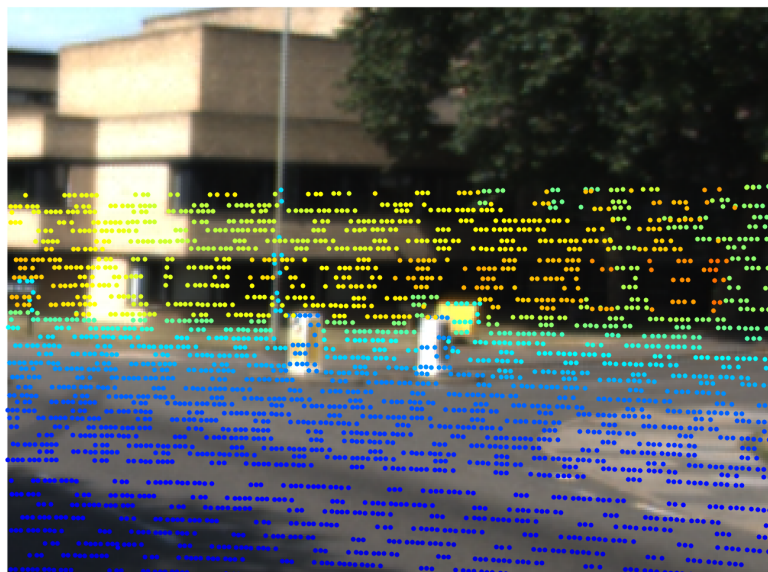
The calibrations can also be verified qualitatively by projecting the points obtained from LIDAR sensors into cameras, using the obtained extrinsic calibrations. This is shown in Figure 5.10. As can be seen, the structure of the points line up well



(a)



(b)



(c)

Figure 5.10: Example of pointcloud from the HDL64ES LIDAR projected into one of the forward-facing Blackfly cameras. (a) Original image. (b) Projected pointcloud, with colour indicating depth: blue = closer, red = further. (c) Detail view of projected pointcloud. The pole and bollards align with the pointcloud, indicating good calibration.

with the image observed by the camera, further verifying that the calibrations obtained are accurate.

5.5 Conclusion

This chapter has presented a data-driven system for obtaining extrinsic calibrations for multi-sensor, multi-modality systems. The system is driven by data collected with the respective sensors, and so does not require accurate direct measurements of the sensor positions, which are often difficult or impossible to obtain. Our method is able to handle more sensor types than existing calibration systems — we have demonstrated calibration of stereo, monocular, and fisheye cameras, as well as both 2D and 3D LIDAR of different resolutions. Furthermore, in contrast to previous systems, we jointly optimise over all sensors on a platform during calibration, giving a globally consistent estimate that uses all the available data for every sensor.

Through comparison with a motion-capture system, we’ve shown that the system we have designed provides accurate calibration. This was further verified qualitatively by viewing the data from the LIDAR sensors in camera images.

Having a simple, effective approach to calibration enables multi-sensor applications, including, but not limited to, those presented in the previous two chapters.

6

Vision-only Mapping and Localisation

Contents

6.1	Introduction	114
6.2	System Overview	116
6.3	NID Tracking	116
6.4	Variations on NID	119
6.4.1	Spatial vs Appearance Weight	119
6.4.2	Depth Uncertainty	121
6.4.3	Histopyramids	122
6.4.4	Overlap Barrier	126
6.5	Depth Estimation	128
6.5.1	Photometric Depth Estimation	129
6.5.2	NID Depth Estimation	130
6.5.3	Initialisation	132
6.6	Evaluation	133
6.6.1	Synthetic Data	133
6.6.2	Real Data	139
6.7	Conclusion	147

6.1 Introduction

The work so far in this thesis has assumed separate vehicles dedicated to mapping using LIDAR, whilst commodity vehicles using only vision localise within that prior map. It would be useful, however, for all vehicles to be able to map

regions, in order to handle alterations in the environment. Such a system also enables an experience-based approach [41], wherein vehicles are able to localise against a number of available maps, laying down a new experience if and when no localisation is successful.

Recently, several systems have been developed for direct visual simultaneous localisation and mapping [15, 17]. Similar to the discussion on pure localisation in Chapter 4, these systems use a direct photometric comparison for both aligning images and estimating depth. As a result, their maps are effectively single-use — they can only be used in the same lighting conditions under which they were generated.

State-of-the-art feature-based approaches, such as ORB-SLAM [28] provide greater robustness to appearance change, as they don't make strong appearance-constancy assumptions. They are still limited, however, by the local nature of their feature descriptors — large appearance changes, or degradations such as motion blur, cause the local descriptors to vary significantly.

One of the most robust existing methods for visual SLAM is SeqSLAM [54], which makes use of sequences of images rather than just individual frames. Whilst SeqSLAM has demonstrated localisation ability across significant environmental change, it is very viewpoint dependent and only provides topological localisation — it is not capable of providing a metric pose estimate. It is therefore not suitable for tasks other than traversing *exactly* the same path with *exactly* the same cameras, and not a candidate for input into a control system which requires metric pose.

This chapter deals with an NID-based metric localisation and mapping system, which we refer to as NID-SLAM. Due to the reliance on NID rather than photometric registration, it is more robust to appearance change than LSD-SLAM, whilst the global nature of NID makes it less susceptible to local changes than ORB-SLAM. Our system relies on vision only, and is able to both localise and map across appearance change, with no LIDAR sensors required. This enables the whole

mapping and localisation system to be implemented on any vehicle with a camera, with no separate survey vehicles with extra sensors required.

6.2 System Overview

Our NID-SLAM system is based on that of LSD-SLAM [17], a state-of-the-art direct SLAM system. NID-SLAM, however, differs in the design of two principal components: *tracking* and *depth estimation*.

Tracking is the process of determining the local camera pose, relative to a keyframe.

Depth Estimation makes use of those tracked poses to produce and refine estimates of the depth values for pixels in the keyframes.

In LSD-SLAM, both of these components are performed using a direct photometric comparison between frames, where as in NID-SLAM, we use NID-based estimation instead.

A schematic of the overall system design is shown in Figure 6.1. The following sections describe our tracking and depth estimation modules in detail.

6.3 NID Tracking

The approach we took towards developing an NID-based localisation and mapping system makes use of semi-dense keyframes as a map representation. In this representation, a keyframe consists of three components:

- a reference image $I_r : \Omega \rightarrow \mathcal{R}^+$;
- an inverse depth map $D : \Omega \rightarrow \mathcal{R}^+$; and
- an inverse depth variance $V : \Omega \rightarrow \mathcal{R}^+$

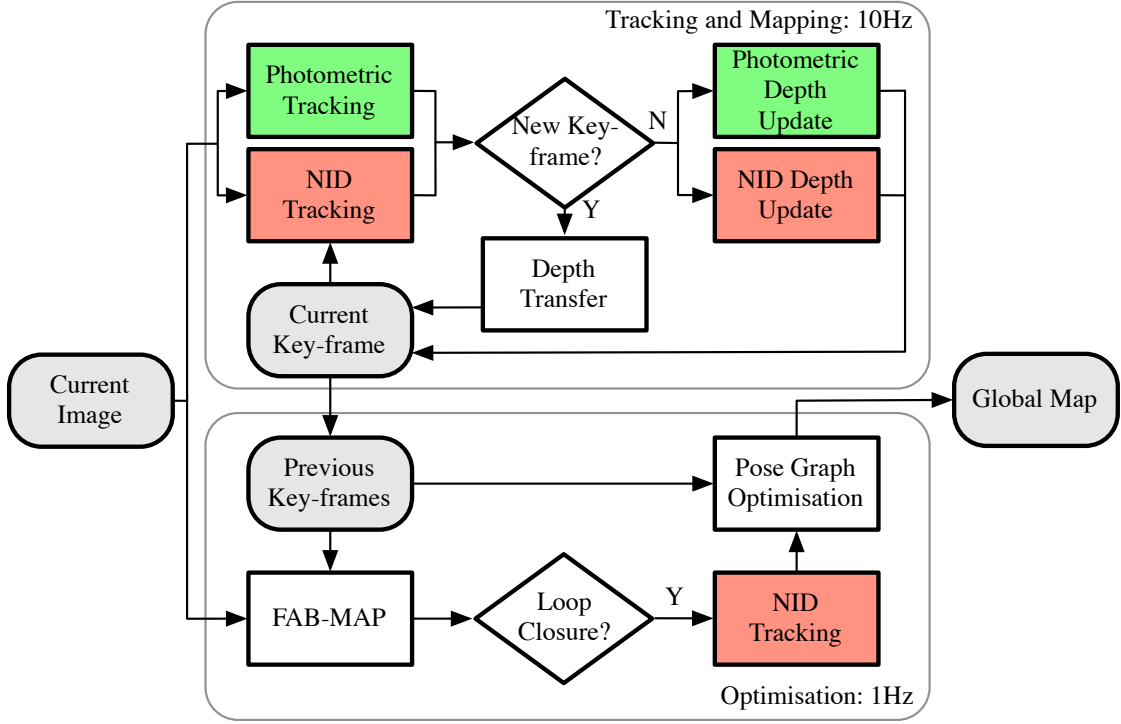


Figure 6.1: The NID-SLAM pipeline. The components shown in green are those from LSD-SLAM, and use direct photometric comparisons between images. By using the NID-based modules shown in red, we increase robustness to appearance change across sessions. Similar to [17], we make use of FAB-MAP [79] to detect loop closures, but use our NID tracker to enable relative pose estimation for loop closures under large appearance changes.

where $\Omega \in \mathcal{R}^2$ are normalised pixel coordinates. The keyframes are referred to as *semi*-dense, as only a subset of pixels $\Omega_D \in \Omega$ are used. The pixels in Ω_D are those for which a sufficiently confident depth estimate can be obtained — see Section 6.5 for details on how the depth estimates are performed.

For each pixel $\mathbf{p}_i \in \Omega_D$, we apply a warp function ω , which transforms the point into a second camera image:

$$\mathbf{q}_i = \omega(\mathbf{p}_i, D(\mathbf{p}_i), \xi) \quad (6.1)$$

where $\xi \in \mathbb{SE}(3)$ is the $\mathbb{SE}(3)$ parameterisation of the relative pose between the two cameras.

The function ω is an inverse projection of $\mathbf{p}_i$ into 3D space using a pinhole

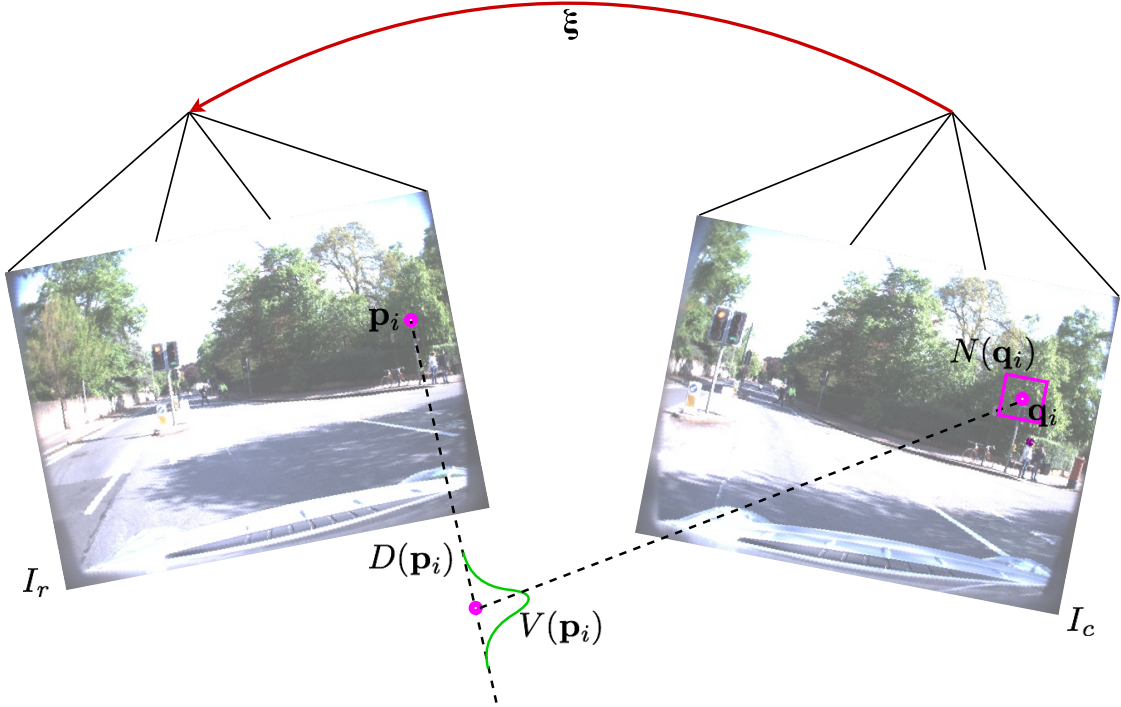


Figure 6.2: Pixels from the keyframe $\mathbf{p}_i$ are reprojected into the new camera image $\mathbf{q}_i$ via their estimated inverse depth $D(\mathbf{p}_i)$ and the estimated pose between the two cameras ξ . N is the neighbourhood around $\mathbf{q}_i$, used for spatial weighting (Section 6.4.1).

camera model and depth $D(\mathbf{p}_i)$, transformation into the new camera frame using ξ , followed by a projection using a pinhole camera model, identical to Equation 3.2. This is illustrated by Figure 6.2.

Similarly to Chapter 4, we compute a joint histogram by reprojecting each point $\mathbf{p}_i \in \Omega_D$ into the live camera image, and evaluating the image intensities $I_r(\mathbf{p}_i)$ and $I_c(\mathbf{q}_i)$. The relative pose between the two cameras can then be found via minimising NID:

$$\arg \min_{\xi} \text{NID} (I_r(\mathbf{p}_i), I_c(\mathbf{q}_i)) \quad (6.2)$$

In the same way as in Chapter 4, we minimise this function by way of a BFGS optimisation.

6.4 Variations on NID

The minimisation problem in Equation 6.2 is at first glance virtually identical to that presented previously in Equation 4.1. There are, however, important differences between the two problems. In this section, we describe the alterations we made to the NID formulation to enable use for this case.

6.4.1 Spatial vs Appearance Weight

The B-spline histogram formulation presented in Section 4.3.2 is only one of several options for constructing a continuous, differential histogram. In particular, [89] presents two weighting methodologies for the B-spline-based histogram: *spatial* weighting and *appearance* weighting.

Appearance weighting each pixel in an image contributes to four histogram bins.

The contribution to each bin is taken from the values of a cubic B-spline centred at the actual appearance value of that pixel. As a result, each pixel will contribute to four adjacent bins.

Spatial weighting a point projected into an image contributes to 16 bins, corresponding to the 4×4 neighbourhood of pixels into which the point projects.

The contribution to each bin is taken from the value of a 2-dimensional B-spline centred at the reprojected location of the point.

These two weighting schemes are illustrated in Figure 6.3.

For the case discussed in Chapter 4, the appearance weighting scheme is not appropriate. With a fully dense prior, we generate a virtual image by sampling the map at the centre of every pixel. As a result, the image coordinates of each sample are not variable, and thus a weighting scheme based on the precise reprojected location of points does not make sense.

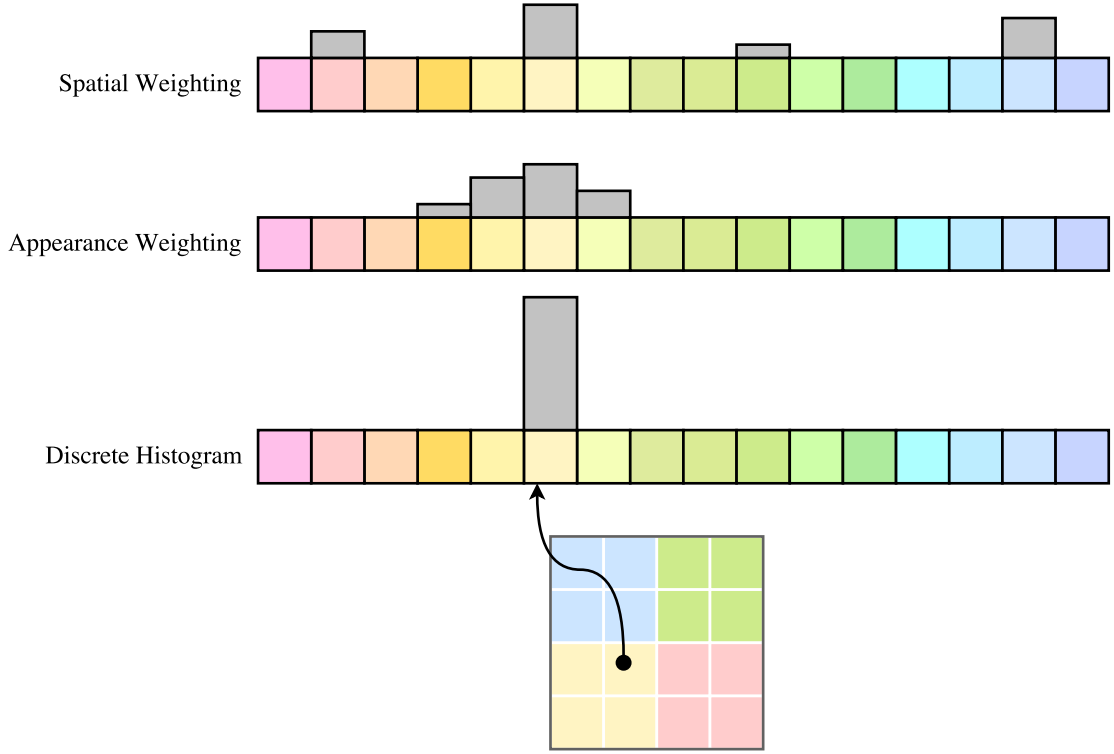


Figure 6.3: Difference between spatial and appearance weighting. This shows the contribution of one point reprojected into an image. The appearance weighting scheme only takes into account the pixel the point reprojects into, whilst the spatial weighting scheme uses information from the neighbourhood.

In the semi-dense keyframe-based case discussed in this chapter, however, the depth map is incomplete and we don't have a solid mesh to render. Instead, we project individual points into a camera image, as shown in Figure 6.2. As a result, pixels from the keyframe will in general reproject to a *sub-pixel* location in the live image. Furthermore, this sub-pixel location varies with the relative pose between the camera and the keyframe. Therefore, in this case, it is reasonable to use a weighting scheme that is dependent on the precise sub-pixel coordinates. Hence both the spatial and appearance weighting schemes are applicable.

In a spatial weighting scheme, the contribution from each sample $\mathbf{p}_i \in \Omega_D$ is added as follows:

$$p_{r,c}(a, b) \leftarrow p_{r,c}(a, b) + \frac{\beta(\mathbf{p}_i, N^{(j)}(\mathbf{q}_i))}{k} \quad (6.3)$$

Here $\beta(\mathbf{p}_i, N^{(j)}(\mathbf{q}_i)) \in \mathbb{R}^+$ represents a 2D cubic B-spline function that weights the contribution of pixel j in the 4×4 neighbourhood $N(\mathbf{q}_i)$ based on proximity to sub-pixel location $\mathbf{q}_i$. The weights are normalised such that $\sum^j \beta(\mathbf{q}_i, N^{(j)}(\mathbf{q}_i)) = 1, \forall \mathbf{q}_i$.

In each dimension, the weighting function α_u and α_v is identical to that presented in Equation 4.18. The 2D B-spline weight β is then the product $\alpha_u \alpha_v$.

Note that by sampling the j neighbouring pixels $N^{(j)}(\mathbf{q}_i)$ of $\mathbf{q}_i$, we never sample I_r or I_c at sub-pixel locations, so no interpolation between pixel intensities is required. This is in contrast to both appearance weighting and photometric methods.

Spatial weighting has the advantage of encoding more spatial information in the histogram, and drawing samples from a wider area, thus increasing robustness. Furthermore, spatial weighting provides more realistic values around edges or discontinuities in the scene. In the case of appearance weighting, moving a point smoothly across an edge results in a jump in the histogram value, as the point moves from one colour to another. Under spatial weighting, however, the histogram will vary smoothly, as colours on both sides of the edge will be sampled, and their weights will gradually change as the point moves. See Figure 6.4 for a visual description of this effect.

6.4.2 Depth Uncertainty

In Chapter 4, we made use of a known, laser-borne prior map with geometry that was assumed to be highly accurate. In the case discussed here, however, our depth estimates are iteratively refined, and are expected to be noisy. Whilst the domain Ω_D does only include pixels for which a reasonable depth estimate is available, it still includes a large range of inverse depth variances. To prevent spurious reprojections due to inaccurate depth values from degrading the pose estimation, whilst still using as many samples as possible, we include an estimate of depth uncertainty

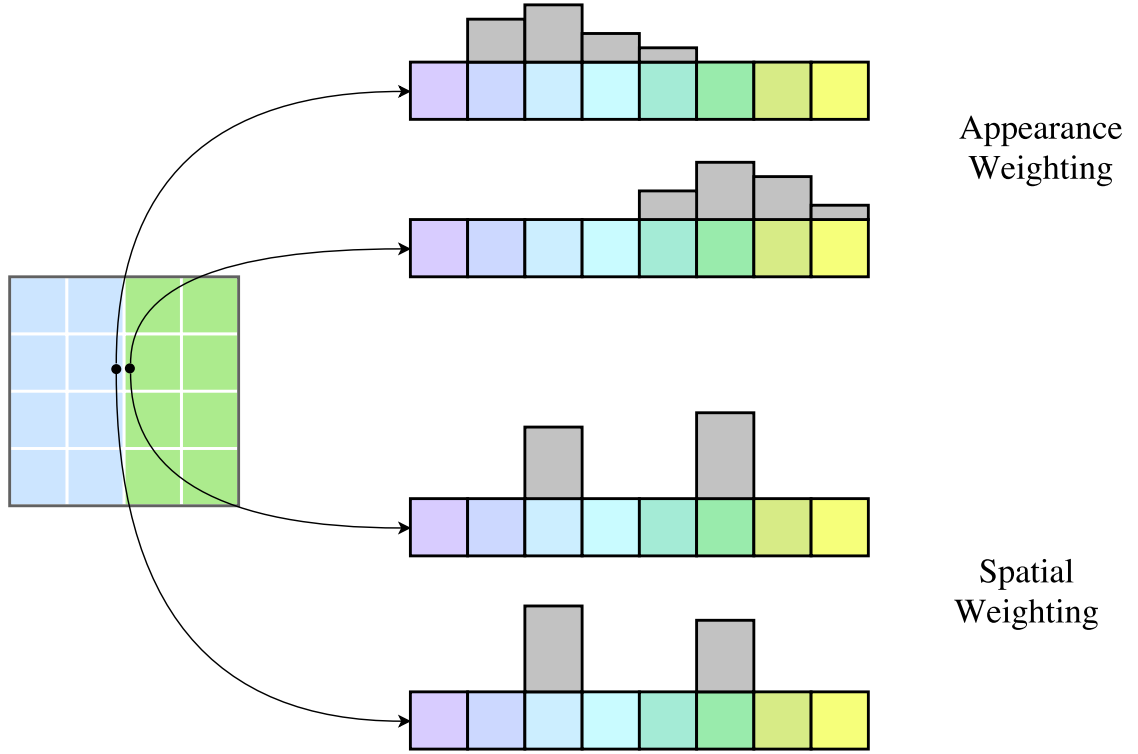


Figure 6.4: When a point reprojects near a discontinuity, the value of the histogram obtained via appearance weighting is highly dependent on the precise reprojected coordinates, and the histogram has a discontinuity when the point moves across an edge (see top). In contrast, the spatially-weighted histogram varies smoothly as the point moves across the edge (bottom).

in the histogram function. Equation 6.3 is altered to become:

$$p_{r,c}(a, b) \leftarrow p_{r,c}(a, b) + \frac{\beta(\mathbf{p}_i, N^{(j)}(\mathbf{q}_i))}{kV(\mathbf{p}_i)} \quad (6.4)$$

where $V(\mathbf{p}_i)$ is the estimated uncertainty in the inverse depth of pixel $\mathbf{p}_i$. This serves to downweight those pixels for which there is large depth uncertainty, thus reducing their affect on the optimisation. The estimation of this value is described in section 6.5.

6.4.3 Histopyramids

Many direct SLAM methods such as [17] or [15] make use of an *image pyramid* to enlarge the convergence basin for their optimiser, thus improving the robustness

of the localisation system. This involves producing a pyramid of images from the camera image, where the image at each level of the pyramid is half the size in each dimension as the level below. Higher levels of the pyramid are obtained by downsampling the original image — in effect, each 2×2 neighbourhood of pixels is averaged to produce one pixel in the next level up. Localisation is first performed at the highest pyramid level, giving a coarse pose estimate. One can then localise successively with each pyramid level, refining the pose estimate at each step until the full resolution image is used.

Localisation at the highest level of the pyramid is more robust to offsets between the initial pose estimate and the true pose, as it relies less on the precise features of the scene, and more on the larger-scale structure. This large scale structure will still be informative further from the true pose, but will not provide as precise localisation as using the full resolution image. Once an approximate pose estimate is found, lower pyramid levels can use the more precise features of the scene for more precise localisation.

Whilst the image pyramid has been shown to significantly improve robustness of direct tracking methods [15], we found that when using NID, the use of an image pyramid had little to no effect, and in some cases actually had a negative effect on robustness. This appears to be due to two effects:

- Fewer samples: lower resolution images have far fewer samples, leading to a sparser sampling of the joint distribution; and
- Pixel averaging: whilst for an SSD cost function, averaging pixels makes sense, and reduces the locality of the cost function, using averaging as an aggregation function in a sampling framework doesn't make sense.

These effects are more pronounced in the semi-dense keyframe representation than the textured mesh representation from Chapter 4 as there are fewer valid

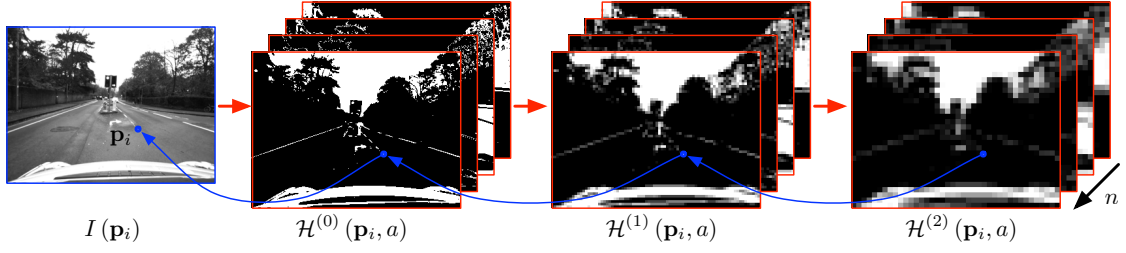


Figure 6.5: Multi-resolution histogram representation. The image $I(\mathbf{p})$ is first converted to a multi-channel binary image $\mathcal{H}^{(0)}(\mathbf{p}, a)$, where the index a selects among the n histogram bins. Each channel a of $\mathcal{H}^{(0)}$ is successively down-sampled using 2×2 block addition, with $\mathcal{H}^{(l)}$ representing the histogram down-sampled l times. By down-sampling the histogram representation instead of building histograms from the down-sampled image I_c , more information is retained at lower image resolutions.

pixels to start with. Furthermore, in the mesh approach, we can generate an image at lower resolution directly from the map, rather than averaging values from a higher resolution image.

To address these limitations, we developed a new framework called *histopyramids*. The intuition behind histopyramids is that they preserve all the samples from the original image, but downscale to reduce the effect of locality. In order to do this, we maintain a histogram for each pixel in the downscaled image.

Formally, we build a pyramid of n -channel histogram images, denoted $\mathcal{H}^{(l)}$ for pyramid level l . Each channel a of the base level $\mathcal{H}^{(0)}$ is computed from the input image I as follows:

$$\mathcal{H}^{(0)}(\mathbf{p}_i, a) = \begin{cases} 1, & a = \mathcal{B}(I(\mathbf{p}_i)) \\ 0, & \text{otherwise} \end{cases} \quad (6.5)$$

Successive levels are produced by down-sampling as follows:

$$\mathcal{H}^{(l+1)}(\mathbf{p}_i, a) = \frac{1}{4} \sum_{j=1}^4 \mathcal{H}^{(l)}(N^{(j)}(2 \cdot \mathbf{p}_i), a) \quad (6.6)$$

where $N(\mathbf{p}_i)$ returns pixels within a 2×2 neighbourhood of $\mathbf{p}_i$. This produces downscaled ‘images’, where each pixel contains a histogram of all the values from the original image that fall into that pixel in the downscaled version; if one took the mean of all these values, the classical downscaled image would be obtained. Each

downscaled histogram image thus loses some of the *spatial* information from the full-resolution image, whilst maintaining all of the *appearance* information. The histogram down-sampling process is illustrated in Fig. 6.5.

The histopyramid can be represented efficiently in OpenGL using a multi-channel, multi-layer, multi-level texture. Multi-level textures are designed for the storage of mipmapped textures — representations of the same texture at multiple resolutions, for efficient rendering of objects at different distances from the camera. Each level in the texture is half the size of the level below in each axis. This maps exactly to the levels in our histopyramids.

To store all the bins of the histogram at each pixel, we use a combination of multiple channels and multiple layers. Multi-layer textures are basically 3-dimensional textures, but unlike a standard 3D texture, there is no interpolation in the 3rd dimension — the layers of the texture are discrete images which are accessed via integral indices. This allows more performant look-up. More importantly, higher levels of a mipmapped multi-layer texture are not downscaled in the third dimension — there are the same number of discrete layers in each texture. This allows layers to be used to represent histogram bins, as the same number of bins is needed at each level.

Conceptually, this is all that is required for the histogram representation — each bin maps to a layer in the multi-layer texture. It is more efficient, however, to use a 4-channel (RGBA) texture, and have each channel represent a bin. Using this representation, the texture has $n/4$ layers, where each layer contains 4 histogram bins, giving a total of n bins in the histogram. This is most efficient when the number of bins in the histogram is an integer multiple of 4 — here we use $n = 16$ bins.

When using a histopyramid for NID evaluation, we compute the joint entropy between *all* the samples in a given pixel. Equation 6.3 is modified to take into account the multiple samples:

$$p_{r,c}^{(l)}(a, b) \leftarrow p_{r,c}^{(l)}(a, b) + \frac{\gamma^{(l)}(a, b) \beta(\mathbf{q}_i, N^{(j)}(\mathbf{q}_i))}{k} \quad (6.7)$$

where the pairwise histogram weighting function $\gamma^{(l)}(a, b) : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}^+$ is defined as follows:

$$\gamma^{(l)}(a, b) = \mathcal{H}_r^{(l)}(\mathbf{p}_i, a) \mathcal{H}_c^{(l)}(N^{(j)}(\mathbf{q}_i), b) \quad (6.8)$$

Importantly, the weighting function $\gamma^{(l)}$ can be non-zero for any combination of bin indices (a, b) , and therefore Eq. 6.7 may update up to n^2 histogram bins in the joint distribution $p_{r,c}^{(l)}$ in contrast to n bins for Eq. 6.3. This increases computation by a constant factor for each level, but higher pyramid levels quadratically reduce the number of samples, which results in an overall computational saving.

The development of NID from these histograms is identical to that described in Chapter 4.

Figure 6.6 shows the effect of histopyramids versus a standard image pyramid at different levels. As the figure shows, the reduced effect of locality in the histopyramid leads to a smoother cost surface with a wider convergence basin.

6.4.4 Overlap Barrier

An issue that came up during the development of this system was that sometimes, a step in the optimisation would produce an estimated camera pose that had very little overlap with the keyframe — ie. very few points from the keyframe would reproject into the camera. On occasion, through chance, with only a small number of samples, this would produce a very low NID, and thus be accepted as a valid step. This led to regular tracking failures.

This problem is less apparent in both the fully dense approach developed in Chapter 4 and the photometric approach in [17]. With a fully dense approach, using a 3D mesh as the map representation, unless the camera is pointing towards the sky, the map will always be visible. In a keyframe approach, the region of space in which 3D information is present, however, is much smaller. When using a photometric comparison, having fewer samples is less of a problem, as the samples

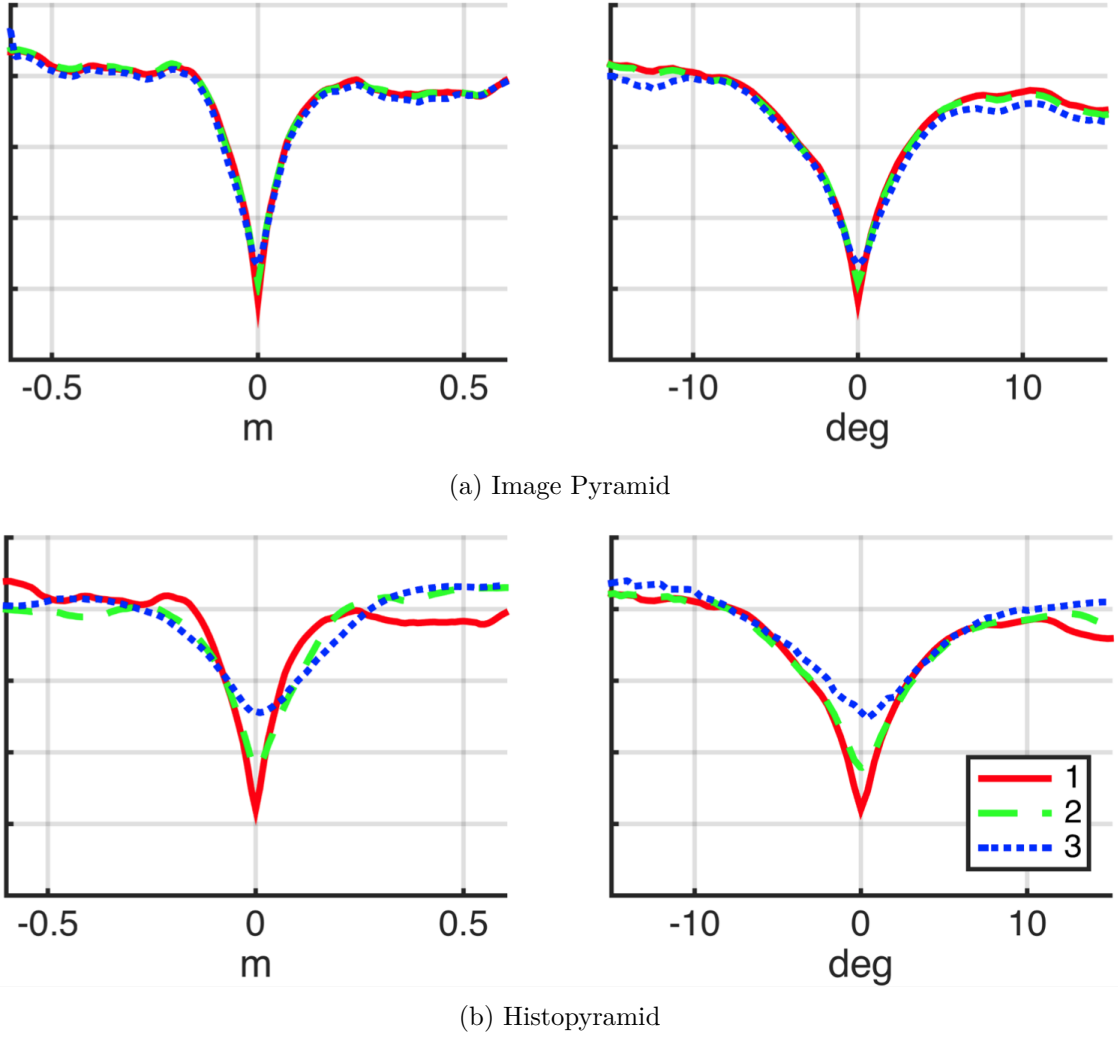


Figure 6.6: NID vs offset from ground truth pose at different levels of (a) standard image pyramid and (b) histopyramid. The histopyramid gives smoother cost function with wider convergence basin at higher levels.

may each produce a high cost, as compared to the NID-based approach, where few samples leads to a poor estimate of the distribution.

To combat this effect, we introduced an overlap barrier into the cost function. As a result, Equation 6.2 becomes:

$$\arg \min_{\xi} \left\{ \text{NID}_{\mathbf{p}_i \in \Omega_D} (I_r(\mathbf{p}_i), I_c(\mathbf{q}_i)) + O(\psi) \right\} \quad (6.9)$$

where $O(\psi)$ is the overlap barrier function defined by:

$$O(I_r(\mathbf{p}_i), I_c(\mathbf{q}_i)) = \frac{3 \times 10^3}{\log((\psi + 0.75)^3)} \quad (6.10)$$

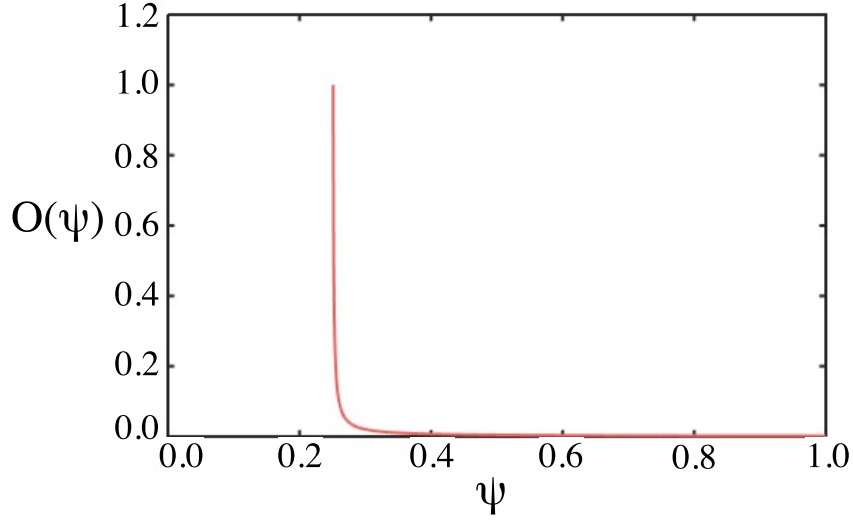


Figure 6.7: Overlap barrier function. The value ψ is the fraction of keyframe pixels that reproject into the camera. The function has little effect when the overlap is greater than 0.4, but prevents the optimiser converging to a pose from which the camera can see only a small fraction of the keyframe.

and ψ is the fraction of points $\mathbf{p}_i$ that reproject into the image I_c :

$$\psi = \frac{|\mathbf{q}_i \in [-1, 1]|}{|\mathbf{p}_i|} \quad (6.11)$$

The function $O(\psi)$ is illustrated in Figure 6.7. This function effectively prevents the optimiser from finding solutions in which the overlap between the keyframe and the estimated camera pose is small. When the overlap ratio is above 0.4 (ie. the camera can see at least 40% of the keyframe’s points), the barrier has negligible effect. If the overlap gets below 0.3, however, the overlap barrier starts to dominate the cost, pushing the optimiser back towards more overlap.

6.5 Depth Estimation

The key information required for mapping using monocular images is an estimate of the depth of each pixel. Combining this information with the colour or intensity information from the images produces a 3-dimensional map, containing both geometric and appearance information, similar to that developed in Chapter 3.

Similar to the approach taken in [17], we represent the map as a series of ‘2.5D’ keyframes, each of which contains an image, depth estimates for pixels, and an uncertainty value for each depth estimate.

After solving for the estimated camera pose ξ , the depth for a given keyframe pixel can be estimated using a small-baseline temporal stereo measurement between the keyframe and the current camera image I_c , as is done for example in [17]. By using multiple such small-baseline measurements, the depth estimate can be progressively refined, resulting in a depth estimate that will, in most cases, converge on the true value.

Whilst we wish to compute depth estimates for as many pixels as possible, in practice, most images are made up of large regions with little intensity gradient, and thus little signal for depth estimation. Many methods have been proposed for estimating depth for such textureless regions, for example [101], however the primary aim of the depth estimation in this case is to produce a map for localisation. Regions with too little intensity gradient for accurate direct depth estimation will also provide little to no gradient in the localisation cost function, so the extra computation required for methods such as [101] is not worthwhile. Therefore those pixels without sufficient intensity gradient are omitted from the keyframe.

6.5.1 Photometric Depth Estimation

For photometric error, the stereo measurement is independent for each pixel $\mathbf{q}_i \in \Omega_D$, and can be estimated efficiently using either quadratic optimisation or a sampling method along an epipolar line. However, when revisiting key-frames after long periods of time, lighting and appearance changes sufficiently to make local photometric depth updates impossible. We therefore developed a global approach to key-frame depth updates using the NID metric to robustly maintain and improve depth estimates across appearance change.

6.5.2 NID Depth Estimation

In addition to wanting to be able to *localise* against a map generated under different conditions, it would also be desirable to be able to update the map itself over time. This would enable refinement of the map through multiple observations of the same place. Furthermore, an updateable map would allow areas of the map that are structurally altered — either through natural processes (eg. leaves falling from trees in winter) or through man-made alterations (eg. scaffolding) — to be kept up-to-date without having to rebuild an entirely new map from scratch.

The map in the system presented here (as well as in other semi-dense approaches such as LSD-SLAM) consists of two components for each pixel in a keyframe: appearance and (inverse) depth. We have shown that much variation in appearance is handled by the use of the NID metric for localisation, so it is mostly the depth (geometric) information that we would like to be able to update across sessions. For this, we turn again to NID to provide a measure of image consistency, which can be used to estimate pixel depths.

The key difference between depth estimation using NID, as compared to photometric error, is that NID is a global measurement; ie. it is a single measure across the whole image. This means that the NID cost value is dependent on the value of every pixel jointly — we cannot compute the NID cost for a single pixel independently. Thus, we can't perform a 1-dimensional linear search to find the optimum depth for each pixel, as is done in LSD-SLAM.

Although we can't define a per-pixel cost with NID, we can compute an individual NID gradient for each pixel. This indicates how the NID would change with small perturbations in the depth of that pixel alone. For fixed camera pose parameters ξ we compute the inverse depth gradient $\nabla_{D_r}(\xi) \in \mathbb{R}^{|\Omega_D|}$ as follows:

$$\nabla_{D_r}(\xi) = \left. \frac{\partial \text{NID}(I_r(\mathbf{p}_i), I_c(N(\mathbf{q}_i)))}{\partial D_r(\mathbf{p}_i)} \right|_{\xi, \mathbf{p}_i \in \Omega_D} \quad (6.12)$$

Now that we have these partial derivatives for each pixel, we once again used a gradient-based optimisation framework to refine the depth estimates. Here, rather than finding the camera pose that minimises NID, we are holding the pose constant and trying to find the depth map that minimises the function:

$$\arg \min_{\hat{D}_r(\mathbf{p}_i)} [\text{NID}(I_r(\mathbf{p}_i), I_c(\mathbf{q}_i))] \Big|_{\mathbf{p}_i \in \Omega_D} \quad (6.13)$$

As this minimisation has a large parameter space (one inverse depth parameter per pixel) for a single-dimensional cost space (NID), we again use BFGS to carry out this optimisation. The new estimated reference inverse depth map $\hat{D}_r$ is iteratively refined using the BFGS update equation:

$$\hat{D}_r(\mathbf{p})_{k+1} = \hat{D}_r(\mathbf{p})_k - \alpha_{D_k} \mathbf{\Sigma}_{D_k} \nabla_{D_r}(\xi) \quad (6.14)$$

where $\alpha_{D_k} \in \mathbb{R}^+$ is a step distance and $\mathbf{\Sigma}_{D_k} \in \mathbb{R}^{|\Omega_D|} \times \mathbb{R}^{|\Omega_D|}$ is the estimated depth covariance after k iterations, obtained from the BFGS inverse Hessian approximation. This matrix is typically sparse.

After optimisation, we obtain a new inverse depth estimate for each pixel $\hat{D}_r(\mathbf{p})$. Additionally, by taking the diagonal of the inverse Hessian approximated by BFGS, we obtain an estimate of the variance in the inverse depth of each pixel. Rather than accepting these new values, we fuse them with the previous estimates to obtain updated inverse depth $D_r(\mathbf{p})$ and variance $V_r(\mathbf{p})$:

$$D_r(\mathbf{p}) = \frac{\hat{D}_r(\mathbf{p})_k \circ V_r(\mathbf{p}) + D_r(\mathbf{p}) \circ \text{diag}(\mathbf{\Sigma}_{D_k})}{V_r(\mathbf{p}) + \text{diag}(\mathbf{\Sigma}_{D_k})} \quad (6.15)$$

$$V_r(\mathbf{p}) = \left(V_r(\mathbf{p})^{-1} + \text{diag}(\mathbf{\Sigma}_{D_k})^{-1} \right)^{-1} + \text{diag}(\sigma_p^2 \mathbf{I}) \quad (6.16)$$

where $\circ$ is the Hadamard (element-wise) product and σ_p^2 is a constant process noise term similar to the update in [17]. This ensures that the inverse depth variances

do not become overconfident. By fusing the depth updates in this manner, we obtain estimates that will become more refined over time.

6.5.3 Initialisation

The method presented in Section 6.5 is sufficient for updating a depth map in which most of the pixels already have a reasonable estimate of depth. This doesn't address the separate problem of initialising a new keyframe with depth values, however.

In LSD-SLAM [17], a similar photometric method, initialisation is carried out using a random value with large variance for the inverse depth of each pixel. By combining this with a robust kernel, a small number of randomly correct (or almost correct) inverse depth values are enough to give an initial pose estimate; after this, the independence of each inverse depth measurement allows the depth of each pixel to be estimated, with the large initial variance giving low weight to the initial random estimate.

In the case of an NID cost function, however, we have found this approach not to work. Because NID does not provide a 'quality' measure for each individual pixel — only for the image as a whole — it is unable to cope with a random initialisation in which most of the depth estimates are incorrect. It fails to converge on a depth map without a reasonable initial estimate.

We therefore use a photometric cost function when initialising a keyframe, and use the NID depth map update when visiting a previously-initialised keyframe. During initialisation, the time difference between frames being used for small-baseline stereo measurements is very small, and thus appearance change is minor, resulting in photometric comparisons performing well. When the time interval is larger, with the possibility of large appearance change, the NID cost function still allows for depth map refinement.

6.6 Evaluation

For the evaluation of the NID-based SLAM system, we once again make use of both a synthetic dataset and a real-world dataset. The datasets we use here are:

- synthetic indoor New Tsukuba dataset [102]; and
- a subset of the RobotCar dataset [23]

We compare our NID-SLAM system’s performance with that of two state-of-the-art systems: the feature-based ORB-SLAM [28] and the direct, semi-dense LSD-SLAM [17].

For ORB-SLAM, we used the implementation from [29], whilst for LSD-SLAM we implemented a pipeline that could perform both LSD-SLAM and NID-SLAM. We benchmarked our LSD-SLAM implementation against the open source version¹, and obtained identical results.

6.6.1 Synthetic Data

For evaluation on synthetic data, we use the indoor New Tsukuba dataset [102]. This dataset provides a single trajectory through an office environment, with 1800 photorealistic images and ground truth poses. The trajectory is rendered for four different lighting conditions — natural daylight, artificial fluorescent light, desk lamps, and a flashlight moving with the camera. An example image from each of the four lighting conditions is shown in Figure 6.8.

The extreme variation in lighting conditions provided by the New Tsukuba dataset provides a good evaluation of the problems our system was designed to address — map reuse under changing appearance.

For each of the systems under evaluation, we ran two passes through the trajectory, for each combination of lighting conditions. On the first pass, no prior

¹https://github.com/tum-vision/lsd_slam

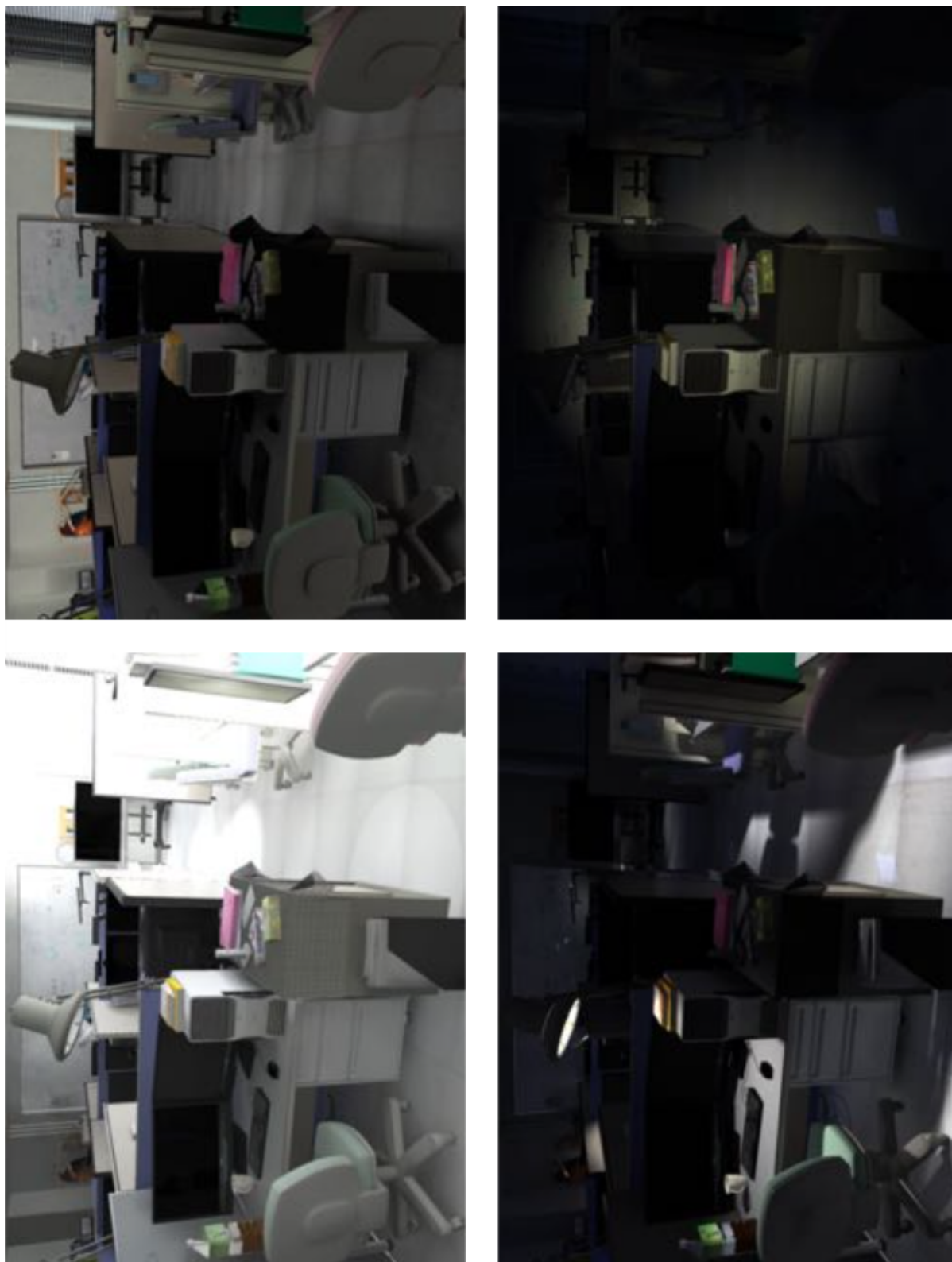


Figure 6.8: Example images from the four lighting conditions provided by the New Tsukuba dataset. Clockwise from top left: daylight, fluorescent, flashlight, lamps.

map is available, so each system was creating a map as it went. We scaled the inter-keyframe poses using the ground truth in order to report metric errors. On the second pass, the map from the first pass was used as a starting point — each system would attempt to localise against the keyframes from the first pass, whilst simultaneously updating those keyframes if possible. As all three systems are monocular, we used the ground truth poses to scale the poses obtained, and report metric errors. Table 6.1 shows the results of this experiment, whilst Figure 6.9 shows some example cross-dataset localisations.

The results in Table 6.1 show that NID-SLAM achieves the most reliable localisation performance (greatest percentage of successful poses) on all combinations except Daylight-Daylight and Fluorescent-Fluorescent. These two are the simplest cases, where the whole scene is well illuminated, and both traversals are under the exact same lighting conditions. In both cases, NID-SLAM is successful on over 99% of poses, but ORB-SLAM does marginally better. Furthermore, whilst on these cases ORB-SLAM does give slightly higher reliability, the denser nature of NID-SLAM allows it to give more precise pose estimates, with lower RMS errors in both translation and orientation.

These results show that NID-SLAM is far more robust under radically altered appearance than the other two systems under evaluation. Many of the combinations achieve over 90% success rate, where as LSD- and ORB-SLAM only see success rates that high in the easiest cases.

By far the worst performance for all three systems is with the first traversal under lamp light, and the second with the flashlight — all three completely fail under this combination. These are the two most challenging conditions, with only small parts of the scene illuminated. The map produced by NID-SLAM under lamp light is very sparse, and the illuminated regions under flashlight in general won't overlap much with those illuminated by the lamps. As a result, localisation fails.

Traversals	Traversals 1	Daylight			Fluorescent			Lamps			Flashlight		
		% Success	RMSE (mm)	RMSE (°)	% Success	RMSE (mm)	RMSE (°)	% Success	RMSE (mm)	RMSE (°)	% Success	RMSE (mm)	RMSE (°)
Daylight	NID	99.3	4.7	0.22	96.7	7.8	0.58	73.9	67.9	14.24	74.6	64.2	2.25
	LSD	96.1	36.4	1.93	48.6	59.8	2.93	26.1	33.2	1.53	25.5	89.5	4.40
	ORB	100.0	25.5	1.20	81.4	21.4	0.20	9.5	20.9	0.91	0.1	12.8	0.48
Fluorescent	NID	95.0	12.0	1.05	99.7	8.5	0.41	85.3	59.4	5.39	95.8	27.9	0.80
	LSD	55.5	52.6	2.44	96.5	14.7	0.53	38.6	76.5	3.67	29.7	70.0	3.80
	ORB	85.4	18.4	0.26	99.7	15.9	0.64	7.1	291.1	1.51	10.7	125.2	3.39
Lamps	NID	88.3	33.8	1.39	93.6	25.2	0.95	93.1	19.6	0.73	84.3	72.8	3.98
	LSD	6.6	90.2	5.01	46.8	93.2	4.30	71.7	8.0	0.05	11.9	72.3	2.80
	ORB	35.4	21.4	0.65	24.6	27.8	0.57	83.1	18.5	0.62	1.6	24.4	0.25
Flashlight	NID	23.8	41.2	0.88	92.2	38.5	1.72	0.00	N/A	N/A	92.0	24.6	1.17
	LSD	15.7	91.0	4.54	27.2	88.4	4.91	0.00	N/A	N/A	88.7	36.4	1.93
	ORB	19.4	17.2	0.29	30.0	24.7	0.25	0.00	N/A	N/A	22.7	50.8	1.28

Table 6.1: Indoor tracking results on the New Tsukuba dataset. The NID system shows the highest successful localisation rates on almost every combination, with the exception of the easiest cases (daylight-daylight and fluorescent-fluorescent) — in these cases NID-SLAM reports over 99% success, but ORB-SLAM does marginally better, though on these cases, NID-SLAM does give more precise pose estimates.

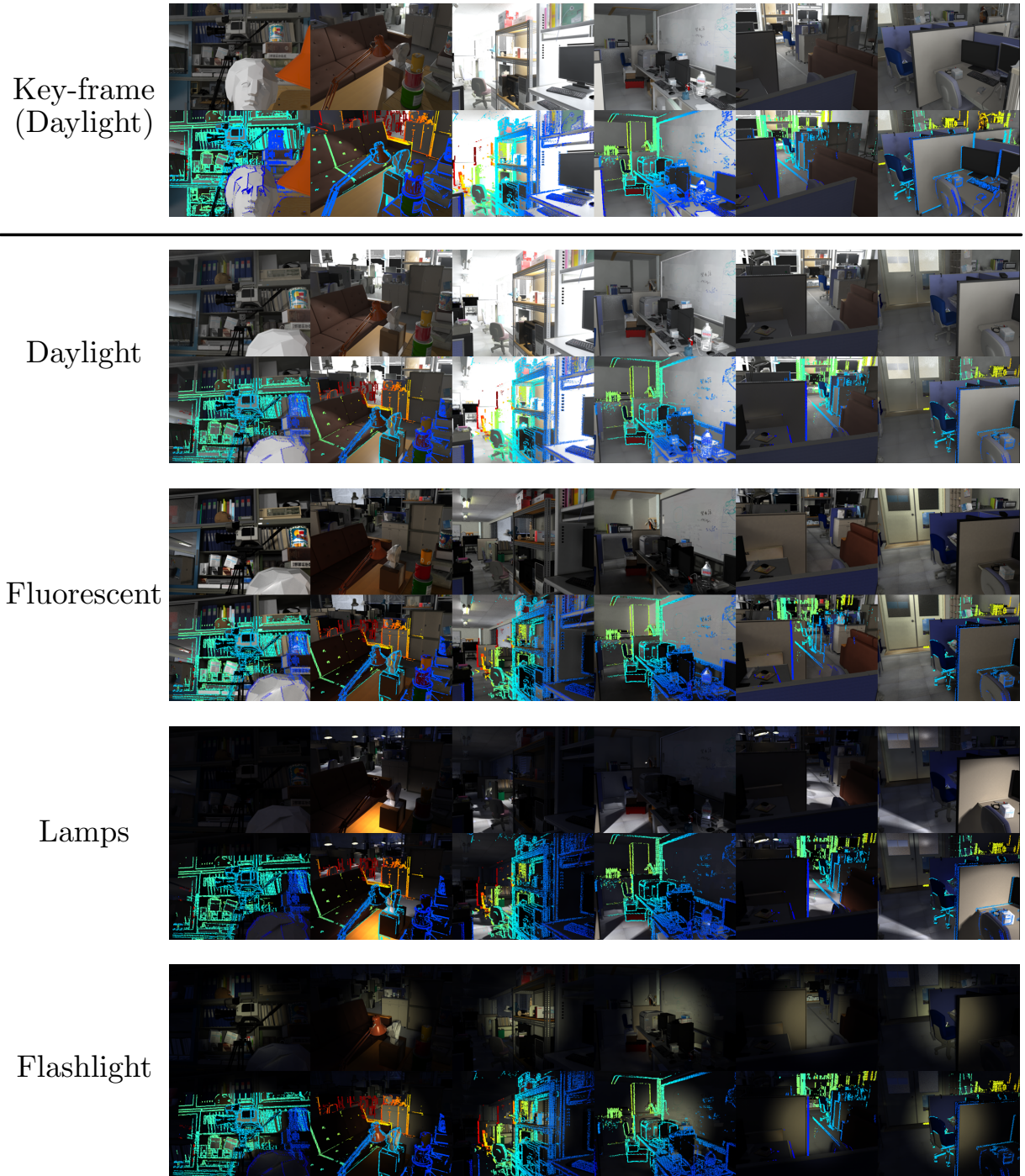


Figure 6.9: Sample depth maps projected into images during indoor localisation against a map built in daylight. Key-frame appearance and depth maps are shown at the top, and the results of the tracking against the key-frame are shown for each of the four different conditions (daylight, fluorescent, lamps and flashlight), demonstrating robust tracking in the presence of indoor appearance change. Note that the images used for tracking are taken from a different location and view-point to the key-frame.

1 \ 2	Daylight	Fluorescent	Lamps	Flashlight
Daylight	60.9	63.3 (-7.05%)	54.5 (-9.31%)	55.4 (-7.20%)
Fluorescent	59.7 (-1.97%)	68.1	55.6 (-7.49%)	61.1 (+2.35%)
Lamps	59.1 (-2.96%)	63.6 (-6.61%)	60.1	61.5 (+3.02%)
Flashlight	52.5 (-12.65%)	68.1 (+0%)	63.7 (+5.99%)	59.7

Table 6.2: Depth map refinement on the New Tsukuba dataset. All measurements are median absolute depth map errors in units of millimetres; percentages in brackets show the change in errors after the second traversal.

The best results for NID-SLAM are those with a first traversal in fluorescent light, followed by those with a first traversal in daylight. In both these conditions, the scene is relatively uniformly illuminated, leading to a relatively dense map. The fluorescent light is slightly better than the daylight due to constant illumination across the room, rather than light coming only from windows — for this reason, the daylight illumination has more shadows. This suggests that an initial traversal under good lighting conditions, allowing a quality initial map to be built, makes subsequent traversals more reliable.

Depth Map Refinement

We also used this experiment to evaluate the ability of NID-SLAM to refine its depth maps under different conditions to those under which they were originally created.

For each of the combinations presented in Table 6.1, we evaluated the depth map errors by comparing the estimated depth maps to the ground truth from the dataset. We computed these errors both after the first traversal and after the second, to evaluate whether the second traversal was able to improve estimates obtained on the first. The results of this evaluation are summarised in Table 6.2.

As can be seen in Table 6.2, under most traversal combinations, the depth estimates improve with the second traversal, even when that second traversal is under different illumination conditions. Unsurprisingly, the worst performance comes when the second traversal is under lamp light. As was discussed with regards to localisation, the lamps illuminate very little of the scene, and thus it would be expected that they would not provide much depth information, particularly when used after an initial traversal in full fluorescent light.

Significantly, though, even under poor lighting conditions, the depth maps do not get less accurate (or negligibly so). This is due to the fact that areas in darkness in the lamps or flashlight traversals will have little effect on NID no matter their depth. This leads to a high variance, and thus very little effect on the depth according to Equation 6.15.

The main exception to this is when the first traversal is under flashlight, but the second under fluorescent or lamps. The increase in error here appears to be due to the flashlight map being quite sparse, and having a lighting distribution that follows the camera, whilst the fluorescent light is fairly constant across the scene, and the lamp light is sparse but static. Thus the distribution of appearance in the map is not only dependent on the geometry, leading to spurious depth estimates when paired with a consistent lighting source. As with localisation, this indicates the importance of the first traversal through a scene being under friendly conditions.

These results show that even under significant appearance changes, this system is able to refine the geometry of its maps, whilst in conditions under which depth estimation is difficult, the maps remain almost unchanged.

6.6.2 Real Data

To evaluate NID-SLAM on real, outdoor data, we ran two experiments. Firstly, a small-scale experiment to compare robustness with ORB-SLAM and LSD-SLAM, and secondly a larger-scale evaluation of NID-SLAM across substantially more

data. Both of these experiments again make use of a subset of the Oxford RobotCar dataset [23].

Robustness Comparison

The previous evaluation using synthetic data indicated that NID-SLAM is better able to handle changes in lighting conditions than ORB-SLAM or LSD-SLAM. To test that hypothesis in the real world, we carried out a small experiment on a section of road from the Oxford RobotCar Dataset.

For this experiment, we selected a section of road approximately 500m long, under six different and challenging illumination conditions. Table 6.3 shows the subsets of the Oxford RobotCar Dataset, whilst example images from these six traversals are shown in Figure 6.10.

Condition	Frames	Start Timestamp	End Timestamp	Dataset
Sun	924	1403622333595478	1403622393158652	2014/06/24 15:03:07
Rain	802	1437747791877082	1437747841995156	2015/07/24 14:17:50
Overcast	908	1417176772138658	1417176830255804	2014/11/28 12:07:13
Night	874	1441138754508641	1441138809134112	2015/09/01 21:16:05
Dusk	1081	1424450359745880	1424450427299121	2015/02/20 16:34:06
Snow	1491	1422953346020135	1422953439195435	2015/02/03 08:45:10

Table 6.3: Traversals from the Oxford RobotCar Dataset [23] for outdoor comparison. The images used cover the same stretch of road under different conditions. Images from each are shown in Figure 6.10. The datasets can be accessed at <http://robotcar-dataset.robots.ox.ac.uk/datasets>.

As with the experiments on the New Tsukuba dataset, we ran each system through two traversals of the scene, under each combination of conditions available. On the first run through, the system doesn't have any prior knowledge of the environment, and thus generates a new map. On the second run through, we disable the creation of new keyframes, to force the system to localise against the keyframes generated on the first traversal. The keyframes on the second traversal, however, are not static, and can be further refined.

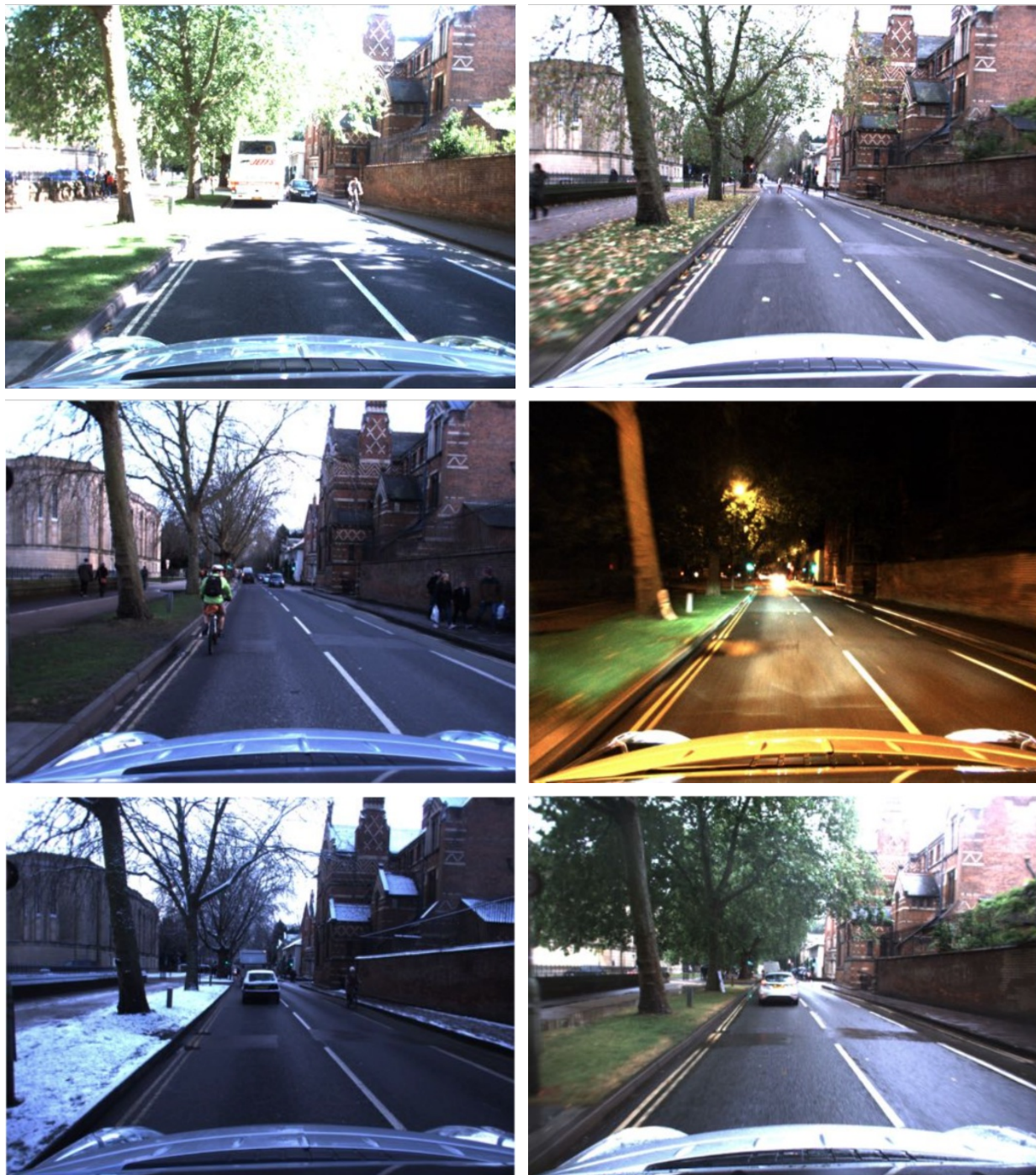


Figure 6.10: Example images from the six traversals used for outdoor comparison. Clockwise from top left: sun, overcast, dusk, night, snow, rain. These traversals show very challenging variation in appearance.

$\frac{1}{2}$	Overcast			Dusk			Snow			Sun			Rain			Night		
	NID	LSD	ORB	NID	LSD	ORB	NID	LSD	ORB	NID	LSD	ORB	NID	LSD	ORB	NID	LSD	ORB
Overcast	100	76.28	100	96.6	0.3	42.6	85.3	0.3	12.9	7.9	0.0	0.1	4.3	0.0	0.0	0.0	0.0	0.0
Dusk	88.2	10.68	100	99.3	64.1	47.8	21.8	18.5	15.2	21.9	1.72	0.1	4.2	0.37	0.0	0.0	0.0	0.0
Snow	97.3	0.0	0.1	91.7	4.94	57.8	99.9	40.5	45.7	0.1	0.8	0.1	0.0	0.0	0.0	0.0	0.0	0.0
Sun	10.1	0.2	0.1	0.0	0.0	0.1	11.4	0.0	0.1	98.7	9.27	100	0.0	0.0	0.0	0.0	0.0	0.0
Rain	6.4	0.8	0.1	4.2	0.0	0.1	0.0	1.2	0.1	0.0	0.1	0.1	94.1	5.0	0.0	0.0	0.0	0.0
Night	1.9	0.1	0.1	0.1	0.2	0.1	5.7	0.0	0.0	0.0	0.0	0.1	1.0	0.1	0.0	0.0	0.0	0.0

Table 6.4: Outdoor tracking success rates using the Oxford RobotCar Dataset. NID-SLAM shows the highest success rates on almost all combinations, but still struggles on the hardest cases. LSD-SLAM, due to its reliance on photometric consistency, does the worst under any appearance change, whilst ORB-SLAM’s features provide some robustness to change, but still does worse than NID-SLAM in most cases.

Figure 6.11 shows some example qualitative results, whilst Table 6.4 shows a quantitative comparison with LSD-SLAM and ORB-SLAM.

Of the three systems under evaluation, NID-SLAM provides the most robust tracking estimates. Whilst ORB-SLAM is able to obtain 100% localised poses for some of the easiest cases, it drops off to negligible success rates below 1% under any significant appearance change. Unsurprisingly, LSD-SLAM performs the worst under any appearance change, as the static scene illumination assumption inherent in any direct photometric method is strongly violated here.

Whilst NID-SLAM provides the most robustness to the appearance changes presented here, it still struggles in the most challenging conditions. In particular, at night time, all three systems are unable to either track without an existing map or localise against a map from a previous traversal. This is not surprising, considering that the system presented in Chapter 4, with a LIDAR-borne map, was also unable to localise at night.

Under rainy conditions, all three systems struggle, except for NID-SLAM with both traversals in the rain. Most of the images in this traversal are highly blurred, due to rain drops on the lenses, so it is unsurprising that ORB-SLAM, relying on point features, fails completely. The fact that NID-SLAM is the most reliable here supports the claim that direct methods are more robust to blur.

Large-scale Evaluation

To evaluate NID-SLAM more rigorously, we made use of the same datasets we used in Chapter 4, as shown in Figure 4.15. These are twelve traversals of a 9.3km route through Oxford, containing over 450,000 images, and recorded over a period of six months. As in Chapter 4, we first used a traversal through dataset 1 to build an initial map, then tested tracking/localisation performance using that map as a starting point for each of the other conditions. We use visual odometry to scale inter-keyframe poses from the first traversal, in order to measure metric errors.

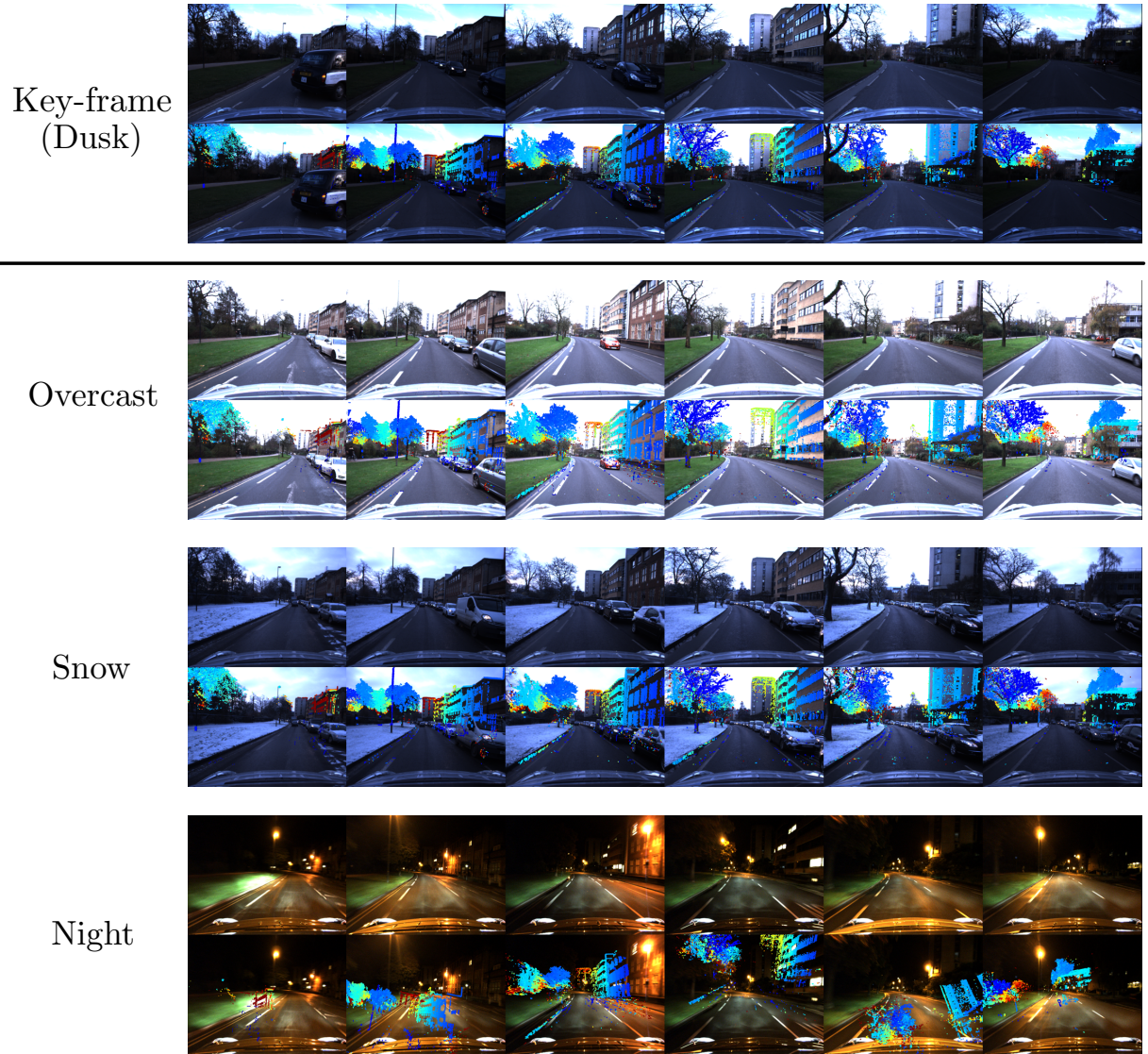


Figure 6.11: Sample depth maps projected into images during outdoor localisation against a map built at dusk. Key-frame appearance and depth maps are shown at the top, and the results of the NID-SLAM tracking against the key-frame are shown for each of three different conditions (overcast, snow and night), demonstrating robust tracking in the presence of outdoor lighting (overcast to dusk) and weather (snow to spring) changes. Note that localisation at night fails with a map built under different conditions.

Dataset	RMS Translation Error (m)	RMS Rotation Error (°)	Outlier Rate (%)
2	0.2865	0.7436	4.70
3	0.2558	0.9659	3.98
4	0.2833	1.0555	9.25
5	0.2359	0.8313	3.83
6	0.3819	1.3779	9.02
7	0.2611	1.3386	6.02
8	0.2401	0.7523	2.73
9	0.2864	0.8384	6.26
10	0.3449	0.8573	6.24
11	0.2768	0.9336	6.91
12	0.2483	1.0337	5.07

Table 6.5: Results of evaluation of NID SLAM on outdoor data from the Oxford RobotCar dataset [23]. The results show robust and precise localisation across a large variation in appearance. Translation error and outlier rate are generally slightly higher than for the method from Chapter 4, but orientation errors are mostly slightly lower. The datasets used are shown in Figure 4.15.

The results of this evaluation are presented in Table 6.5. The errors presented in Table 6.5 are all in a local frame — they do not represent global localisation error. Rather they represent error relative to the keyframe they are tracking against. As with the maps built in Chapter 3, we do not require nor expect the maps to be globally consistent; only local relative consistency is required.

The results in Table 6.5 show similar localisation performance to those presented in Chapter 4. Translation errors across most datasets are slightly larger here, as are the outlier rates, whilst orientation errors are mostly lower. The higher translation errors may be due to the difficulty in correctly determining depth to scale — incorrectly scaled keyframes would lead to small errors in translation estimates. This, however, would have a smaller effect on orientation.

The higher outlier rate with this method is likely due to the more limited map available for localisation. Where the localiser in Chapter 4 had an entire 3D model, with 360° visibility, in the system here, only a semi-dense keyframe is available. Furthermore, using semi-dense keyframes makes the approach more viewpoint sensitive. Thus if the optimiser enters a state in which the view of the

keyframe is not good, it is likely to diverge, where a more complete map might still provide enough context to localise.

These problems may be able to be obviated by using multiple keyframes at a time for localisation. If paired with an experience-based approach [41], this would provide the localiser with much more information, and further increase robustness.

The only dataset for which the SLAM formulation in this chapter achieves a higher success rate than the fully dense formulation from Chapter 4 is traversal 9. As noted in Chapter 4, most of the failures in traversal 9 were due to large dynamic obstacles, as this dataset has the heaviest traffic. The fact that NID-SLAM has a lower error rate on this dataset indicates that it is less affected by this traffic — this is due to the fact that, for those areas of the image covered by a vehicle, the depth variances could be expected to grow (as the depth doesn't match that from the original traversal), and thus those points would be downweighted in the cost function. Whilst this traversal does still have a higher error rate than many others, it does indicate that the depth uncertainty weighting increases robustness to dynamic obstacles.

Convergence Range

One important consideration when using a system such as this is the size of the convergence basin for the optimisation — the cost function is non-convex, so the initial hypothesis needs to be close to the true pose for the optimiser to converge to the correct solution. To determine how large this convergence basin is, we ran the optimiser, starting at a random offset of fixed magnitude from the ground truth pose, for each pose along a trajectory. By varying the magnitude of the offset, we can get a measure of how large the initial offset can be in order to still converge. We ran this experiment on three trajectories:

- New Tsukuba dataset (keyframes from daylight, tracking frames from fluorescent)

- Subset of the RobotCar dataset on a narrow road (keyframes from dataset 1, tracking frames from dataset 5) — see Figure 6.12(a)
- Subset of the RobotCar dataset on a wide road (keyframes from dataset 1, tracking frames from dataset 5) — see Figure 6.12(b)

Figure 6.13 shows the percentage of successful localisations for each scenario, with varying offset magnitudes. As can be seen, the convergence basin varies in size from approximately 0.5m for the indoor data to 1.6m for the wide road, whilst the convergence basin for orientation is approximately 15-17° for all datasets. This is due to the fact that on the wider road, the structure is further from the camera, and thus small translations lead to smaller changes in image space. In the indoor environment, however, even a small translation can significantly alter the image, taking it out of the convergence basin for the optimiser. Orientation, on the other hand, affects all scales approximately the same, and so the distance to structure doesn't have a significant effect on the convergence basin.

Run Time

The localisation system presented here has a run time marginally faster than that presented in Chapter 4, running at approximately 10Hz on the same hardware. This is primarily due to faster histogram evaluations with less dense maps.

Whilst this run time is not fast enough to operate at frame rate, it is enough to provide regular local updates to a fast local tracking system, whether another visual system or an inertial sensor.

6.7 Conclusion

In this chapter we have introduced NID-SLAM, a system capable of performing both mapping and localisation across multiple sessions with significant appearance changes. This system builds on both existing work on semi-dense monocular



(a)



(b)

Figure 6.12: Example images from trajectories used to test convergence range. (a) A narrow road, with most of the structure relatively close to the vehicle. (b) A wider road, with buildings set back some distance from the curb.

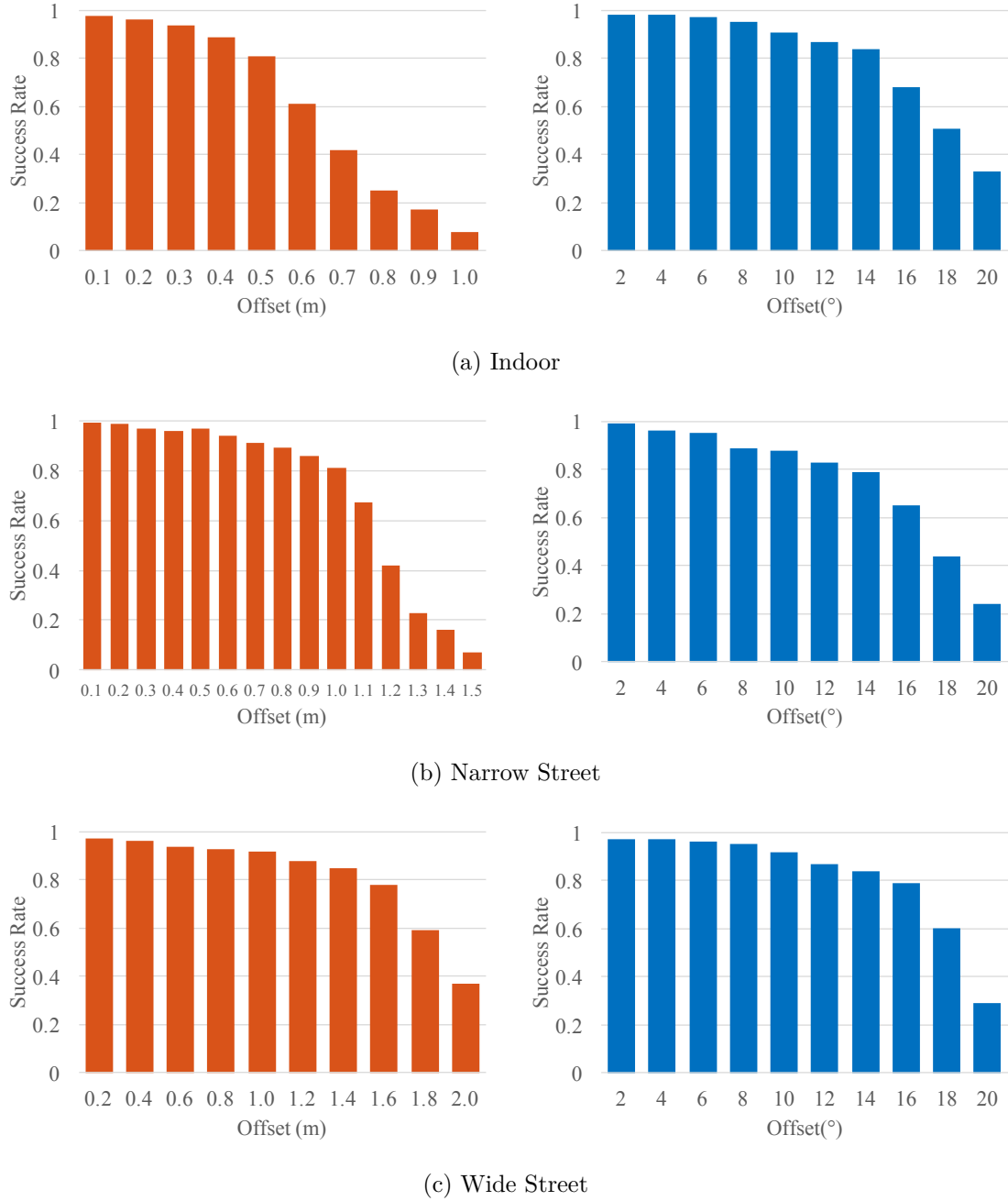


Figure 6.13: Success rates for localisation starting from different offsets from the true pose. In indoor data, localisation is reasonably reliable within 0.5m of the true pose, whilst outdoor that increases to approximately 1m for a narrow street and 1.6m for a wide street. The orientation bounds for localisation are similar for all environments, at around 15-17°.

SLAM as well as the NID-based dense localisation system presented in Chapter 4. Combining mapping and localisation in a single vision-only system enables greater flexibility, with maps that can be refined and updated continuously over time.

Through the use of synthetic data, we showed greater robustness to large changes in appearance when compared to state of the art direct and feature-based SLAM systems. Unlike previous methods that demonstrate cross-seasonal localisation [42, 54], we produce metric pose estimates, not just topological localisation. Whilst our NID-SLAM system can still fail with an initial traversal under poor lighting conditions, with a reasonable initial map to start with, the system is able to robustly localise. Furthermore, we showed the ability of this system to refine its maps when coming back under different conditions, something that is not possible for photometric methods.

Through a large-scale experiment on outdoor data, we were able to show that the system works under real-world conditions, not just those present in simulation. We demonstrated greater robustness under varying lighting and weather conditions than state-of-the-art feature-based and direct SLAM methods. Furthermore, we showed metric accuracy with RMS errors mostly below 30cm on a dataset of over 450,000 images captured over a period of six months, with large appearance changes over that time. Whilst the NID-SLAM system still has limitations, particularly operation at night time or in direct sunlight, but performs well across a large variety of lighting and weather conditions.

7

Conclusion

Contents

7.1	Summary	151
7.2	Contributions	153
7.3	Future Work	155
7.4	Concluding Remark	156

7.1 Summary

In this thesis we have outlined two methods for localisation using cameras over long time periods in changing outdoor environments.

We began in Chapter 2 by looking at prior work in the area of localisation, introducing the concepts of *indirect* (feature-based) and *direct* localisation. Until recently, most visual localisation solutions were indirect, and relied on the detection and matching of feature points. This has computational benefits, but results in much of the information in the images being discarded. Some more recent methods directly match an image to a map, but impose very strong assumptions on the appearance constancy of the environment.

Chapter 2 also introduced some background on tools that we use throughout the thesis. OpenGL allows us to perform fast, highly parallel computation across a wide variety of GPU hardware. It is this GPU hardware, its drop in price, and rise in power that has enabled more computationally intensive vision processing in recent years. We also reviewed $\mathbb{SE}(3)$ transformations, which allow us to represent and optimise vehicle and sensor poses in a 3D environment.

In Chapter 3 we developed a method for producing textured 3D maps from a survey vehicle equipped with a 2D LIDAR scanner and a number of cameras. We used a LIDAR scanner mounted in a ‘pushbroom’ configuration, such that it produces line scans perpendicular to the road service and the direction of travel of the vehicle. Using vehicle motion, we showed how we can stitch multiple such 2D LIDAR scans into 3D geometry. We then make use of the cameras on the vehicle to produce textures for that geometry. Compared to previous methods that make use of 3D LIDAR, our method enables mapping with much lower cost hardware, whilst still producing maps of a fidelity that is suitable for localisation, as is shown in Chapter 4.

Chapter 4 showed how to localise a camera in textured 3D maps such as those produced in Chapter 3. We discussed the shortcomings of using photometric consistency in outdoor environments in which appearance can vary significantly from that at the time of mapping. Instead, we made use of the NID metric, which is rooted in information theory, and is more robust to the sorts of appearance changes observed outdoors. We showed how we could localise by finding the pose that minimises the value of this metric computed between a camera image and the map — this was achieved through the use of a BFGS optimisation. This localisation method was implemented in low-level OpenGL for high performance, whilst maintaining compatibility across as wide as possible a range of graphics computing hardware. An evaluation on both synthetic and real-world data demonstrated the ability of this system to localise even under large appearance changes.

In Chapter 5, we demonstrated a method for obtaining extrinsic calibration of a multi-sensor system. This is a key requirement for any system that makes use of multiple sensors, such as those developed in the preceeding chapters. For all but the most trivial platforms, the relative poses of the sensors can not be adequately determined via direct measurements. The system takes a data-driven approach, making use of co-observations of a target board with a chequerboard pattern on it. We are able to obtain extrinsic calibrations for a wide range of sensors: stereo, monocular, and fisheye cameras, as well as both 2D and 3D LIDAR of varying resolution. Our calibration system optimises over all sensor poses and target observations simultaneously, producing a single, globally consistent estimate for the extrinsics of all sensors on the platform. We compared the results of the purely data-driven system to those obtained with a commercial motion capture system, demonstrating high precision.

Finally, in Chapter 6, we developed a system capable of both mapping and localisation across large appearance change using only cameras. This removes the need for a survey vehicle with extra sensors, and also unlocks the use of a multi-experience localisation framework. Maps can be updated by any vehicle equipped with cameras, and both localisation and map refinement can be performed across significant appearance change. This method was implemented in OpenGL for high performance and compatibility. We demonstrated greater robustness in localisation across traversals under changed conditions than state-of-the-art feature-based and direct SLAM methods, whilst maintaining the metric precision of those methods.

7.2 Contributions

The primary contributions outlined in this thesis are:

- A method for building maps, consisting of textured 3D meshes, using only a 2D LIDAR scanner and camera. This enables lower-cost mapping than prior

systems that depend on 3D LIDAR, whilst still producing maps suitable for localisation.

- A robust visual localisation system that estimates the $\mathbb{SE}(3)$ pose of a camera with respect to a 3D map. Our system makes use of an information theoretic metric (NID) to align images with the map, leading to robustness to large changes in environmental appearance. We demonstrated the efficacy of this system with localisation across seasonal variation over a period of six months, whilst obtaining metric errors below 30cm.
- A SLAM system that utilises NID to provide robustness in both mapping and localisation, allowing a full navigation solution that only depends on cameras. Our NID-SLAM framework shows greater robustness than both state-of-the-art feature-based and direct methods, whilst still providing precise metric pose estimates, rather than the topological localisation produced by systems such as SeqSLAM [54] or appearance change prediction [42].
- A method for estimating the relative poses of different types of sensors on a multi-sensor platform. Our method is able to simultaneously calibrate all sensors on the platform, leading to a globally consistent calibration estimate, and can handle more sensor types than existing calibration systems. We demonstrated its precision via a comparison with a commercial motion capture system.
- GPU implementations of our mapping and localisation systems, allowing fast operation on commodity hardware. The algorithms we have developed are not specific to our OpenGL implementations, but the structure we have outlined would enable implementation on any highly parallel architecture.

7.3 Future Work

The work in this thesis makes progress towards the goal of robust localisation in outdoor environments. There are however, many avenues of further research that would enhance the work presented here. Some areas worth exploring are:

Learned cost functions This thesis has shown that the NID cost function is useful for image alignment and localisation under changing appearance. Some conditions, however, still frustrate this method — particularly those with extreme lighting variation, such as night time or direct sunlight. It may be possible to develop a better cost function that is robust to these changes through the use of a CNN. This could combine the robustness of CNN-based localisation [51] with the precision of direct methods.

Multi-experience localisation Experience-based localisation has been demonstrated with feature-based methods [41], and significantly increases robustness by storing multiple representations of the environment. Such a framework, when used with the keyframe-based approach from Chapter 6 could further improve the robustness we have demonstrated.

Outlier rejection We make use of an external frame-to-frame pose source to classify our localisations as successful or outliers, as our system is unable to internally determine the quality of the localisations produced. Whilst developing the results in Section 6.6.2, we observed that both ORB-SLAM [28] and LSD-SLAM [17], whilst having internal methods for determining this, often reported successful localisation for clearly erroneous poses. It would thus be useful to develop a robust method for classifying the output from a localiser without the need for an external relative pose source, both specifically for the localiser presented in this thesis, and for localisation more generally.

Such a method was proposed in [89] using a support vector machine (SVM) — either this method or other learning-based methods could be explored.

Dense maps from cameras The maps we produced in Chapter 6 are relatively sparse, and limited by processing-time constraints. There has been significant research into fusing multiple such depth maps into dense maps similar to those produced in Chapter 3 — the system outlined in [103] is able to produce city-scale maps, but does not run in real time. Fusing the depth maps we produce using an offline processing step may enable the production of a more complete map of the environment, increasing both precision and robustness in localisation.

Visual-inertial fusion As noted in Chapter 6, our systems do not currently operate at frame rate. Integrating our system for tracking against a map with an inertial system for high-frequency pose updates would provide both fast updates and robustness to appearance change. This visual-inertial combination is common in other tracking systems in the literature, for example [39].

7.4 Concluding Remark

This thesis produced two methods for localising in large outdoor environments using cameras. The first required a fleet of specialist survey vehicles equipped with LIDAR to produce maps in the forms of meshes. It demonstrated robustness to appearance change over large time periods, whilst maintaining high precision.

The second method removes the need for LIDAR equipped survey vehicles, whilst maintaining almost the same levels of robustness and precision. This enables all vehicles — or other platforms equipped with cameras — to build and refine maps.

The ability to reliably localise in changing environments is a key requirement for the long term operation of robots, including autonomous vehicles. We hope this the methods presented here can advance the development of such platforms.

References

- [1] Henry Chen, Austin S Lee, Mark Swift, and John C Tang. “3D Collaboration Method over HoloLens and Skype End Points”. In: *Proceedings of the 3rd International Workshop on Immersive Media Experiences*. ACM. 2015, pp. 27–30.
- [2] Larry Matthies, Mark Maimone, Andrew Johnson, Yang Cheng, Reg Willson, Carlos Villalpando, Steve Goldberg, Andres Huertas, Andrew Stein, and Anelia Angelova. “Computer vision on Mars”. In: *International Journal of Computer Vision* 75.1 (2007), pp. 67–92.
- [3] Julian Surber, Lucas Teixeira, and Margarita Chli. “Robust Visual-Inertial Localization with Weak GPS Priors for Repetitive UAV Flights”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2017.
- [4] Jesse Levinson, Michael Montemerlo, and Sebastian Thrun. “Map-Based Precision Vehicle Localization in Urban Environments”. In: *Proc. of Robotics: Science and Systems (RSS)* (2007).
- [5] Jesse Levinson and Sebastian Thrun. “Robust vehicle localization in urban environments using probabilistic maps”. In: *2010 IEEE International Conference on Robotics and Automation* (May 2010), pp. 4372–4378.
- [6] Will Maddern, Geoffrey Pascoe, and Paul Newman. “Leveraging Experience for Large-Scale LIDAR Localisation in Changing Cities”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA, USA, May 2015.
- [7] Ryan W Wolcott and Ryan M Eustice. “Robust LIDAR localization using multiresolution Gaussian mixture maps for autonomous driving”. In: *The International Journal of Robotics Research* 36.3 (2017), pp. 292–319.
- [8] Eric Royer, Maxime Lhuillier, Michel Dhome, and Jean-Marc Lavest. “Monocular Vision for Mobile Robot Localization and Autonomous Navigation”. In: *International Journal of Computer Vision* 74.3 (Jan. 2007), pp. 237–260.
- [9] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. “MonoSLAM: Real-time single camera SLAM”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.6 (2007), pp. 1052–1067.
- [10] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. “Fast image-based localization using direct 2d-to-3d matching”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 667–674.
- [11] Simon Lynen, Torsten Sattler, Michael Bosse, Joel Hesch, Marc Pollefeys, and Roland Siegwart. “Get Out of My Lab: Large-scale, Real-Time Visual-Inertial Localization”. In: *Robotics Science and Systems*. 2015.

- [12] Nathan Piasco, Désiré Sidibé, Cédric Demonceaux, and Valérie Gouet-Brunet. “A Survey on Visual-Based Localization: On the Benefit of Heterogeneous Data”. In: *Pattern Recognition* (2017).
- [13] Arren J Glover, Will Maddern, Michael J Milford, and Gordon F Wyeth. “FAB-MAP+ RatSLAM: Appearance-based SLAM for multiple times of day”. In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 3507–3512.
- [14] Christoffer Valgren and Achim J Lilienthal. “SIFT, SURF and Seasons: Long-term Outdoor Localization Using Local Features.” In: *EMCR*. 2007.
- [15] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. “DTAM: Dense tracking and mapping in real-time”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on* (Nov. 2011), pp. 2320–2327.
- [16] Ryan W. Wolcott and Ryan M. Eustice. “Visual Localization within LIDAR Maps for Automated Urban Driving”. In: *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*. 2014.
- [17] Jakob Engel, Thomas Sch, and Daniel Cremers. “LSD-SLAM: Large-Scale Direct Monocular SLAM”. In: *Eccv* (2014), pp. 1–16.
- [18] Alexander D Stewart and Paul Newman. “LAPS - localisation using appearance of prior structure: 6-DoF monocular camera localisation using prior pointclouds”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2625–2632.
- [19] Geoffrey Pascoe, Will Maddern, Alexander D. Stewart, and Paul Newman. “FARLAP: Fast Robust Localisation using Appearance Priors”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA, USA, May 2015.
- [20] Geoffrey Pascoe, Will Maddern, and Paul Newman. “Robust Direct Visual Localisation using Normalised Information Distance”. In: *British Machine Vision Conference (BMVC)*. Swansea, Wales, 2015.
- [21] Geoffrey Pascoe, Will Maddern, and Paul Newman. “Direct Visual Localisation and Calibration for Road Vehicles in Changing City Environments”. In: *IEEE International Conference on Computer Vision: Workshop on Computer Vision for Road Scene Understanding and Autonomous Driving*. Santiago, Chile, Dec. 2015.
- [22] Geoffrey Pascoe, Will Maddern, Michael Tanner, Pedro Pinies, and Paul Newman. “NID-SLAM: Robust Monocular SLAM using Normalised Information Distance”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, July 2017.
- [23] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. “1 Year, 1000 km: The Oxford RobotCar dataset”. In: *The International Journal of Robotics Research* 36.1 (2017), pp. 3–15.
- [24] Matthias Hentschel, Oliver Wulf, and Bernardo Wagner. “A GPS and laser-based localization for urban and non-urban outdoor environments”. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2008), pp. 149–154.

- [25] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. “Visual SLAM algorithms: a survey from 2010 to 2016”. In: *IPSS Transactions on Computer Vision and Applications* 9.1 (Dec. 2017), p. 16.
- [26] Velodyne. *A High Definition Lidar Sensor for 3-D Applications*. http://velodynelidar.com/lidar/products/white_paper/HDLwhitepaper_OCT2007_web.pdf. 2007.
- [27] Sebastian Thrun and Michael Montemerlo. “The graph SLAM algorithm with applications to large-scale mapping of urban structures”. In: *The International Journal of Robotics Research* 25.5-6 (2006), pp. 403–429.
- [28] Raul Mur-Artal, JMM Montiel, and Juan D Tardós. “ORB-SLAM: a versatile and accurate monocular SLAM system”. In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163.
- [29] Raúl Mur-Artal and Juan D. Tardós. “ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras”. In: *arXiv preprint arXiv:1610.06475* (2016).
- [30] Georg Klein and David Murray. “Parallel Tracking and Mapping for Small AR Workspaces”. In: *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. Nov. 2007, pp. 225–234.
- [31] Sven Middelberg, Torsten Sattler, Ole Untzelmann, and Leif Kobbelt. “Scalable 6-DOF Localization on Mobile Devices”. In: *ECCV*. 2014.
- [32] Julius Ziegler, Henning Lategahn, Markus Schreiber, Christoph G Keller, Carsten Knoppel, Jochen Hipp, Martin Haueis, and Christoph Stiller. “Video based localization for Bertha”. In: *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE. 2014, pp. 1231–1238.
- [33] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.
- [34] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. “Speeded-up robust features (SURF)”. In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.
- [35] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. “Brief: Binary robust independent elementary features”. In: *Computer Vision–ECCV 2010* (2010), pp. 778–792.
- [36] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. “ORB: An efficient alternative to SIFT or SURF”. In: *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE. 2011, pp. 2564–2571.
- [37] Martin A Fischler and Robert C Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Communications of the ACM* 24.6 (1981), pp. 381–395.
- [38] Hauke Strasdat, JMM Montiel, and Andrew J Davison. “Real-time monocular SLAM: Why filter?” In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 2657–2664.

- [39] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. “Keyframe-based visual-inertial odometry using nonlinear optimization”. In: *The International Journal of Robotics Research* 34.3 (2015), pp. 314–334.
- [40] Paul Furgale and Timothy D Barfoot. “Visual teach and repeat for long-range rover autonomy”. In: *Journal of Field Robotics* 27.5 (2010), pp. 534–560.
- [41] Winston Churchill and Paul Newman. “Practice makes perfect? Managing and leveraging visual experiences for lifelong navigation”. In: *2012 IEEE International Conference on Robotics and Automation*. Minnesota, USA: Ieee, May 2012, pp. 4525–4532.
- [42] Peer Neubert, Niko Sunderhauf, and Peter Protzel. “Appearance change prediction for long-term navigation across seasons”. In: *Mobile Robots (ECMR), 2013 European Conference on*. IEEE. 2013, pp. 198–203.
- [43] Nan Yang, Rui Wang, and Daniel Cremers. “Feature-based or Direct: An Evaluation of Monocular Visual Odometry”. In: *arXiv:1705.04300 [cs.CV]* (2017), pp. 1–12. arXiv: 1705.04300.
- [44] Jakob Engel, Vladlen Koltun, and Daniel Cremers. “Direct Sparse Odometry”. In: *arXiv:1607.02565*. July 2016.
- [45] Steven Lovegrove, Andrew J Davison, and Javier Ibanez-Guzmán. “Accurate visual odometry from a rear parking camera”. In: *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE. 2011, pp. 788–793.
- [46] Kumar Shaurya Shankar and Nathan Michael. “Robust direct visual odometry using mutual information”. In: *Safety, Security, and Rescue Robotics (SSRR), 2016 IEEE International Symposium on*. IEEE. 2016, pp. 9–14.
- [47] Hatem Alismail, Michael Kaess, Brett Browning, and Simon Lucey. “Direct Visual Odometry in Low Light Using Binary Descriptors”. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 444–451.
- [48] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.
- [49] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. “Segnet: A deep convolutional encoder-decoder architecture for image segmentation”. In: *arXiv preprint arXiv:1511.00561* (2015).
- [50] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).
- [51] Alex Kendall, Matthew Grimes, and Roberto Cipolla. “PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization”. In: *()*, pp. 2938–2946. arXiv: arXiv:1505.07427v3.
- [52] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. “CNN-SLAM: Real-Time Dense Monocular SLAM With Learned Depth Prediction”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

- [53] Alex Kendall and Roberto Cipolla. “Geometric Loss Functions for Camera Pose Regression With Deep Learning”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.
- [54] Michael J. Milford and Gordon F. Wyeth. “SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2012, pp. 1643–1649.
- [55] Ronald Clark, Sen Wang, Andrew Markham, Niki Trigoni, and Hongkai Wen. “VidLoc: A Deep Spatio-Temporal Model for 6-DoF Video-Clip Relocalization”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.
- [56] Amaury Dame and Eric Marchand. “Mutual Information-Based Visual Servoing”. In: *IEEE Transactions on Robotics* 27.5 (Oct. 2011), pp. 958–969.
- [57] François Chaumette and Seth Hutchinson. “Visual servo control. I. Basic approaches”. In: *IEEE Robotics & Automation Magazine* 13.4 (2006), pp. 82–90.
- [58] Christophe Collewet and Eric Marchand. “Photometric visual servoing”. In: *IEEE Transactions on Robotics* 27.4 (2011), pp. 828–834.
- [59] Amaury Dame and Eric Marchand. “Using mutual information for appearance-based visual path following”. In: *Robotics and Autonomous Systems* (2012), pp. 1–32.
- [60] Claude E. Shannon. “A mathematical theory of communication”. In: *Bell system technical journal* 27 (1948), pp. 379–423, 623–656.
- [61] Paul Viola and William M Wells III. “Alignment by maximization of mutual information”. In: *International journal of computer vision* 24.2 (1997), pp. 137–154.
- [62] Frederik Maes, Andre Collignon, Dirk Vandermeulen, Guy Marchal, and Paul Suetens. “Multimodality image registration by maximization of mutual information”. In: *IEEE transactions on medical imaging* 16.2 (1997), pp. 187–198.
- [63] Josien PW Pluim, JB Antoine Maintz, and Max A Viergever. “Mutual-information-based registration of medical images: a survey”. In: *IEEE transactions on medical imaging* 22.8 (2003), pp. 986–1004.
- [64] Robert R Schaller. “Moore’s law: past, present and future”. In: *IEEE spectrum* 34.6 (1997), pp. 52–59.
- [65] Khronos. *OpenGL - The Industry’s Foundation for High Performance Graphics*. <http://www.khronos.org/opengl/>. accessed 9/8/2017.
- [66] NVIDIA. *Parallel Programming and Computing Platform*. <https://developer.nvidia.com/cuda-zone>. accessed 9/8/2017.
- [67] Khronos. *OpenCL - The open standard for parallel programming of heterogeneous systems*. <http://www.khronos.org/opencl/>. accessed 9/8/2017.
- [68] NVIDIA. *Self-Driving Vehicles Development Platform / NVIDIA DRIVE PX*. <http://www.nvidia.com/object/drive-px.html>.
- [69] Dave Shreiner, Bill The Khronos OpenGL ARB Working Group, et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009.

- [70] Richard S Wright Jr, Nicholas Haemel, Graham M Sellers, and Benjamin Lipchak. *OpenGL SuperBible: comprehensive tutorial and reference*. Pearson Education, 2010.
- [71] Daniel Ruijters. “Efficient GPU-based texture interpolation using uniform B-splines”. In: *Journal of Graphics, GPU . . .* (2008), pp. 61–69.
- [72] Richard M Murray, Zexiang Li, S Shankar Sastry, and S Shankara Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 1994.
- [73] Hauke Strasdat. “Local accuracy and global consistency for efficient SLAM”. PhD thesis. London: Imperial College, 2014.
- [74] Winston Churchill. “Experience Based Navigation: Theory, Practice and Implementation”. PhD thesis. Oxford, United Kingdom: University of Oxford, 2012.
- [75] Alastair Harrison and Paul Newman. “TICSync: Knowing when things happened”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 356–363.
- [76] Ken Shoemake. “Animating rotation with quaternion curves”. In: *ACM SIGGRAPH computer graphics*. Vol. 19. 3. ACM. 1985, pp. 245–254.
- [77] Alexandri Zavodny, Patrick J. Flynn, and Xin Chen. “Textured Mesh Generation of Extracted Regions From Urban Range-Scanned Lidar Data”. In: *Multimedia and Expo (ICME), . . .* (2011).
- [78] Chen Zhu and Wee Kheng Leow. “Textured mesh surface reconstruction of large buildings with multi-view stereo”. In: *The Visual Computer* 29.6-8 (May 2013), pp. 609–615.
- [79] M. Cummins and P. Newman. “FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance”. In: *The International Journal of Robotics Research* 27.6 (June 2008), pp. 647–665.
- [80] Reinhard Koch. “Dynamic 3-D scene analysis through synthesis feedback control”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 15.6 (1993), pp. 556–568.
- [81] Andrew I Comport, Ezio Malis, and Patrick Rives. “Real-time quadrifocal visual odometry”. In: *The International Journal of Robotics Research* 29.2-3 (2010), pp. 245–266.
- [82] Frank Steinbrucker, Jürgen Sturm, and Daniel Cremers. “Real-time visual odometry from dense RGB-D images”. In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE. 2011, pp. 719–722.
- [83] Ming Li, Xin Chen, Xin Li, Bin Ma, and P M B Vitanyi. “The similarity metric”. In: *Information Theory, IEEE Transactions on* 50.12 (2004), pp. 3250–3264.
- [84] Ankur Handa, Thomas Whelan, John McDonald, and Andrew J. Davison. “A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM”. In: *IEEE Intl. Conf. on Robotics and Automation, ICRA*. Hong Kong, China, May 2014.
- [85] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Vol. 43. 1999, pp. 164–75.

- [86] Les Piegl and Wayne Tiller. “The NURBS book. 1997”. In: *Monographs in Visual Communication* (1997).
- [87] Daniel Ruijters and P Thévenaz. “GPU prefilter for accurate cubic B-spline interpolation”. In: *The Computer Journal* 55 (2012), pp. 15–20.
- [88] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. “A flexible technique for accurate omnidirectional camera calibration and structure from motion”. In: *Proceedings of the Fourth IEEE International Conference on Computer Vision Systems, ICVS 06*. Vol. 2006. 2006, p. 45.
- [89] Alexander D. Stewart. “Localisation using the Appearance of Prior Structure”. PhD thesis. Oxford, United Kingdom: University of Oxford, 2014.
- [90] Sameer Agarwal, Keir Mierle, et al. *Ceres Solver*. <http://ceres-solver.org>.
- [91] Peter Nelson, Winston Churchill, Ingmar Posner, and Paul Newman. “From Dusk till Dawn: Localisation at Night using Artificial Light Sources”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA, USA, May 2015.
- [92] Davide Scaramuzza, Ahad Harati, and Roland Siegwart. “Extrinsic self calibration of a camera and a 3d laser range finder from natural scenes”. In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE. 2007, pp. 4164–4169.
- [93] Ranjith Unnikrishnan and Martial Hebert. “Fast extrinsic calibration of a laser rangefinder to a camera”. In: (2005).
- [94] Andreas Geiger, Frank Moosmann, Ömer Car, and Bernhard Schuster. “Automatic camera and range sensor calibration using a single shot”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 3936–3943.
- [95] Carlos Guindel, Jorge Beltran, David Martin, and Fernando Garcia. “Automatic Extrinsic Calibration for Lidar-Stereo Vehicle Sensor Setups”. In: *arXiv preprint arXiv:1705.04085* (2017).
- [96] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. “A toolbox for easily calibrating omnidirectional cameras”. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE. 2006, pp. 5695–5701.
- [97] Gary Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).
- [98] Jon Louis Bentley. “Multidimensional binary search trees used for associative searching”. In: *Communications of the ACM* 18.9 (1975), pp. 509–517.
- [99] Peter J Huber et al. “Robust estimation of a location parameter”. In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101.
- [100] Vicon. *Vicon Vero*. <https://www.vicon.com/products/camera-systems/vero>.
- [101] Pedro Piniés, Lina Maria Paz, and Paul Newman. “Dense mono reconstruction: Living with the pain of the plain plane”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. May 2015, pp. 5226–5231.

- [102] Sarah Martull, Martin Peris, and Kazuhiro Fukui. “Realistic CG stereo image dataset with ground truth disparity maps”. In: *ICPR Workshop: TrakMark2012*. Vol. 111. 430. 2012, pp. 117–118.
- [103] Michael Tanner, Pedro Piniés, Lina Maria Paz, and Paul Newman. “DENSER Cities: A System for Dense Efficient Reconstructions of Cities”. In: *ArXiv e-prints* (2016). arXiv: 1604.03734 [cs.CV].