

Driven by Data: City-Scale Localisation at Night

Peter D. Nelson
New College



Supervisor:
Professor Paul Newman

Mobile Robotics Group
Department of Engineering Science
University of Oxford

October 2016

Peter D. Nelson
New College

Doctor of Philosophy
October 2016

**Driven by Data:
City-Scale Localisation at Night**

Abstract

This thesis is about facilitating the use of vast quantities of data for robotics applications, particularly for the tasks of mapping and localisation in the context of lifelong learning. Further, we tackle the specific problem of localising an autonomous vehicle at night, in urban environments, using vision.

In general, robotics applications have some unique and unusual data requirements. The curation and management of such data are often overlooked. An emerging theme is the use of large corpora of spatiotemporally indexed sensor data which must be searched and leveraged both offline and online. Without careful thought to their organisation, however, this leads to vast quantities of data which quickly become unmanageable. The current paradigm of collecting data for specific purposes and storing them in ad-hoc ways will not scale to meet this challenge. In the first part of this thesis, we present the design and implementation of a data management framework that is capable of dealing with large datasets, and provides functionality required by many offline and online robotics applications. We systematically identify the data requirements of these applications and design a relational database that is capable of meeting their demands.

This database framework is then used to facilitate the storage and retrieval of maps in our visual nighttime navigation system, a problem which has its own unique set of challenges. Despite it being dark exactly half of the time, surprisingly little work has addressed these challenges. A defining aspect of nighttime urban scenes is the presence and effect of artificial lighting. By building a model of the environment which includes a representation of the spatial location of every light source, localisation becomes possible using monocular cameras. One major challenge we face is the gross change in light appearance as a function of distance, due to flare, saturation, and bleeding. To overcome this, we model the appearance of each light as a function of vehicle location, using this to inform our data association decisions and to regularise the cost function which is used to infer vehicle pose. We demonstrate that our system is able to localise successfully at night, over 40km, in situations where a traditional point feature based system fails.

Statement of Authorship

This thesis is submitted to the Department of Engineering Science, University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy. This thesis is entirely my own work, and except where otherwise stated, describes my own research.

Peter Nelson, New College

Funding

The work described in this thesis was funded by the Engineering and Physical Sciences Research Council (EPSRC).

To my beloved grandparents

Acknowledgements

First and foremost I would like to thank my supervisor, Professor Paul Newman, for his invaluable guidance and support throughout the whole DPhil process.

Huge thanks go to the many members of the Mobile Robotics Group (MRG), both past and present, who have made it such a fantastic team to be a part of. Also to my colleagues at Oxbotica with whom I have the privilege to work with on some amazing projects.

Specific thanks go in no particular order to Winston Churchill, Alex Stewart, Colin McManus, Pedro Piniés, Geoff Hester, and Chris Prahacs for all of their help, including with proofreading, answering my many questions, and nighttime data collection missions.

Last but definitely not least, I am very grateful to my family and friends for supporting me during my time at Oxford.

Contents

1	Introduction	1
1.1	Motivations	1
1.2	Contributions	3
1.3	Thesis Structure	4
1.4	Publications and Exploitation	4
2	Visual Mapping and Localisation	6
2.1	Introduction	6
2.2	Preliminaries: Rigid-Body Transformations	7
2.3	Problem Definition	8
2.4	What is a Map?	9
2.4.1	Globally Consistent vs Locally Consistent Maps	10
2.4.2	Data and Query Requirements	13
2.5	Overview of Teach and Repeat	15
2.5.1	Mapping	17
2.5.2	Localisation	18
2.6	State Parameterisation	20
2.6.1	Rotations	21
2.6.2	Chosen Representation	22
2.6.3	3-DoF Versus 6-DoF Parameterisation	23
2.7	Non-linear Optimisation	24
2.7.1	Problem Formulation	24

2.7.2	Data Association	28
2.7.3	Robust Estimation	32
2.8	Visual Sensors	35
2.8.1	Pinhole Camera Model	35
2.8.2	Lens Distortion	37
2.9	Visual Structure from Motion	38
2.9.1	Structure from Motion Derivatives	39
2.10	Visual Pose Estimation	41
2.10.1	Infinitesimal Perturbation Parameterisation	41
2.10.2	Pose Estimation Derivatives	43
2.10.3	Depth Measurements	45
2.10.4	Incorporating the Motion Model	46
2.11	Summary	47
3	Database Design and Implementation	48
3.1	Introduction	48
3.2	Relational Database Fundamentals	51
3.2.1	SQL	53
3.3	System Requirements	57
3.3.1	Concrete Requirements	58
3.4	Background and Related Work	59
3.4.1	Data Management within Robotics	60
3.4.2	Data Management in a Broader Context	61
3.4.3	Relational DBMS vs NoSQL	63
3.5	Architectural Overview	65
3.6	Filesystem Layer	66
3.6.1	Serialisation with Google Protocol Buffers	68

3.7	Database Layer	69
3.7.1	Raw sensor data	69
3.7.2	Annotations	72
3.7.3	Multigraph Maps	74
3.7.4	Miscellaneous Data	76
3.8	System Implementation	77
3.8.1	Common Subqueries	78
3.8.2	Creating and Managing Maps for Online Use	79
3.9	Application APIs	83
3.10	Optimising Performance with Indexes	85
3.10.1	Equality Selections	86
3.10.2	Temporal Selections	86
3.10.3	Spatial Selections	87
3.10.4	Performance Tradeoffs	87
3.11	PostgreSQL vs SQLite	89
3.11.1	Experiments and Analysis	89
3.12	Example Use Cases	92
3.12.1	Scene Priors	92
3.12.2	Scene Variation	94
3.12.3	Experience Based Navigation	97
3.13	Summary	98
4	Navigation at Night	100
4.1	Introduction	100
4.2	Motivation	101
4.3	Related Work	104
4.4	System Overview	107

4.5	Mapping	107
4.5.1	Light Detection and Tracking	108
4.5.2	Structure from Motion	111
4.5.3	Modelling Appearance	112
4.6	Localisation	112
4.6.1	Pose Prediction	116
4.6.2	Points Correspondence	116
4.6.3	Pose Update	119
4.7	Database Usage	121
4.7.1	Queries	122
4.8	Systems and Datasets	123
4.8.1	Hardware	123
4.8.2	Datasets	124
4.9	Results and Analysis	130
4.9.1	Map Analysis	132
4.9.2	Light Appearance and Classification	132
4.9.3	How Many Landmarks Are Sufficient?	139
4.9.4	How Many Appearances Are Sufficient?	144
4.9.5	Other Parameters	146
4.9.6	Overall Localisation Performance	147
4.10	Summary	152
5	Conclusions	155
5.1	Summary of Contributions	155
5.2	Future Work	157
5.3	Concluding Remarks	160
	Bibliography	161

Symbols and Notation

Symbol	Description
$\mathbf{1}$	The identity matrix
$\mathbf{0}$	The zero matrix
a	A real-valued scalar: $a \in \mathbb{R}$
$\mathbf{a}$	A real-valued $N \times 1$ column vector: $\mathbf{a} \in \mathbb{R}^N$
$\mathbf{A}$	A real-valued $N \times M$ matrix: $\mathbf{A} \in \mathbb{R}^{N \times M}$
∇f	Gradient of scalar-valued function $f : \mathbb{R}^N \rightarrow \mathbb{R}$, i.e. a vector of its first partial derivatives: $\nabla f : \mathbb{R}^N \rightarrow \mathbb{R}^N$
$\nabla^2 f$	Hessian of scalar-valued function $f : \mathbb{R}^N \rightarrow \mathbb{R}$, i.e. a matrix of its second partial derivatives: $\nabla^2 f : \mathbb{R}^N \rightarrow \mathbb{R}^{N \times N}$
$\oplus$	Transformation composition operator
$\times$	Scalar, vector, or matrix multiplication operator
$\ \mathbf{a}\ $	Euclidean norm (magnitude) of vector $\mathbf{a}$
$\sim \mathcal{N}(\mathbf{x}, \mathbf{P})$	Normally distributed with mean $\mathbf{x}$ and covariance $\mathbf{P}$
$p(y \mid x)$	Probability density of a conditional random variable y given x
$\mathbf{T}_{a,b}$	A 4×4 transformation matrix $\in \text{SE}(3)$ that transforms points from reference frame b to reference frame a

Chapter 1

Introduction

“ *Not all those who wander are lost.* ”

J. R. R. Tolkien, *The Fellowship of the Ring*, 1954

1.1 Motivations

The realisation of lifelong learning, in the context of autonomous vehicles, has two major implications for roboticists and the systems we develop.

The first implication is that we ultimately need to deal with the vast amount of information and experience that will accumulate over the course of entire lifetimes. This ever-expanding repository of knowledge must be somehow stored and curated in a systematic way. We require the ability to recall individual ‘memories’ when needed, and the ability to augment the totality of stored experience with *new* memories. These problems relate to the *learning* aspect of lifelong learning.

The second implication is that our autonomous vehicles will be expected to operate continuously over long periods of time, and therefore will be required to

handle wide variations in environmental conditions. Perhaps one of the most extreme (albeit regular and predictable) forms of such variation occurs as a result of the day-night cycle – from a visual perspective, at least, the appearance of the world changes drastically as night descends. This relates to the *lifelong* aspect of lifelong learning.

In this thesis we explore both of these aspects. We begin with the concrete problem of how best to store, manage, curate, and query vast amounts of raw and processed data, such that they can be leveraged for use within robotics applications. Of these, one application that we look at in particular is the storage and retrieval of maps that are consumed by localisation systems. These have some challenging requirements, such as the need to scale efficiently as maps grow ever larger.

It has become clear to us that ad-hoc strategies for dealing with such data do not scale sufficiently to meet the challenges involved in realising long-term autonomy. When it comes to handling large datasets, we find that without a consistent and systematic approach to data management, valuable data become difficult and slow to access, tools get needlessly rewritten, and we often find it easier to simply collect more data than to reuse what we already have. Instead, we would like the ability to query our existing data for what we need, which crucially may be for a different purpose than that for which the data were originally collected.

Once we have a data management system in which we can effectively store maps of the environment, we focus on the challenge of localising an autonomous vehicle at night – a challenge that has not received nearly as much attention as the daytime case. Our goal is to overcome this using conventional cameras due to their low cost, however, we face issues such as lens flare, over-saturation, and motion blur. These conditions create difficulties for many current state-of-the-art systems, which perform localisation by detecting and matching small *point features* in the camera images. We therefore propose the use of larger, more stable landmarks that are common in urban nighttime scenes: artificial light sources such as street lights,

illuminated signs, and brightly lit windows. These are easy to detect and track, and moreover appear very different from different viewpoints. We make use of this additional appearance information, easily accessible via our database framework, in order to improve the performance and robustness of our navigation system.

In the remainder of this chapter we summarise the main contributions of the thesis and explain how the following chapters are organised.

1.2 Contributions

The main contributions of this thesis are:

- We design and implement a scalable database framework that is suitable for robotics applications. User-defined annotations coexist with raw sensor data, resulting in a living, growing repository of all data we have ever collected. We demonstrate its ability to handle almost 90TB of data collected over a period of 4.5 years.
- We optimise and tune the database framework for the storage and retrieval of different types of map, including metric and topometric, by a robot, at runtime. The system’s suitability for such online tasks is assessed, and its utility is demonstrated via several motivating use cases, including our own nighttime navigation system.
- We develop NightNav: a vision-based teach and repeat navigation system for an autonomous vehicle, operating in urban areas at night. Bright, artificial light sources are used as landmarks, and their highly variable appearance characteristics are modelled in order to improve data association and robustness. We demonstrate the system’s ability to localise on 40km of nighttime data, collected around Oxford over a period of five months.

1.3 Thesis Structure

Chapter 2 provides an overview of the theory underpinning mapping and localisation. This material is required to understand the nighttime navigation system described in Chapter 4. We formulate the mapping and localisation problems first in general terms, then detail how they can be solved using vision. Additionally, we examine the requirements for storing and accessing different kinds of map, providing motivation for the database system that we develop next.

Chapter 3 sets out the design and implementation of our database framework that is used for the storage and querying of maps and to manage our large datasets. It is here where we motivate the need for a systematic and consistent approach to data management within the context of robotics research. In an evaluation of the database framework we compare the suitability of two different storage back-ends for both online and offline applications, and analyse the performance of different types of query. The utility of the system is demonstrated through a selection of use cases.

We then turn our attention to the problem of navigation at night in Chapter 4. A teach-and-repeat system is presented that builds a map of artificial light sources in an offline phase, and localises against that map in a subsequent online phase. We compare our approach against a standard point-feature-based localisation system and demonstrate how it performs favourably under challenging nighttime conditions.

Finally we summarise our contributions, offer some ideas for future work, and conclude the thesis in Chapter 5.

1.4 Publications and Exploitation

The localisation system described and evaluated in Chapter 4 was presented as “From Dusk till Dawn: Localisation at Night using Artificial Light Sources” at

IEEE International Conference on Robotics and Automation (ICRA) in Seattle, USA, May 2015 [66].

The database system described and analysed in Chapter 3 was presented as “Building, Curating, and Querying Large-scale Data Repositories for Field Robotics Applications” at the International Conference on Field and Service Robotics (FSR) in Toronto, Canada, June 2015 [67].

Chapter 2

Visual Mapping and Localisation

“ *It’s like driving a car at night. You never see further than your headlights, but you can make the whole trip that way.* ”

E. L. Doctorow, *Writers at Work*, 1988

2.1 Introduction

In this chapter we introduce the concepts of mapping and localisation in the context of robotics. We begin by defining what a map is before examining the data and query requirements of systems that will build and localise against such a map. These requirements will inform the design of our database system that is the subject of Chapter 3. Later, we turn our attention to the background theory and algorithms that are utilised in the development of our nighttime navigation system, NightNav, in Chapter 4. A high-level overview of the concept of a *teach and repeat* system that performs mapping and localisation as separate, independent phases is provided, then these problems are defined in general. We show how their probabilistic formulations

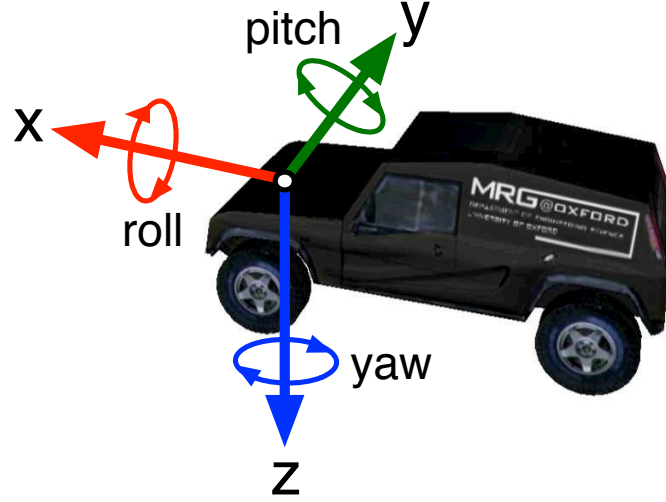


Figure 2.1: The coordinate convention used throughout this thesis, with positive x pointing forward, positive y pointing right, and positive z pointing down. Rotations around these axes are named roll, pitch, and yaw, respectively.

can lead to a non-linear system of equations which can be solved using standard iterative algorithms, as well as how the system can be made robust to outliers. Finally, we detail the specifics of solving these problems with visual sensors.

2.2 Preliminaries: Rigid-Body Transformations

Before we delve into the details of visual mapping and localisation, we first review the concepts of rigid-body transformations. These are used extensively throughout the work presented in this thesis.

A rigid-body transformation $\mathbf{T}_{A,B} \in \text{SE}(3)$ maps 3D points in the origin reference frame B into 3D points in the destination frame A . $\mathbf{T}_{A,B}$ can be represented as a 4×4 homogeneous matrix with the following form:

$$\mathbf{T}_{A,B} = \begin{bmatrix} \mathbf{R}_{A,B} & \mathbf{t}_{A,B} \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (2.1)$$

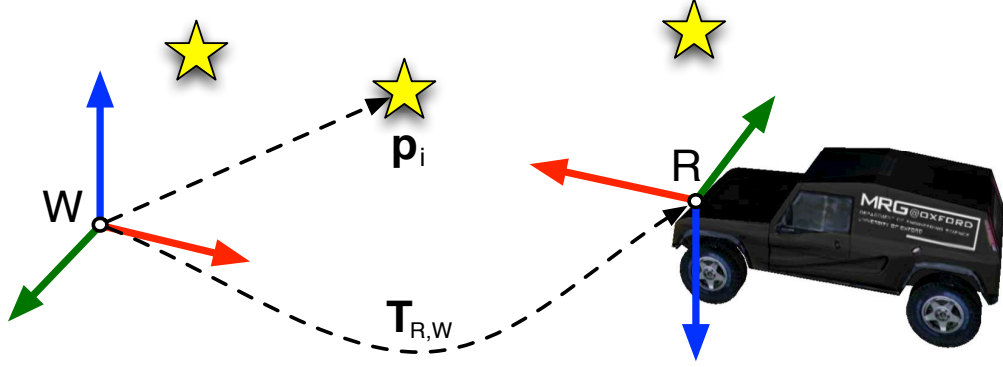


Figure 2.2: A robot moves through the world which contains some landmarks at points $\mathbf{p}_i$, defined in frame W . The robot senses these landmarks relative to its local frame R . The transform $\mathbf{T}_{R,W}$ parameterises the pose of the robot, i.e., its position and orientation relative to W .

where $\mathbf{R}_{A,B} \in \text{SO}(3)$ is a 3×3 *rotation matrix* and $\mathbf{t}_{A,B} \in \mathbb{R}^3$ is a *translation vector*. Together these encode the orientation and position of the destination frame A relative to B .

Given a 3D point in frame B represented by the 4×1 homogeneous vector $\mathbf{p}_B$, its corresponding position in frame A is given by $\mathbf{p}_A = \mathbf{T}_{A,B}\mathbf{p}_B$.

We use the standard right-handed aeronautical coordinate system convention for camera and vehicle base frames as shown in Figure 2.1. In this convention, positive x points forwards, positive y points right, and positive z points down.

2.3 Problem Definition

Consider a robot moving through a 3D world as shown in Figure 2.2. The world contains some landmarks which are located at points $\mathbf{p}_i \in \mathbb{R}^3$, defined in the world frame W . The robot is able to sense these landmarks relative to its local frame R . The robot may have never seen any landmarks before, or it may have seen a *lifetime's* worth of landmarks. The problems we consider in this scenario are:

1. How do we build a map of the world (i.e. estimate landmark locations $\mathbf{p}_i$)?

2. How do we localise the robot within the map (i.e. estimate the transform $\mathbf{T}_{R,W}$ that parameterises the *pose* of the robot)?
3. How do we store the map for later retrieval, particularly as it grows ever larger over the robot’s lifetime?
4. How do we query a large and growing map efficiently (i.e. from the potentially millions of landmarks in our map, how do we quickly find ones that are nearby)?

Problems (1) and (2) are the subject of this chapter. Problems (3) and (4) are examined later in Chapter 3. However, in order to solve any of them, we must first define what a map actually is and how it is represented.

2.4 What is a Map?

Fundamentally, a map is a representation of the structure of the environment surrounding the robot. Typically it will have some *metric* structure, meaning the location of both the robot and the landmarks in the world can be expressed as a coordinate vector defined in some reference frame. In general it does not have to be metric, for example, the iconic ‘tube map’ of the London Underground¹ is a *topological* map as it specifies only the topology of the network without any metric information. A *topometric* map is a hybrid representation consisting of a topological map that incorporates additional metric information. A simple example is shown in Figure 2.3. As we are ultimately interested in navigating within a Euclidean space, for the remainder of this thesis we will consider only metric and topometric representations.²

¹<http://content.tfl.gov.uk/standard-tube-map.pdf>

²We note however that the database system we develop in Chapter 3 is fully capable of handling purely topological map representations as well.

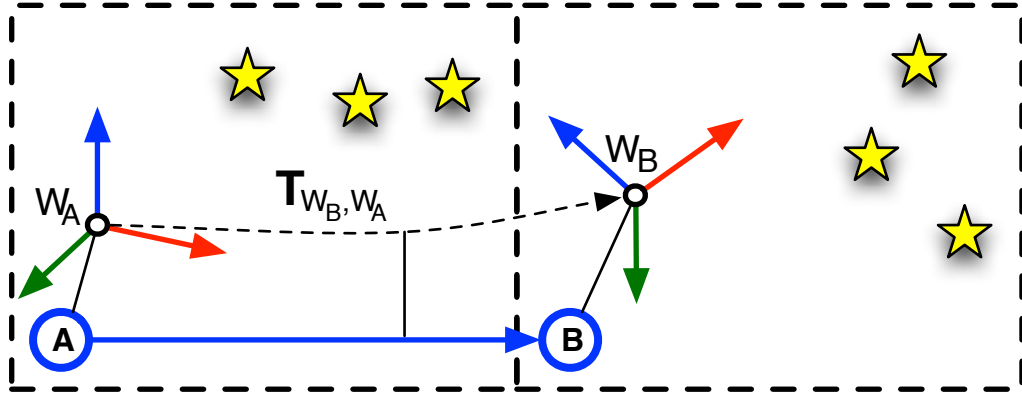


Figure 2.3: Example topometric map. The topological graph structure is shown in blue and consists of two nodes connected by a single edge. Solid black lines show the association between elements of the graph structure with the metric map sub-elements. Each node represents a small, locally metric sub-map with an associated base reference frame, and landmarks defined in that frame. The edge represents a transform that maps points from the source node’s frame to the destination node’s frame. With this type of map, the robot typically seeks to localise in one sub-map using the landmarks in that sub-map. Failure to localise in the current sub-map may induce a search in neighbouring ones, i.e. in those nodes connected by edges to the current node.

2.4.1 Globally Consistent vs Locally Consistent Maps

One important property of a metric or topometric map is its *consistency*. Informally, a map’s consistency refers to how similarly it behaves to a Euclidean space. It’s important for us to know the degree of this similarity as the mathematics that govern how robots interact with their environment typically depend on them operating within a such a space. In a *globally consistent* map, every distinct map location represents a distinct location in the world. Moreover, the distance and bearing between any two arbitrary map locations (however far apart) accurately and uniquely represents the distance and bearing between the corresponding real-world locations.

This sounds ideal, but the task of building and maintaining large globally consistent maps is challenging. It has been known for many years that uncertainty caused by noisy sensor readings will lead to inconsistency, unless global corrections can be applied [6] (e.g. by detecting loop closures). The effort involved in applying these corrections increases with the size of the map, a fact which has been evident

since the early days of SLAM. As an example, it was once popular to use nonlinear variants of the Kalman Filter [44] (e.g. Extended Kalman Filter (EKF)) to solve the classical *simultaneous localisation and mapping* (SLAM) problem. However, when used naively to build a single global map these techniques quickly become computationally intractable as the map grows – complexity growing quadratically with the total number of landmarks. Thus the community began to question whether global consistency was really necessary.

In [6], Brooks posited that in order to enable robot autonomy, maps need only be *locally consistent*. This means the map is only locally Euclidean: every map location is surrounded by a small region that can be considered approximately metrically consistent. Beyond this region, the guarantees afforded by a Euclidean space no longer hold. One consequence of this is that many distinct points in a locally consistent map can refer to the same physical location in the world. Figure 2.4 shows the result of this consequence graphically, and further illustrates the differences between globally and locally consistent map representations.

Despite these limitations, locally consistent maps are more than sufficient for robot navigation and planning tasks. As long as at each instant in time the robot’s sensor observations are confined to its local surroundings, we can assume that it is in a metric space. Map trajectories can be built by simply integrating open-loop odometry: even though there is nothing to stop errors from accumulating indefinitely, over short distances the error will be small. The crucial advantage is that by dropping the requirement for global consistency, we can build and localise within large maps that do not need to be expensively globally optimised.

This insight has since been taken advantage of in many works. Brooks proposed an abstract graph-based topological map representation in [6]. The Atlas Framework, presented by Bosse et al. in [5], uses a graph structure to represent an interconnected set of local coordinate frames, each containing a map. In

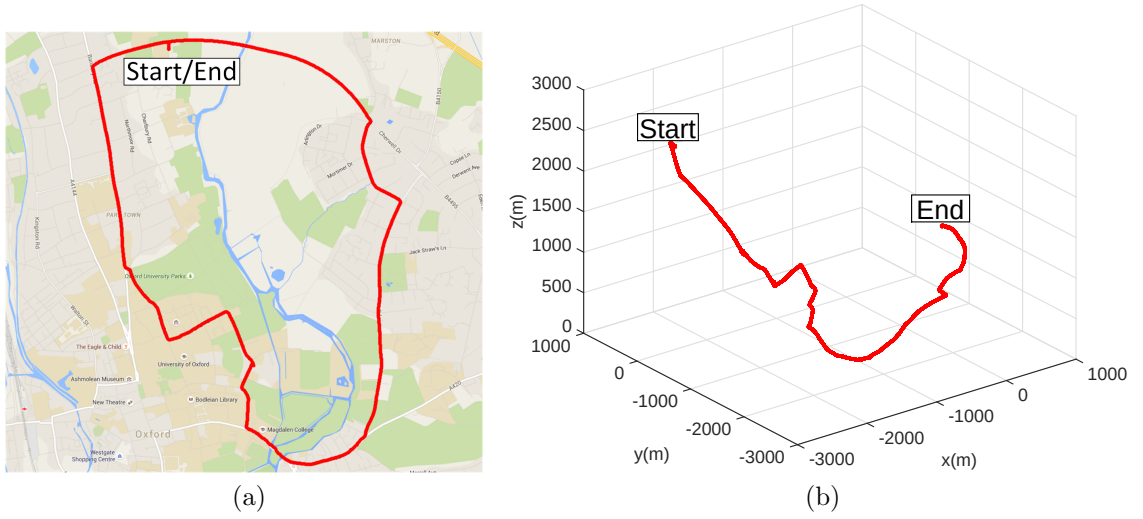


Figure 2.4: Illustration of the difference between globally and locally consistent maps. On the left is shown a robot’s route driven in a globally consistent reference frame, starting and ending at the same point. On the right is shown the output of our visual odometry (VO) system, integrated over the whole route, embedded in a global frame. This can be thought of as a locally consistent manifold embedded in a higher dimensional space. The start and end points do not line up in this representation, even though physically they are the same place, demonstrating that VO is prone to drift significantly over the course of the route. Within small local regions, however, this drift is negligible. Therefore, as we are only interested in observations of landmarks within these regions, we are still able to navigate. One disadvantage is that we are constrained to move only along the manifold: a globally consistent map would allow us to take ‘shortcuts’ off of the driven route.

[38], Howard proposed topometric mapping on a manifold embedded in a higher-dimensional space, and so built locally consistent ‘overlapping’ 2D planar sub-maps with no global consistency. Furgale and Barfoot extend this sub-mapping approach in 3D allowing them to do teach-and-repeat in an unstructured outdoor environment [24]. These approaches can be thought of as discretising the space around the manifold shown in Figure 2.4b and connecting the discrete sub-maps via a graph structure. An alternative approach by Sibley et al. in [78] instead uses a continuous relative representation where only the observations within a small window around the robot’s current location are bundle adjusted.

2.4.2 Data and Query Requirements

We now consider the data and query requirements of metric and topometric maps. This will inform our design of a suitable database system in Chapter 3. To serve as motivation, we focus on some concrete map implementations (including those that we use in Chapter 4) but also make clear the requirements that are common to all metric and topometric map representations.

2.4.2.1 Metric Maps

The map we use for NightNav can be considered a locally consistent metric map, similar to the manifold shown in Figure 2.4b. In this map representation, a landmark is a 3D point, $\mathbf{p} \in \mathbb{R}^3$, defined in a single global world frame. As explained in more detail later in Chapter 4, each landmark is associated with some additional information about its appearance (i.e. image patches capturing what it looked like at the time it was mapped).

To localise within a metric map, we will want to recall candidate landmarks that are within the vicinity of the vehicle’s current position, i.e. a nearest-neighbour query. We also want to query for the data that are associated with each landmark, e.g. in the case of NightNav, image patches representing its appearance.

These base requirements hold for other forms of metric map. For example, nearest-neighbour is a common query to run on a GPS-based map. Landmarks in this case will be 2D coordinates in latitude and longitude. More generally, landmarks may be arbitrary geometric objects such as lines, planes, and polyhedrons. Independent of their geometry, they will have additional data associated with them that needs to be interrogated at runtime.

2.4.2.2 Topometric Maps

Topometric maps have their own set of requirements. As an example, Experience-Based Navigation (EBN) [12] is a state-of-the-art mapping and localisation framework that models the world as a ever-growing set of ‘experiences’. This system is explained in more detail in Chapter 3 where we present it as a use case for our database system. The EBN map representation can be considered a locally consistent topometric map like that shown in Figure 2.3, meaning that landmarks are 3D points $\mathbf{p} \in \mathbb{R}^3$ defined relative to some node (in this case, nodes represent a stereo camera keyframe) in the topological graph. Adjacent keyframes are linked together in linear chains, but are also linked across experiences and loop closures, resulting in a highly connected and vast graph structure. Localisation requires finding all landmarks that are defined relative to a particular keyframe, amounting to the need to query data that is attached to that node. When the robot is unable to localise within the current keyframe, EBN tries to localise in neighbouring nodes. This therefore requires graph traversal operations such as depth and breadth-first search. Moreover, these queries need to run in real-time, even over huge maps.

Other graph-based maps have similar requirements, the main differences being what data they attribute to nodes and edges. As another example, ‘scene prior’ maps (given as another use case in Section 3.12.1) store coloured point cloud sub-maps on directed graph edges, with nodes representing the coordinate frames in which these point clouds are defined. In general, topometric maps make use of a full suite of graph creation, annotation, and traversal operations. They require the ability to create nodes, link nodes together via edges, and to attach data (e.g. keyframes, landmarks, transforms) to both nodes and edges.

2.4.2.3 Concrete Requirements

To support both metric and topometric maps we therefore require the ability to perform the following operations:

1. Create landmarks that are 2D or 3D geometric objects.
2. Create an arbitrary graph structure with nodes linked by edges.
3. Annotate landmarks, nodes, and edges with arbitrary data.
4. Fast (i.e. suitable for real-time systems) nearest-neighbour queries on geometric objects.
5. Fast (i.e. suitable for real-time systems) graph traversal, e.g. find neighbouring nodes, depth-first search.

We return to these requirements in Chapter 3 when we consider them in the design of our database system. For the remainder of this chapter we turn our attention to the dual questions of how a map is built, and how we localise within it.

2.5 Overview of Teach and Repeat

In this section we begin by examining how both mapping and localisation can be solved concurrently, before introducing the concept of teach and repeat that performs one after the other.

We have already established that for a mobile robot to successfully navigate and interact with the world it needs to have some representation of the structure of the world—a map—as well as an idea of where it is located in the world. To do this, it must be capable of *sensing* the world. A robot’s sensors may be divided into two broad categories: *interoceptive*, which measure some aspect of the robot’s internal

state (e.g. steering angle, velocity, acceleration); and *exteroceptive*, which provide measurements of local map features (e.g. distance or angle to a landmark).

Solving the classical *Simultaneous Localisation and Mapping* (SLAM) problem has been the focus of much early work in robotics. At time t , the problem can be expressed probabilistically as estimating the joint posterior density:

$$p(\mathbf{x}_{1:t}, \mathcal{M} \mid \mathbf{Z}_{1:t}, \mathbf{U}_{1:t}) \quad (2.2)$$

where:

$\mathbf{x}_{1:t}$ is the set of robot poses at each time step.

$\mathcal{M}$ is the map, e.g. in the case of NightNav: $\mathcal{M} = \{\mathbf{p}_i \in \mathbb{R}^3\}$.

$\mathbf{Z}_{1:t} = \{\mathbf{z}_1, \dots, \mathbf{z}_t\}$ is the set of exteroceptive sensor measurements observed at each time step.

$\mathbf{U}_{1:t} = \{\mathbf{u}_1, \dots, \mathbf{u}_t\}$ is the set of interoceptive control inputs at each time step.

Intuitively, both the robot's pose and the environment are initially unknown and the task is to build a map while simultaneously localising within it. However, in many situations this problem is unnecessarily general. For example, rarely if ever should an autonomous car find itself in an environment it knows *nothing* about, and yet still be expected to function as normal. It is far more likely that it will already have a map within which it can localise.

A *teach and repeat* system is one in which the mapping and localisation processes are decoupled in this manner. We first perform a mapping phase where a survey vehicle is driven manually around a route, passively recording data from its sensors which are then used to build a map offline (and therefore not subject to the limitations of real-time processing). On subsequent traversal phases the robot is initialised

to somewhere within this map and is able to repeat the route autonomously. This reduces both the conceptual and computational complexity of the problem as the map can now be considered immutable and independent of the location of the robot. This approach has recently been applied with great success in large-scale, unstructured outdoor environments, for example in [25, 59]. Our system takes inspiration from these approaches and is therefore structured in the same way.

We examine these two problems separately in the following sections.

2.5.1 Mapping

Probabilistically, solving the mapping problem on its own (the *teach* phase of teach and repeat) can be formulated as finding the most likely estimate of the map, $\hat{\mathcal{M}}$:

$$\hat{\mathcal{M}} = \arg \max_{\mathcal{M}} p(\mathcal{M} \mid \mathbf{x}_{1:t}, \mathbf{Z}_{1:t}) \quad (2.3)$$

Note that we can drop the control inputs as they are conditionally independent of $\mathcal{M}$, given the trajectory, $\mathbf{x}_{1:t}$. At first glance, we appear to have a causality dilemma: how is it possible to construct the map $\hat{\mathcal{M}}$ in isolation when it is conditioned on a somehow independent set of poses $\mathbf{x}_{1:t}$? To answer the question we first note that would be indeed possible to solve the full SLAM problem in the teach phase, then replay the immutable map and trajectory through it in subsequent localisation phases. Alternatively we can derive the poses from another source, for example by integrating the output of an open-loop odometry stream (this is the approach that NightNav takes). The next question that should be asked is: won't the ever-increasing errors in the odometry source's estimates cause our map to diverge wildly and become nonsense? The answer to this ties back to the comparison between locally consistent and globally consistent maps given in Section 2.4.1: yes, on a global scale, as errors from the odometry source build up, the map will become

inconsistent. However, within local regions, the error is relatively small and so although the ‘global’ pose is nonsense, the robot is still able to infer where it is relative to landmarks within its local vicinity.

2.5.2 Localisation

Localisation is the problem of inferring a robot’s pose (or *state* – typically position and orientation) relative to some known map, given measurements obtained from its sensors. In a probabilistic sense, a robot’s pose estimate $\hat{\mathbf{x}}_t$ at time t can be thought of as finding the *maximum a posteriori* estimate of the joint distribution:

$$\hat{\mathbf{x}}_t = \arg \max_{\mathbf{x}_t} p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{Z}_{1:t}, \mathbf{U}_{1:t}, \hat{\mathcal{M}}) \quad (2.4)$$

As new measurements and control inputs are received at time $t + 1$, the evolution of the posterior state distribution $\mathcal{P}$ can be defined recursively using Bayes’ theorem [4] and the Markov assumption, which postulates that the next state $\mathbf{x}_{t+1}$ is dependent solely on the current state $\mathbf{x}_t$:

$$\mathcal{P} = p(\mathbf{x}_{t+1} \mid \mathbf{x}_{1:t}, \mathbf{Z}_{1:t+1}, \mathbf{U}_{1:t+1}, \hat{\mathcal{M}}) = \quad (2.5)$$

$$\eta p(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, \mathbf{Z}_{1:t}, \mathbf{U}_{1:t}, \hat{\mathcal{M}}) \quad (2.6)$$

$$\times p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{u}_{t+1}) \quad (2.7)$$

$$\times p(\mathbf{z}_{t+1} \mid \mathbf{x}_{t+1}, \hat{\mathcal{M}}) \quad (2.8)$$

where:

Equation 2.6 is the recursive localisation posterior from the previous time step (multiplied by a constant scalar normalisation factor η).

Equation 2.7 represents the *motion model* which describes how the robot’s

pose evolves as a function of the control inputs. Note that the Markov assumption allows us to ignore the history of states that precede $\mathbf{x}_t$.

Equation 2.8 represents the *sensor model* which describes the likelihood of observing measurements $\mathbf{z}_{t+1}$ given the current state of the robot and map. Similarly due to the Markov assumption, only the state $\mathbf{x}_{t+1}$ need be considered.

If we make the assumption that both interoceptive and exteroceptive measurements are corrupted by some zero-mean Gaussian distributed noise, with covariance $\mathbf{Q}$ and $\mathbf{R}$ respectively, we can formulate the motion and sensor models as:

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mathbf{w}_{t+1}, \quad \mathbf{w}_{t+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \quad (2.9)$$

$$\mathbf{z}_{t+1} = \mathbf{h}(\mathbf{x}_{t+1}, \hat{\mathcal{M}}) + \mathbf{v}_{t+1}, \quad \mathbf{v}_{t+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \quad (2.10)$$

We can assume by induction that our estimator is *unbiased*, meaning that the expected value of its estimate $\mathbb{E}[\mathbf{X}]$ is equal to the ‘true’ parameter value $\bar{\mathbf{X}}$. Consequently we assume $\bar{\mathbf{x}}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_{t+1})$ and $\bar{\mathbf{z}}_{t+1} = \mathbf{h}(\mathbf{x}_{t+1}, \hat{\mathcal{M}})$ are the ‘true’ values, and so we can consider $\mathbf{x}_{t+1}$ and $\mathbf{z}_{t+1}$ as also being Gaussian distributed random variables with means $\bar{\mathbf{x}}_{t+1}$ and $\bar{\mathbf{z}}_{t+1}$, respectively. Therefore we can reformulate Equations (2.7) and (2.8), using the definition of a multivariate Gaussian distribution, as:

$$p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_{t+1}) = \eta_f \exp \left(-\frac{1}{2} (\mathbf{x}_{t+1} - \mathbf{f}(\mathbf{x}_t, \mathbf{u}_{t+1}))^T \mathbf{Q}^{-1} (\mathbf{x}_{t+1} - \mathbf{f}(\mathbf{x}_t, \mathbf{u}_{t+1})) \right) \quad (2.11)$$

$$p(\mathbf{z}_{t+1} | \mathbf{x}_{t+1}, \hat{\mathcal{M}}) = \eta_h \exp \left(-\frac{1}{2} (\mathbf{z}_{t+1} - \mathbf{h}(\mathbf{x}_{t+1}, \hat{\mathcal{M}}))^T \mathbf{R}^{-1} (\mathbf{z}_{t+1} - \mathbf{h}(\mathbf{x}_{t+1}, \hat{\mathcal{M}})) \right) \quad (2.12)$$

where η_f and η_h are constant scalar normalisation factors that do not depend on the input variables. Both Equations (2.11) and (2.12), and therefore also the posterior

$\mathcal{P}$, are maximised when their exponents are zero. Equivalently, we can take logs of both sides and minimise:

$$\hat{\mathbf{x}}_{t+1} = \arg \min_{\mathbf{x}_{t+1}} (-\log \mathcal{P}) \quad (2.13)$$

By simplifying (and again ignoring constant factors), we can see that this takes the form of a *weighted least-squares* system:

$$\hat{\mathbf{x}}_{t+1} = \arg \min_{\mathbf{x}_{t+1}} \left(\frac{1}{2} (\mathbf{e}_f^T \mathbf{Q}^{-1} \mathbf{e}_f) + \frac{1}{2} (\mathbf{e}_h^T \mathbf{R}^{-1} \mathbf{e}_h) \right) \quad (2.14)$$

where $\mathbf{e}_f = \mathbf{x}_{t+1} - \mathbf{f}(\mathbf{x}_t, \mathbf{u}_{t+1})$ and $\mathbf{e}_h = \mathbf{z}_{t+1} - \mathbf{h}(\mathbf{x}_{t+1}, \hat{\mathcal{M}})$ are the *residuals* – the difference between what was observed and what was predicted by our motion and sensor models. By gathering terms we obtain the final cost function that we wish to minimise:

$$C(\mathbf{x}) = \frac{1}{2} \mathbf{e}^T \boldsymbol{\Sigma}^{-1} \mathbf{e} \quad (2.15)$$

where $\mathbf{e} = [\mathbf{e}_f^T, \mathbf{e}_h^T]^T$ and $\boldsymbol{\Sigma} = \text{diag}(\mathbf{Q}, \mathbf{R})$.

If the motion and sensor models $\mathbf{f}(\cdot)$ and $\mathbf{h}(\cdot)$ are both linear functions then Equation (2.15) could be solved in closed form. However, in general they are non-linear and so must be solved iteratively. The process by which non-linear functions of this type are minimised is explained in Section 2.7 and we derive the definition of $\mathbf{h}(\cdot)$ for cameras in Section 2.8.1.

2.6 State Parameterisation

So far we have not specified what form our pose state variable $\mathbf{x}$ takes. In this section we compare some representations, paying particular attention to how we parameterise rotations.

2.6.1 Rotations

Much debate has been had about the most appropriate way to parameterise rotations in 3D space for various applications. This is due to the fact that rotations are non-Euclidean in nature: although they have three degrees of freedom (in 3D), any attempt to parameterise the entire set of rotations by mapping into $\mathbb{R}^3$ will result in singularities³ in the parameter space [84]. Alternative formulations that employ more than three parameters can avoid singularities, however, we must be careful to preserve the constraints between these necessarily non-independent parameters, thus increasing complexity. Here we will briefly consider the merits and downsides of some commonly used parameterisations.

Rotation Matrices

A 3×3 rotation matrix $\mathbf{R}$ has a total of nine parameters and so is an overparameterised representation. A total of six constraints ensure its three columns are mutually orthogonal and of unit length. The set of all such matrices forms the *special orthogonal group* in dimension three, or $\text{SO}(3)$. More generally, as noted previously in Section 2.2, a 4×4 rigid-body transformation matrix contains an embedded rotation matrix and also encodes a translation. The set of these matrices forms the *special Euclidean group* in dimension three, $\text{SE}(3)$.

Euler Angles

Euler angles minimally parameterise 3D rotations by three angles: one for each of the principal axes. In our convention, rotations about the x , y , and z axes are called *roll*, *pitch*, and *yaw*, respectively. This is convenient if we wish to, for example, put constraints on roll and pitch in order to bias solutions towards planar motion. Euler angles, however, suffer from the well-known *gimbal lock* problem, caused by

³A singularity is a point or subspace at which the function becomes degenerate or undefined.

alignment of two of the three axes of rotation and thereby resulting in the loss of one degree of freedom.

Quaternions

Quaternions were invented as a generalisation of complex numbers [34] and have since been used to parameterise rotations in 3D space. A unit quaternion representing a rotation consists of four parameters, $\mathbf{q} = [q_x, q_y, q_z, q_w]$, with the constraint $\|\mathbf{q}\| = 1$. They are commonly used in computer graphics and animation applications due to their intuitive and aesthetically pleasing interpolation properties. However, as with any non-minimal formulation, we must be careful to ensure that our optimiser does not move the quaternion (too far) off the non-Euclidean manifold that maps to valid rotations (in this case, the unit sphere in $\mathbb{R}^4$).

Exponential Map

The exponential map [29] is another minimal parameterisation in which vectors in $\mathbb{R}^3$ describe the axis and magnitude of corresponding 3D rotations. Although not immune to singularities, it is shown in [29] that they are readily avoidable with this parameterisation. Unlike Euler angles, however, it is less intuitive to relate exponential map parameters to the principal axes, making planar movement constraints more complicated to enforce.

2.6.2 Chosen Representation

As mentioned in Section 2.2, we already use transformation matrices extensively as the representation with which we compose two (or more) transformations together. Therefore we implicitly use rotation matrices as part of this representation. This is convenient as they do not suffer from singularities, and transformation matrices are generally easy to compose and are intuitive to reason about.

However, we prefer a minimal state parameterisation on which to optimise over as this allows us to do simpler, unconstrained optimisation, described later in Section 2.7. For this representation we choose Euler angles due to the simplicity of biasing them towards planar motion, in comparison to the exponential map parameterisation, despite the latter’s more favourable distribution of singularities. For the most part, we are able to avoid gimbal lock caused by axis alignment with the former by only considering “small” Euler angles at any particular instance in time – we explain how this is done in Section 2.10.

Our minimal state parameterisation is therefore a 6×1 vector:

$$\mathbf{x} = [x, y, z, \theta_r, \theta_p, \theta_y]^T \quad (2.16)$$

To construct a transformation matrix $\mathbf{T}(\mathbf{x})$ from this parameterisation, we use the ‘ZYX’ Euler convention. This specifies the order in which we apply rotations around the principle axes (see Figure 2.1): we first apply roll, then pitch, then yaw, giving us the following formulation:

$$\mathbf{T}(\mathbf{x}) = \begin{bmatrix} \mathbf{R}_z(\theta_y)\mathbf{R}_y(\theta_p)\mathbf{R}_x(\theta_r) & [x, y, z]^T \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (2.17)$$

2.6.3 3-DoF Versus 6-DoF Parameterisation

Roads are typically approximately planar, at least within the local vicinity of a vehicle at any instant in time. This provides some strong prior information about a vehicle’s movement – it is more likely to move straight forward along the plane than to suddenly translate up away from the plane.

There are two ways we can make use of this information: a hard constraint, meaning we parameterise our world and movement in 2D only, or a soft constraint, meaning we allow limited movement out of the plane, but heavily bias towards

planar movement.

In Section 4.9.3 we compare these approaches in simulation (albeit incorporating real-world vehicular odometry data) and find that the latter soft constraint approach yields the best performance. Therefore for the rest of this chapter we consider only the soft constrained 6-DoF parameterisation.

2.7 Non-linear Optimisation

Now that we know what form our pose state variable $\mathbf{x}$ takes, we can show how to solve non-linear least square problems like Equation (2.15). This is the goal of optimisation: to minimise (or maximise) an objective function subject to some constraints. Here we will only consider the unconstrained case where the objective function is non-linear: this is the form of Equation (2.15) that we wish to minimise from Section 2.5.2.

2.7.1 Problem Formulation

Consider a non-linear function $f(\mathbf{x}) : \mathbb{R}^N \rightarrow \mathbb{R}$ that we wish to minimise with respect to the input variables $\mathbf{x} \in \mathbb{R}^N$. Given a current best guess for the minimum, $f(\mathbf{x}_n)$, we seek a $\delta\mathbf{x}$ such that $f(\mathbf{x}_n + \delta\mathbf{x}) < f(\mathbf{x}_n)$. This leads to the iterative update:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \delta\mathbf{x} \tag{2.18}$$

and so the problem can be reduced to a series of 1D line searches: $\delta\mathbf{x} = \alpha\mathbf{m}$ where α is a *step length* and $\mathbf{m}$ is a *step direction*. Solving this search problem can be done in a number of ways.

2.7.1.1 Gradient Descent

One of the simplest solutions is the *gradient descent* method. At each iteration we choose the steepest direction based on the gradient of the function at the current operating point: $\mathbf{m} = -\nabla f(\mathbf{x}_n)$. The step length α is then chosen (often according to some heuristic) such that $f(\mathbf{x}_n + \alpha\mathbf{m})$ is sufficiently less than $f(\mathbf{x}_n)$. Although effective, this method has a slow rate of convergence as it tends to ‘zigzag’ inefficiently down shallow valleys.

2.7.1.2 Newton’s Method

The shortcomings of gradient descent can be partially overcome by taking second order derivative information into account. Intuitively, Newton’s Method approximates f locally as a quadratic at each iteration. It is then able to find the minimum of that quadratic in a single computational step. We approximate $f(\mathbf{x}_n + \delta\mathbf{x})$ with a second order Taylor series expansion:

$$f(\mathbf{x}_n + \delta\mathbf{x}) \approx f(\mathbf{x}_n) + \delta\mathbf{x}\nabla f(\mathbf{x}_n) + \frac{1}{2}\delta\mathbf{x}^T\nabla^2 f(\mathbf{x}_n)\delta\mathbf{x} \quad (2.19)$$

We then minimise Equation (2.19) with respect to $\delta\mathbf{x}$ and set to $\mathbf{0}$:

$$\frac{d}{d\delta\mathbf{x}}f(\mathbf{x}_n + \delta\mathbf{x}) = \nabla f(\mathbf{x}_n) + \delta\mathbf{x}^T\nabla^2 f(\mathbf{x}_n) = \mathbf{0} \quad (2.20)$$

and therefore obtain a closed-form expression for $\delta\mathbf{x}$:

$$\delta\mathbf{x} = -(\nabla^2 f(\mathbf{x}_n))^{-1} \nabla f(\mathbf{x}_n) \quad (2.21)$$

2.7.1.3 Gauss-Newton

Newton's method has the disadvantage that it requires an exact value for the Hessian $\nabla^2 f(\mathbf{x}_n)$ which can be expensive to compute. However, for a least-squares problem we note that f has the following form:

$$f(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^m \mathbf{e}_j(\mathbf{x})^T \mathbf{e}_j(\mathbf{x}) = \frac{1}{2} \|\mathbf{e}(\mathbf{x})\|^2 \quad (2.22)$$

where each function $\mathbf{e}_j : \mathbb{R}^N \rightarrow \mathbb{R}^M$ is smooth and referred to as a *residual*. The first and second order derivatives of f with respect to $\mathbf{x}$ are therefore:

$$\nabla f(\mathbf{x}) = \sum_{j=1}^m \mathbf{e}_j(\mathbf{x}) \nabla \mathbf{r}_j(\mathbf{x}) = \mathbf{J}(\mathbf{x})^T \mathbf{e}(\mathbf{x}) \quad (2.23)$$

$$\nabla^2 f(\mathbf{x}) = \mathbf{J}(\mathbf{x})^T \mathbf{J}(\mathbf{x}) + \sum_{j=1}^m \mathbf{e}_j(\mathbf{x}) \nabla^2 \mathbf{e}_j(\mathbf{x}) \quad (2.24)$$

where $\mathbf{J}(\mathbf{x})$ is the Jacobian of the residuals, $\frac{d\mathbf{e}}{d\mathbf{x}}$.

The Gauss-Newton method approximates the Hessian using only the first term of Equation (2.24), i.e. $\nabla^2 f(\mathbf{x}) \approx \mathbf{J}(\mathbf{x})^T \mathbf{J}(\mathbf{x})$. This is a reasonable approximation for 'well behaved' problems where the residuals are small near the solution, and therefore the first term dominates Equation (2.24). The advantage over Newton's method is that we no longer need to compute a costly Hessian. Therefore, by substituting Equation (2.23) and the above approximation into Equation (2.21), we obtain the following closed-form expression for $\delta \mathbf{x}$:

$$\delta \mathbf{x} = - \left(\mathbf{J}(\mathbf{x})^T \mathbf{J}(\mathbf{x}) \right)^{-1} \mathbf{J}(\mathbf{x})^T \mathbf{e}(\mathbf{x}) \quad (2.25)$$

2.7.1.4 Levenberg-Marquardt

One of the weaknesses of Gauss-Newton is its tendency to diverge when $\mathbf{J}(\mathbf{x})$ is (almost) rank-deficient. To accommodate the fact that under these conditions, $\mathbf{J}(\mathbf{x})^T \mathbf{J}(\mathbf{x})$ does not have a (well behaved) inverse, the Levenberg-Marquardt algorithm [46] proposes adding a damping factor $\lambda \mathbf{1}$ (where $\mathbf{1}$ is the identity matrix and $\lambda > 0$) to the approximate Hessian in Equation (2.25), obtaining:

$$\delta \mathbf{x} = -(\mathbf{J}(\mathbf{x})^T \mathbf{J}(\mathbf{x}) + \lambda \mathbf{1})^{-1} \mathbf{J}(\mathbf{x})^T \mathbf{e}(\mathbf{x}) \quad (2.26)$$

It can be seen that we are able to interpolate between gradient descent and Gauss-Newton methods by respectively increasing or decreasing the value of λ during the optimisation process. The choice of which to prefer is based on the properties of the function at each step. This makes Levenberg-Marquardt very robust, often allowing it to converge even when the initial estimate is far from the local minimum.

The algorithm, adapted from that in [72], is outlined below:

1. Set an initial value for the damping parameter, $\lambda = 0.001$.
2. Solve Equation (2.26) to obtain $\delta \mathbf{x}$.
3. If $|\delta \mathbf{x}| < \text{threshold}$ then stop.
4. Otherwise if $f(\mathbf{x} + \delta \mathbf{x}) > f(\mathbf{x})$, set $\lambda = \lambda \times 10$.
5. Otherwise if $f(\mathbf{x} + \delta \mathbf{x}) < f(\mathbf{x})$, set $\lambda = \lambda \div 10$ and update $\mathbf{x} = \mathbf{x} + \delta \mathbf{x}$.
6. Return to step 2.

Due to its aforementioned robustness, the Levenberg-Marquardt is our optimisation procedure of choice for our work in Chapter 4.

2.7.2 Data Association

So far in considering our sensor model $\mathbf{h}(\cdot)$, we have made the implicit assumption that we know precisely to which map feature a sensor measurement corresponds. In the real world, of course, there may be significant ambiguity: landmarks may produce multiple (or no) measurements, and spurious observations might be induced by noise or dynamic objects. To reliably determine these correspondences is to solve the *data association* problem.

2.7.2.1 Joint Compatibility Branch and Bound

Joint Compatibility Branch and Bound (JCBB) is a batch data association algorithm that finds the maximal *jointly compatible* set of measurement-landmark correspondences. Informally, compatibility between a measurement $\mathbf{z}_i$ and a landmark $\mathbf{p}_j$ means, given the estimated state of the system $\hat{\mathbf{x}}$, it is plausible that the landmark gave rise to that measurement. Intuitively, the measurement $\mathbf{z}_i$ must lie in the vicinity of the predicted observation for that landmark, in sensor space $\mathbf{h}(\hat{\mathbf{x}}, \mathbf{p}_j)$ (e.g. for a camera, they must only be a few pixels apart).

The boundary of the acceptable vicinity around $\mathbf{z}_i$ is defined by the *validation gate* and is commonly measured using the squared *Mahalanobis distance* [55] (also referred to as *normalised innovation squared* (NIS)). For a residual $\mathbf{e}_{ij} = \mathbf{z}_i - \mathbf{h}(\hat{\mathbf{x}}, \mathbf{p}_j)$, with covariance $\mathbf{S}_{ij}$, the Mahalanobis distance M_{ij} is defined as:

$$M_{ij}^2 = \mathbf{e}_{ij}^T \mathbf{S}_{ij}^{-1} \mathbf{e}_{ij} \quad (2.27)$$

Intuitively, Mahalanobis distance can be thought of as Euclidean distance in a space that has been scaled according to the inverse covariance $\mathbf{S}_{ij}^{-1}$ – that is, distances are shorter along more uncertain axes. In the case where $\mathbf{S}_{ij} = \mathbf{I}$, it is exactly equivalent to Euclidean distance. The validation gate test, therefore, is a test against a max-

imum distance threshold γ : if $M_{ij}^2 < \gamma$ holds, then measurement $\mathbf{z}_i$ is *individually compatible* with landmark $\mathbf{p}_j$.

This kind of nearest neighbour individual compatibility test can be used on its own for data association: simply associate measurement $\mathbf{z}_i$ with the landmark $\mathbf{p}_j$ with which it has the smallest squared Mahalanobis distance, M_{ij}^2 . This is the approach used in for example [9, 23]. However, individual compatibility does not preserve the correlations between the set of measurements, predicted observations, and sensor model. For example, the discrepancy between a measurement and a prediction could be explained by panning the camera to the right. However, we should not be able to simultaneously explain (in the same hypothesis) another discrepancy that might be consistent with panning the camera to the left – this would violate the rigid-body constraints inherent in the camera’s sensor model. An example set of camera measurements, predicted observations, and their corresponding validation gates are shown in Figure 2.5.

The *joint compatibility* test preserves these correlations. Consider a set of N measurements $\mathbf{z} = [\mathbf{z}_1^T, \dots, \mathbf{z}_N^T]^T$ with covariance $\mathbf{R}$, and a set of M landmarks $\mathbf{p} = [\mathbf{p}_1, \dots, \mathbf{p}_M]$. A data association hypothesis consists of a set of N associations (one for each measurement) $\alpha = [\alpha_1, \dots, \alpha_N]$, such that measurement $\mathbf{z}_i$ corresponds to landmark $\mathbf{p}_{\alpha_i}$, and $\mathbf{p}_\alpha = [\mathbf{p}_{\alpha_1}, \dots, \mathbf{p}_{\alpha_N}]$ is a set of such corresponding landmarks. We represent the entire state of the system as $\hat{\mathbf{x}}'_\alpha = [\hat{\mathbf{x}}^T, \mathbf{p}_{\alpha_1}^T, \dots, \mathbf{p}_{\alpha_N}^T]^T$, with covariance $\mathbf{P}$, which includes both the current pose of the camera $\hat{\mathbf{x}} \in \mathbb{R}^6$, and the N hypothesised landmarks. The joint predicted observation $\hat{\mathbf{z}}_\alpha$ for hypothesis α is therefore:

$$\hat{\mathbf{z}}_\alpha(\hat{\mathbf{x}}'_\alpha) = \begin{bmatrix} \mathbf{h}(\hat{\mathbf{x}}, \mathbf{p}_{\alpha_1}) \\ \vdots \\ \mathbf{h}(\hat{\mathbf{x}}, \mathbf{p}_{\alpha_N}) \end{bmatrix} \quad (2.28)$$

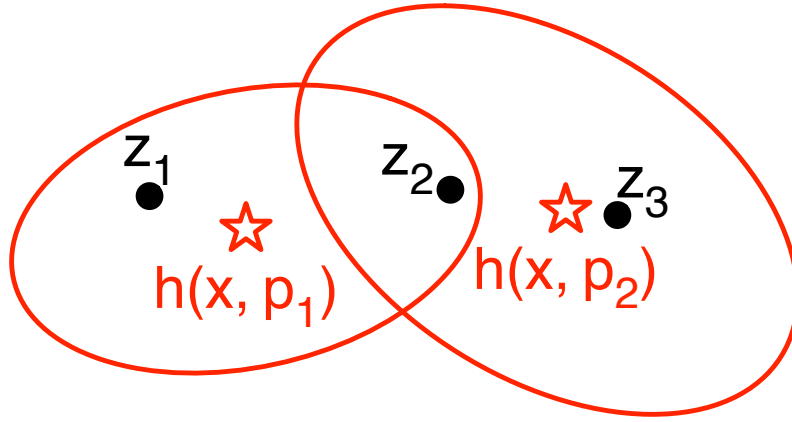


Figure 2.5: Diagram showing three measurements z_1 , z_2 , and z_3 , and the predicted locations of two landmarks p_1 and p_2 , projected into the image via camera sensor model function h . Shown around the landmark predictions are their Mahalanobis distance boundary ellipses. Measurements falling into these ellipses are individually compatible with the corresponding landmark. In this example, a nearest-neighbour algorithm would match measurements z_1 and z_3 to landmarks p_1 and p_2 respectively. However, if joint compatibility is taken into account, a more convincing hypothesis would instead match measurement z_2 to landmark p_2 – e.g. a slight rotation of the camera would align both pairs almost exactly.

The joint residuals $\mathbf{e}_\alpha$ are then given by:

$$\mathbf{e}_\alpha = \mathbf{z} - \hat{\mathbf{z}}_\alpha(\hat{\mathbf{x}}'_\alpha) \quad (2.29)$$

and the joint residual covariance $\mathbf{S}_\alpha$ is then computed by linearising the state covariance $\mathbf{P}$ around the current state $\hat{\mathbf{x}}'_\alpha$:

$$\mathbf{S}_\alpha = \mathbf{J}(\hat{\mathbf{x}}'_\alpha)\mathbf{P}^{-1}\mathbf{J}(\hat{\mathbf{x}}'_\alpha)^T + \mathbf{R} \quad (2.30)$$

where $\mathbf{J}(\hat{\mathbf{x}}'_\alpha) = \frac{d\hat{\mathbf{z}}_\alpha}{d\hat{\mathbf{x}}'_\alpha}(\hat{\mathbf{x}}'_\alpha)$. The joint validation gate test then takes the form:

$$M_\alpha^2 = \mathbf{e}_\alpha^T \mathbf{S}_\alpha^{-1} \mathbf{e}_\alpha < \gamma \quad (2.31)$$

In order to find the maximal set of jointly compatible associations, JCBB searches an *interpretation tree* [30] that spans the space of possible hypotheses. Figure 2.6 shows the interpretation tree for the hypotheses arising from data in Figure 2.5. Level i of the tree (with the root being level 0 and the leaves level N) corresponds to measurement $\mathbf{z}_i$ and each node represents an association between that measurement and a single landmark, or the null hypothesis (i.e. no corresponding landmark). Every full descending path therefore corresponds to a full hypothesis α . To preserve a one-to-one correspondence, a landmark will only appear once along any descending path (and only alongside measurements with which it is individually compatible).

The algorithm performs depth-first search on the interpretation tree⁴, testing for joint compatibility between an increasingly larger set of associations as it descends. A sub-tree is culled from the search if the joint validation test at its root node fails, or if it has insufficient non-null nodes along any path to produce a set of associations with size equal to or greater than the current largest set. When the

⁴Various heuristics can be used to prioritise which branches are searched first – [65] uses minimum joint NIS, whereas [1] suggests using maximum joint likelihood.

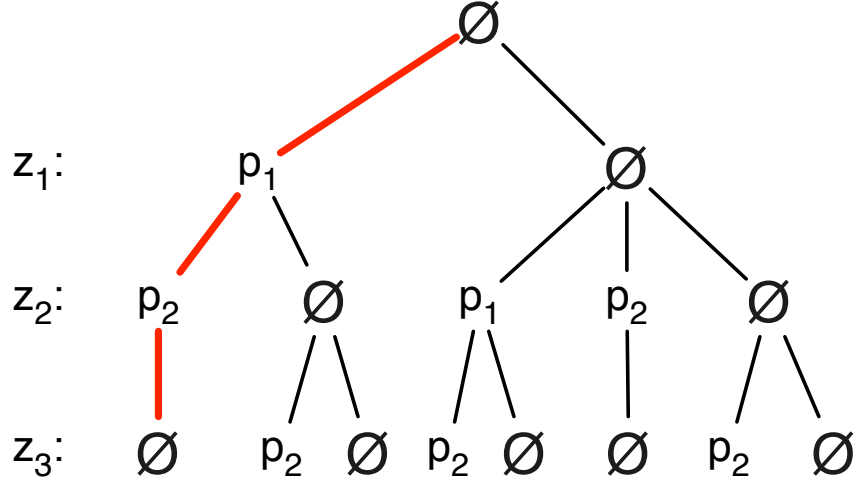


Figure 2.6: The full interpretation tree for the configuration shown in Figure 2.5. The path corresponding to the maximal jointly compatible hypothesis is shown in red, associating measurement $\mathbf{z}_1$ with landmark $\mathbf{p}_1$ at level 1 of the tree, and $\mathbf{z}_2$ with $\mathbf{p}_2$ at level 2. ‘Null’ associations are denoted with $\emptyset$.

search is complete, the current largest set of associations is returned as the best hypothesis (in case of a tie, the set with the lowest NIS score is returned).

2.7.3 Robust Estimation

As our data association algorithms are never perfect, it is desirable that our estimation machinery be robust to the presence of outliers. Unfortunately, the squared error cost function is unduly affected by outliers as larger errors have an increasingly dominant influence over the optimisation solution. One way to overcome this is to re-weight the errors using a *robust estimator* function $\rho(\cdot)$ (alternatively known as an *M-estimator*). Recall our cost function given by Equation (2.22) which can be rewritten as:

$$f(\mathbf{x}) = \frac{1}{2} \sum_{j=1}^m \mathbf{e}_j(\mathbf{x})^T \mathbf{e}_j(\mathbf{x}) = \sum_{j=1}^m c_j^2 \quad (2.32)$$

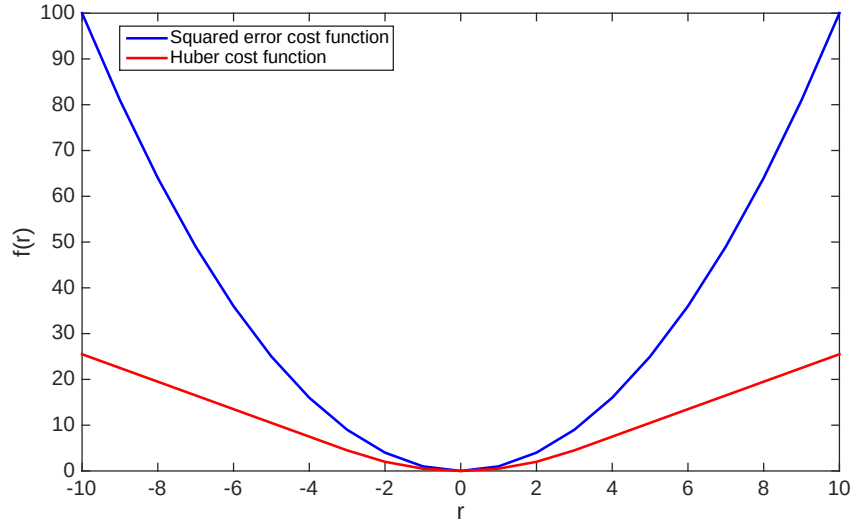


Figure 2.7: Huber cost function (shown in red) vs squared error cost function (shown in blue), for $k = 3$.

We transform this into *re-weighted least squares* form by incorporating ρ :

$$\mathbf{g}(\mathbf{x}) = \sum_{j=1}^m \rho(c_j) \quad (2.33)$$

The Huber cost function [42] has been recommended for almost all situations [91]. It is quadratic for small values of the error and becomes linear for errors beyond a threshold k , therefore preserving the maximum-likelihood properties of squared error cost for inliers while reducing the influence of outliers. It has the following definition:

$$\rho(r) = \begin{cases} \frac{r^2}{2} & \text{if } |r| \leq k \\ k \left(|r| - \frac{k}{2} \right) & \text{otherwise} \end{cases} \quad (2.34)$$

The Huber and squared error cost functions are compared in Figure 2.7.

Taking this approach results in the following changes to the Jacobians of our

system. Differentiating Equation (2.33) we get:

$$\frac{d\mathbf{g}}{d\mathbf{x}} = \sum_{j=1}^m \frac{d\rho(c_j)}{d\mathbf{x}} \quad (2.35)$$

$$= \sum_{j=1}^m \left(\frac{d\rho(c_j)}{dc_j} \right) \left(\frac{dc_j}{d\mathbf{x}} \right) \quad (2.36)$$

$$= \sum_{j=1}^m \left(\frac{d\rho(c_j)}{dc_j} \right) \left(\frac{1}{2c_j} \right) \left(\frac{dc_j^2}{d\mathbf{x}} \right) \quad (2.37)$$

With the derivative of Equation (2.34) given as:

$$\frac{d\rho}{dr}(r) = \begin{cases} r & \text{if } |r| \leq k \\ k & \text{otherwise} \end{cases} \quad (2.38)$$

we obtain:

$$\frac{d\mathbf{g}}{d\mathbf{x}} = \begin{cases} \frac{1}{2} \sum_{j=1}^m \left(\frac{dc_j^2}{d\mathbf{x}} \right) & \text{if } |r| \leq k \\ \frac{1}{2} \sum_{j=1}^m \frac{k}{c_j} \left(\frac{dc_j^2}{d\mathbf{x}} \right) & \text{otherwise} \end{cases} \quad (2.39)$$

This can be conveniently expressed as:

$$\frac{d\mathbf{g}}{d\mathbf{x}} = \frac{1}{2} \sum_{j=1}^m w_j \left(\frac{dc_j^2}{d\mathbf{x}} \right) \quad (2.40)$$

where:

$$w_j = \begin{cases} 1 & \text{if } |r| \leq k \\ \frac{k}{c_j} & \text{otherwise} \end{cases} \quad (2.41)$$

As intended this results in a reweighted closed-form expression for $\delta\mathbf{x}$ (as previously given by Equation (2.25)):

$$\delta\mathbf{x} = - \left(\mathbf{J}(\mathbf{x})^T \mathbf{\Sigma} \mathbf{J}(\mathbf{x}) \right)^{-1} \mathbf{J}(\mathbf{x})^T \mathbf{\Sigma} \mathbf{e}(\mathbf{x}) \quad (2.42)$$

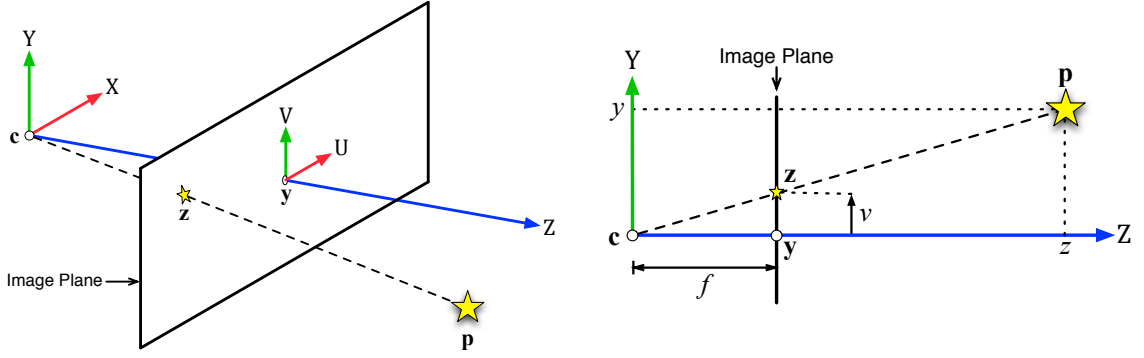


Figure 2.8: The pinhole camera model shown from two different perspectives. A ray of light from a landmark $\mathbf{p}$ projects onto a pixel $\mathbf{z}$ on the image plane through the camera’s focal point $\mathbf{c}$.

where $\Sigma = \text{diag}(w_1, w_2, \dots, w_m)$. It can be seen that this has the effect of down-weighting large residuals, therefore reducing their impact on the solution.

In the remainder of this chapter we consider how to apply this machinery to solve problems resulting from the use of visual sensors specifically.

2.8 Visual Sensors

Part of the motivation behind our work in Chapter 4 is to achieve autonomy at night using low-cost vision-based sensors. We therefore must tailor the machinery we have presented so far in this chapter to these types of sensor.

A camera is a device that focuses ambient rays of light (emitted or reflected by objects in a scene) onto a photosensitive sensor in order to capture a still image. Mathematically, a camera maps points in the 3D world onto a 2D image. In this section we detail the ubiquitous *pinhole camera model* which gives rise to such a mapping that is applicable to some hypothetical ‘ideal’ camera.

2.8.1 Pinhole Camera Model

We are interested in defining a function $\mathbf{h}(\cdot) : \mathbb{R}^6 \times \mathbb{R}^3 \mapsto \mathbb{R}^2$ that maps 6-DoF camera poses and 3D points in the world to 2D points in the image plane. This

will form the basis of our sensor model that will be used to infer the position of landmarks during mapping, as well as the pose of the camera during localisation.

Consider a coordinate frame centred at the camera's *focal point* $\mathbf{c} \in \mathbb{R}^3$ as shown in Figure 2.8. Perpendicular to, and at a distance f metres along the z -axis (*principal axis*) lies the (virtual) *image plane*. Where the principle axis intersects this plane lies the *principle point* $\mathbf{y} = (y_u, y_v)$. A 3D point $\mathbf{p} = [x, y, z]^T$ projects onto a point $\mathbf{z} = [u, v]^T$ on the image plane along a ray which passes through $\mathbf{c}$. Using similar triangles it is easy to verify that:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{fx}{z} \\ \frac{fy}{z} \end{bmatrix} \quad (2.43)$$

If we are content to represent points using homogeneous coordinates, Equation 2.43 can be expressed as a simple linear transformation:

$$\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = \begin{bmatrix} f & 0 \\ & f & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{1}^{3 \times 3} & \mathbf{0}^{3 \times 1} \end{bmatrix} \mathbf{p} = \mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{p} \quad (2.44)$$

Here, $\mathbf{K}$ is the *camera calibration matrix*. For a fully general *finite projective* camera, $\mathbf{K}$ has five degrees of freedom:

$$\mathbf{K} = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.45)$$

where $\alpha_u = fm_u$ and $\alpha_v = fm_v$ are the focal length expressed in terms of pixel dimensions in u and v respectively, m_u and m_v being the number of pixels per metre

along the u and v dimensions of the image plane respectively. This is to account for CCDs with non-square pixel sizes. $u_0 = y_u m_u$ and $v_0 = y_v m_v$ are offsets expressed in pixel dimensions that align the origin of the image plane with the principal point $\mathbf{y}$. s is a skew parameter and is only non-zero for cameras which have non-perpendicular image plane axes.

In order to obtain pixel values from a homogeneous representation, we must pass the result of Equation (2.44) through a non-linear function $\mathbf{g}(\cdot)$:

$$\mathbf{g} \left(\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} \right) = \begin{bmatrix} \frac{u'}{w} \\ \frac{v'}{w} \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix} \quad (2.46)$$

Therefore $\mathbf{h}(\cdot) : \mathbb{R}^6 \times \mathbb{R}^3 \mapsto \mathbb{R}^2$ takes the following full form:

$$\mathbf{h}(\mathbf{x}, \mathbf{p}) = \mathbf{g} \left(\mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}_{b,w}(\mathbf{x}) [\mathbf{p}^T, 1]^T \right) \quad (2.47)$$

where $\mathbf{T}_{c,b}$ is an arbitrary transformation from the camera's base coordinate frame to its image coordinate frame (if these are different⁵), and $\mathbf{T}_{b,w}(\mathbf{x})$ is a transformation matrix constructed from the x , y , z , roll, pitch, and yaw components of $\mathbf{x}$ using Equation (2.17), representing the transformation from the world coordinate frame into the camera's base coordinate frame (i.e. the camera's pose).

2.8.2 Lens Distortion

Real-world camera lenses typically do not behave perfectly like a pinhole camera: they exhibit some radial and tangential distortion. The effects are seen most obvi-

⁵A camera unit has a single *base coordinate frame*. Each separate image sensor in the camera unit has its own *image coordinate frame*. Therefore this distinction is useful for stereo (and other multi-lens) cameras, particularly as the extrinsic calibration values $\mathbf{T}_{c,b}$ are often provided by the camera manufacturer for each unit.

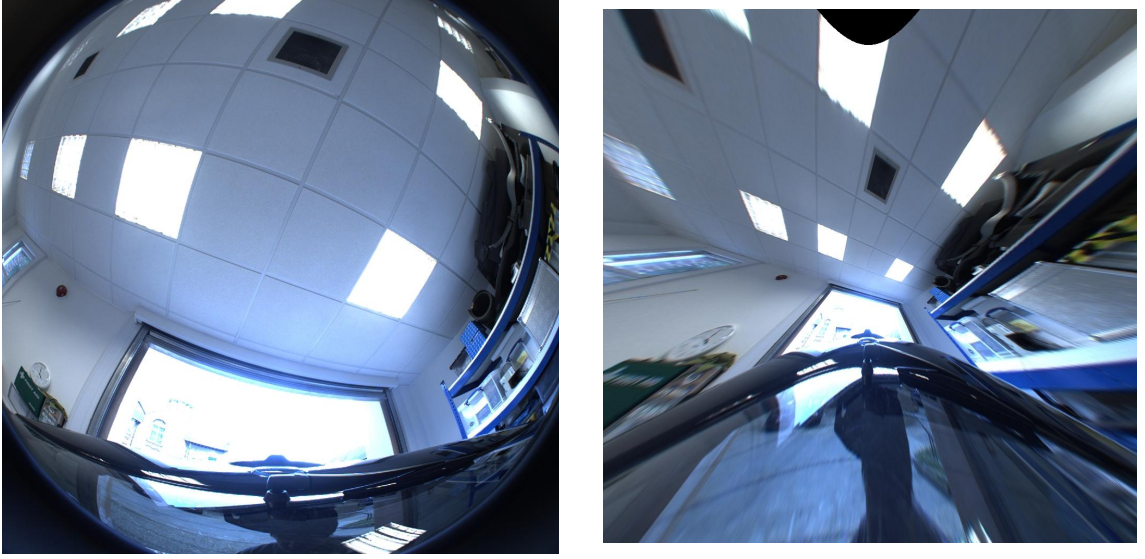


Figure 2.9: The image on the left, captured using a Point Grey Grasshopper2 fisheye camera, exhibits some extreme radial distortion. On the right is the undistorted image: notice that straight lines in the world now correctly appear straight. The image on the right now exhibits pinhole camera model behaviour.

ously in wide angle fisheye lenses, for example as shown in Figure 2.9. The parameters defining the distortion can be estimated using a calibration procedure such as that in [76]. The aim of this procedure is to generate a function that transforms distorted camera images into undistorted images for which the pinhole camera model applies. This function takes the form of a lookup table that maps pixel coordinates in the output image to subpixel coordinates in the input image. In the remainder of this thesis we assume our images are undistorted in this manner.

Now that we understand the theory underpinning visual sensors, we apply it to concrete mapping and localisation problems in the following sections.

2.9 Visual Structure from Motion

Recall from Section 2.5 that in our teach-and-repeat mapping phase, we are assumed to know the poses of the vehicle from some source (visual odometry is usually sufficiently locally accurate for this task), and our goal is to infer the 3D position of

a landmark from a number of 2D pixel measurements. Consider then a set of N 6-DoF camera poses $\mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^T$ and a set of N observations of a single landmark $\mathbf{z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]^T$, where each $\mathbf{z}_i = (u_i, v_i)$ is a pixel measurement, and such that measurement $\mathbf{z}_i$ corresponds to the camera's pose $\mathbf{x}_i$. Our task is to infer the location of the landmark $\mathbf{p} \in \mathbb{R}^3$.

2.9.1 Structure from Motion Derivatives

To apply Equation (2.25) to this optimisation problem to infer the state $\mathbf{p}$ we need to define the residuals $\mathbf{e}(\mathbf{p})$, and the Jacobian of the residuals $\mathbf{J}(\mathbf{p})$.

Let the residual between a single measurement $\mathbf{z}_i$ and the predicted observation of its corresponding landmark at pose $\mathbf{x}_i$ be $\mathbf{h}(\mathbf{x}_i, \mathbf{p}) - \mathbf{z}_i$. Therefore the combined residual vector for the whole system $\mathbf{e}(\mathbf{p})$ is:

$$\mathbf{e}(\mathbf{p}) = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1, \mathbf{p}) - \mathbf{z}_1 \\ \mathbf{h}(\mathbf{x}_2, \mathbf{p}) - \mathbf{z}_2 \\ \vdots \\ \mathbf{h}(\mathbf{x}_N, \mathbf{p}) - \mathbf{z}_N \end{bmatrix} \quad (2.48)$$

and so the combined Jacobian $\mathbf{J}(\mathbf{p})$ is:

$$\mathbf{J}(\mathbf{p}) = \frac{d\mathbf{e}}{d\mathbf{p}} = \begin{bmatrix} \frac{\partial \mathbf{h}}{\partial \mathbf{p}}(\mathbf{x}_1, \mathbf{p}) \\ \frac{\partial \mathbf{h}}{\partial \mathbf{p}}(\mathbf{x}_2, \mathbf{p}) \\ \vdots \\ \frac{\partial \mathbf{h}}{\partial \mathbf{p}}(\mathbf{x}_N, \mathbf{p}) \end{bmatrix} \quad (2.49)$$

We therefore require an expression for the Jacobian of a single residual:

$$\frac{d}{d\mathbf{p}} (\mathbf{h}(\mathbf{x}_i, \mathbf{p})) = \frac{d}{d\mathbf{p}} \left(\mathbf{g} \left(\mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}_{b,w}(\mathbf{x}_i) [\mathbf{p}^T, 1]^T \right) \right) \quad (2.50)$$

Using the chain rule we obtain two factors that we will deal with separately:

$$= \mathbf{g}' \left(\mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}_{b,w} (\mathbf{x}_i) [\mathbf{p}^T, 1]^T \right) \quad (2.51)$$

$$\times \frac{d}{d\mathbf{p}} \left(\mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}_{b,w} (\mathbf{x}_i) [\mathbf{p}^T, 1]^T \right) \quad (2.52)$$

Homogeneous Derivative

We consider the second factor, Equation (2.52). By repeated application of the product rule and taking the derivative we obtain:

$$\mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}_{b,w} (\mathbf{x}_i) \frac{d}{d\mathbf{p}} [\mathbf{p}^T, 1]^T = \mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}_{b,w} (\mathbf{x}_i) \quad (2.53)$$

Inhomogeneous Derivative

Considering now the first factor, Equation (2.51), recall the definition of $\mathbf{g}(\cdot) : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ from Equation (2.46) which is repeated below:

$$\mathbf{g} \left(\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} \right) = \begin{bmatrix} \frac{u'}{w} \\ \frac{v'}{w} \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix}$$

The Jacobian corresponding to Equation (2.51) takes the form:

$$\begin{bmatrix} \frac{\partial u}{\partial u'} & \frac{\partial u}{\partial v'} & \frac{\partial u}{\partial w} \\ \frac{\partial v}{\partial u'} & \frac{\partial v}{\partial v'} & \frac{\partial v}{\partial w} \end{bmatrix} = \begin{bmatrix} \frac{1}{w} & 0 & -\frac{u'}{w^2} \\ 0 & \frac{1}{w} & -\frac{v'}{w^2} \end{bmatrix} = \frac{1}{w} \begin{bmatrix} 1 & 0 & -\frac{u'}{w} \\ 0 & 1 & -\frac{v'}{w} \end{bmatrix} \quad (2.54)$$

Full Jacobian

The full Jacobian of the system is therefore:

$$\frac{d\mathbf{h}}{d\mathbf{p}}(\mathbf{x}_i) = \frac{1}{w} \begin{bmatrix} 1 & 0 & -\frac{u'}{w} \\ 0 & 1 & -\frac{v'}{w} \end{bmatrix} \mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}_{b,w}(\mathbf{x}_i) \quad (2.55)$$

2.10 Visual Pose Estimation

For the localisation phase of teach and repeat, we will have already built a map of the environment, as in Section 2.9, and our goal is to infer the pose of the camera using observations of landmarks. Consider then an already mapped set of N 3D landmarks $\mathbf{p} = [\mathbf{p}_1, \dots, \mathbf{p}_N]$ and a new set of N corresponding observations $\mathbf{z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]$, where each $\mathbf{z}_i = (u_i, v_i)$ is a pixel measurement. Our task is to infer the frame-to-frame relative pose of the camera $\mathbf{x} \in \mathbb{R}^6$.

2.10.1 Infinitesimal Perturbation Parameterisation

Similarly to Section 2.9, we must define the residual vector and Jacobian of the whole system. For reasons that will become clear soon, we rewrite our state $\mathbf{x} = \mathbf{\Delta} \oplus \bar{\mathbf{x}}$ such that $\mathbf{\Delta} = [\delta x, \delta y, \delta z, \delta \theta_r, \delta \theta_p, \delta \theta_q] \approx \mathbf{0}$ represents a small perturbation and $\oplus$ denotes the transformation composition operator. Our sensor model function $\mathbf{h}(\cdot)$ is then rewritten as:

$$\mathbf{h}(\mathbf{\Delta}, \mathbf{p}) = \mathbf{g} \left(\mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}(\mathbf{\Delta}) \mathbf{T}_{b,w}(\bar{\mathbf{x}}) [\mathbf{p}^T, 1]^T \right) \quad (2.56)$$

This allows us to reformulate our non-linear optimisation process such that $\mathbf{\Delta}$ is the state variable we solve for at each iteration. Additionally, we will see that by assuming $\mathbf{\Delta} \approx \mathbf{0}$ and therefore necessarily evaluating the Jacobian $\frac{d\mathbf{h}}{d\mathbf{\Delta}}$ only at $\mathbf{\Delta} = \mathbf{0}$, the form of the derivative is much simpler to compute. However, recall that

in the Levenberg-Marquardt solve Equation (2.26), we evaluate the Jacobian at the current best estimate $\mathbf{x}_n$ which in general $\neq \mathbf{0}$. We therefore must slightly modify our optimisation algorithm.

The full residual vector and Jacobian are now:

$$\mathbf{e}(\Delta) = \begin{bmatrix} \mathbf{h}(\Delta, \mathbf{p}_1) - \mathbf{z}_1 \\ \mathbf{h}(\Delta, \mathbf{p}_2) - \mathbf{z}_2 \\ \vdots \\ \mathbf{h}(\Delta, \mathbf{p}_N) - \mathbf{z}_N \end{bmatrix} \quad (2.57)$$

$$\mathbf{J}(\Delta) = \frac{d\mathbf{h}}{d\Delta} = \begin{bmatrix} \frac{\partial \mathbf{h}}{\partial \Delta}(\Delta, \mathbf{p}_1) \\ \frac{\partial \mathbf{h}}{\partial \Delta}(\Delta, \mathbf{p}_2) \\ \vdots \\ \frac{\partial \mathbf{h}}{\partial \Delta}(\Delta, \mathbf{p}_N) \end{bmatrix} \quad (2.58)$$

Remember that in iteration $n+1$ of the optimisation process we solve Equation (2.26) (repeated below), getting a value for (in this case) Δ :

$$\Delta = -(\mathbf{J}(\Delta)^T \mathbf{J}(\Delta) + \lambda \mathbf{1})^{-1} \mathbf{J}(\Delta)^T \mathbf{e}(\Delta) \quad (2.59)$$

Now, instead of updating our state variable Δ directly, as in Equation (2.18), we aggregate Δ into $\bar{\mathbf{x}}$ as we need to ensure $\Delta = \mathbf{0}$ at each iteration. The iterative update then becomes:

$$\bar{\mathbf{x}}_{n+1} = \Delta \oplus \bar{\mathbf{x}}_n \quad (2.60)$$

2.10.2 Pose Estimation Derivatives

We again require an expression for the Jacobian of a single residual:

$$\frac{d}{d\Delta} (\mathbf{h}(\Delta, \mathbf{p}_i)) = \frac{d}{d\Delta} \left(\mathbf{g} \left(\mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}(\Delta) \mathbf{T}_{b,w}(\bar{\mathbf{x}}) [\mathbf{p}_i^T, 1]^T \right) \right) \quad (2.61)$$

Using the chain rule we obtain the two factors:

$$= \mathbf{g}' \left(\mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}(\Delta) \mathbf{T}_{b,w}(\bar{\mathbf{x}}) [\mathbf{p}_i^T, 1]^T \right) \quad (2.62)$$

$$\times \frac{d}{d\Delta} \left(\mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \mathbf{T}(\Delta) \mathbf{T}_{b,w}(\bar{\mathbf{x}}) [\mathbf{p}_i^T, 1]^T \right) \quad (2.63)$$

Notice that Equation (2.62) has the same form as Equation (2.51) and therefore the Jacobian for this factor is the same as given in Equation (2.54). We therefore turn our attention to the derivative of Equation (2.63).

Transform Derivative

Repeated application of the product rule to Equation (2.63) results in:

$$\mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \frac{d}{d\Delta} (\mathbf{T}(\Delta)) \mathbf{T}_{b,w}(\bar{\mathbf{x}}) [\mathbf{p}_i^T, 1]^T \quad (2.64)$$

leaving us to evaluate the derivative $\frac{d}{d\Delta} (\mathbf{T}(\Delta))$.

From Equation (2.17), our transformation matrix $\mathbf{T}(\Delta)$ is therefore defined as the following 4×4 matrix:

$$\mathbf{T}(\Delta) = \begin{bmatrix} \mathbf{R}_x(\delta\theta_r) \mathbf{R}_y(\delta\theta_p) \mathbf{R}_z(\delta\theta_q) & [\delta x, \delta y, \delta z]^T \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (2.65)$$

where $\mathbf{R}_a(\theta)$ is a 3×3 matrix representing a rotation of θ around the specified axis

a. The full Jacobian takes the form of a $4 \times 4 \times 6$ tensor:

$$\frac{d}{d\Delta}(\mathbf{T}(\Delta)) = \begin{bmatrix} \frac{\partial \mathbf{T}}{\partial \delta x} & \frac{\partial \mathbf{T}}{\partial \delta y} & \frac{\partial \mathbf{T}}{\partial \delta z} & \frac{\partial \mathbf{T}}{\partial \delta \theta_r} & \frac{\partial \mathbf{T}}{\partial \delta \theta_p} & \frac{\partial \mathbf{T}}{\partial \delta \theta_q} \end{bmatrix} \quad (2.66)$$

We note that in particular the rotational Jacobians will contain many compound $\sin(\cdot)$ and $\cos(\cdot)$ terms which we will not reproduce here. However, recall that we linearise $\mathbf{T}(\Delta)$ around $\Delta = \mathbf{0}$ and therefore each component simplifies down to the following, known as the canonical generator matrices of $\text{SE}(3)$ [80]:

$$\begin{aligned} \frac{\partial \mathbf{T}}{\partial \delta x} &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} & \frac{\partial \mathbf{T}}{\partial \delta \theta_r} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \frac{\partial \mathbf{T}}{\partial \delta y} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} & \frac{\partial \mathbf{T}}{\partial \delta \theta_p} &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \frac{\partial \mathbf{T}}{\partial \delta z} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} & \frac{\partial \mathbf{T}}{\partial \delta \theta_q} &= \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \quad (2.67)$$

Right-multiplying Equation (2.66) by a homogeneous coordinate vector $\mathbf{p} = [x, y, z, 1]$ yields:

$$\frac{d}{d\Delta}(\mathbf{T}(\Delta)) \mathbf{p} = \begin{bmatrix} \mathbf{1}^{3 \times 3} & [\mathbf{p}]^\times \\ \mathbf{0}^{1 \times 3} & \mathbf{0}^{1 \times 3} \end{bmatrix} \quad (2.68)$$

where $[\cdot]^\times$ is an operator that given an arbitrary homogeneous vector $\mathbf{v} = [x, y, z, w]^T$

returns a 3×3 skew-symmetric matrix:

$$[\mathbf{v}]^\times = \frac{1}{w} \begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix} \quad (2.69)$$

Full Jacobian

Substituting Equation (2.64) into Equation (2.63), and combining with Equation (2.54), gives us an expression for the full Jacobian of the system:

$$\frac{d}{d\Delta} (\mathbf{h}(\mathbf{x}, \mathbf{p}_i)) = \frac{1}{w} \begin{bmatrix} 1 & 0 & -\frac{u'}{w} \\ 0 & 1 & -\frac{v'}{w} \end{bmatrix} \mathbf{K} \mathbf{P}_{\text{pr}} \mathbf{T}_{c,b} \begin{bmatrix} \mathbf{1}^{3 \times 3} & [\mathbf{v}]^\times \\ \mathbf{0}^{1 \times 3} & \mathbf{0}^{1 \times 3} \end{bmatrix} \quad (2.70)$$

where $\mathbf{v} = \mathbf{T}_{b,w}(\bar{\mathbf{x}}) [\mathbf{p}_i^T, 1]^T$.

2.10.3 Depth Measurements

In Chapter 4 we are able to use the observed size of landmarks to obtain a coarse estimate of depth. We model this with another sensor function (also parameterised by Δ), $\mathbf{h}_d(\Delta, \mathbf{p}_i)$:

$$\mathbf{h}_d(\Delta, \mathbf{p}_i) = \sqrt{(\delta x + \bar{x} - p_x)^2 + (\delta y + \bar{y} - p_y)^2 + (\delta z + \bar{z} - p_z)^2} \quad (2.71)$$

where the x, y, z translational components of $\bar{\mathbf{x}}$ are $\bar{x}, \bar{y}$, and $\bar{z}$, respectively. The Jacobian $\frac{\partial \mathbf{h}_d}{\partial \Delta}$ is computed simply as:

$$\frac{\partial}{\partial \Delta} \mathbf{h}_d(\Delta, \mathbf{p}_i) = \frac{1}{\mathbf{h}_d(\Delta, \mathbf{p}_i)} [\delta x + \bar{x} - p_x, \delta y + \bar{y} - p_y, \delta z + \bar{z} - p_z, 0, 0, 0] \quad (2.72)$$

Therefore the residual vectors and Jacobians for M depth measurements are:

$$\mathbf{e}_d(\Delta) = \begin{bmatrix} \mathbf{h}_d(\Delta, \mathbf{p}_1) - \mathbf{z}_1 \\ \vdots \\ \mathbf{h}_d(\Delta, \mathbf{p}_M) - \mathbf{z}_M \end{bmatrix} \quad (2.73)$$

$$\mathbf{J}_d(\Delta) = \frac{d\mathbf{e}_d}{d\Delta} = \begin{bmatrix} \frac{\partial \mathbf{h}_d}{\partial \Delta}(\Delta, \mathbf{p}_1) \\ \vdots \\ \frac{\partial \mathbf{h}_d}{\partial \Delta}(\Delta, \mathbf{p}_M) \end{bmatrix} \quad (2.74)$$

2.10.4 Incorporating the Motion Model

Recall from Section 2.5.2 that we have a motion model $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_{t+1})$ that provides a new estimate of the pose $\mathbf{x}_{t+1}$ given the previous pose $\mathbf{x}_t$ and a control input $\mathbf{u}_{t+1}$. In our system, we consider the frame-to-frame pose estimate from visual odometry as our control input, which we apply to the pose estimate from the previous time step, $\mathbf{x}_t$, to obtain an initial estimate $\bar{\mathbf{x}}_{t+1|1}$ for the pose at the current time step (denoting the input to the i th Levenberg-Marquardt iteration as $\bar{\mathbf{x}}_{t+1|i}$):

$$\bar{\mathbf{x}}_{t+1|1} = \mathbf{x}_t \oplus \mathbf{u}_{t+1} \quad (2.75)$$

This is then used in the first iteration of our Levenberg-Marquardt process, as in Equation (2.60). We then bias our cost function towards this estimate with an additional term $\mathbf{f} : \mathbb{R}^6 \rightarrow \mathbb{R}^6$:

$$\mathbf{f}(\Delta) = \Delta \oplus \bar{\mathbf{x}} \quad (2.76)$$

The residual $\mathbf{e}_f(\Delta)$ and Jacobian $\mathbf{J}_f(\Delta)$ for this term are then:

$$\mathbf{e}_f(\Delta) = \mathbf{f}(\Delta) - \bar{\mathbf{x}}_{t+1|1} \quad (2.77)$$

$$\mathbf{J}_f(\Delta) = \bar{\mathbf{x}} \quad (2.78)$$

2.10.4.1 Soft Kinematic Constraints

Occasionally we may wish to impose a soft kinematic constraint on some of the components of $\mathbf{x}$. For example, to bias the system towards planar movement, we would like to penalise movement along the z -axis, as well as rotations around x (roll) and y (pitch). Recall from Equation (2.15) that the motion model term is weighted by an inverse covariance, $\mathbf{Q}^{-1}$. Such a planar constraint can be realised by ensuring that z , roll, and pitch have a very low covariance, for example $\mathbf{Q} = \text{diag}([1, 1, 10^{-3}, 10^{-3}, 10^{-3}, 1])$.

2.11 Summary

In this chapter we have introduced the problems of mapping and localisation. We have shown how the problems can be decoupled, resulting in the teach and repeat paradigm. We defined each sub-problem in a general sense, and detailed how they are solved specifically using visual sensors. In Chapter 4 we will apply these theoretical principles and algorithms to our implementation of a teach-and-repeat system designed for an autonomous vehicle operating at night. Additionally, we have specified the data and query requirements for a map representation. We address and expand upon these in the next chapter as we look to design a database system suitable for robotics applications.

Chapter 3

Database Design and Implementation

“ *“I checked it very thoroughly,” said the computer, “and that quite definitely is the answer. I think the problem, to be quite honest with you, is that you’ve never actually known what the question is.”* ”

Douglas Adams, *The Hitchhiker’s Guide to the Galaxy*, 1979

3.1 Introduction

Lifelong learning for robotic systems requires large quantities of data to be collected and stored over long periods of time. As these data accumulate, they become increasingly difficult to manage and query. Without a scalable system in place, finding useful data becomes ever more complex and this undermines our goal of achieving long-term autonomy. As of yet, this is a largely neglected area of robotics research. Figure 3.1 shows an all-too-common pattern of data storage that exists today.

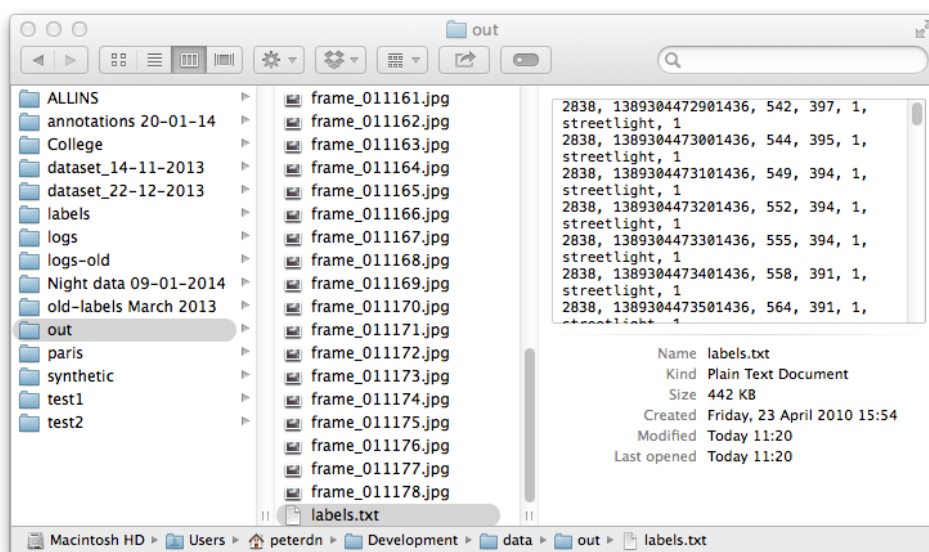


Figure 3.1: A typical legacy method of data ‘organisation’, complete with inconsistently and vaguely named directories, and ad-hoc, poorly specified labels in a standalone text file. While manageable for small datasets, this organisation strategy does not scale to the many terabytes of data we deal with every day.

Many publications focus on the mechanics of lifelong autonomy but very few explicitly deal with the problem of storing and accessing the required data in a way designed to aid long-term, large-scale mobile autonomy. Given the need for field roboticists to build coherent systems, it is time for this subject to be addressed.

Mobile robotics applications have some unusual data needs that cannot always be anticipated in advance. An example is illustrative. We often want to evaluate the efficacy of a new feature detector for visual odometry and thus testing under differing lighting conditions is vital. What we actually want to do is automatically collate image sequences that satisfy complicated compound queries such as ‘find sequences $> 50\text{m}$ of stereo images, captured while driving into the sun over wet ground in the absence of dynamic obstacles’. This should run over all data ever collected and return a pristine dataset as if this had been the sole purpose of our experimentation over the past four years.

As another example, suppose we need images of street lights to train a new

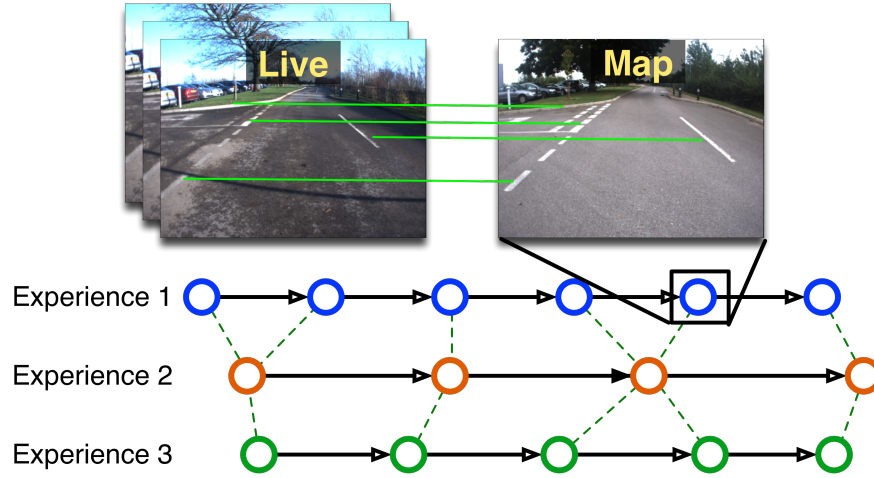


Figure 3.2: Experience Based Navigation (EBN) [12] is a state-of-the art visual navigation framework for autonomous robots. Maps, in the form of ‘experiences’ with associated graph structure, camera images, and landmarks, accumulate over many years. One of the questions this thesis aims to answer is: ‘how do we store and retrieve this kind of data in a flexible and efficient way?’

state-of-the-art street light detector for data association purposes and to ultimately map their locations. Instead of wasting time collecting a whole new set of data for this specific purpose, we should first look to our existing data. It is probably the case that we inadvertently have images of street lights from previous data collection missions, and therefore we would like the ability to search for them.

To aid in solving this problem, we have designed and implemented a relational database framework that is applicable to a wide range of robotics applications. A data and query model is presented which cleanly distinguishes between sensor data and user-defined metadata. This makes it trivial for a user to decorate the database with their own contributions, and makes those contributions accessible to other users in a consistent way. For example, if someone builds the aforementioned street light detector and runs it over 100,000 images, they are then able (and encouraged) to add those results to the database for others to use in the future.

Our framework not only makes offline batch processing tasks much easier, but also supports the needs of online tasks, for example the storage and retrieval of

maps used by a robot’s navigation system at runtime. This massively reduces the overhead of implementing and testing new navigation systems as data back-ends do not need to be written from scratch. We consequently use the system in our night-time localisation system that is detailed in Chapter 4. Another motivating use case is our research group’s Experience Based Navigation (EBN) system [12], a visual navigation framework designed to deal with vast maps that grow continuously over a robot’s entire lifetime (see Figure 3.2). EBN utilises our database framework in order to fulfil these challenging data storage and retrieval demands.

In this chapter we present the design and implementation of this framework. We analyse its performance under a variety of conditions, and demonstrate its real-world utility via a selection of use cases, including the aforementioned EBN.

3.2 Relational Database Fundamentals

In this section we introduce some fundamental concepts about relational databases and SQL that will be used later in the chapter. Readers already familiar with this material may skip ahead.

The *relational model* for databases was first introduced by Codd in 1970 [15] and has since become the dominant data model upon which most popular database management systems (*DBMS’s*) are based. These are known as relational database management systems (*RDBMS’s*).

Under the relational model, a database is a collection of tables (known as *relations* in the traditional literature). A table is defined by its *schema* which specifies the table’s name, its columns (*fields*), and the datatypes of those columns (e.g. numeric, text, binary, etc). The data in a table are then a set of rows (*tuples* or *records*), each with the same number of fields as defined in the table’s schema.

An important concept in relational databases is that of *keys*. A table’s *primary*

<i>sensor_id</i>	name	type
1	Bumblebee2	Camera
2	Ladybug2	Camera

(a) ‘Sensor’ table.

<i>sensor_id</i>	<i>timestamp</i>	snow
1	1000	yes
2	1010	no
1	2000	no

(b) ‘Snow’ table.

<i>sensor_id</i>	<i>timestamp</i>	sun
1	1002	no
2	1012	yes

(c) ‘Sun’ table.

Table 3.1: Example tables in a relational database. Primary key fields are shown in *italics*. Table 3.1a records the names and types of various sensors. Tables 3.1b and 3.1c show tables ‘Snow’ and ‘Sun’, respectively, representing the outputs of some virtual sensors that detect these conditions in sensor data.

key is defined as one or more columns whose combined value uniquely identifies a single row. This provides a way to search for and refer to specific rows. A *foreign key* in table A is one or more columns whose combined value ‘points to’ a primary key *in another table* B. That is, the foreign key in A is a logical pointer that can be used to look up the row in B that is identified by that key.

Informally, we may define a schema by specifying its name followed by a comma-separated list of column name and datatype pairs. We may omit the datatype declarations in cases where they are clear from context. The primary key column (or columns in the case of a *composite primary key* are underlined. For example, consider three tables defined by the following schema:

$$\begin{aligned} \text{Sensor}(\underline{\text{sensor_id : integer}}, \\ \text{name : text, type : text}) \end{aligned} \tag{3.1}$$

Snow(*sensor_id* : integer,
timestamp : integer, *snow* : boolean) (3.2)

Sun(*sensor_id* : integer,
timestamp : integer, *sun* : boolean) (3.3)

In this example, rows in the Sensor table are identified by a unique numeric ID (the table's primary key) and two additional columns store the sensor's name and type. Both the Snow and Sun tables have composite primary keys consisting of a sensor ID and timestamp. An additional column stores whether snow or sun was detected in the data from that sensor at that time. Note that the *sensor_id* columns in these tables are foreign keys that refer to records in the Sensor table.

We occasionally denote columns as Table.column_name when it is necessary to distinguish between different columns that have the same name.

Example instances of these tables are shown in Tables 3.1a to 3.1c.

3.2.1 SQL

The Structured Query Language (SQL) was originally developed at IBM in the early 1970s as a way of manipulating and retrieving data stored in their seminal RDBMS, System R. Today, SQL is an ISO standard¹ and dialects of it can be found in practically every modern RDBMS.

Here we provide a basic reference covering one subset of SQL: the Data Manipulation Language (DML) which provides operations to query tables. For a more detailed overview of the language, including how to insert, update, delete rows, and

¹ISO/IEC 9075-1:2011

create tables and indexes, see [73] or the PostgreSQL documentation.²

3.2.1.1 Querying Data

Rows are queried for using the **SELECT** statement that specifies which columns should be returned, which tables should be queried, and which conditions should be satisfied for a row to be returned. Therefore the output of a **SELECT** statement is a set of rows (which can be considered simply as another table). The statement takes the following form:

```
SELECT [DISTINCT] select-list
FROM from-list
WHERE conditions-list
```

where:

select-list is a list of column names (or more generally, expressions involving columns, e.g. `R1.amount + R2.amount`). The optional **DISTINCT** keyword will eliminate duplicate rows from the query output.

from-list is a list of table names (and optional aliases). If multiple tables are specified separated by commas, then their *cross product* is computed (alternatively known as *cross join*; see Section 3.2.1.2).

conditions-list is a list (separated by boolean operators **AND** or **OR**) of boolean conditions of the form *expression* **OP** *expression*, where **OP** is a comparison operator (i.e. `=`, `<>`, `<`, `>`, `<=`, `>=`). Sometimes conditions can be also be arbitrary boolean-valued functions, for example `ST_DWithin(A, B, x)` is a function we use later that returns true if geometric objects *A* and *B* are within *x* metres of each other.

²<http://www.postgresql.org/docs/>

Example: querying sensors with a particular name. Note that the $\star$ wildcard can be used to select all columns from the specified tables:

```
SELECT  $\star$  FROM Sensor WHERE name = 'Bumblebee2'
```

 (3.4)

Example: querying times when no snow detected in sensor with ID 1:

```
SELECT timestamp FROM Snow WHERE snow = no AND sensor_id = 1
```

 (3.5)

The tables output by Queries 3.4 and 3.5 are shown in Tables 3.2a and 3.2b respectively.

3.2.1.2 Joins

A *join* operation is one that pairs together rows from two tables based on some condition, outputting them as a single row. Most commonly, joins are used to dereference foreign keys, outputting rows that contain columns from both the referencing and referenced tables. In this work we also make use of *temporal joins* which associate rows with timestamps that represent (almost) the same time, and *spatial joins* which associate rows based on some spatial condition (e.g. GPS fields that are nearby each other). Joins are specified using the following construction as part of a `SELECT` statement's *from-list* (multiple joins can be chained together):

left-table JOIN right-table ON conditions-list

For example, to join our Snow table with Sensor in order to dereference *sensor_id*:

```
SELECT  $\star$  FROM Snow JOIN Sensor ON Snow.sensor_id = Sensor.sensor_id
```

 (3.6)

3.2 Relational Database Fundamentals

sensor_id	name	type
1	Bumblebee2	Camera

(a) Sensors named ‘Bumblebee2’ (Query 3.4).

timestamp
2000

(b) Times when no snow detected in sensor with ID 1 (Query 3.5).

sensor_id	timestamp	snow	name	type
1	1000	yes	Bumblebee2	Camera
2	1010	no	Ladybug2	Camera
1	2000	no	Bumblebee2	Camera

(c) Join of the ‘Snow’ and ‘Sensor’ tables on sensor_id column (Query 3.6).

Sun.sensor_id	timestamp	sun	Sensor.sensor_id	name	type
1	1002	no	1	Bumblebee2	Camera
1	1002	no	2	Ladybug2	Camera
2	1012	yes	1	Bumblebee2	Camera
2	1012	yes	2	Ladybug2	Camera

(d) Cross product of the ‘Sun’ and ‘Sensor’ tables (Query 3.7).

resulting in the output table shown in Table 3.2c. Note that this table has columns from both schemas that were joined. Also note that *conditions-list* can contain arbitrary conditions and so rows in *left-table* may be paired with multiple rows in *right-table* and vice versa. In particular, if `JOIN ... ON TRUE` is specified, the output table is the *cross product* of the input tables, containing every possible pairing of rows. This is alternatively specified with the `CROSS JOIN` construction. For example, the cross product of Sun and Sensor tables can be computed with the following query:

`SELECT * FROM Sun CROSS JOIN Sensor` (3.7)

or equivalently:

`SELECT * FROM Sun, Sensor` (3.8)

This results in the output shown in Table 3.2d.

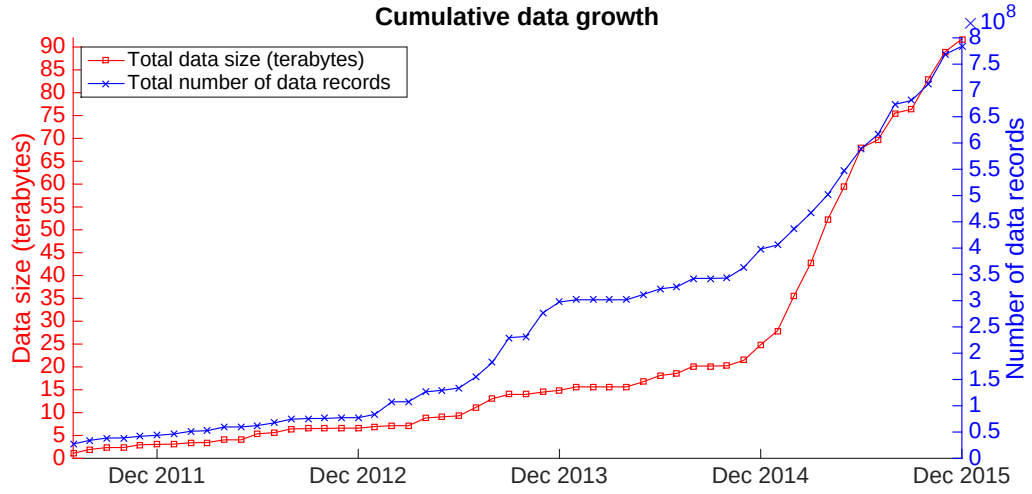


Figure 3.3: Cumulative amount of sensor data we have amassed each month from July 2011 to December 2015. Shown in red is the total size of these data in terabytes and shown in blue is the approximate number of ‘individual’ data records (i.e. single camera images, GPS readings, laser scans).

We can now apply SQL as part of our toolset for designing a database system that meets the requirements laid out in Chapter 2 and those in the following section.

3.3 System Requirements

Our own experience serves as motivation for what is to follow, and we suspect our requirements are not unique to us. Figure 3.3 shows how our data have accumulated almost exponentially over the past 4.5 years. As of December 2015 we have amassed almost 90TB of data, comprising of nearly 800 million ‘individual’ records, and more are added on a daily basis. It is now intractable to manage all of this by hand.

Firstly there is the problem of reuse. In a previous time, data were collected for a specific purpose, used by one or two people, then forgotten about. Almost no metadata describing particular datasets were stored and when they were, usually in a non-standard ad-hoc way (such as in notes on a wiki, or in a README). Metadata and semantic information (*annotations*) extracted from processed datasets suffered from the same problems. When gigabytes of data are collected and processed like

this it becomes increasingly difficult to reuse what is considered useful and instead is easier to collect and process new data for each new purpose. To address this problem, we require that existing useful data can be found effectively and we therefore need a way to index it, as well as efficient ways to add new data when necessary. We also require a standard method for data annotation and require that these annotations can be easily traced to their underlying data.

Next, we have the problem of retrieval. Once we know a dataset contains relevant data, we must somehow retrieve them. For batch tasks, this would have been done by manually inspecting the dataset, chopping out the required parts and discarding the rest – a time-consuming and boring process. For online tasks, data would typically be accessed through a custom-written ad-hoc back-end, leading to bugs and time wasted on re-implementing common components.

Lastly, we have the problem of physical storage and organisation of data. Previously, datasets were organised solely by directory structure and filename. This reliance on physical location presents many problems when datasets must be moved. Users should not need to care on which machines their data physically reside. Therefore we aim to build a modular system that is platform and tool agnostic. Additionally, users should be able to issue queries and access data using any programming language on any operating system. Ideally, queries should run online on robotic platforms with little or no modification.

3.3.1 Concrete Requirements

We combine the system requirements outlined above with the requirements for storing and querying maps, given in Section 2.4.2.3 and repeated here for convenience, to obtain a complete set of concrete requirements on what capabilities our system should have:

- (i) Create landmarks that are 2D or 3D geometric objects.
- (ii) Create a graph structure with nodes linked by edges.
- (iii) Annotate landmarks, nodes, and edges with arbitrary data.
- (iv) Fast (i.e. suitable for real-time systems) nearest-neighbour queries on geometric objects.
- (v) Fast (i.e. suitable for real-time systems) graph traversal, e.g. find neighbouring nodes, depth-first search.
- (vi) Index into our existing raw data files to allow for efficient lookup (e.g. based on record ID or timestamp).
- (vii) Issue queries from any programming language on any operating system.
- (viii) Run queries on robotic platforms in the field, with no network connection.
- (ix) Deploy easily on a robotic platform.

Given these requirements, we now examine some existing systems and tools and assess their ability to fulfil them.

3.4 Background and Related Work

In the context of robotics and lifelong learning, very little work has so far addressed the problems of organising, maintaining, and utilising the vast amounts of data that will accumulate over extended periods of time. These topics have historically been a focus of the file system and database communities, in which they remain active areas of research to this day.

3.4.1 Data Management within Robotics

Traditionally, (relatively) small datasets are collected for specific purposes and are often discarded or forgotten about once they have served their purpose. Various datasets have been released to the robotics community along with associated tools and documentation. Many of these have been widely referenced in other publications and serve as convenient benchmarks for comparing performances of related techniques. Some recent and frequently cited examples include the Victoria Park 2D dataset (2000) [31], the New College Vision and Laser dataset (2009) [81], the DARPA Urban Challenge dataset (2010) [41], the UQ St Lucia dataset (2010) [89], the Devon Island Rover Navigation dataset (2012) [26], and the KITTI Vision Benchmark Suite (2013) [28]. Although these resources are invaluable to the robotics and computer vision communities, they are also utterly static – the data are a snapshot of a single period in time and adding to them in a way that is accessible for others to use is difficult. Additionally, as these datasets are often formatted differently and require non-standard tools to use, it is cumbersome to mine data from more than one at a time. To be useful in the context of lifelong learning, we wish to move away from the idea of disjoint, immutable datasets and towards a living, growing repository of everything we record and have ever recorded.

The Radish Robotics Dataset Repository [39] is an attempt at addressing these issues by providing a place for researchers to access and contribute to a repository of Creative Commons³ licensed data. As of writing, however, the most recent addition is from 2011 and the entire repository consists of only about 1GB of data. This pales in comparison to the terabytes of data we need to manage every day. In addition, the provided datasets are all very disjoint and heterogeneous – having all of our data in a common format is a requirement for an effective system of management.

RoboEarth [88] is an ambitious project that proposes to solve some of these

³<http://creativecommons.org/licenses/>

problems by building a ‘World Wide Web’ for robots. A distributed cloud database is used to store machine-readable semantic data about environments, objects, and actions. Generic robots can access this prior knowledge to help complete tasks and can upload their own knowledge once they succeed. A modular software architecture enables generic actions (e.g. moving, grasping) to be realised on specific hardware. A subset of RoboEarth’s vision is close in spirit to what we want to achieve, however it places more of an emphasis on the storage, retrieval, and reconciliation of knowledge required for high-level planning and reasoning tasks. It does not, for example, define a format for storing graph-based maps (requirement (ii)), and the dependency on a cloud database precludes requirement (viii).

By far the most related work we are aware of in this area is by Mühlfellner et al. [64] wherein a relational database system designed for managing visual maps is presented. The authors implement schemas that represent components of visual maps such as keypoints, landmarks, and reference frames, and describe the relationships between them. They demonstrate how maps can be augmented with new data as they are gathered, and how compact summaries can be generated through the use of queries. Their system incorporates the concept of a pose graph, which they store in a serialised form, but it does not allow for arbitrary graph structures (requirement (ii)). It also has no support for storing or managing our raw sensor data (requirement (vi)). Ultimately, our database system aims to be more general and extensible, in particular to support laser and GPS-based maps as well as visual maps.

3.4.2 Data Management in a Broader Context

More generally outside of robotics, DBMS’s have been the subject of academic research for many decades. RDBMS’s have traditionally been the most popular. Most support SQL as the standard programming language interface. As databases

are often too large to fit into main memory, one particularly important area of research is concerned with improving the throughput of reads and writes to and from slow secondary storage devices such as hard drives and SSDs. Efficient index data structures have been developed for searching data sorted along a single dimension (e.g. B-Trees [3]) as well as for geospatial and multi-dimensional queries (e.g. R-Trees [32]).

In recent years, many alternatives to relational databases have been developed to cope with the rise of ‘big data’ and the scalability challenges involved with servicing thousands of concurrent users. These so-called ‘NoSQL’⁴ systems are typically characterised by the fact they do not adhere to the relational model and instead place an emphasis on being *horizontally scalable* – that is, their data is distributed and/or replicated over multiple servers that do not share memory.

Included among these are simple key-value stores like Riak⁵ and Redis⁶. These systems provide operations to put and get data based on a single key value (much like a C++ map). Relationships and links between data are difficult to express in this paradigm, making it unsuitable for many of our requirements, particularly arbitrary annotation (requirement (iii)). Document-based data stores like MongoDB⁷ and CouchDB⁸ are more complex, allowing arbitrary, schemaless annotation of data stored in hierarchical documents. Specialised graph databases such as Neo4j⁹ would fulfil our graph-based map storage and query requirements, however they are unsuitable for storing non-graph data and therefore in particular would fail to meet requirement (vi). Distributed multidimensional key-value databases like HBase¹⁰ and Cassandra¹¹ have taken inspiration from the state-of-the-art but proprietary

⁴NoSQL is usually understood to stand for “not only SQL” rather than “no SQL.”

⁵<http://basho.com/products/#riak>

⁶<http://redis.io/>

⁷<https://www.mongodb.org/>

⁸<http://couchdb.apache.org/>

⁹<https://neo4j.com/>

¹⁰<http://hbase.apache.org/>

¹¹<http://cassandra.apache.org/>

Google BigTable [10] and Amazon Dynamo [20] systems. These suffer from the same inflexibility as simpler key-value stores.

The aforementioned systems tend to shine when managing very specific types of data, up to the order of many hundreds of petabytes stored across thousands of machines. Although this is far beyond our (current) need, they are all still worth bearing in mind and have influenced some aspects of our system architecture. Next, we discuss in general terms our rationale behind preferring a traditional relational database model to any of these alternatives.

3.4.3 Relational DBMS vs NoSQL

1. NoSQL approaches are typically more suitable for unstructured data, where each record may have hundreds of fields that are unique to it. The process of adding and removing fields is designed to be very quick, compared to in relational systems where such structural modifications can be more time consuming. The type of data our system is designed to deal with, however, will likely have a consistent structure across thousands of records.
2. RDBMS's have a solid mathematical basis, have been widely researched for decades, and a great deal of work has been put into optimising their performance. We can take advantage of this work in popular open-source implementations such as PostgreSQL. NoSQL systems can perform better for specific types of data, for example unstructured web documents, however there is no indication that we would see any benefit with our data. Neo4j would be an exception to this for our graph-based data, however we would need to combine it with another system to store non-graph data, drastically increasing system complexity.
3. Implementations of RDBMS's have been around longer than NoSQL offerings

and are generally regarded as being more mature, stable, and consequently safer to entrust with valuable mission-critical data. Indeed, many NoSQL systems *deliberately* trade safety for performance (ACID¹² vs BASE¹³ guarantees).

4. SQL is ISO standardised¹⁴ and SQL-based data stores exist for almost all platforms and operating systems. Numerous commercial and free implementations support a myriad of use cases, and although they may differ in non-standard ways, it is regardless easier to migrate data and queries from one SQL-based system to another than to a completely different NoSQL model.
5. The majority of NoSQL systems are designed to be run as a server process, to be communicated with over a network or inter-process (IPC) connection. This introduces significant performance overheads that we explore in Section 3.11 with two SQL systems. One of these is a fully-featured SQL system that runs as an embedded library, eliminating costly network or IPC communication. Unfortunately, NoSQL systems such as MongoDB and Neo4j have no such embedded alternative, therefore fail to meet requirement (v). Additionally, the setup and administration of such servers has a significant learning curve, violating requirement (ix).

Given all of these reasons, we have chosen to implement our system on top of existing SQL-based RDBMS's, specifically PostgreSQL and SQLite. We compare and discuss the differences between these two back-ends later in Section 3.11.

¹²Relational DBMS's are designed to guarantee Atomicity, Consistency, Isolation, and Durability (ACID). This amounts to assurances that, under normal operation, transactions (i.e. atomic groups of operations) will never leave data in an inconsistent state or interfere with other transactions.

¹³In contrast, many NoSQL systems guarantee a much weaker condition of Basically Available, Soft state, Eventual consistency (BASE). Such systems are able to break data consistency (i.e. return old data) in order to instead guarantee system liveness (i.e. respond to every query within a finite period of time).

¹⁴ISO/IEC 9075-1:2011

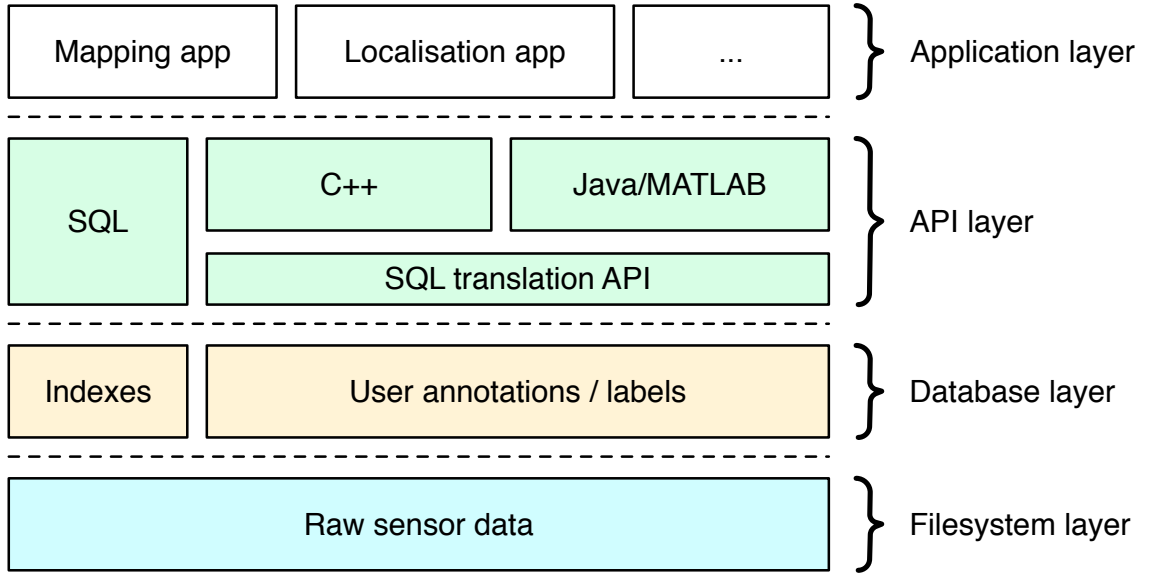


Figure 3.4: Database architecture hierarchy. Raw sensor data are stored at the lowest layer, directly on the file system. Annotations and indexes are stored in the DBMS itself. APIs for various programming languages are built on top of this to provide convenient and consistent ways to access data and issue queries. These APIs are consumed by client applications that are the actual producers and consumers of data.

3.5 Architectural Overview

We begin with a high-level overview of the database system architecture. A conceptual hierarchical view is shown in Figure 3.4. From bottom to top, the main layers of the system are as follows:

1. Filesystem layer – concerned with how raw sensor data are physically stored at the level of files and directories. The format of these files is detailed in Section 3.6.
2. Database layer – concerned with how data (annotations in particular) are structured and organised at a higher level in the DBMS, as well as how raw data are indexed and referenced. It is the structure of this layer that influences the types of queries we can perform efficiently. Towards this end, we develop data and query models in Section 3.7.

3. API layer – concerned with the various interfaces that provide access to the underlying data and support the execution of queries in a consistent way. We take a look at various implementation issues and decisions in Section 3.8. A brief discussion of the programming language APIs to the system is found in Section 3.9.
4. Application layer – the applications that actually consume, produce, and manipulate data via the underlying APIs. We consider some use cases in Section 3.12.

3.6 Filesystem Layer

We have standardised the way raw sensor data are stored in an attempt to eliminate fragmentation of our tools and methods. A *monolithic file* (or simply *monolithic*) contains a sequence of atomic data records. Each record consists of an XML header and a binary blob (*message*) containing serialised data. The XML header contains sensor-agnostic metadata about the message such as a timestamp indicating when it was collected, its length in bytes, and an identifier indicating the type and format of the serialised data. This structure is illustrated in the far left of Figure 3.5. Our serialisation method of choice is described later in Section 3.6.1.

As sensor data are being collected, monolithic files are constructed on the fly. Raw sensor data are streamed through a driver, which converts them into messages of the appropriate format. Both the message and header are then appended to the output monolithic. As they are simply an unlinked sequence of distinct and atomic records, monolithics can be constructed, split, and concatenated easily, and their length is limited only by the underlying file system.

Raw sensor data are stored in monolithic files in a well-defined directory structure that is organised by the platform and sensors used to collect them, and time of collection. In our centralised data repository, the top-level directory is located on a

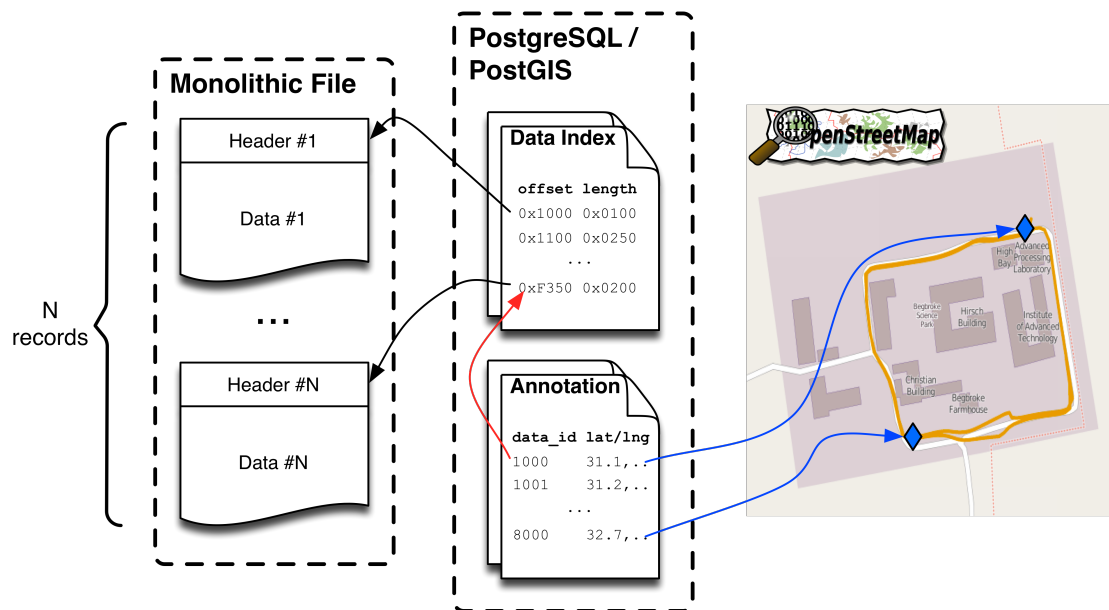


Figure 3.5: Overview of the data model structure. At the lowest level, raw data are stored in sequential records in flat ‘monolithic’ files. An index table in the DBMS holds offsets that point to these records (represented by black arrows). Annotations exist in separate tables with each individual annotation pointing to some record in the index (red arrow) and optionally to any other data – in this example, to locations in OpenStreetMap (blue arrows).

network drive that allows any user on the network to mount it to anywhere in their own filesystem. This is ideal for offline and batch processing. For online use, for example on a robot without a network connection, copies of relevant monolithic files can be stored and accessed locally. This process and its implications are discussed further in Section 3.8.2.2.

3.6.1 Serialisation with Google Protocol Buffers

Robotics systems are complex, often consisting of many individual processes written in different programming languages, running on different operating systems and processor architectures. This diversity makes the task of data serialisation¹⁵ non-trivial. To simplify this task, we use Google Protocol Buffers¹⁶ as our serialisation method of choice. Message schemas are defined using a simple Protocol Buffer interface description language that is syntactically similar to defining C structures. These schemas are then passed through a code generator for another programming language such as Java or C++. This results in a linkable library that handles the serialisation and deserialisation of messages transparently, meaning we do not have to separately write and test such boilerplate code for those languages. The schemas are both forward and backward compatible, allowing us to add new data fields to messages in the future without breaking existing software and workflows. For example, legacy messages that are missing new fields can be deserialised without them (the fields will simply be empty). Similarly, legacy software that is unaware of an updated message format is able to deserialise new messages, but can only access the legacy fields. Additionally, messages are automatically compressed, greatly reducing the storage and bandwidth required to handle them.

¹⁵*Serialisation* is the process of converting a data structure or object into a stream of bytes such that it can be stored in a file or transmitted across a network link. *Deserialisation* is the reverse process: recreating the data structure or object from a stream of bytes.

¹⁶<https://developers.google.com/protocol-buffers/>

3.7 Database Layer

Here we describe the main data models that underpin our framework. At the lowest level is *raw sensor data*: typically recorded directly from physical hardware and subjected to minimal, if any, processing. Although we have specified how these are physically stored in monolithic files, we now need to consider how they are represented in and referenced from our DBMS. *Annotations* exist as a layer above this and are related directly to the data they annotate. Data that do not fall neatly into either of these two categories must simply conform to the standard relational model (as described in Section 3.2).

In many ways these categories are arbitrary and some data can fall into multiple categories depending on context. However, they help us identify common types of queries which we take advantage of to reduce complexity. Returning to our street light example from Section 3.1: raw data include images captured by a camera during a data collection mission. Annotations include the output of a street light detector on these images (perhaps bounding boxes and labels). Other data include everything else that isn't directly causally related, for example an OpenStreetMap [33] road network map.

The following subsections make some use of SQL in order to express queries. Introductions to this topic can be found in Section 3.2 and also in [73].

3.7.1 Raw sensor data

Raw sensor data form the bedrock of our data model and many of our subsequent processing needs. As the collection of sensor data is such a fundamental part of our workflow, it seems appropriate that they be given special consideration.

Raw data are characterised by the fact that they represent part of the state of the world, as observed by some robotic platform, at a particular moment in time.

A fundamental query we wish to support is to find the *entire* state of the world, as observed by a platform, at a particular moment in time. For example, our detector finds a street light in some camera image that was captured at time t . We therefore may want to find, from our collection of raw data, the corresponding GPS reading or integrated visual odometry pose giving the location of the robot at time $t' \approx t$. This will be the first and most coarse step in many applications' processing pipelines.

Consider a robotic platform $r \in R$ representing an entity equipped with a set of sensors $S(r)$. The output datum of sensor $s^i \in S(r)$ at time t is denoted s_t^i . We assume that a single sensor can be mounted on more than one platform, though not at the same time. We also assume that, by definition and without loss of generality, sensors cannot produce multiple data at the same time (i.e. in cases such as stereo cameras, we treat both images as a single datum). Given these constraints, it can be seen that $\langle \text{platform, sensor, timestamp} \rangle$ tuples map to individual data via a lookup function $\bar{\Omega}$:

$$s_{t'}^i \leftarrow \bar{\Omega}(r, s^i, t) \quad (3.9)$$

where t' is the nearest¹⁷ discretised timestamp for which a sensor reading exists. It follows that $\langle \text{platform, timestamp} \rangle$ tuples map to the state of the world as observed by the entire platform via Ω :

$$\{s_{t^i}^i\} \leftarrow \Omega(r, t) \quad (3.10)$$

where t^i is the nearest discretised timestamp for which an output from s^i exists.

Using the fact that our underlying raw data are stored in monolithic files (as explained in Section 3.6), these mappings can be modelled as four tables whose rows respectively represent individual robotic platforms, individual sensors, individual

¹⁷The definition of 'nearest' may differ between queries but note that it is beyond the scope of the system to, for example, interpolate between records or verify annotation correctness – how that is handled is up to individual client applications.

monolithic files, and individual data:

$$\text{Platforms}(\underline{\text{platform_id}}, \dots) \quad (3.11)$$

$$\text{Sensors}(\underline{\text{sensor_id}}, \dots) \quad (3.12)$$

$$\text{Monolithics}(\underline{\text{monolithic_id}}, \text{platform_id}, \text{sensor_id}, \text{filepath}) \quad (3.13)$$

$$\text{Data}(\underline{\text{data_id}}, \text{monolithic_id}, \text{timestamp}, \text{offset_bytes}, \text{length_bytes}) \quad (3.14)$$

where the *Monolithics.platform_id*, *Monolithics.sensor_id*, and *Data.monolithic_id* fields are foreign keys referencing rows in the *Platforms* and *Sensors* relations respectively. Note that as raw data are not stored directly in the database, but rather within monolithic files, the *Monolithics* and *Data* tables provide all the information required to extract individual records from those files: the path to the file, the location where the record begins, and the length of the record in bytes. These tables therefore form an easily queryable index into the raw data as per requirement (vi). This index structure is illustrated back in Figure 3.5.

Expressed in SQL, Equation (3.9) becomes a **SELECT** query over the *Data* and *Monolithics* tables joined together:

$$\begin{aligned} &\text{SELECT offset_bytes, length_bytes, filepath, ABS(timestamp - t) AS time_diff} \\ &\text{FROM Data JOIN Monolithics ON Data.monolithic_id = Monolithics.monolithic_id} \\ &\text{WHERE platform_id = } r \text{ AND sensor_id = } s^i \text{ AND ABS(timestamp - t) < } T \\ &\text{ORDER BY time_diff ASC LIMIT 1} \end{aligned} \quad (3.15)$$

where in order to select the temporally closest datum to our query timestamp t , we have: (1) used the **SELECT** ... **AS** construction to compute the absolute difference between data timestamps and t , naming this value *time_diff*; (2) used the **ORDER**

BY ... ASC clause to sort the output by ascending order of *time_diff*; (3) used the LIMIT 1 clause to return only the first row (i.e. with the smallest *time_diff* value). An application would use the data returned from this to load data directly from the specified locations in the specified monolithic files.

Similarly, the mapping Ω given by Equation (3.10) is captured by the following SQL query:

```
SELECT offset_bytes, length_bytes, filepath, ABS(timestamp - t) AS time_diff
FROM Data JOIN Monolithics ON Data.monolithic_id = Monolithics.monolithic_id
WHERE platform_id = r AND ABS(timestamp - t) < T
```

(3.16)

where T is some small time delta value.

3.7.2 Annotations

Annotations are characterised by the fact that they mandatorily relate to other data – whether raw data, other annotations, or otherwise. In other words, they are ‘meaningless’ without context. To expand on our street light example: one detector may create annotations consisting of bounding boxes that identify the locations of street lights in camera images. Another detector might annotate *these* annotations, indicating the type of street light, or other higher-level properties. Metadata, such as the version of the algorithm used to generate the labels, might also be included.

This tree-like annotation hierarchy is encoded in a recursive table named Annotations:

$$\text{Annotations}(\underline{\text{annotation_id}}, \text{data_id}, \text{parent_id}) \quad (3.17)$$

The *data_id* field references a row in the Data relation, and the *parent_id* field references a lower-level annotation. One and only one of these fields must be non-

empty as it can either annotate some raw data or another annotation.

Annotation-specific data (for example, the bounding boxes from our street light detector) are then stored in their own tables. Records in these tables link to their corresponding records in the Annotations hierarchy table. For example, our street light annotation table may have the following schema:

$$\begin{aligned} \text{Lights}(\underline{id}, \text{annotation_id}, \text{bbox_x}, \text{bbox_y}, \\ \text{bbox_width}, \text{bbox_height}, \text{type}) \end{aligned} \quad (3.18)$$

Queries such as ‘find all raw data with a particular annotation θ ’ are simply a `SELECT` query over the three-way join of the Data, Annotations, and specific annotation data tables. For example, to get raw data for streetlights with a type θ :

$$\begin{aligned} &\text{SELECT Data.} \star \text{ FROM Lights} \\ &\text{JOIN Annotations ON Lights.annotation_id = Annotations.annotation_id} \\ &\text{JOIN Data ON Annotations.data_id = Data.data_id} \\ &\text{WHERE type} = \theta \end{aligned} \quad (3.19)$$

We can also query annotations by time by joining them with their respective raw data, for example, ‘find all lights that were found in data collected close to time t :’

$$\begin{aligned} &\text{SELECT Lights.} \star \text{ FROM Lights} \\ &\text{JOIN Annotations ON Lights.annotation_id = Annotations.annotation_id} \\ &\text{JOIN Data ON Annotations.data_id = Data.data_id} \\ &\text{WHERE ABS(timestamp} - t) \leq T \end{aligned} \quad (3.20)$$

Figure 3.6 shows graphically how annotations and raw data are related.

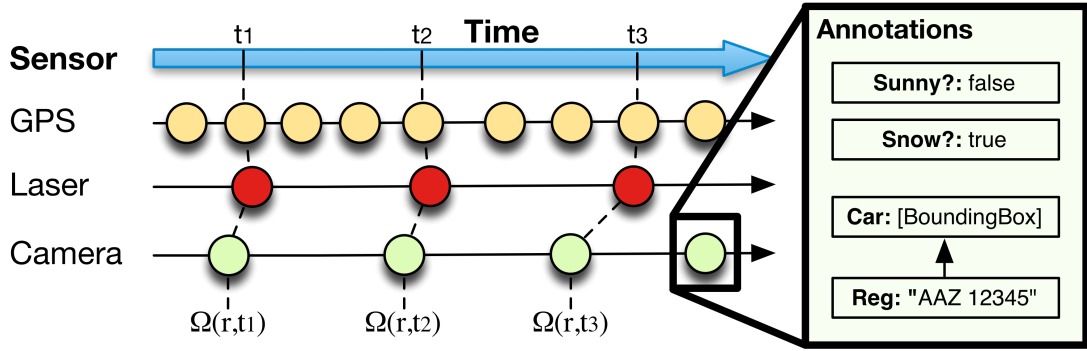


Figure 3.6: Visualisation of some raw GPS, laser, and camera data and the relative times they were captured. An example annotation hierarchy for a single image datum is shown – note that the ‘Car’ annotation has a child that contains the detected registration number. Links between data records that are temporally joined at times t_1 , t_2 , and t_3 are shown as dashed lines.

3.7.3 Multigraph Maps

One ubiquitous pattern that manifests itself time and again is the use of a directed multigraph¹⁸ structure (*GraphDB*) to represent maps for use by localisation and navigation systems, for example EBN. The graph is defined by two relations and is encoded in an edge-list representation, i.e., each row in the Edges relation corresponds to an edge that connects from one node to another. Two additional relations store links representing annotations for nodes and edges:

$$\text{Nodes}(\underline{\text{node_id}}) \quad (3.21)$$

$$\text{Edges}(\underline{\text{edge_id}}, \text{node_from_id}, \text{node_to_id}) \quad (3.22)$$

$$\text{NodeAnnotations}(\underline{\text{id}}, \text{node_id}, \text{data_id}, \text{annotation_id}) \quad (3.23)$$

$$\text{EdgeAnnotations}(\underline{\text{id}}, \text{edge_id}, \text{data_id}, \text{annotation_id}) \quad (3.24)$$

where the *data_id* and *annotation_id* fields are foreign keys referencing rows in the Data and Annotations tables respectively. One and only one of these must be non-

¹⁸A multigraph is permitted to have multiple, parallel edges connecting the same end nodes.

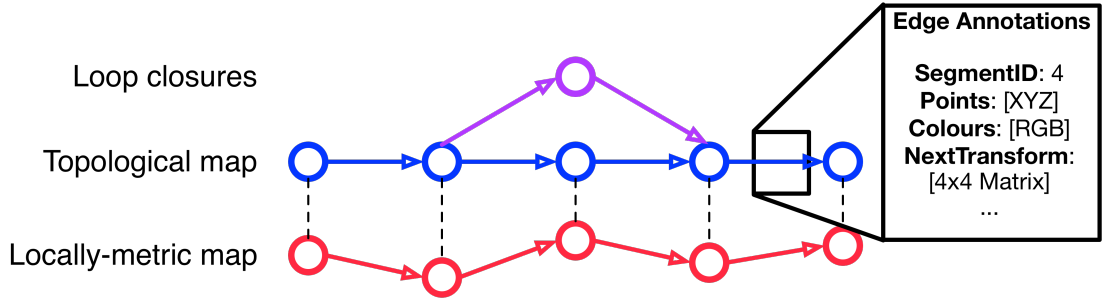


Figure 3.7: Example GraphDB structure representing a metric map, topological map, and loop closures. These types of map are explained previously in Section 2.4. Example annotations attached to a single edge are shown.

empty, allowing Node and EdgeAnnotations to refer to both raw data or existing annotations.

For example, to get a list of ids of nodes that succeed node n in the directed graph, we would do a simple `SELECT` query over the Edges table:

```
SELECT node_to_id FROM Edges WHERE node_from_id = n
```

(3.25)

To get specific annotations (e.g. lights) associated with an edge v , we select over the join of the Annotations, EdgeAnnotations, and Lights tables:

```
SELECT Lights.* FROM Lights
JOIN Annotations ON Lights.annotation_id = Annotations.annotation_id JOIN
EdgeAnnotations ON Annotations.annotation_id = EdgeAnnotations.annotation_id
WHERE edge_id = v
```

(3.26)

An example of how a map might be represented using this structure is shown in Figure 3.7.

3.7.4 Miscellaneous Data

This class encompasses any other arbitrary data that do not fall neatly into the two aforementioned categories. For example, it includes completely standalone data, such as OpenStreetMap maps and sensor calibration information. Harnessing the power of the relational model, these data can still link to raw sensor data and annotations if the need arises. For example, our street light annotations could include GPS locations and therefore implicitly spatially relate to OpenStreetMap region data. If we wanted to find all street lights in Oxford, we would do a spatial join on the StreetLights and OpenStreetMap Regions tables:

```
SELECT LightsGPS.★ FROM Lights
JOIN Regions ON ST_Within(LightsGPS.location, Regions.geometry)    (3.27)
WHERE Regions.name = 'Oxford'
```

where `ST_Within(A, B)` is a function that returns true if geometric object A is entirely contained within geometric object B. As another example, if we wanted to find all street lights within x metres of a GPS point p , we could execute the following query:

```
SELECT LightsGPS.★ FROM Lights WHERE ST_DWithin(location, p, x)    (3.28)
```

where `ST_DWithin(A, B, x)` is another function that returns true if geometric object A is within x metres of geometric object B.

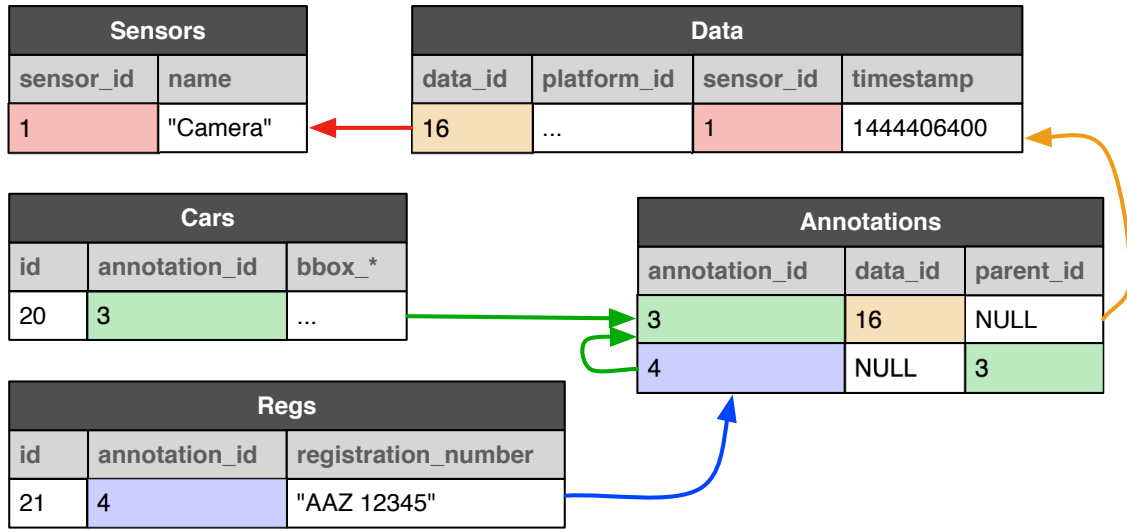


Figure 3.8: Example concrete table layout showing data corresponding to the ‘car’ annotation hierarchy shown in Figure 3.6. Foreign keys and the primary keys they reference are displayed with the same colour. Similarly coloured arrows point from the row containing the foreign key to the referenced row. Only a subset of the tables and data are shown – Figure 3.9 provides a more complete picture of the concrete database schema.

3.8 System Implementation

We choose to use either PostgreSQL¹⁹ or SQLite²⁰ as our RDBMS, depending on our particular use case. PostgreSQL is designed for a multi-user environment and has full transactional and locking abilities, allowing many concurrent connections to the database without the risk of data corruption. These guarantees come at the cost of performance – queries must be sent over pipe or network to the PostgreSQL server process which executes them and returns results over the same medium. For real-time, online applications, we therefore prefer SQLite which stores all necessary data in a single file. Applications access this file via a shared library, removing the requirement for costly inter-process or network communication.

We keep a subset of OpenStreetMap [33] consisting of all roads and regions in England. This is stored locally in our centralised PostgreSQL instance using the

¹⁹<http://www.postgresql.org/>

²⁰<http://www.sqlite.org/>

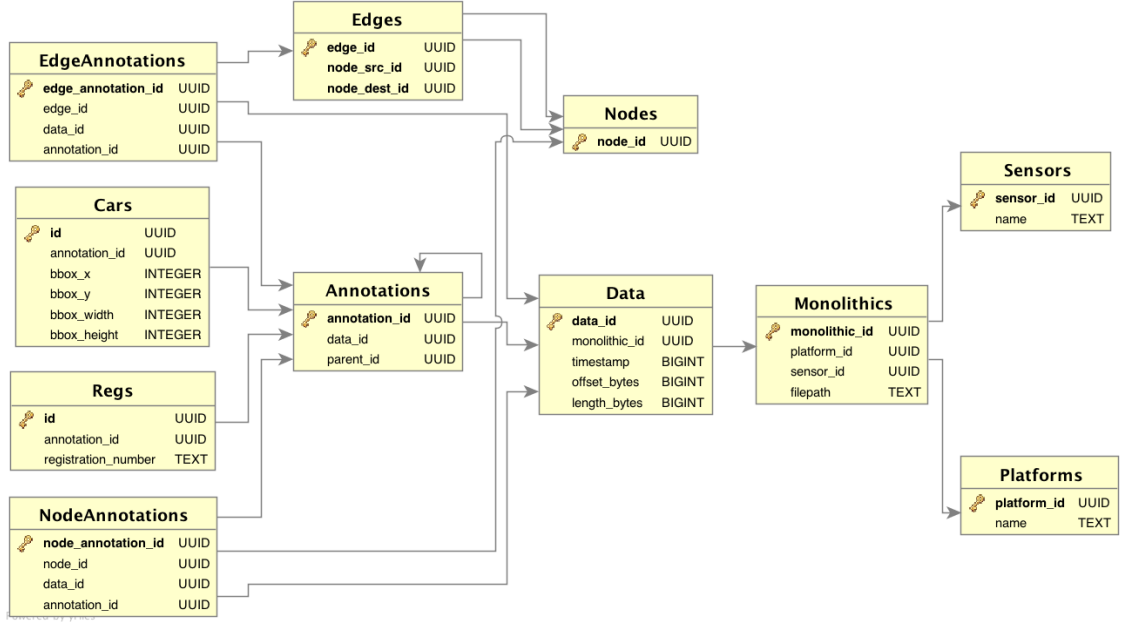


Figure 3.9: Diagram showing the database schema including the four ‘core’ tables: Annotations, Data, Platforms, and Sensors; the Monolithics table that provides links between the database and filesystem layers; the GraphDB tables: Nodes, Edges, NodeAnnotations, and EdgeAnnotations; and some example annotation tables representing those shown in Figure 3.6 and Figure 3.8: Cars and Regs.

PostGIS²¹ extension, which allows us to perform geospatial queries very efficiently. PostGIS implements the functions `ST_DWithin` and `ST_Within` that we used previously.

The full relational schema of the database system is shown in Figure 3.9.

3.8.1 Common Subqueries

Here we identify several common patterns in the `SELECT ... WHERE` conditions that can be seen in the SQL queries in the previous sections:

1. Equality conditions take the form `WHERE c = v` in SQL for some field c and scalar value v , for example in Equation (3.25). This is the simplest type of condition.

²¹<http://postgis.refrations.net/>

2. Temporal selections take the form `WHERE ABS( $c - t$ ) <  $T$` in SQL for some field c and timestamp t , for example in Equation (3.16). The threshold T can be set according to the needs of each individual application.
3. Spatial selections take the form `ST_Within( $A, B, x$ )` for some geometric objects A and B (e.g. regions, polylines, points), and distance x , for example as part of the spatial joins in Equation (3.27). This category includes related PostGIS functions like `ST_Within`.

As these types of selection operations are fundamental to the majority of queries we want to execute, we will specifically analyse their performance in Section 3.10.

3.8.2 Creating and Managing Maps for Online Use

One of the stated goals of our database system is to provide a centralised repository for all of our data. For many of the batch-style queries we might want to issue from our workstations while connected to a wired network, this makes sense. However, for a robot operating in the field with an unreliable or non-existent network link, it does not. It is also intractable for such a robot to carry with it a duplicate of the entire database. We therefore require a way to copy small subsets of the database, for example a map of a particular area, such that it can be stored and accessed locally on the robot.

3.8.2.1 Master-Clone Architecture

As a solution to this problem, we created a tool that takes as input a root node $\mathcal{R}$ of a GraphDB map and outputs a new *strongly connected* GraphDB containing only the nodes, edges, and data that are reachable from $\mathcal{R}$. The expected workflow of this is to store a master copy of every map in a central PostgreSQL database. Individual maps can be cloned to single SQLite files which can then be easily ‘xcopy’-deployed

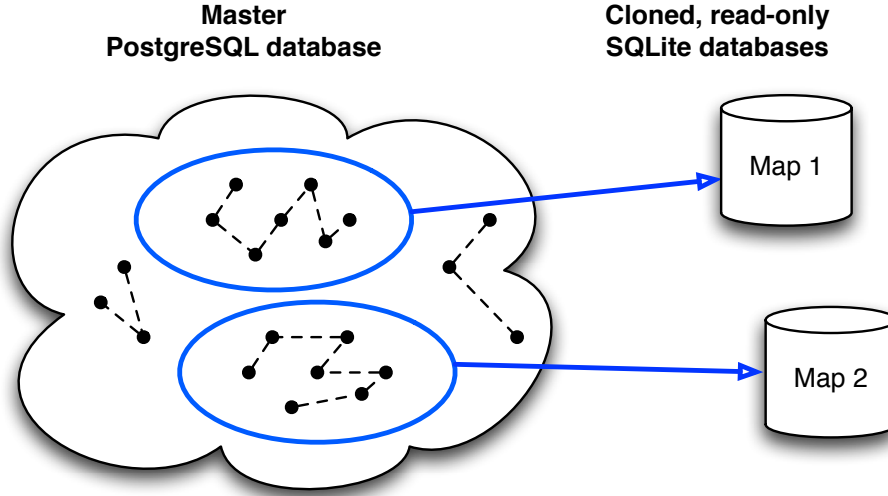


Figure 3.10: In our master-clone architecture, a master PostgreSQL instance holds the entirety of our data, including maps used for robot localisation. Because it is difficult to access this data from outside our network and impractical to host a copy locally on the robot, we have made it possible to clone small subsets of the GraphDB. This diagram shows two distinct and strongly connected (therefore necessarily non-overlapping) subsets of the GraphDB being cloned to their own instances. These are then compact enough to store locally on a robot. We typically disallow writes to these instances to prevent conflicts and confusion from arising.

(in addition to any referenced monolithic files) to a robot, as required. This is illustrated in Figure 3.10. Note that because references to monolithic files are stored in the database as relative paths, no changes to these paths need to be made so long as the relative directory structure is preserved during this copy.

3.8.2.2 Implementation Considerations

Having data duplicated across multiple machines raises the question of whether we should deal with synchronisation, and if so, how. Although it is beyond the scope of this thesis to develop a fully distributed version control system, complete with diffs, merges, branches, and conflict resolution, in certain circumstances we are able to support a subset of these tasks. Nonetheless, we should ideally make design decisions that do not preclude such future improvements.

Perhaps the most important design consideration is our method of generating

primary key IDs (e.g. the `Data.data_id` column). Recall that a record's primary key is supposed to uniquely identify that record: it is invalid to have two with the same ID. When this happens, it is known as a *collision*. Collisions introduce undesirable ambiguity between records – for example we may not be able to distinguish between two sections of road in our map.

By default, DBMS's such as PostgreSQL and SQLite support the use of an *auto-incrementing* numeric type for primary keys, typically a 32 or 64-bit unsigned integer. Each newly created record is assigned an ID that is simply the next greatest available integer, i.e. records 1, 2, 3 will respectively have IDs 1, 2, 3. This is enough to guarantee uniqueness only for this table in this particular database instance. Complications arise when we attempt to merge two such databases, each having records with IDs 1, 2, 3. One database will need to have its IDs remapped to a new range of values, for example 4, 5, 6. However this alone is not enough – every corresponding foreign key reference in other tables will also need to be updated, otherwise these will now point to completely different records. Worse, if these IDs were stored anywhere outside of the database (e.g. in a configuration file) they will need to be updated there as well. This remapping process is clearly error-prone and unscalable.

One approach to solve this issue is to generate new IDs randomly instead of deterministically. This would allow us to support trivial merges of disjoint maps that contain no intersecting IDs that might conflict with one another, and hence no existing IDs or references would need to be updated. If the space of possible IDs (*key space*) is large enough, then collisions would be extremely unlikely. Calculating the probability of such a collision occurring is best known as the *birthday problem*²² which can be reformulated as:

²²The birthday problem is believed to have been first proposed by Richard von Mises in 1939.

Theorem 1: *The probability $P(n, k)$ of a collision occurring in a set of n keys drawn randomly from a key space of size k is:*

$$P(n, k) = 1 - \frac{k!}{k^n(k - n)!} \quad (3.29)$$

where $n < k$. When k is large, Equation (3.29) can be approximated as:

$$P(n, k) \approx 1 - e^{-\frac{n^2}{2k}} \quad (3.30)$$

In an attempt to find a reasonable value for k , we consider a ‘conservative’ value of $n = 5 \times 10^8$. Recall from Figure 3.3 that this is the same order of magnitude as the number of individual data records we currently manage, or in terms of our EBN use case is approximately half a million kilometres worth of stereo camera keyframes – equivalent to the entire road networks of Great Britain and Ireland²³ mapped just once. If our key size is 32-bits, i.e. $k = 2^{32}$, then Equation (3.30) gives us (up to machine precision) $P(n, k) \approx 1$. These keys are clearly too small even for relatively modest n . When $k = 2^{64}$ then $P(n, k) \approx 0.00675$. At first glance this might appear sufficiently unlikely, however there is deceptively little room for growth: a tenfold increase in n results in an almost 50% probability of collision. Trying $k = 2^{128}$, conveniently the size of most implementations of *universally unique identifiers* (UUIDs), gives us (up to machine precision) $P(n, k) \approx 0$. This is likely to be suitable for the foreseeable future – it would require n as large as 8.3×10^{17} before we have even a 0.1% probability of collision.²⁴

We therefore opt for 128-bit UUIDs as our primary key datatype of choice.

²³<https://www.cia.gov/library/publications/the-world-factbook/rankorder/2085rank.html>

²⁴In EBN terms, on the order of fully mapping every road in the world, every day, for the next 35,000 years.

```

1 // Connect to an SQLite database file.
2 DatabaseClient db;
3 boost::filesystem::path path("/Users/peterdn/test.db");
4 db.Connect(path);
5
6 // Create two nodes.
7 NodeID node1 = db.CreateNode();
8 NodeID node2 = db.CreateNode();
9
10 // Create an edge linking node1 -> node2.
11 EdgeID edge = db.CreateEdge(node1, node2);
12
13 // Annotate edge with a Google protocol buffer message.
14 db.CreateNodeAnnotation<pbTuple3d>(edge, ANNOTATION_NAME,
15     ANNOTATION_DATA);
16
17 // Read annotation data.
18 std::vector<boost::shared_ptr<pbTuple3d > > edge_data;
19 db.GetEdgeData<pbTuple3d>(edge, ANNOTATION_NAME, edge_data);

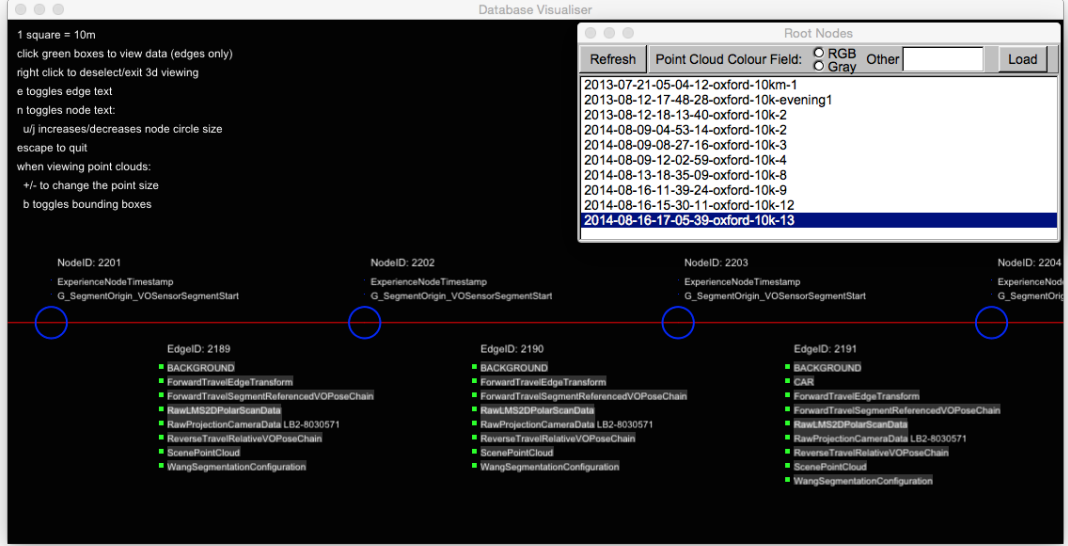
```

Listing 3.1: Simple example showing usage of the GraphDB API in C++ to create some nodes, an edge linking them, and creating and reading annotation data. Some boilerplate code has not been shown.

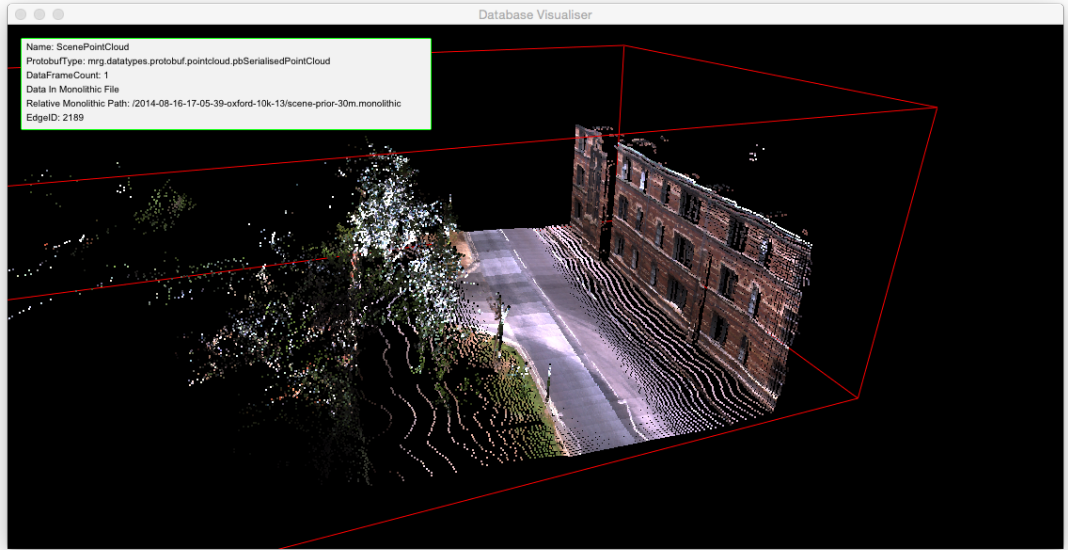
3.9 Application APIs

Both C++ and Java/MATLAB libraries have been implemented which provide common APIs that simplify access to the underlying database. Although SQL is the ‘native’ language of both PostgreSQL and SQLite, we prefer that users use only the aforementioned libraries where possible. Additional tools have been created within our research group to further simplify maintenance and management tasks, for example the database visualisation application shown in Figure 3.11.

Our GraphDB C++ library provides methods that support all standard graph operations such as creating and deleting nodes and edges, and traversing the graph structure. It also supports adding annotation data to nodes and edges as well as reading the corresponding data back from them. This way, users of the library do not need to write a single line of SQL themselves. A toy C++ example is shown in Listing 3.1.



(a)



(b)

Figure 3.11: Screenshots of the MRG database visualiser application. This allows users to view the nodes, edges, and annotations of a GraphDB (top), as well as to inspect and visualise annotation data (below). Pictured is a coloured point cloud segment *scene prior* [82] – see the use case in Section 3.12.1 for more details.

Query	Execution Time
Insert approx. 250,000,000 index records	12 hours approx.
Select single datum by ID	0.06ms
Select single datum by platform, sensor, and timestamp	0.06ms
Temporal join to select data by platform and timestamp	7.5ms
Temporal join for every record in an approx. 60GB dataset	14.7s

Table 3.2: Measured mean execution times for some example queries. Performed using a test PostgreSQL instance containing a 250,000,000-record subset of our data (approximately 20TB).

The performance of our framework is analysed in the following section, and a selection of motivating use cases are presented.

3.10 Optimising Performance with Indexes

The importance we ascribe to run-time performance of the database system depends on the specific use case in question. For example, batch tasks that process our entire 90TB of data have very different performance expectations to those of navigation systems running on live robots.

Both PostgreSQL and SQLite support indexes on data tables which we use to improve the performance of certain queries. An index is a data structure that stores a copy of a particular column’s data, typically in some order that enables efficient searches on that data. The physical address of the entire corresponding record is also stored so it can be retrieved quickly. Indexes are often implemented using well known B-Tree [3], R-Tree [32], and hash table data structures. For example, we create a B-Tree index on the *timestamp* field in the Data table. This reduces the search for a record identified by a specific *timestamp* from $O(n)$ to $O(\log n)$. For most long-running batch tasks, the speedups afforded by these indexes are sufficient. Table 3.2 shows measured execution times for some example queries of this nature.

In the rest of this section we analyse some of the performance characteristics of PostgreSQL under particular configurations using the common selection subqueries

we identified in Section 3.8.1.

All experiments in this chapter were performed on an early 2011-era MacBook pro with 8GB RAM and a 500GB SSD, with a PostgreSQL server running locally.

3.10.1 Equality Selections

Recall from Section 3.8.1 that equality selections are those with conditions of the form `WHERE  $c = v$` in SQL for some field c and scalar value v . We evaluate the performance implications of hash table and B-Tree indexes on some synthetic data for these types of query. Following our decision in Section 3.8.2.2 to use 128-bit UUIDs as our ID fields, we fill a table with N randomly generated UUIDs. We then attempt to `SELECT` a known existing UUID from the table and record the execution time. The resulting execution times with the aforementioned index types, and with no index, are shown in Figure 3.12a.

The timing results are consistent with our understanding that in the average case, hash tables have $O(1)$ search time, B-Trees, $O(\log(N))$, and with no index our only recourse is a slow $O(N)$ sequential scan through the entire table.

3.10.2 Temporal Selections

Recall that temporal (or inequality) selections are those with conditions of the form `WHERE  $\text{ABS}(c - t) < T$` in SQL for some field c , timestamp t , and threshold T . We fill a table with N randomly generated 64-bit timestamps (representing microseconds since midnight 1st January 1970) distributed between 1st June 2011 and 1st June 2015. We then attempt to `SELECT` all timestamps falling within a 1s interval in the middle of this distribution. As before, execution times were recorded for varying values of N and with different index types, and are shown in Figure 3.12b.

These results are consistent with the fact that a hash table does not preserve

ordering and therefore is useless for resolving inequality conditions – it must fall back to a sequential scan of every record. On the other hand, the leaves of a B-Tree are sorted and therefore the inequality condition resolves to a contiguous subset of the leaves.

3.10.3 Spatial Selections

Recall that spatial selections are those with conditions of the form `ST_Within(A, B, x)` for some geometric objects A and B (e.g. regions, polylines, points), and distance x . We fill a table with N (latitude, longitude) coordinates recorded by GPS from around the roads of Oxford. We then attempt to `SELECT` all records that lie within an 80m radius of a particular intersection (chosen such that most of our data collection routes pass through it). Execution times comparing R-Tree index performance against no index, for varying values of N , are shown in Figure 3.12c.

Again, the results confirm our understanding that an R-Tree index has average case $O(\log(N))$ search performance for 2D spatial data, and without an index, sequentially scanning though all records in $O(N)$ time is the only option.

3.10.4 Performance Tradeoffs

The downside to indexes—and the reason we don’t simply index *every* column in *every* table—is the decreased performance of inserting, modifying, and deleting data. This is because the index on disk must be modified as well as the actual data files. Therefore, only columns that often participate as a selection or join condition should be indexed. For our system, this includes every table’s primary key column, as well as foreign keys (such as `Annotations.data_id`), temporal fields (such as `Data.timestamp`), and any annotation columns that store (latitude, longitude) coordinates, or other spatial data.

3.10 Optimising Performance with Indexes

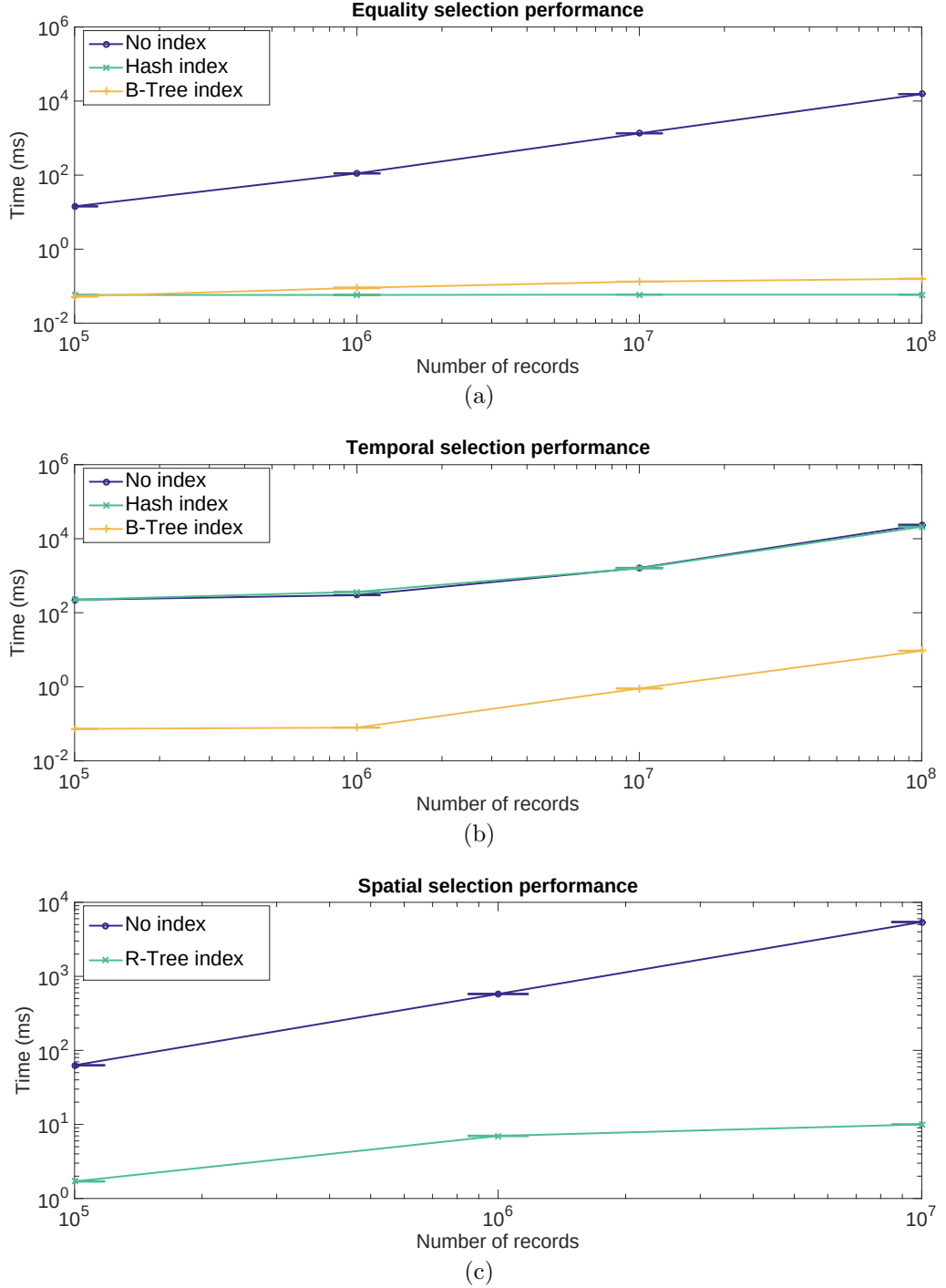


Figure 3.12: Log-log plots showing time taken to execute queries with different types of selection conditions: equality (i.e. of the form $c = v$), temporal (inequality; i.e. of the form $c \approx t$), and spatial (i.e. of the form $c \approx r$ for some geometric entity r). Various indexing strategies are compared against a no-index baseline in each case. Each experiment was repeated 5 times and the mean was taken. Error bars, where visible, show the maximum and minimum times taken (although due to the constant seek time of an SSD, there is unsurprisingly little variation).

3.11 PostgreSQL vs SQLite

We now evaluate our choice of DBMS back-end. PostgreSQL is a server-based DBMS, meaning queries are issued via relatively costly interprocess communication or network calls. In return, it provides full transactional and fault-tolerant behaviour and therefore allows many users to read and write data simultaneously without interfering with each other.

For real-time use cases, we instead prefer SQLite. It is linked statically and stores all of its data, metadata, and indexes in a single file which vastly reduces the overhead of its API calls.

It's worth noting that as the large monolithic files (containing raw data) are stored independently of the DBMS, directly on the filesystem, the actual DBMS files are relatively small (depending on the exact use case). For example, a scene prior [82] database which references coloured point cloud monolithics, in addition to storing serialised transforms between sub-maps, consumes on the order of 5-10MB per kilometre (the referenced monolithics themselves are approximately 400-500MB per kilometre).

Here we give performance results for the creation and traversal of a 'worst case' linked list graph structure. Although somewhat contrived, this example allows us to identify certain bottlenecks and observe how different configurations perform relative to one another. In the real world, access patterns and disk speed would also be major performance factors.

3.11.1 Experiments and Analysis

To test write performance, we create linear chains (successive nodes connected by single edges) of varying lengths n in the GraphDB. PostgreSQL and SQLite are tested in both *synchronous* (every write is fully flushed to disk) and *buffered* (writes

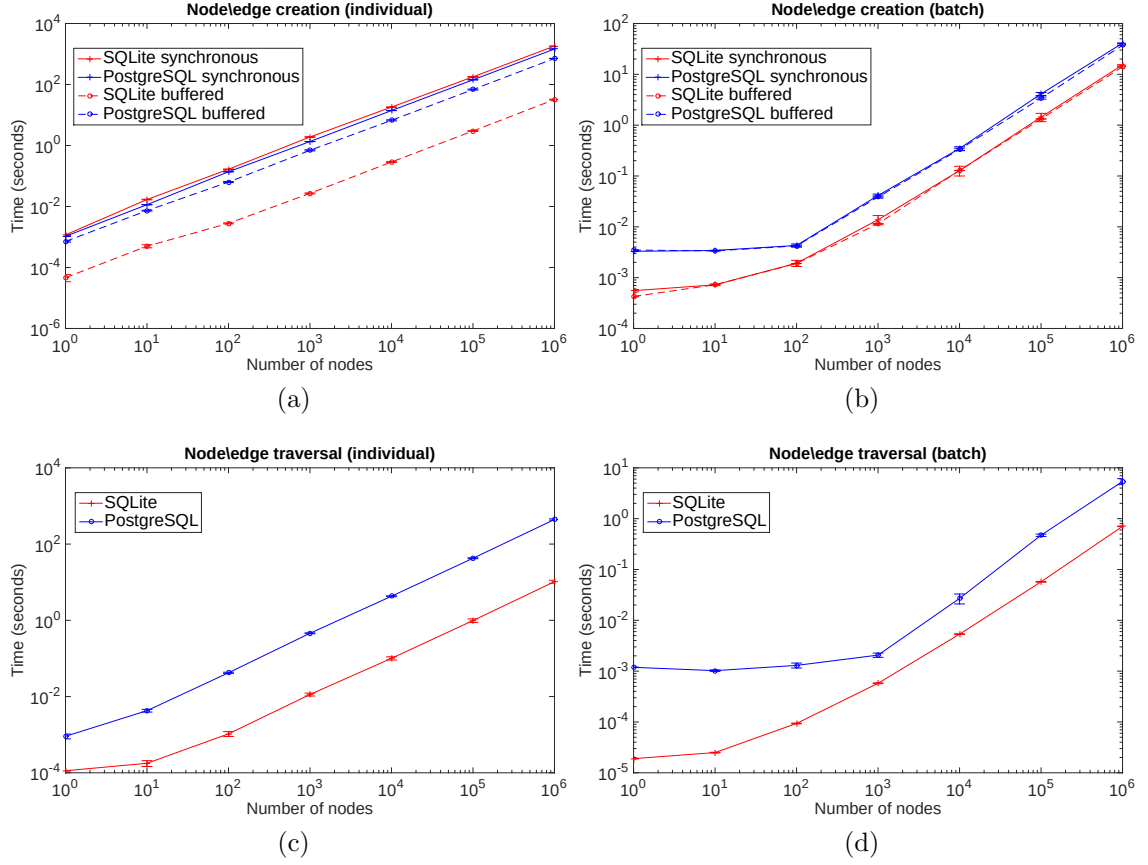


Figure 3.13: Figures 3.13a and 3.13b show log-log plots showing time taken to create linear chains of various lengths in PostgreSQL and SQLite, in both synchronous and buffered modes. On the left, nodes were all created individually – i.e. n calls to a `CreateNode` API function. On the right, nodes were created in batch using a single API call. Figures 3.13c and 3.13d show log-log plots showing time taken to read linear chains of various lengths in both PostgreSQL and SQLite. On the left, nodes were all read individually – i.e. n calls to a `GetNextNode` API function. On the right, the entire chain was read in batch using a single API call. Each experiment was repeated 5 times and the mean was taken. Error bars, where visible, show the maximum and minimum times taken.

are buffered in memory by the host operating system) modes. In the synchronous case, a write-ahead log²⁵ is also used. In addition, we compare the creation of all nodes and edges in one operation (*batch*) versus creating them individually. The former case is more likely to be the behaviour of an offline batch task that has all data already available to it, whereas the latter behaviour is more likely to be exhibited by an online task that is constantly processing new data from its environment.

Timing results for write operations are shown in Figures 3.13a and 3.13b. Unsurprisingly, buffering individual writes is faster for both PostgreSQL and SQLite, although much more dramatically for the latter. Buffering batch writes makes virtually no difference. Batch writes of more than about 10-100 nodes are significantly faster than writing them individually. This is likely due to the overhead of API calls which begins to dominate in longer chains, particularly for PostgreSQL.

To test read performance, we load the previously created linear chains. Again we test how reading the whole graph in one operation (*batch*) compares with traversing the chain one node at a time.

Timing results for read operations are shown in Figures 3.13c and 3.13d. Similarly we see that reading in batch is far quicker than traversing nodes individually. Additionally, SQLite is faster than PostgreSQL by almost 1-2 orders of magnitude in all cases.

Overall, SQLite consistently outperforms PostgreSQL, although the difference is less marked in batch than individual cases. We believe this is because of the overhead associated with interprocess communication return trips, of which there are $O(1)$ in the batch case compared with $O(n)$ individually. We use this as justification for our use of PostgreSQL as our centralised DBMS, suitable for use with long-running offline tasks—where the multi-user and fault tolerance properties outweigh slight

²⁵Write-ahead logging involves keeping a record of every modification before it is performed. In the event of catastrophic program or system failure, the log can be used to restore data to a consistent state.

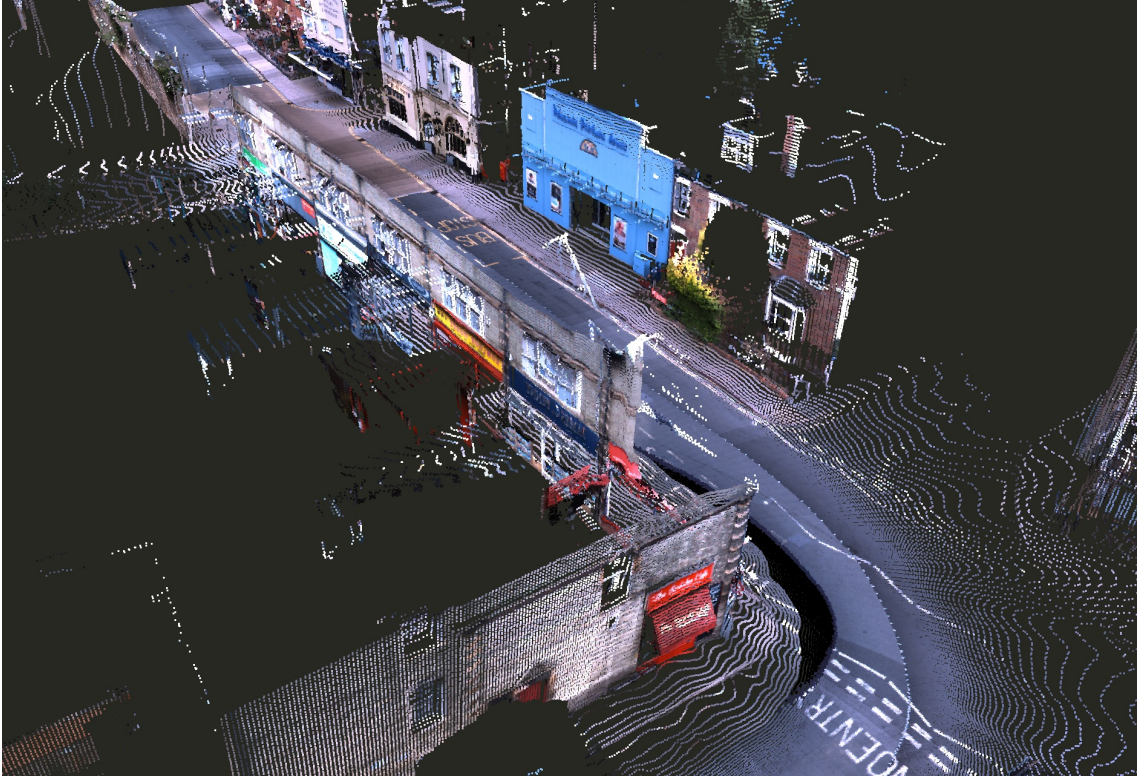


Figure 3.14: Example scene prior coloured point cloud ‘segment’, showing a section of Walton Street in Oxford. This point cloud data is stored in a monolithic file and referenced from an edge annotation in the scene prior graph database. Figure reproduced with permission from [82].

performance gains—and SQLite for real-time tasks on robots.

3.12 Example Use Cases

We now present a selection of use cases that demonstrate both the offline batch data mining capabilities of our system, as well as its ability to run real-time as the map storage and retrieval component of a vast-scale localisation system.

3.12.1 Scene Priors

A *scene prior* is a map format detailed by Stewart in [82]. Being a topometric representation, the map is structured as a collection of locally-metric sub-maps

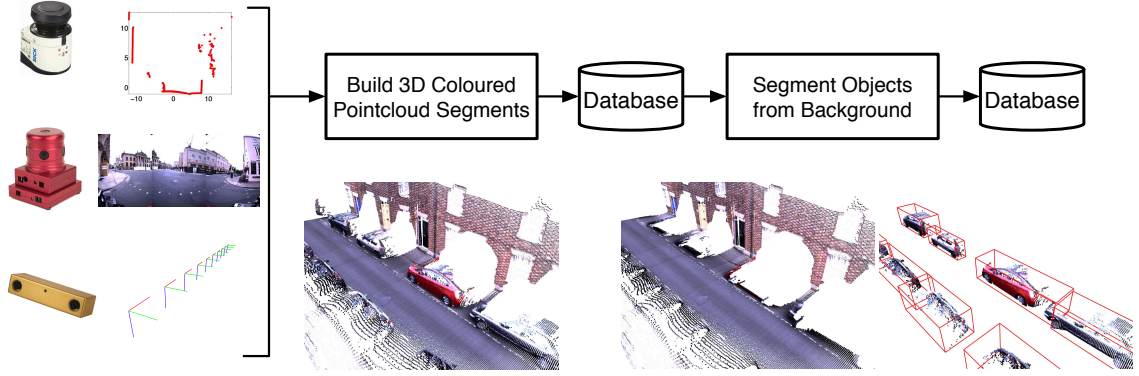


Figure 3.15: Pipeline used to build scene priors, as described in [82]. From left to right, 2D laser scans, camera images, and visual odometry are fused to generate coloured point cloud segments. These are written as annotations on top of a graph structure to our database. As a post-processing step, dynamic obstacles (in this case cars) are detected and segmented from the point cloud. These are then written back to the database as additional annotations. Figure reproduced with permission from [82].

connected together via a directed graph. Edges in the graph contain these sub-maps which are metric 3D coloured point clouds. Nodes correspond to specific visual odometry frames, representing the coordinate frame in which point clouds in emanating edges are defined. In [82], nodes are spaced approximately 30m apart along the map route, meaning each locally-metric point cloud ‘segment’ is around 30m in length. An example segment is shown in Figure 3.14.

The scene prior graph structure is stored in GraphDB. Nodes and edges are annotated with 6-DoF visual odometry transforms and coloured point cloud data, respectively. The underlying point cloud data is stored in monolithic files and the annotations index into these monolithics via the Data table. Using our database framework in this way makes it easy for users to build, distribute, and query scene priors. It is straightforward for users to add their own data and annotations: for example, Stewart further demonstrates dynamic obstacle segmentation where coloured point clouds are post-processed to detect and remove cars. These are then added back to the database as annotations. This pipeline is shown in Figure 3.15.

Additionally, Stewart demonstrates a system that localises within these scene

prior maps (named *LAPS*), successfully testing it on approximately 57km of real data collected around Oxford (representing approximately 8 hours of driving). The database is queried in real-time for new edges when the vehicle crosses into their corresponding sub-maps, resulting in execution of a single query every few seconds.

3.12.2 Scene Variation

Dealing with the challenges inherent in drastic appearance change requires large quantities of images of the same place, under different appearance conditions. In our research group this includes our Experience-Based Navigation [12], [48] and Experience-Based Classification [37] work.

With the PostGIS geospatial extension for PostgreSQL we are able to very quickly find and group images by corresponding GPS location. As an example query, we require stereo images taken near (i.e. ± 5 metres) 25 key points, $\mathcal{P}$, around the Begbroke Science Park near Oxford, captured only on counter-clockwise traversals of the loop (to ensure consistent camera orientation). These points are shown as markers in Figure 3.16. INS data at these points were quantised into cardinal directions north, east, south and west, and we defined a mapping from points $p \in \mathcal{P}$ to direction d via $\mathcal{D}$. These quantised data were then added as annotations to the GPS raw data. For brevity we express the query we wish to execute in pseudocode in Listing 3.2 – the underlying SQL was built dynamically, consists of multiple joins and unions of subqueries and so is too large to give in full.

3.12.2.1 Results

The query completed in around four minutes, producing approximately 6GB of output images. Some of these are shown in Figure 3.17 – images that were taken at very nearly the same position and orientation, but months apart. It is difficult to imagine how one might achieve these results by hand in the same length of time.


```

1 stereo_images = {}
2 FOR EACH  $p \in \mathcal{P}$  DO
3   (times, platforms) = SELECT timestamps, platforms of GPS data
4     captured within 5m of  $p$  while moving in direction  $\mathcal{D}(p)$ 
5   FOR EACH  $(t, r) \in (\text{times}, \text{platforms})$  DO
6     stereo_image = SELECT stereo data captured by platform  $r$ 
7       closest to time  $t$ 
8     IF stereo_image  $\notin$  stereo_images THEN
9       add stereo_image to stereo_images
10 RETURN stereo_images

```

Listing 3.2: Pseudocode query for the selection of all images taken at specific locations.

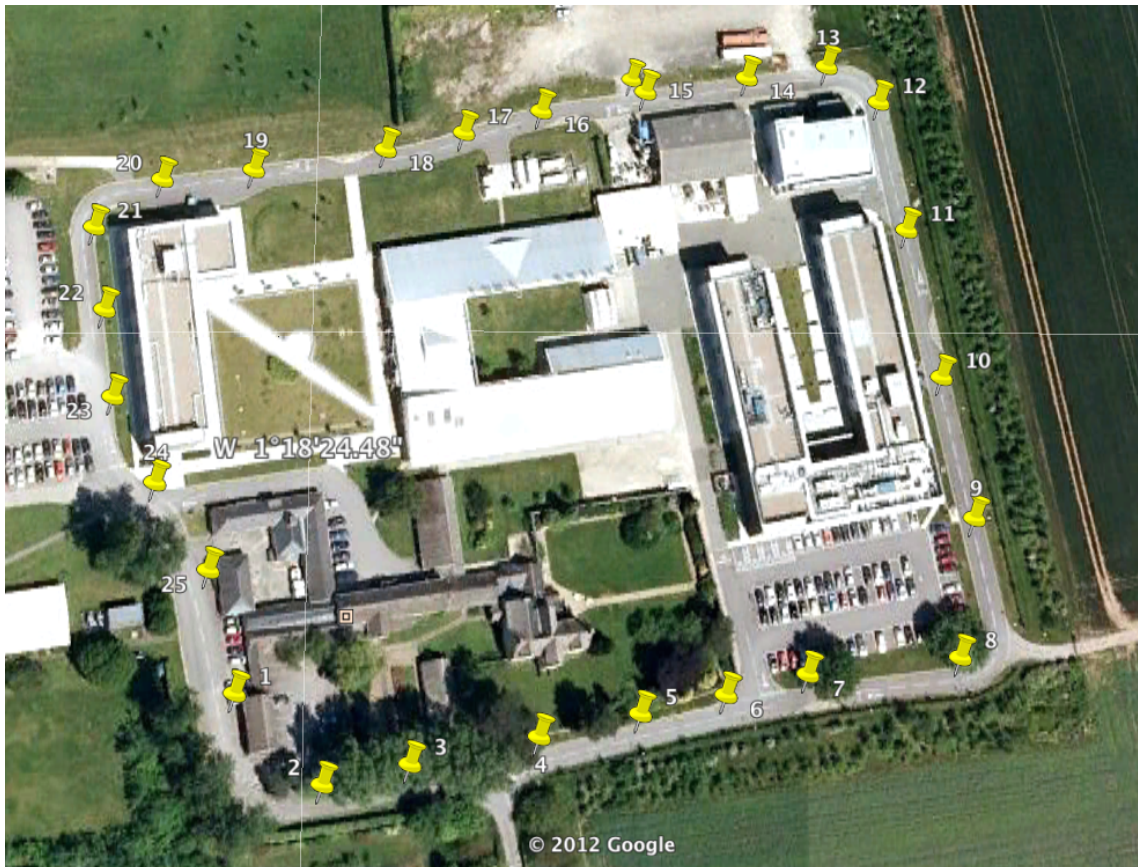


Figure 3.16: Points around the Begbroke Science Park at which to find stereo images. Imagery © 2015 Infoterra Ltd & Bluesky. Map data © 2015 Google.



(a) Place 1, travelling south.



(b) Place 6, travelling east.



(c) Place 11, travelling south.



(d) Place 17, travelling west.

Figure 3.17: Examples from the output of the query, described in Section 3.12.2, to find stereo images captured at particular points around the Begbroke Science Park. Images from four of the places are shown, spanning multiple seasons.

3.12.3 Experience Based Navigation

Experience-Based Navigation (EBN) [12] is a general-purpose framework for vast-scale, lifelong visual navigation. Outdoor environments exhibit challenging appearance change as the result of variation in lighting, weather, and season. EBN models the world using a series of overlapping ‘experiences’, where each experience models the world under particular conditions. New experiences are added in real-time when the robot is unable to sufficiently localise live camera images in the map of experiences. Additionally, the robot learns from its previous use of the experience map in order to more accurately retrieve relevant experiences from memory at runtime. The system has been tested on datasets totalling 206km of travel (representing approximately 25 hours of driving), successfully running in real-time at 20Hz [48]. This was extended to 1060km in total (representing over 150 hours of driving) in [47].

Since EBN must operate over vast distances and throughout the lifetime of a robot, it is essential that the map of experiences is persisted in long-term storage. In order to maintain reasonable memory requirements, the system needs to be able to selectively load relevant portions of the map into memory, while leaving others on disk. To meet real-time constraints, these data must also be efficiently accessible.

The GraphDB meets these requirements and therefore provides the mechanism for storing and retrieving data. Raw sensor data (e.g. images) and processed data (e.g. visual landmarks, FABMAP [17] bags of words) are linked to corresponding nodes as annotations. Edges are annotated with 6-DoF relative transformations, giving the graph a local metric structure.

The pipeline for localisation is shown in Figure 3.18. The breadth-first search enables EBN to load relevant portions of the map (nodes nearby the robot) into memory at runtime where they may be cached for later use. Since this search may require a large number of read requests (on the order of 10,000 per second, 90%+ of which will be read from cache), the SQLite implementation of the GraphDB is

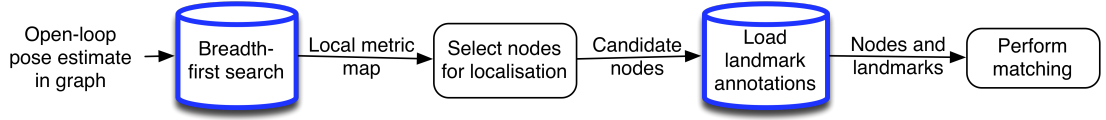


Figure 3.18: Overview of the EBN localisation pipeline. This process is strictly required to run at 20Hz – the system has 50ms to complete, including data access tasks (highlighted in blue).

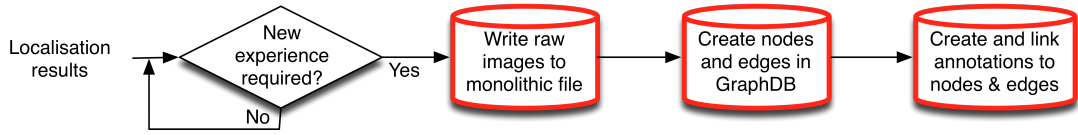


Figure 3.19: Overview of the EBN experience creation pipeline. This process has less strict runtime requirements and runs in a background thread. Writes to the database are highlighted in red.

used in buffered mode to maintain real-time performance.

A core feature of EBN is the ability to add new experiences to the map in real-time (pipeline shown in Figure 3.19). These experience creation tasks are processed in an independent thread so that slow disk write speeds do not impact on real-time performance. Figure 3.20 shows timing measurements for both the localisation and experience creation pipelines.

3.13 Summary

In this chapter we have motivated the need to move away from privileged datasets and ad-hoc data storage and annotation. These lack-lustre habits do not allow us to fully utilise our resources. Data take valuable time to collect and annotate and we make the case that it is impossible to predict which will be useful as they are collected. It is often only after the fact—weeks or months later—that we realise the full potential of some data. Without a consistent framework like ours in place, we would likely forget these useful data, not to mention waste time reimplementing

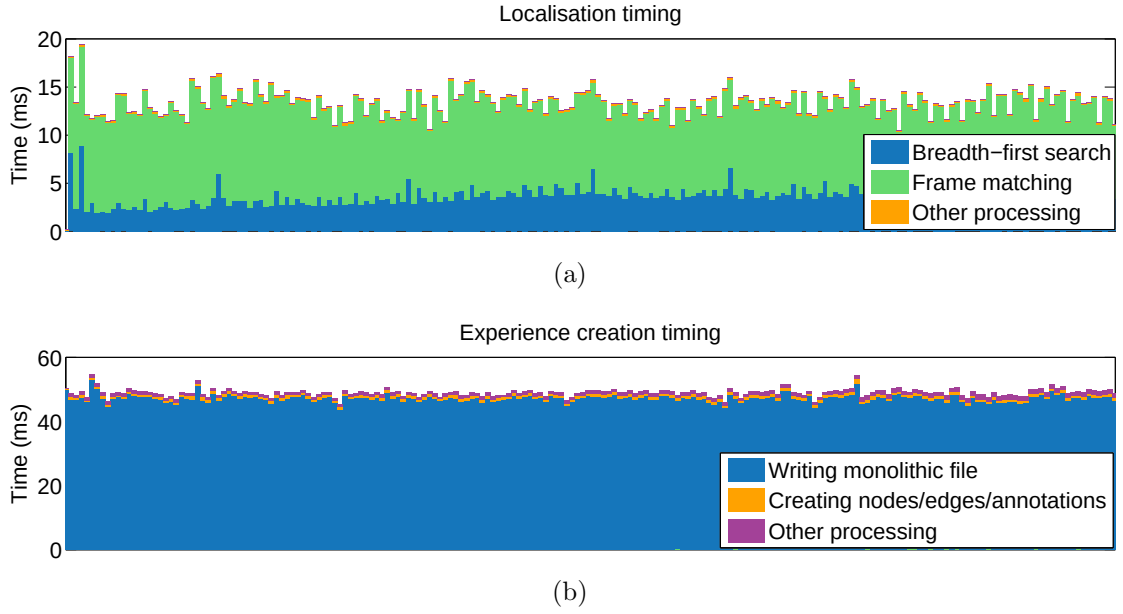


Figure 3.20: Sampled timing measurements for the localisation and experience creation pipelines. Dominant database access periods are shown in blue. It can be seen that during localisation, these accesses comfortably meet the 50ms target. Experience creation takes in general longer however still runs under 50ms for majority of the time. For this task, occasional spikes in disk access latency do not impact on real-time performance as it is handled in an independent thread.

separate storage and retrieval back-ends. We firmly believe that a system like this is the way forward for robotics applications – text files, READMEs and wikis are no longer sufficient for many of our data management needs.

We have also presented an analysis of the performance characteristics of the database system. We verify that no single DBMS back-end is ideal for *both* online and offline tasks: hence our decision to use a centralised master PostgreSQL instance for the latter, and lightweight SQLite clones for the former. We have shown both of these in action on a selection of real-world use cases, including several real-time localisation systems that have been tested on over 1000km of data combined.

We are now in a position to make use of this framework in solving the problem of localising a vehicle at night. It means we do not need to worry about how to store or retrieve maps, and instead can focus on the challenges that make navigation at night difficult.

Chapter 4

Navigation at Night

“ *What, so everyone’s supposed to sleep every single night now?*
You realise that night time makes up half of all time? ”

Rick Sanchez, *Rick and Morty*, S01E01 “Pilot”

4.1 Introduction

In this chapter we combine the visual mapping and localisation algorithms described in Chapter 2, with our database framework developed in Chapter 3 to implement a system that enables an autonomous vehicle to navigate in urban environments at night. Our database system gives us the ability to store and retrieve maps; now we need the ability to build maps and localise within them.

We present a teach and repeat system that: (i) automatically builds a map of artificial light sources from an offline sequence of images; and (ii) detects and matches observations of lights in an online sequence in order to localise within this map. Additionally, we take advantage of how the appearance of bright lights often

changes radically as their distance from the camera varies. By modelling how a light’s appearance changes as a function of camera location during the map building process, we can refer back to these place-dependent models in order to improve data association and better inform our pose estimate. We demonstrate that this approach allows our system to localise under conditions where a conventional point feature based teach-and-repeat system fails.

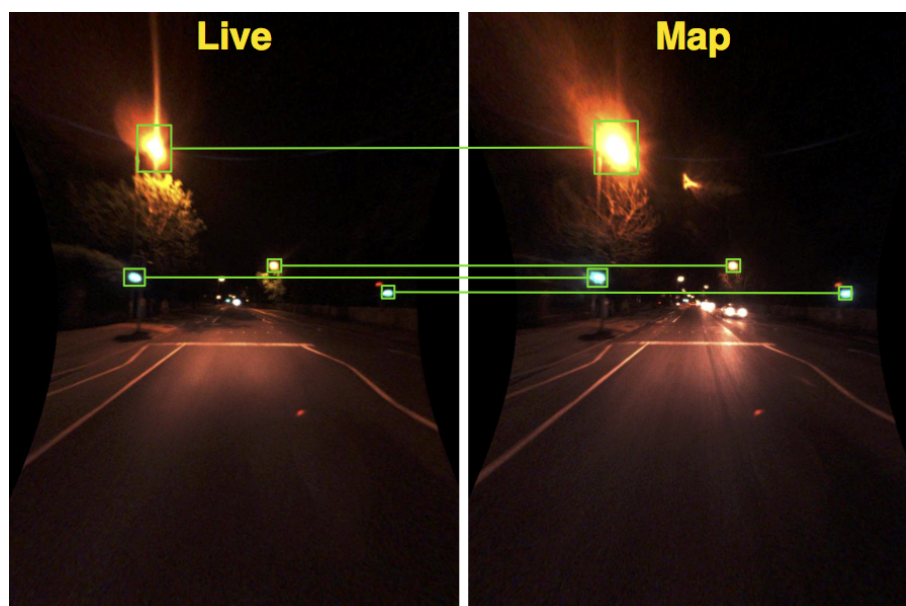
4.2 Motivation

An ongoing challenge for autonomous vehicles is the problem of localisation: knowing where the vehicle is positioned relative to a map of its environment. We addressed this problem in general terms in Section 2.5.2. As it is dark exactly half of the time, some consideration must be given to the challenges posed under these conditions. Localisation has been successfully demonstrated many times using active sensors such as lidar [51, 31], radar [14], and sonar [45, 22]. These sensors, which emit their own radiation, are largely invariant to environmental lighting conditions and thus operate normally in darkness. The solution to these challenges using passive vision sensors, however, remains an open question. The vast majority of research to date has only tackled the problem in daylight or under similarly well-lit conditions such as indoors.

Unfortunately, conventional cameras typically perform poorly at night, often lacking the dynamic range required to capture scenes which can exhibit large luminance ranges when bright lights are present in otherwise total darkness. Figure 4.1 shows some examples of the degraded quality of imagery we must contend with at night. At this point we have a dilemma: either increase camera exposure time (and/or sensitivity) in order to better resolve poorly lit background, or decrease exposure time in order to reduce saturation and bleeding caused by light sources.



(a) A traditional point feature based system struggles to localise a live stream of images against its map under night-time conditions. Data association in particular is made difficult by effects such as significant motion blur, saturation, lens flaring, lack of well defined edges, and overall lack of visibility – most of the scene is pitch black.



(b) By specifically detecting and matching lights in the live scene with those in a map, we are able to successfully localise. In addition to position in image space, we also take into account the expected appearance of each light, based on how far away it is. This greatly improves the robustness of our data association and serves to better inform our pose estimate.

Figure 4.1: A visual comparison of how localisation at night fares using traditional, small point features versus permanent, static light sources.

These compromises can increase noise or cause blurring, and subsequently decrease information in an image. Choosing the best strategy depends on the task at hand. For example, [62] uses the former approach to localise along dark rural roads while [70] uses the latter to detect car rear lamps. These problems can be avoided by using cameras that are designed specifically for use at night, however these are more expensive than the consumer variety and often have limited use in daylight. As one of our aims is to achieve cost-effective autonomy using vision, we eschew these alternatives in favour of low-cost conventional cameras.

A consequence of increased noise in images is that low-level point features—widely used in visual mapping and pose estimation systems—are not as effective as they would be under optimal lighting conditions. Fortunately, many urban areas are brightly illuminated at night by artificial light sources of all kinds – street lights, traffic lights, road signs, billboards, and windows. In images captured with a conventional camera using typical exposure and shutter settings, these appear as large saturated blobs, and hence on first consideration would appear to be easy to detect and track. Most importantly, these lights are often permanent, static, and in many cases are specifically designed to be easily visible and distinguishable from the road. Therefore, they make suitable candidate landmarks of which a localisation system can take advantage. We must also however contend with dynamic light sources, for example headlamps of oncoming vehicles, reflections, and occasionally, celestial objects.

Figure 4.1 shows some examples of nighttime imagery and gives a visual introduction to how artificial light sources are utilised in our localisation system.

4.3 Related Work

Visual landmark-based mapping and pose estimation has been an active topic of research for decades. Many previous works use interest-point *feature detectors* (such as Harris Corners [35] or FAST [74]) to find small, distinctive image features as landmarks for vehicle localisation and visual odometry (for example, [19, 68, 56, 79]). In order to perform matching, *feature descriptors* are often computed from a small region of the image surrounding the features. Commonly used feature descriptors include the Scale-Invariant Feature Transform (SIFT) [50], Speeded Up Robust Features (SURF) [2], and Binary Robust Independent Elementary Features (BRIEF) [8]. However, as these patches are often on the order of 11×11 pixels in size, they are highly sensitive to noise and image variability such as drastic, non-linear illumination changes. In addition, many rely on sharp images in order to discern features such as corners and therefore are particularly affected by motion blur, as the evaluation in [27] determined. Low-light conditions often exacerbate these effects for all but the most expensive and specialised cameras, and therefore systems that rely on these point features typically do not perform well in darkness.

In [86], limitations of both SIFT and SURF were identified when matching features across drastic seasonal appearance change and in the presence of significant lighting variation, albeit only during the day. One thread of work that deals with the latter problem uses RGB images that have been transformed into an illumination-invariant chromacity space by approximating daylight as a black body illuminant. This appears to hugely reduce the impact of shadows and other daylight artifacts, and has been applied to the problem of localisation in several publications [16, 58].

Some recent work has looked at how aforementioned specialised cameras can improve computer vision performance at night. In [7], Brunner et al. demonstrate localisation in adverse environments using a combination of both standard visual

and passive infrared imagery, concluding that this combination performs better than either sensor modality in isolation. Maddern et al. report similar findings in [54] when using the same combination of sensors to solve visual place recognition under different lighting and illumination conditions.

Exotic varieties of colour filter arrays (CFAs)—the tiny filters placed over camera photosensors to capture colour information—have also been used in some computer vision applications at night. In [49], four Real-AdaBoost classifiers are trained to detect small and large, front and rear vehicle lights, respectively. A novel “clear red” CFA is utilised which consists of three monochrome pixels and one red selective pixel. This is in contrast to both entirely monochrome sensors and the common colour *Bayer* filter with its mosaic pattern of one red, one blue, and two green-selective pixels. The “clear red” CFA is thus able to combine the better low-light performance of monochrome sensors while retaining the ability to distinguish front and rear lights by colour.

Attempts to emulate a high dynamic range (HDR) sensor with standard hardware include [69, 40], in which multiple images captured at different exposures are fused together (*exposure bracketing*). This allows a greater range of illumination to be captured but proves to be challenging when the camera moves during a capture sequence. In [43] the technique is used to improve robustness of SIFT key point detection for localisation under varying illumination conditions, however it is unknown how their system performs at night.

Our approach of using light blobs as landmarks has similarities to star tracking which is widely used in aerospace applications to determine a spacecraft’s orientation. Modern star trackers are fully autonomous, have comprehensive star catalogs, and are highly accurate and robust [21]. Due to the extreme distances involved, however, star tracking is only able to determine orientation of the spacecraft, not position. In contrast, most light sources visible from a vehicle navigating a well-lit

urban area will be sufficiently close to also provide an indication of relative position. Moreover, vehicles experience many environmental challenges when navigating on Earth that don't occur in space: landmarks can move, can be obstructed (possibly by other landmarks), and can disappear entirely without warning.

One of the few examples of work that addresses the problem of navigating a vehicle at night using consumer cameras is SeqSLAM [63], in which localisation is achieved by recognising coherent sequences of heavily down-sampled images. SeqSLAM demonstrates the ability to recognise a brightly lit urban environment at night despite having previously seen it only during the day. In a subsequent work, localisation was successfully achieved even on unlit rural roads by maximising camera exposure and gain settings [62]. While effective, these techniques only provide an estimate of topological position: in order to control an autonomous vehicle we require a metric pose estimate. In addition, they require long sequences of images to be processed before a position can be determined.

In [53], Maddern et al. demonstrate 24-hour localisation using the appearance of prior structure combined with an illumination-invariant colour space. It is found that a midday prior (map) does not perform well at night as the dominant illuminants—streetlights and headlights—can not be sufficiently modelled as black-body radiators. However, by using a greyscale prior built after nightfall the system is able to localise effectively. They therefore conclude that (at least in this instance), no one map is sufficient for localisation under both day and night conditions.

McManus et al. present an unsupervised approach to robust localisation by building a bank of spatially indexed detectors for large, distinctive image patches named *scene signatures* [60]. These elements are used for localisation in place of small point features, and the system appears to be able to successfully localise at night using a map built and trained on both daytime and nighttime imagery. However, their system differs from ours in several key ways. Firstly, a number of

carefully aligned images of each place in the world, taken at different times (ideally spanning a variety of seasonal, weather, and lighting conditions) are required to build scene signatures for that place. McManus et al. perform this image alignment using GPS/INS and report failures in areas where inaccuracies caused by these sensors mean that training images are not well aligned [57]. Additionally, as the effect of slight misalignment is more pronounced in the image foreground than in the far background, scene signatures tend to represent distant features. This means that the translational estimate of their localiser is not well constrained [61]. In contrast, our system builds a map from a single dataset and therefore does not require any error-prone image alignment. Also, as our system learns the appearance of individual light sources at varying distances, including in the very near foreground, we are able to effectively constrain our translational estimate.

4.4 System Overview

We implement a teach and repeat system as detailed in Section 2.5. The following sections describe the two main tasks our system is concerned with: (i) creating a map of light sources; and (ii) localising within a map of light sources. In both cases we process 2D image sequences from a number of appropriately calibrated monocular cameras for which we assume the standard pinhole camera model. We therefore also assume that, as a preprocessing step, the images are undistorted as described in Section 2.8.2.

4.5 Mapping

Building a map of suitable light sources is fundamentally a *structure from motion problem*, described in general terms in Section 2.9. Given a set of observations of

landmarks from a camera stream and a corresponding set of 6-DoF vehicle poses, our main aims are twofold. Firstly, we need to estimate a 3D location for each static light, ideally discarding those that move – car headlamps, for example. Secondly, we wish to create an individual appearance model for every light – what it looks like, and how its appearance changes as we move relative to it.

We describe the map creation process for a sequence of K RGB colour frames $\mathcal{I} = \{\mathcal{I}_1, \dots, \mathcal{I}_K\}$ from a single camera. From this point on we use the term *blob* to mean a 4-connected region of high intensity pixels. An overview of the steps for building a map are as follows:

1. Detect individual lights in each frame by extracting blobs.
2. Track blobs between frames to obtain a set of light tracks.
3. For each light track, solve the structure from motion problem for every consecutive pair of observations to obtain a set of candidate 3D locations. Tracks that exhibit a large variance of candidate locations are discarded.
4. For the remaining light tracks, solve the structure from motion problem in batch using all observations to obtain a final 3D location.
5. Extract and save appearance patches to our database.

4.5.1 Light Detection and Tracking

To perform light detection we first convert each image $\mathcal{I}_k \in \mathcal{I}$ to greyscale before segmenting using a fixed intensity threshold function, obtaining a binary image $\mathcal{I}_k^B$:

$$\mathcal{I}_k^B(u, v) = \begin{cases} 1 & \text{if } \mathcal{I}_k(u, v) > \lambda_B \\ 0 & \text{otherwise} \end{cases} \quad (4.1)$$

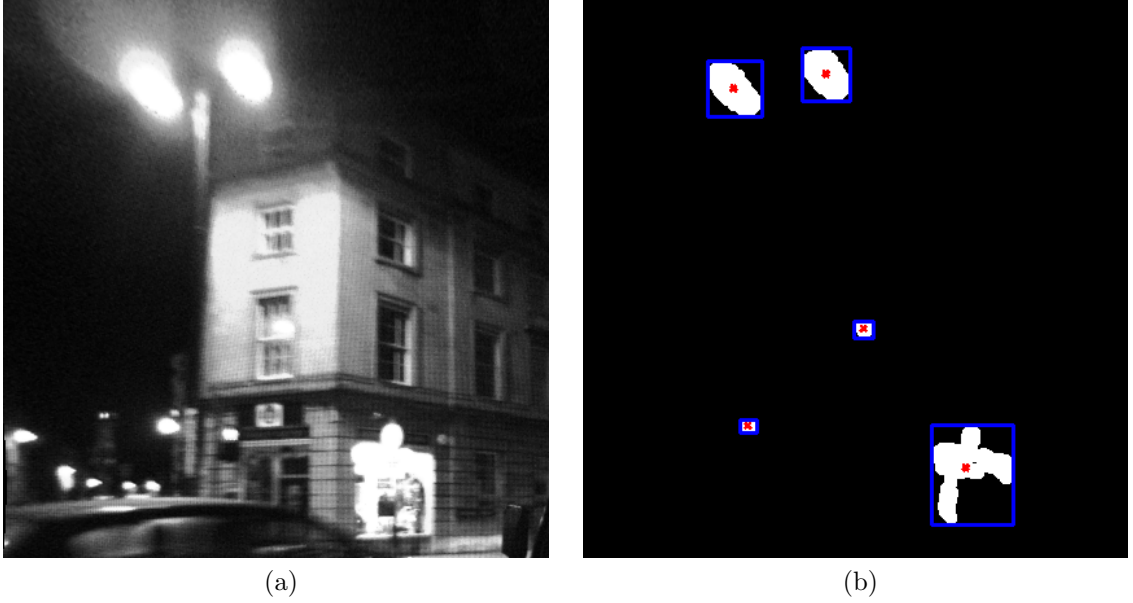


Figure 4.2: Light detection on a single frame. Figure (a) shows the raw image converted to greyscale. Figure (b) shows the binary thresholded image after erosion with the structuring element given in Equation (4.2). Computed bounding boxes and blob centroids are also shown.

Each $\mathcal{I}_k^B$ is eroded [77] using the 5×5 square structuring element $\mathbf{B}$:

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (4.2)$$

This is performed in order to remove noise and lights that are too dim or small to reliably observe. We found for our setup that using a structuring element of this size, along with setting $\lambda_B = 240$ (where $\lambda_B \in [0, 255]$ is a greyscale value) is a suitable configuration that preserves bright light centres but not reflections.

The blobs in each frame are then extracted from the eroded binary image and their bounding boxes and centroids are computed. This process is illustrated in Figure 4.2. We denote the set of blobs extracted from frame k as $\mathcal{B}_k = \{\mathcal{B}_k^1, \mathcal{B}_k^2, \dots\}$

We track lights across multiple frames using a nearest-neighbour approach. Our

aim is to construct candidate *light tracks* that consist of sequences of n_i blobs observed over consecutive frames: $\mathcal{T}_i = \langle \mathcal{B}_{k-n_i+1}^{\dots}, \mathcal{B}_{k-n_i+2}^{\dots}, \dots, \mathcal{B}_k^{\dots} \rangle$.

Intuitively, we don't expect lights to dramatically change shape or move very far frame-to-frame in image space. A heuristic cost function that considers bounding box overlap and distance between blob centroids is used to find the best match for a blob in the previous frame. We define the cost $\mathbf{c}(\mathcal{B}, \mathcal{B}')$ between two bounding boxes $\mathcal{B}$ and $\mathcal{B}'$ with centroids at $\mathbf{p}_{\mathcal{B}}$ and $\mathbf{p}_{\mathcal{B}'}$ respectively as:

$$\mathbf{c}(\mathcal{B}, \mathcal{B}') = \|\mathbf{p}_{\mathcal{B}} - \mathbf{p}_{\mathcal{B}'}\| \frac{\mathcal{B} \cup \mathcal{B}' - \mathcal{B} \cap \mathcal{B}'}{\mathcal{B} \cap \mathcal{B}'} \quad (4.3)$$

where the $\cap$ and $\cup$ operators return the area intersection and union of two bounding boxes respectively. We associate each blob in the current frame, $\mathcal{B}_k^i \in \mathcal{B}_k$, with one in the previous frame, $\mathcal{B}_{k-1}^j \in \mathcal{B}_{k-1}$, that minimises $\mathbf{c}(\mathcal{B}_k^i, \mathcal{B}_{k-1}^j)$, where $\mathbf{c}(\mathcal{B}_k^i, \mathcal{B}_{k-1}^j) < \lambda_C$ for some threshold λ_C , and additionally where no other blob $\mathcal{B}_k^l \in \mathcal{B}_k$ exists such that $\mathbf{c}(\mathcal{B}_k^l, \mathcal{B}_{k-1}^j) < \mathbf{c}(\mathcal{B}_k^i, \mathcal{B}_{k-1}^j)$. $\mathcal{B}_k^i$ is then appended to the light track that contains $\mathcal{B}_{k-1}^j$. However, if $\mathbf{c}(\mathcal{B}_k^i, \mathcal{B}_{k-1}^j) \geq \lambda_C$ then $\mathcal{B}$ is considered a previously unobserved light and a new light track is created $\mathcal{T}' = \langle \mathcal{B}_k^i \rangle$.

Lights are tracked until they go out of view; occlusions, splittings, and mergings are not explicitly considered as these may indicate an unreliable light source. After processing all frames we therefore obtain a set of n light tracks $\mathcal{T} = \{\mathcal{T}_1, \dots, \mathcal{T}_n\}$.

This tracking procedure works well for sparse lights that can be disambiguated very easily (for example solitary street lights) and less well for clusters that exhibit similar movement patterns within the image (for example groups of windows). We note, however, that this is not an issue as the latter are likely to also be ambiguous during localisation and make data association more difficult. In any case, lights that are tracked incorrectly in image space should not converge to a consistent location in 3D space and so will be culled in the next step.

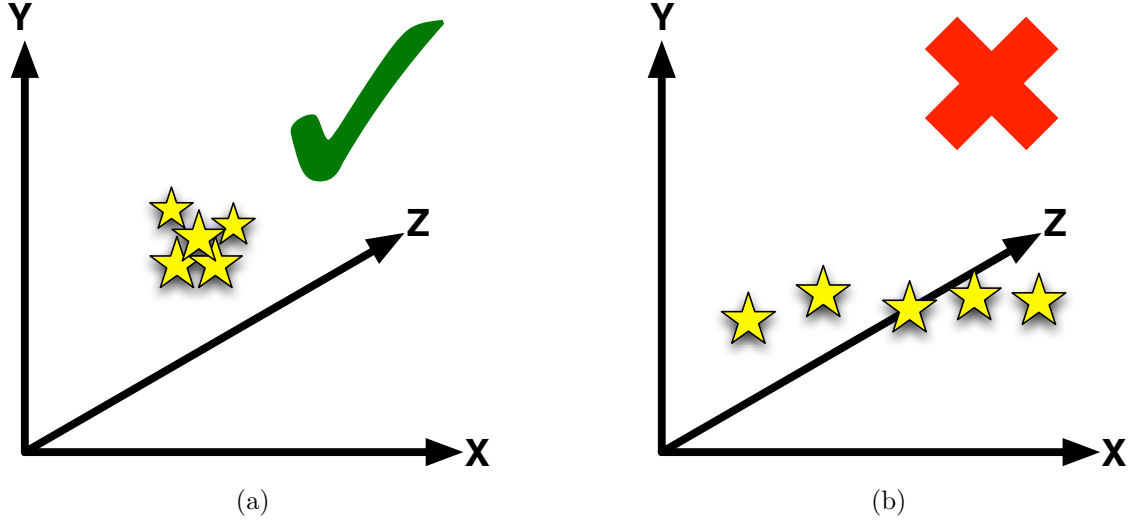


Figure 4.3: Illustration of how the distribution of candidate locations for a landmark, created from consecutive pairs of observations, is used to discard moving (or otherwise unreliable) lights. In the left-hand figure, the candidate locations are clumped together with low variance and so this is a desirable landmark. Conversely, in the right-hand figure, the candidate locations are spread out with high variance (it is possibly a moving light source) and therefore this landmark is discarded.

4.5.2 Structure from Motion

Light tracks that contain fewer than $\lambda_M \approx 10$ observations are discarded immediately as these are more likely to originate from unstable light sources or from spurious observations. For each blob track $\mathcal{T}_i$, we generate $n_i - 1$ estimates of the 3D position of the light using each consecutive pair of blobs. If the variance of these candidate locations is greater than a threshold λ_T , the blob track is discarded as this may indicate a moving light source or an otherwise incorrectly tracked object. This is illustrated in Figure 4.3.

Surviving blob tracks are input to a final optimisation process to obtain a 3D position by minimising the reprojection error over all measurements, as detailed in Section 2.9. From this we obtain a set of N lights $\mathcal{L} = \{\mathcal{L}_1, \dots, \mathcal{L}_N\}$ where each $\mathcal{L}_i \in \mathbb{R}^3$.

4.5.3 Modelling Appearance

For every observation in each light track, we extract and save a full-colour image patch from its corresponding frame, centred on the blob, resulting in a ‘stream’ of $\langle \text{appearance}, \text{camera pose} \rangle$ pairs for that light. Some examples of these appearance patches captured at different distances are shown in Figure 4.4. Additionally, the appearance stream is used to establish a 2-parameter functional relationship between blob area and distance from the camera. For light $\mathcal{L}_i$, we posit that area A is related to distance d by:

$$d = \frac{a_i}{\sqrt{A}} + b_i \quad (4.4)$$

as the inverse square law $A \propto \frac{1}{d^2}$ holds for the pinhole camera model [36]. Values for the two parameters a_i and b_i are obtained by solving linear least squares. Figure 4.5 shows some examples of how light area relates to distance from the camera, and the resulting graphs fitted to Equation 4.4.

Both the appearance stream and functional relationship given by Equation 4.4 form the basis of our individual appearance models for each light.

Finally, all relevant data about each light – its location, appearance patches, and learned parameters for Equation (4.4) are stored in our database for retrieval during the localisation phase. This is detailed later in Section 4.7.

4.6 Localisation

Given a set of observations of lights from a camera stream and a map of known light sources $\mathcal{L}$, we wish to estimate the camera pose. This is known as pose estimation or the perspective from n points problem. Although in multi-camera systems, each camera has its own independent map (i.e. we do not explicitly attempt to detect lights visible in one camera from another), we are able to combine 2D-to-3D point

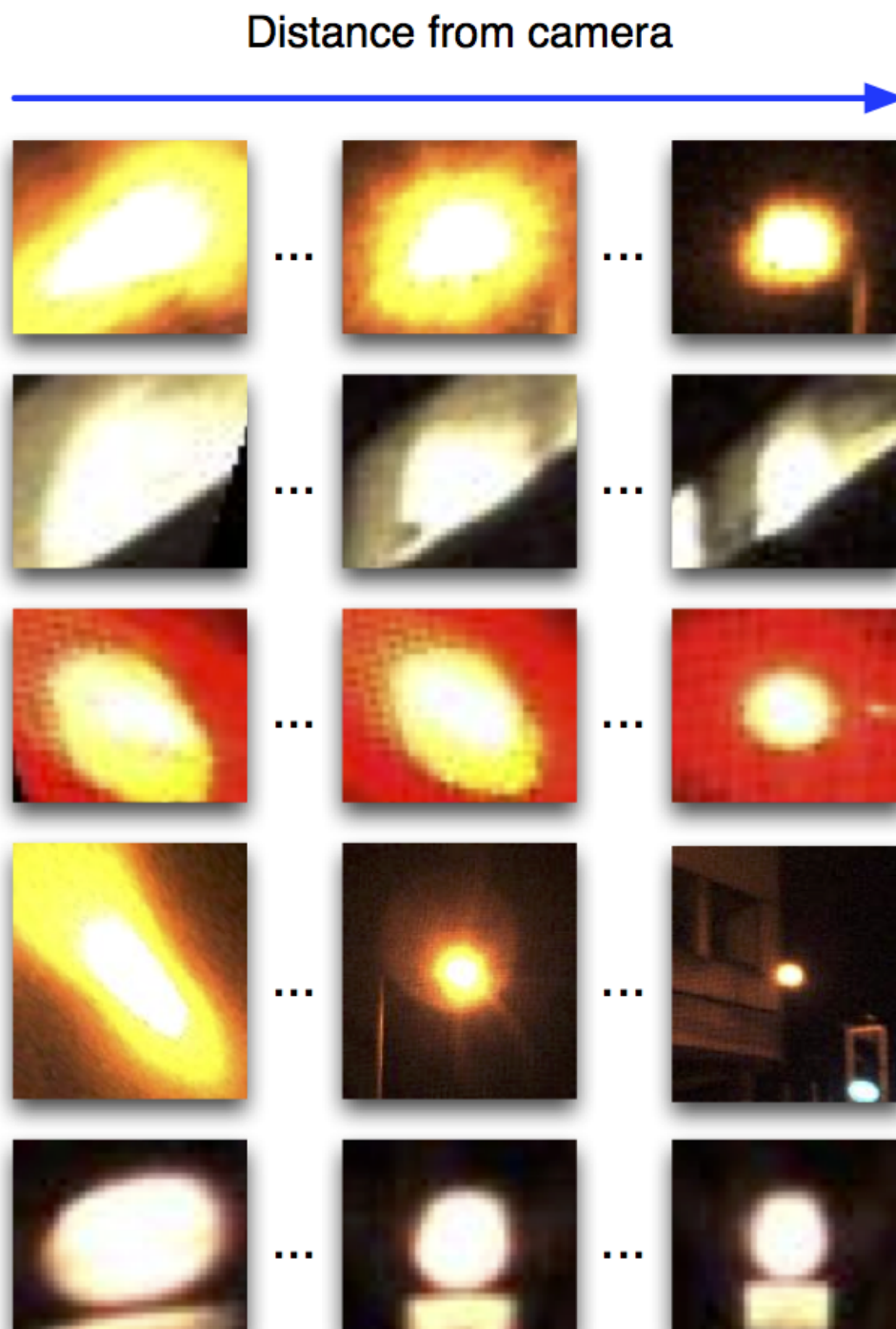


Figure 4.4: Example appearances of lights at varying distances from the camera. Note how drastically their appearance changes as the camera moves towards them – particularly those exaggerated by flaring effects. In a point feature based system this would be a problem, however it allows our system to distinguish between different lights at varying distances with ease.

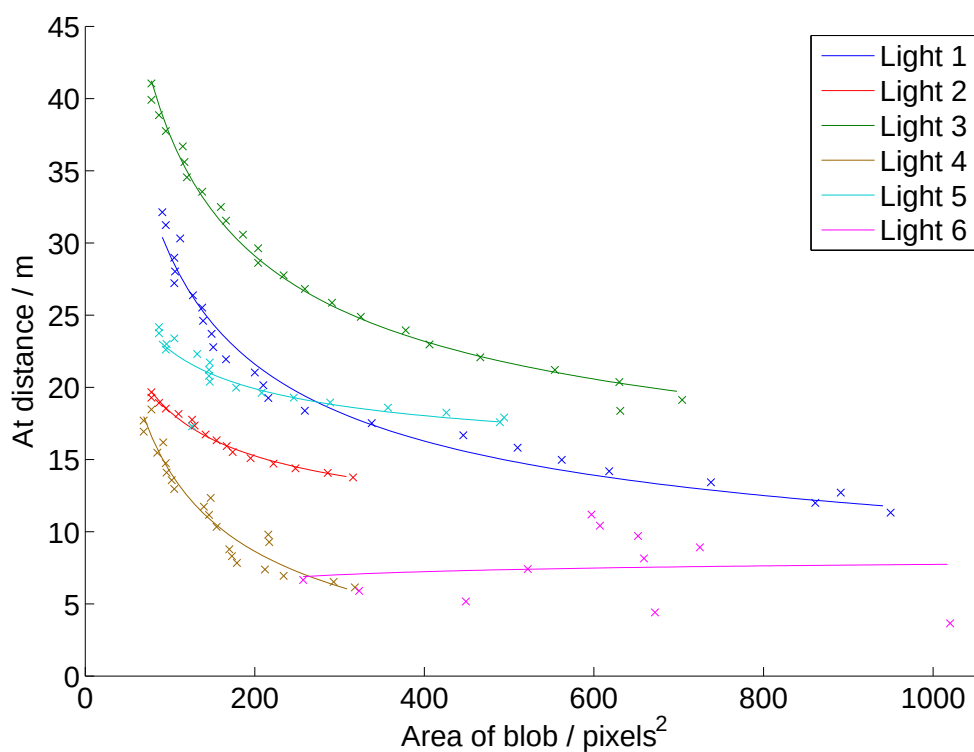


Figure 4.5: Graphs of light blob area vs distance from camera for six different lights fitted to Equation 4.4. This illustrates in particular how by referring back to these models next time we see the corresponding lights, we can accurately infer how far we are from them. It also demonstrates, along with Figure 4.4, just how diverse they look. This is very much to our advantage during data association. It is not perfect, however: light 6 is given as an example that *does not* fit the model correctly. This is mainly true for planar and irregular light sources, as well as those that are tracked incorrectly.

correspondences from all cameras in order to solve for pose. It is assumed that all necessary intrinsic and extrinsic camera calibration parameters are known in advance.

One of the key ideas that should be evident from Figure 4.4 is that not only do lights look very different from one another, but their individual appearances also change dramatically as we move closer or further away from them. We therefore firstly aim to use each light’s individual appearance model to make our data association more robust. Secondly, Figure 4.5 shows us that we can infer valuable information about how far away certain lights are by their observed area. This relationship allows us to derive coarse depth measurements from these appearance models that can influence our pose optimisation cost function.

A high-level overview of the steps for localising with a new set of camera frames is given below:

1. Make a prediction, $\bar{\mathbf{x}}_{t+1|1}$, for the current vehicle pose using the previous pose, $\mathbf{x}_t$, and some estimate of incremental motion, $\mathbf{u}_{t+1}$ (i.e. incorporate the motion model as in Section 2.10.4).
2. Detect lights in the current frame by extracting blobs.
3. Using the current pose estimate, $\bar{\mathbf{x}}_{t+1|1}$, predict where lights in the map should reproject to in the current frame. With these predictions, and the detected light blobs, use the joint compatibility branch-and-bound algorithm to compute a putative but mutually compatible set of 2D-to-3D point correspondences.
4. Perform a second pass rejection step based on light appearance, taking advantage of the fact that we know what these lights should look like from our current estimated location.

5. Estimate distance from camera to lights using their appearance models and Equation 4.4, yielding depth measurements for use in pose optimisation.
6. Finally, optimise to correct the pose estimate.

4.6.1 Pose Prediction

Assume we are given an estimate of the vehicle pose at time t which we write as $\mathbf{x}_t \in \mathbb{R}^6$. Additionally, we are given an estimate of incremental motion between time t and time $t + 1$, $\mathbf{u}_{t+1}$. Our predicted vehicle pose at time $t + 1$ is given by Equation (2.75):

$$\bar{\mathbf{x}}_{t+1|1} = \mathbf{u}_{t+1} \oplus \mathbf{x}_t$$

where $\mathbf{u}_{t+1}$ in our case comes from a visual odometry (VO) system. We have found that even at night, VO offers performance levels commensurate with those required to produce a seed solution to the localiser.

4.6.2 Points Correspondence

Light detection is performed in the same manner as described in Section 4.5.1. From this we obtain a set of n_k candidate blobs $\mathcal{B}^k = \{\mathcal{B}_1^k, \dots, \mathcal{B}_{n_k}^k\}$. All m_k lights $\mathcal{L}_k$ in the map within a distance λ_D of the current pose estimate $\hat{T}_k$ are considered to be observable candidates. Their predicted positions in the camera image are computed by reprojecting them into the current frame using our sensor model Equation (2.47).

We found that setting $\lambda_D = 80\text{m}$ is a suitable maximum distance for candidate lights, as this is around the distance at which a typical streetlight becomes visible in the intensity-thresholded image.

The joint compatibility branch-and-bound algorithm (JCBB) [65], which we described in Section 2.7.2.1, is used to obtain an initial set of point correspondences. JCBB computes the maximal set of jointly compatible correspondences and as such

preserves the correlation between measurements and predictions. However, as its running time is exponential in the number of measurements, we must invoke a policy to limit the number of candidate blobs to $\lambda_J \approx 20$. Therefore, if more than λ_J blobs are detected in a frame, we use a nearest-neighbour approach to prioritise candidate blobs that are closest to predicted locations of lights.

4.6.2.1 Modelling Uncertainty

Recall that JCBB requires computation of the joint residual covariance $\mathbf{S}$ which incorporates the measurement and state covariances $\mathbf{R}$ and $\mathbf{P}$, respectively. Equation (2.30) is repeated below for reference:

$$\mathbf{S}_\alpha = \mathbf{J}(\hat{\mathbf{x}}'_\alpha) \mathbf{P}^{-1} \mathbf{J}(\hat{\mathbf{x}}'_\alpha)^T + \mathbf{R}$$

We consider the measurement uncertainties for each light $\mathcal{L}_j \in \mathcal{L}_k$ independently so that the full measurement covariance matrix takes the form $\mathbf{R} = \text{diag}(\mathbf{R}_1, \dots, \mathbf{R}_{m_k})$, where each $\mathbf{R}_j = \text{diag}(r_u, r_v)$ represents the uncertainty of that light's centre in image space. We reason that due to over-exposure and flaring effects, the observed centre of closer lights (i.e. larger blobs) is more uncertain than the observed center of distant lights. Therefore, for each light $\mathcal{L}_k$, we scale r_u and r_v by the width and height of the closest appearance patch, respectively. Our state covariance matrix $\mathbf{P} = \text{diag}(\mathbf{P}_\mathbf{x}, \mathbf{P}_\mathbf{p})$ is constructed from a pose state covariance matrix $\mathbf{P}_\mathbf{x}$ (chosen to simply allow for a small amount of variance in each pose parameter), and a set of landmark covariance matrices $\mathbf{P}_\mathbf{p} = \text{diag}(\mathbf{P}_{\mathbf{p}_1}, \dots, \mathbf{P}_{\mathbf{p}_N})$. Each $\mathbf{P}_{\mathbf{p}_i} \in \mathbb{R}^{3 \times 3}$ is the inverse Hessian matrix resulting from the structure from motion non-linear least squares solve for the landmark's 3D location during the map building process.

Note that our state covariance matrix $\mathbf{P}$ is not updated and propagated from one time step to the next, like for example in an Extended Kalman Filter. This

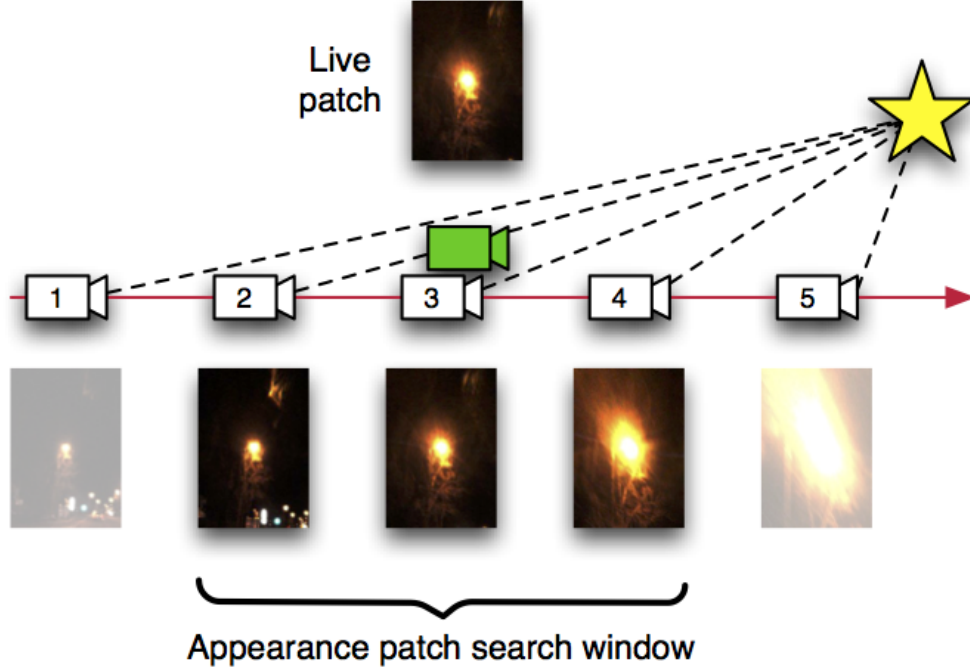


Figure 4.6: As part of our outlier rejection step, performed after JCBB returns a set of tentative associations, a window of stored appearances is searched for a match for our live light observation. In this example, five stored appearances of the candidate light and corresponding camera poses at their time of capture are shown. Our *predicted* live camera pose, relative to the map, is shown in green. Map pose 3 is closest (by distance) to our predicted location and so we search a window centred at the corresponding appearance.

means that $\mathbf{P}$ is sparse block diagonal. $\mathbf{J}(\mathbf{x})$ however is dense, and so the residual covariance $\mathbf{S}$ will still contain useful off-diagonal correlations.

4.6.2.2 Appearance Matching

The set of 2D-to-3D correspondences returned by JCBB may still contain outliers, but we have yet to consider the crucial deciding factor: whether the lights look how we remember them.

Recall that for every light $\mathcal{L}_i$ in our map, we have a set of appearance patch, camera pose pairs $\langle A_{i,j}, \mathbf{x}_{i,j} \rangle$, stored in our database, representing every observation of that light and where it was observed from. Intuitively, we would expect our observation to best match the appearance that was captured at a location closest to our predicted pose. If this closest appearance fails to match our current observa-

tion, however, due to the uncertainty in our pose estimate, we also consider nearby appearances. This is as a result of the potential for a light's appearance to change considerably in the space of a few metres.

Therefore, we query our database system for a window of stored appearances:

$$\bar{\mathcal{A}}_i(\bar{\mathbf{x}}) = \{\mathcal{A}_{i,j+k} : j = \arg \min_j \|\mathbf{x}_{i,j} - \bar{\mathbf{x}}_{t+1|1}\|, |k| \leq \omega\} \quad (4.5)$$

where parameter ω , determining the window size, can be fixed or could for example vary according to our pose uncertainty. $\bar{\mathbf{x}}_{t+1|1}$ is the current pose estimate. Figure 4.6 shows how this matching process works.

We quantify appearance similarity by comparing image patches through some appearance matching function $\Gamma(\cdot)$ which returns a score. We select the appearance $\mathcal{A}_{i,j} \in \bar{\mathcal{A}}_i$ with the highest score. We compare zero-mean normalised cross correlation and support vector machine implementations of Γ later in Section 4.9.2.

4.6.3 Pose Update

Recall from Section 2.10 that we now wish to compute a delta pose estimate, Δ , by minimising some cost function. This is where our individual appearance models come into play again – not only do we want to take reprojection errors into account, we also want to consider how far away we appear to be from each observed light, in order to obtain coarse depth measurements. These distances can be calculated based on the area of each light's observed blob using Equation 4.4. Incorporating depth information allows the system to arrive at reasonable solutions for configurations not possible with simple reprojection alone.

Therefore, our objective function $C(\Delta)$, given N pixel measurements and M

depth measurements, is:

$$C(\Delta) = \frac{1}{2} (\mathbf{r}_h(\Delta)^T \Sigma_h \mathbf{r}_h(\Delta) + \mathbf{r}_d(\Delta)^T \Sigma_d \mathbf{r}_d(\Delta) + \mathbf{r}_f(\Delta)^T \mathbf{Q}^{-1} \mathbf{r}_f(\Delta)) \quad (4.6)$$

where $\mathbf{x} \in \mathbb{R}^6$ is the 6-DOF transformation we are solving for, parameterised by translational components x, y, z , and Euler angles $\theta_r, \theta_p, \theta_q$. $\mathbf{r}_h$ is the residual vector for the pixel measurements given in Equation (2.57). $\mathbf{r}_d$ is the residual vector for the depth measurements given in Equation (2.73). $\mathbf{r}_f$ is the residual vector for the motion model given in Equation (2.77). Σ_h and Σ_d are diagonal matrices containing the Huber weights for each pixel and depth measurement, respectively, computed as detailed in Section 2.7.3. $\mathbf{Q}^{-1}$ is a diagonal matrix containing the weights we wish to apply to the estimate from the motion model, as described in Section 2.10.4.1.

Our optimisation approach is to use a slight variation of Levenberg-Marquardt with a robust cost function (Huber kernel [42]), as detailed in Chapter 2. Therefore the normal equations [85] for the least-squares minimisation are:

$$\mathbf{J}^T \Sigma \mathbf{J} \mathbf{x} = -\mathbf{J}^T \Sigma \mathbf{r} \quad (4.7)$$

where $\mathbf{J}$ is the $(3N + M + 6) \times 6$ Jacobian:

$$\mathbf{J}(\Delta) = \begin{bmatrix} \mathbf{J}_h(\Delta) \\ \mathbf{J}_d(\Delta) \\ \mathbf{J}_f(\Delta) \end{bmatrix} \quad (4.8)$$

and $\mathbf{r}$ is the $(3N + M + 6) \times 1$ vector:

$$\mathbf{r} = \begin{bmatrix} \mathbf{r}_h(\Delta) \\ \mathbf{r}_d(\Delta) \\ \mathbf{r}_f(\Delta) \end{bmatrix} \quad (4.9)$$

where $\mathbf{J}_h$, $\mathbf{J}_d$, and $\mathbf{J}_f$ are the Jacobians given in Equation (2.58), Equation (2.74), and Equation (2.78), respectively. Σ is the diagonal weighting matrix:

$$\Sigma = \text{diag}(\Sigma_h, \Sigma_d, \mathbf{Q}^{-1}) \quad (4.10)$$

Our final pose estimate for frame $k + 1$ is then computed with the update:

$$\mathbf{x}_{k+1} = \Delta \oplus \mathbf{x}_k \quad (4.11)$$

4.7 Database Usage

We store the NightNav map using our database framework developed in Chapter 3. The schema is augmented with several additional tables. The first stores data about each light (i.e. one row per light) and has the following schema:

$$\begin{aligned} \text{Lights}(\underline{\text{light_id}} : \text{integer}, \text{monolithic_id} : \text{integer}, \\ \text{camera_index} : \text{integer}, \text{location} : \text{geometry}, \\ \text{covariance} : \text{double precision}[], \\ \text{parametric_params} : \text{double precision}[]) \end{aligned} \quad (4.12)$$

where column *monolithic_id* is a foreign key referencing a monolithic file in the Monolithics table containing sensor data the map was built from. Column *camera_index* is the identifier for the camera. Columns *location* and *covariance* contain the 3D location of the light and 3×3 uncertainty covariance matrix, respectively. Column *parametric_params* contains the parameters a_i and b_i from Equation (4.4) for the light. An R-Tree index is created on the *location* column to improve the performance of spatial queries.

Another table stores data about each light appearance (i.e. one row per obser-

vation of a light) and has the following schema:

$$\begin{aligned} \text{LightAppearances}(\underline{\text{light_id} : \text{integer}, \text{data_id} : \text{integer}}, \\ \text{location} : \text{geometry}, \text{width} : \text{integer}, \\ \text{height} : \text{integer}, \text{image} : \text{bytea}, \\ \text{svm} : \text{bytea}) \end{aligned} \quad (4.13)$$

where columns *light_id* and *data_id* are foreign keys into the Lights and Data tables respectively, representing the light that this appearance belongs to, and the record in the monolithic file containing the camera image that the appearance was extracted from. Column *location* is the 3D location of the survey vehicle when this appearance was seen (i.e. obtained via integrated visual odometry). Columns *width*, *height*, and *image* store a cached image patch (and its pixel dimensions) containing the appearance. Column *svm* stores a serialised detector that has been trained to recognise this appearance. As above, an R-Tree index is created on the *location* column.

4.7.1 Queries

During localisation the following types queries are executed. Mean execution times are given having been performed on an early 2011-era MacBook pro with 8GB RAM and a 500GB SSD, with a PostgreSQL server running locally:

1. Retrieve all lights within 80m of the vehicle's current location. As described in Section 4.6.2, this is performed once when a new live image is available. Mean execution time: $\approx 2.6\text{ms}$.
2. Retrieve closest appearance for a light. As described in Section 4.6.2.2, this is performed once per light observed in the live image. Mean execution time: $\approx 2.7\text{ms}$.

3. Retrieve a window of the closest $\omega \approx 5$ appearances for a light. As described in Section 4.6.2.2, this is performed for a observed light if its does not match the closest predicted appearance. Mean execution time: $\approx 7\text{ms}$.

Therefore, the number of queries executed in real-time during localisation depends on the number of lights detected, but is typically seen to be on the order of 100 per second.

4.8 Systems and Datasets

Our system was tested, verified, and analysed on real-world data collected around the roads of Oxford. In this section we provide details of the hardware and datasets that were used in the following analysis.

4.8.1 Hardware

For both mapping and localisation we used our survey vehicle, a Bowler Wildcat modified by BAE Systems to include fully autonomous capability, pictured in Figure 4.7. Of relevance to this work, the sensor payload includes a front-facing Point Grey Bumblebee2 stereo camera (used by VO and the visual teach-and-repeat system we compare to NightNav in Section 4.9.6, named VT&R) and a front-mounted Point Grey Ladybug2 (LB2) omni-directional camera system (used by NightNav). The Ladybug2 itself consists of six separate cameras: five are arranged radially providing 360° field of view and a sixth points directly up. Our system uses only the five radially mounted cameras as these provide the best views of lights surrounding the vehicle but it is worth noting again that our technique generalises to any number of monocular cameras in any configuration. The Automatic Multi-Camera Calibration Toolbox [90] was used for camera extrinsic calibration, with intrinsic parameters provided by the manufacturer.



Figure 4.7: Our Wildcat autonomous survey vehicle with which we collected data at night. The omnidirectional Point Grey Ladybug2 (LB2) camera is can be seen mounted in an elevated position at the front of the vehicle. The downward-facing Bumblebee2 camera is mounted next to it. The compute server to which data are logged is visible in the rear compartment.

The Wildcat is also equipped with a roof-mounted Velodyne HDL-64E lidar, a SICK LMS-151 lidar in pushbroom configuration, and an OxTS RT 3042 GPS INS system, however these sensors are not utilised in this work.

Also onboard is a Dell PowerEdge R710 compute server with 1.5TB of SSD storage where sensor data are logged during data collection missions in the format described in Section 3.6.

4.8.2 Datasets

For our results and system testing we used 12 datasets collected from two reasonably well-lit urban routes around central Oxford (both shown in Figure 4.8), each measuring just over 4km in length. Eight datasets on Route 1, totalling around 34km, were collected over a period of five months and so exhibit significant appearance change caused by seasonal, weather, and traffic differences. The remaining four datasets on Route 2, totalling around 17km, were collected over several days in late spring and are therefore all quite similar in appearance. Figure 4.9 shows the routes

ID	Date/Time Collected	Total LB2 Frames	Median Inter-Frame Distance	Mean Logging Frequency
D1a	9th January 21:10 GMT	5139	0.92m	10Hz
D1b	9th January 21:35 GMT	5660	0.93m	10Hz
D1c	9th January 21:50 GMT	4400	0.98m	10Hz
D1d	1st May 22:30 GMT+1	1726	1.56m	3.8Hz
D1e	12th June 23:00 GMT+1	2445	1.16m	3.8Hz
D1f	12th June 23:15 GMT+1	1885	1.78m	3.8Hz
D1g	17th June 23:00 GMT+1	4820	0.98m	10Hz
D1h	17th June, 23:25 GMT+1	5280	0.92m	10Hz

(a) Route 1

ID	Date/Time Collected	Total LB2 Frames	Median Inter-Frame Distance	Mean Logging Frequency
D2a	12th June, 23:30 GMT+1	2650	0.76m	3.8Hz
D2b	12th June, 23:50 GMT+1	2500	0.76m	3.8Hz
D2c	17th June, 23:40 GMT+1	6342	0.94m	10Hz
D2d	18th June, 00:00 GMT+1	6305	0.98m	10Hz

(b) Route 2

Table 4.1:

Table of datasets collected along both routes, showing when they were collected, the total number of Ladybug2 image frames that were logged, and the median distance between each image frame. Note also that due to a software bug, datasets D1d-f and D2a-b logged at roughly a third of the normal rate. This is the reason for the larger distances between frames, however this does not appear to negatively impact our system as VO is sufficiently locally accurate to fill in these gaps. The datasets used to create the maps for each route are highlighted in blue.

divided into three broad categories of region:

1. Street-lit only – characterised by street lights on one or both sides of the road and generally very few buildings that exhibit visible interior or exterior lighting. These regions roughly correspond to an overall low density of landmarks.
2. Intersections – areas where we might expect the vehicle to wait at signal controlled junctions, meaning in addition to street lights, illumination sources include traffic lights, illuminated road signs, and lights of stationary cars. These regions roughly correspond to an overall medium density of landmarks.
3. Dense city – characterised by a high density of buildings with visible interior and exterior lighting, illuminated billboards, road signs, and (typically moving) traffic. These regions roughly correspond to an overall high density of landmarks.

Table 4.1 provides additional details about these datasets including the date and times they were collected, the number of camera frames logged, and the average distance between consecutive camera frames. Datasets are classified as either map (i.e. used to build the map; coloured blue in Table 4.1), or evaluation (i.e. used for localisation analysis).

4.8.2.1 Ground Truth Appearance Correspondences

In anticipation of needing to verify our appearance matching functions, we created a manually labelled dataset of corresponding appearances.

The labelling was carried out as follows. Random camera frames were selected from an evaluation dataset. For every mapped light $\mathcal{L}_i$ visible in a particular frame $\mathcal{I}_k$, we extracted: (a) an image patch around $\mathcal{L}_i$ in $\mathcal{I}_k$; and (b) the closest corresponding appearance for $\mathcal{L}_i$ stored in our map database. We also extracted lights from

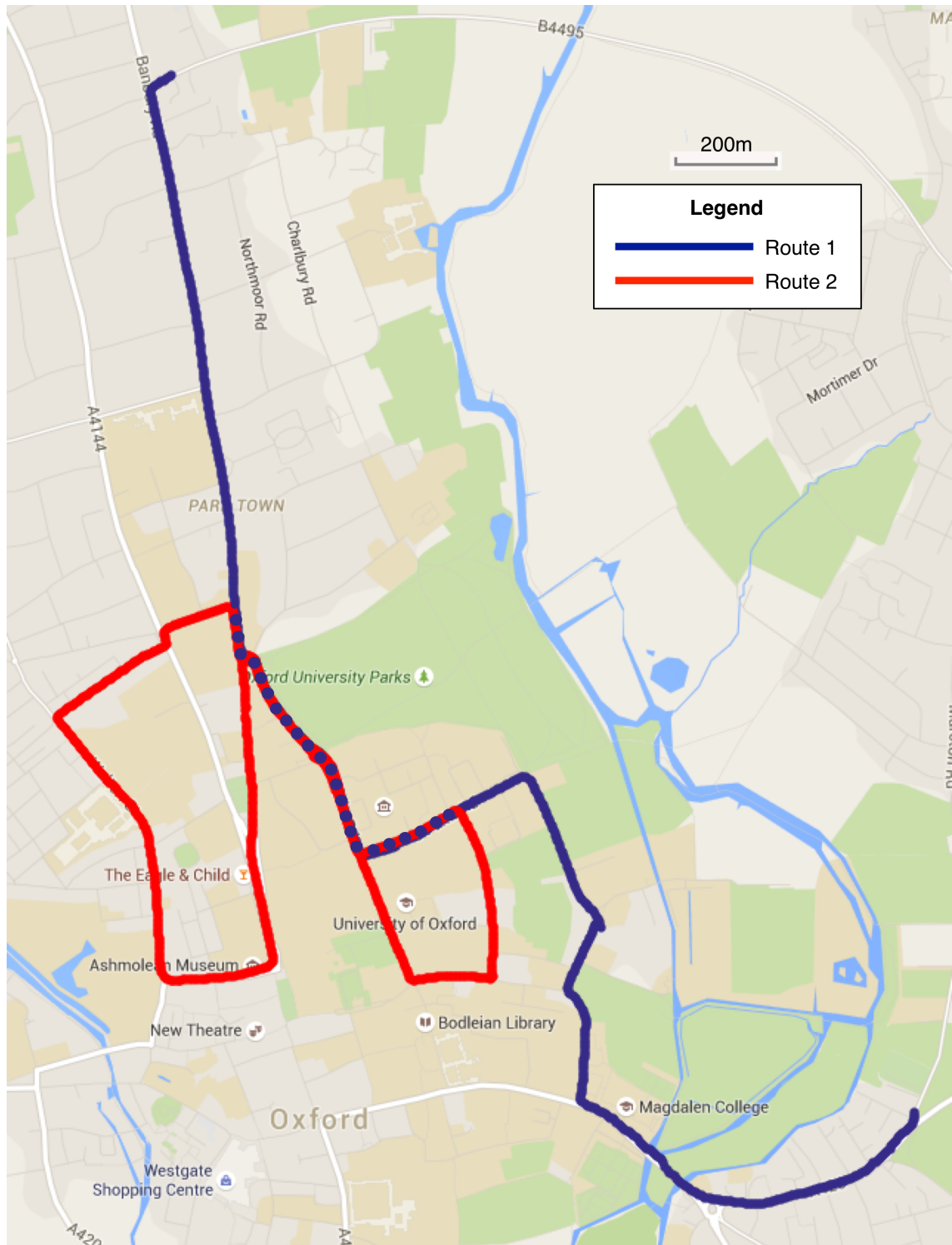


Figure 4.8: A map showing both of the routes through central Oxford used in system testing. Route 1 is shown in blue and Route 2 is shown in red. Both are roughly 4km in length and share a common section which is shown in dotted blue and red. Map data © 2015 Google.

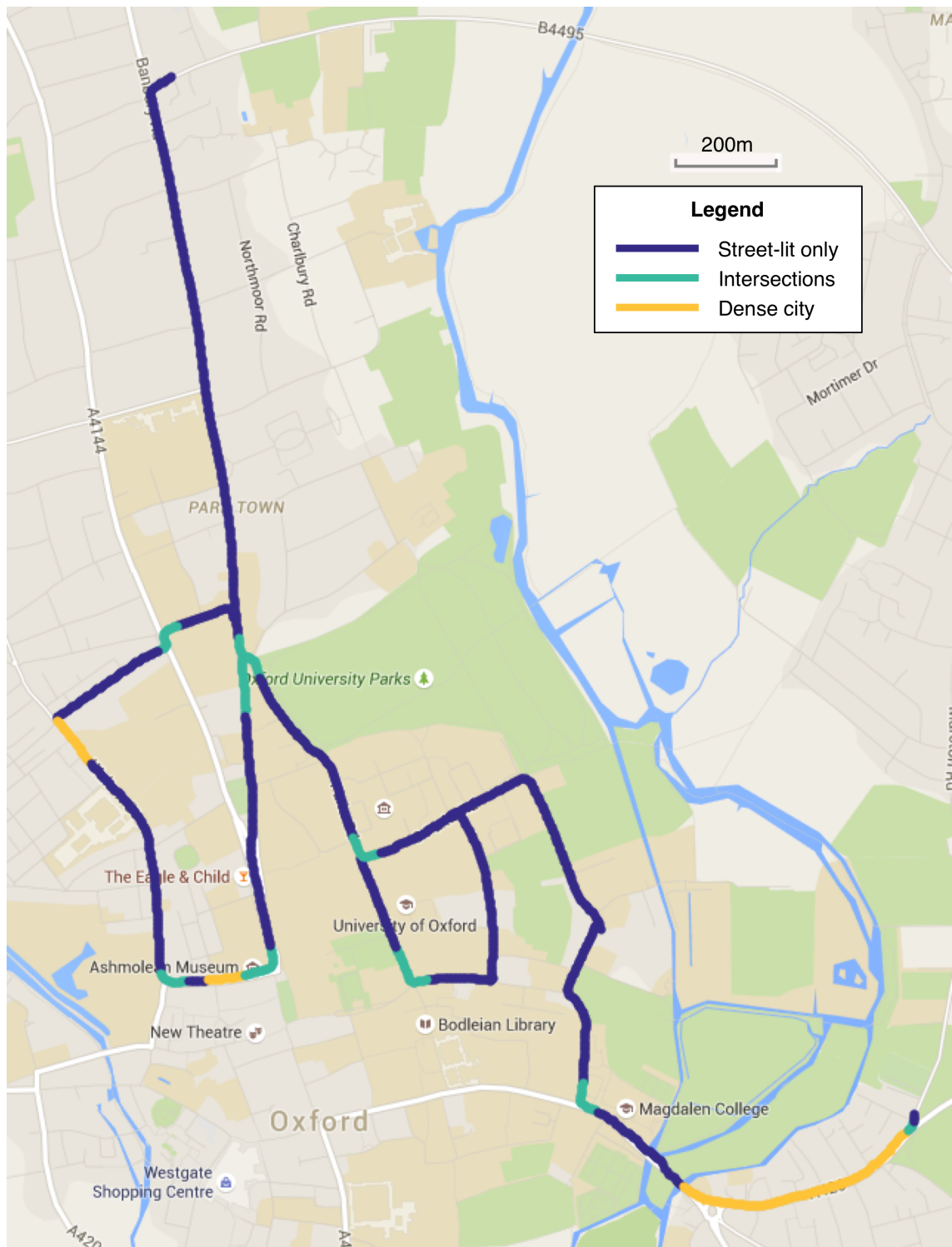


Figure 4.9: A map showing both of the routes divided into three broad categories of region. In blue are areas that are relatively sparsely lit by street lights. Green are areas with medium density lighting, usually intersections with traffic lights, illuminated road signs, and the lights of stationary vehicles. The most densely lit areas are shown in yellow which typically have interior and exterior lighting from buildings and shops.

ID	Evaluation Dataset ID	Map Dataset ID	No. Corresponding Image Pairs
L1	D1e	D1b	271
L2	D2a	D2c	208

(a)

Table 4.2:

Table of labelled appearance datasets, showing the evaluation and map datasets from which appearance patches have been extracted. Refer to Table 4.1 for details about these. The number of corresponding pairs of images that were extracted is also given.

$\mathcal{I}_k$ that did not correspond to anything in the map, to use as additional negative examples – this would include, for example, car headlights.

To ensure our labelled dataset was wholly representative of the regions and times encompassed by our data as a whole, we labelled an evaluation dataset from each route: D1e (collected in June), therefore corresponding map appearances extracted from dataset D1b (January); and D2a (June), corresponding to dataset D2c (June).

Several examples of extracted image patches and corresponding map appearances are shown in Figures 4.10a and 4.10b respectively.

Table 4.2 provides some details about both labelled appearance datasets.

4.8.2.2 Variations in Dataset Imagery

One camera configuration issue of particular interest to us is exposure level. Because the landmarks we are interested in are by definition bright objects, we might expect that our system is highly sensitive to variations in exposure settings. A simple remedy would be to fix the shutter speed and aperture size, however we instead opted to use our software auto-exposure controller that was developed for use in [17]. This was desirable for several reasons: (1) it reduces complexity in the camera driver; (2) it allows us to better co-exist with conventional localisation systems operating in daylight; and (3) it is reasonable to expect a localisation system to be

robust to small variations in exposure, and therefore it is prudent to capture such variations in our test data. Here we analyse the extent of this variation in order to understand the effect it has on appearance matching algorithms we evaluate later in Section 4.9.2.

To assess the typical variations in exposure that might be encountered in our urban night time environment, we computed intensity statistics for the labelled appearances in datasets L1 and L2. Figure 4.10 shows several corresponding pairs of image patches from L1, as well as normalised pixel intensity histograms for all image patches in both L1 and L2. These histograms show clearly that the dataset collected in January (D1b) is more exposed than that collected in June (D1e), as there are fewer pixels with lower (darker) values and more with higher (brighter) values, however the difference is not enormous. Figures 4.10e and 4.10f show the differences between the normalised intensity histograms for the L1 and L2 datasets, respectively. It is again clear that the difference between the January and June datasets is much more pronounced than between the two June datasets.

Having established that our chosen auto-exposure controller produces small but non-trivial differences in exposure levels between datasets, we are able to take this issue into account when we evaluate various appearance matching techniques in Section 4.9.2.

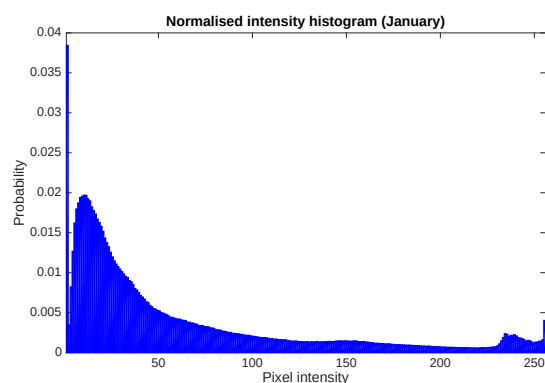
4.9 Results and Analysis

In this section we evaluate the performance of our localisation system (NightNav), before comparing it against a FAST [74] point feature based teach-and-repeat localiser (VT&R) [12].

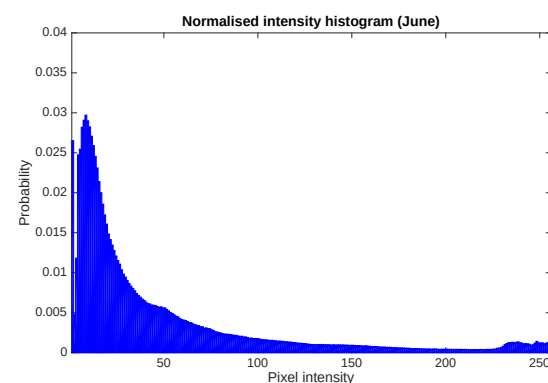


(a) Light appearances in January (dataset D1b).

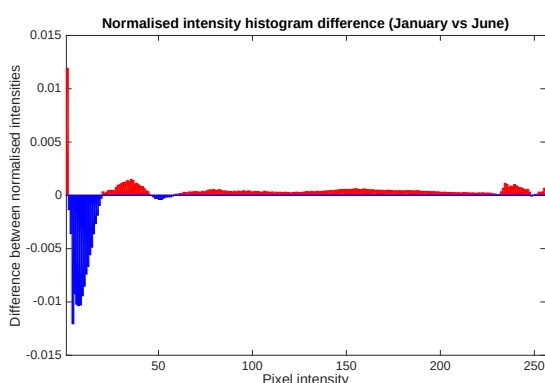
(b) Corresponding light appearances in June (dataset D1e).



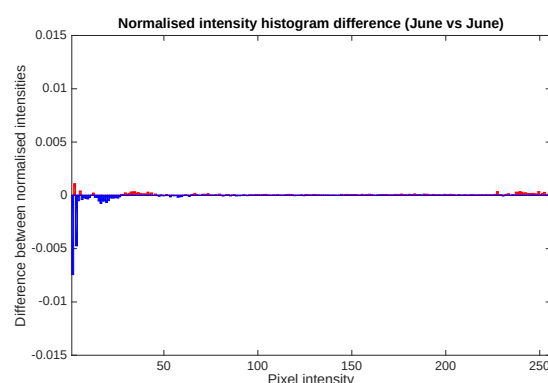
(c) Normalised pixel intensity histogram for all light appearance patches in January (dataset D1b).



(d) Normalised pixel intensity histogram for all light appearance patches in June (dataset D1e).



(e) Differences between January (dataset D1b) and June (dataset D1e) normalised pixel intensity histograms.



(f) Differences between June (dataset D2a) and June (dataset D2c) normalised pixel intensity histograms.

Figure 4.10:

4.9.1 Map Analysis

Maps of lights along each route were built using the method described in Section 4.5 using datasets D1b and D2c. Our VO system [11] determined the dead-reckoned map trajectory. We found it to be sufficiently locally accurate despite not being optimised for low-light conditions.

Figures 4.11 and 4.12 respectively show the density of mapped lights along routes 1 and 2. It can be seen that this density roughly corresponds to the region categories described in Section 4.8.2.

4.9.2 Light Appearance and Classification

Recall from Section 4.5.3 that we use an appearance matching function $\Gamma(\cdot)$ to detect additional outliers at the end of our data association process. Image patches extracted from around lights detected in a live camera frame are compared with the expected appearance of that light stored in our map database. If the two are not ‘similar enough’, the system rejects the match as an outlier.

Here we analyse and compare the performance of various image matching approaches, including zero-mean normalised cross correlation (ZNCC) score with a threshold, and a support vector machine (SVM) classifier.

4.9.2.1 Zero-Mean Normalised Cross Correlation

The zero-mean normalised cross correlation (ZNCC) score between a greyscale template image $\mathcal{I}_t(u, v)$ and a greyscale query image $\mathcal{I}_q(u, v)$ provides a measure of their statistical similarity and is computed as:

$$\Gamma_{ZNCC}(\mathcal{I}_t, \mathcal{I}_q) = \frac{1}{n} \sum_{u,v} \frac{(\mathcal{I}_t(u, v) - \hat{t})(\mathcal{I}_q(u, v) - \hat{q})}{\sigma_t \sigma_q} \quad (4.14)$$

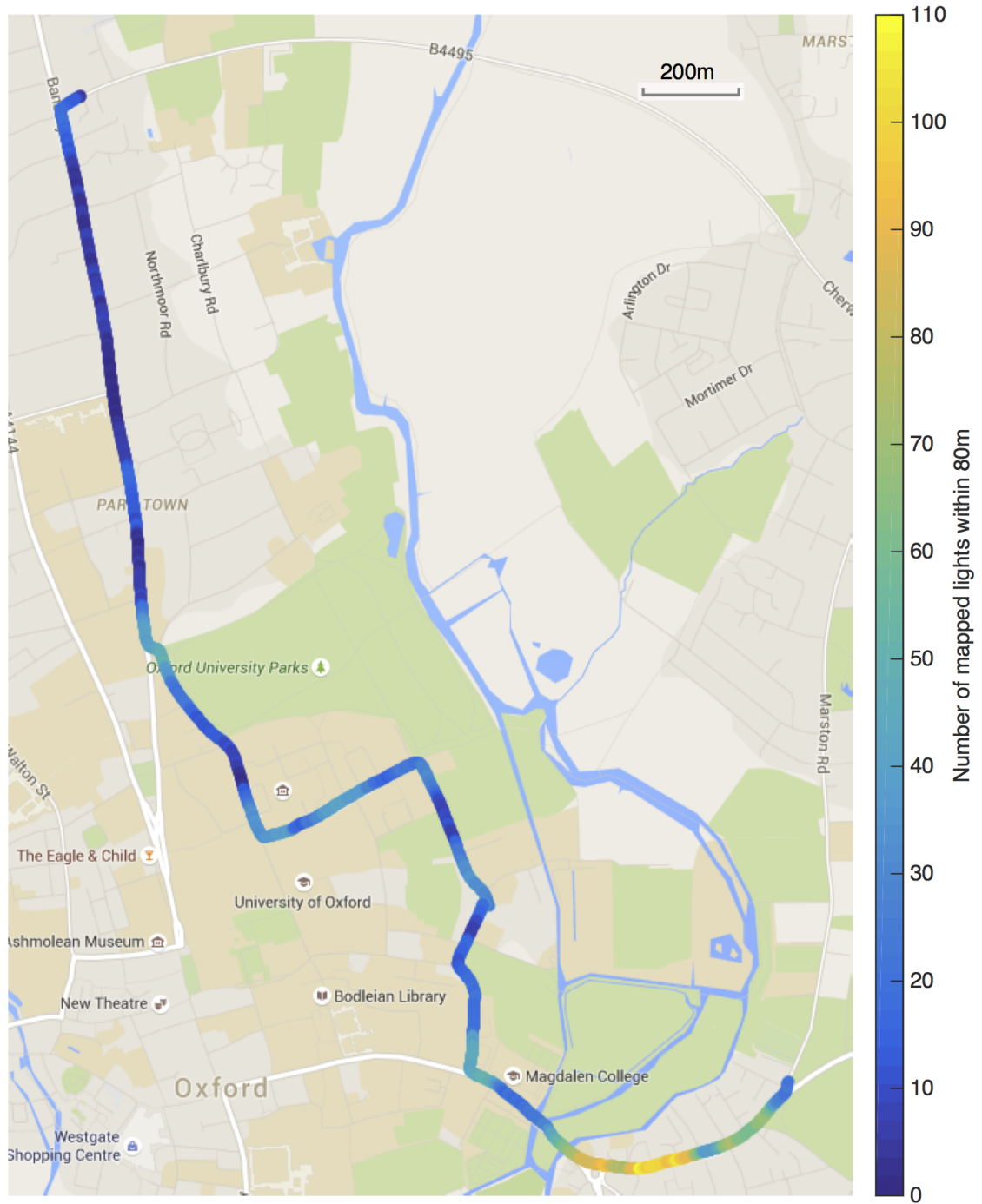


Figure 4.11: Light density for Route 1 map. Points along the map are coloured according to the number of landmarks within an 80m radius of that point.

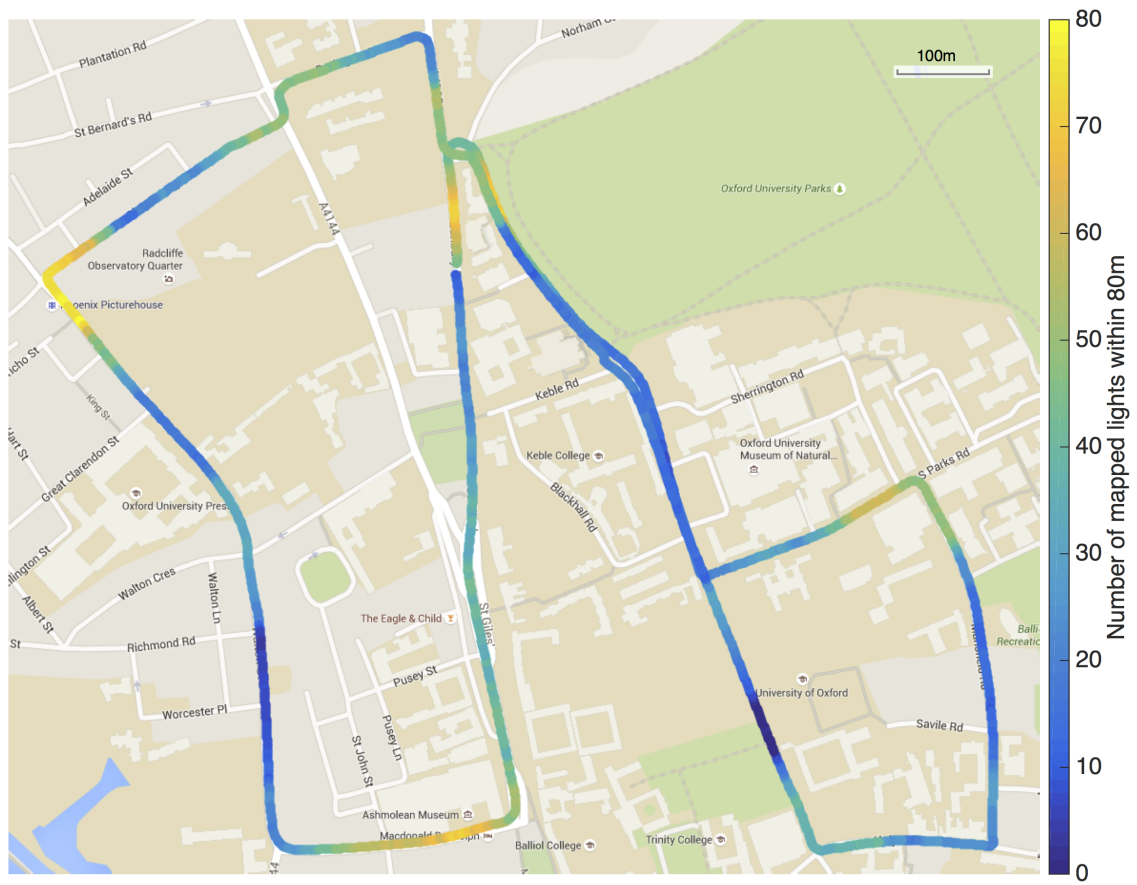


Figure 4.12: Light density for Route 2 map. Points along the map are coloured according to the number of landmarks within an 80m radius of that point.

where n is the number of pixels in the images, $\hat{t}$ and $\hat{q}$ are the mean pixel values, and σ_t and σ_q are the standard deviations. The score ranges from between -1 (completely dissimilar) to 1 (completely similar). We apply this to RGB images by concatenating the three colour channels together, resulting in a single greyscale image (three times the width of the original). In our image matching function, we wish to reject dissimilar matches (i.e. $\Gamma_{ZNCC}(\mathcal{I}_t, \mathcal{I}_q)$ less than some threshold T_{ZNCC}) – but exactly how dissimilar?

First we must realise that there is a trade-off to be made between accepting *false positives* as matches, and rejecting *false negatives* as outliers. There are two extreme cases. If we set $T_{ZNCC} = 1$, then we reject every match and therefore minimise the number of false positives, however we are left with no measurements with which to infer our pose. On the opposite extreme, we set $T_{ZNCC} = -1$ and so accept every match, thereby minimising the number of false negatives. In this case we will have enough measurements, however there may be outliers amongst them which will have a negative impact on the quality of our solution. We therefore assess multiple values of T_{ZNCC} in our analysis.

4.9.2.2 Support Vector Machines

A Support Vector Machine (SVM) [87] is a supervised learning model that is widely used for classification problems in machine learning. Given some training data points, a binary linear SVM attempts to best separate the two classes with a hyperplane. An SVM’s cost function incorporates a parameter C that determines the relative influence of misclassifying each training example versus the size of the *margin* – the distance from the hyperplane to the nearest training example on either side. Large values of C result in an SVM that prefers to correctly classify training examples even if that results in a smaller margin. Conversely, small values of C express a preference for the margin to be maximised, even if that means misclassifying

more training examples. The ‘best’ value of C depends on the given type of data and is often chosen by experimentation, therefore we assess multiple values of it in our analysis.

We use the Histogram of Oriented Gradients (HOG) [18] image descriptor to generate feature vectors from images that can be fed to an SVM for training or classification purposes. The HOG implementation provided by Ludwig et al. [52] is used. Input images are divided into 9 overlapping rectangular cells (3 rows and 3 columns) each with 9 orientation bins. All 81 histogram bins are concatenated to form an 81-dimensional (independent of input image size) feature vector. These parameter values are recommended in [52], albeit for pedestrian detection. We therefore verified their performance on our own data by experimentation with a range of values: the recommended configuration indeed yielded the most accurate classification results in the proceeding analysis.

4.9.2.3 Results

We examine the performance of ZNCC with varying values of T_{ZNCC} and SVMs trained on HOG descriptors derived from the (1) greyscale images only; (2) RGB channels only; (3) both RGB and greyscale information. For each SVM we vary their C parameter by multiples of 10 from 10^{-5} to 10^{-1} (no additional effect was found by varying them beyond this range). We validate these methods on the labelled appearance datasets described in Section 4.8.2.1. From this we are able to measure the true/false positive/negative rates and so better inform our choice of appearance matching method.

Figure 4.13 shows *miss rate* plotted against *false positives per image* for labelled datasets L1 and L2. We can see that the SVMs out-perform ZNCC for every value of T_{ZNCC} , and the SVMs that incorporate colour information perform marginally better than those without. Interestingly, we also see that no single value of T_{ZNCC}

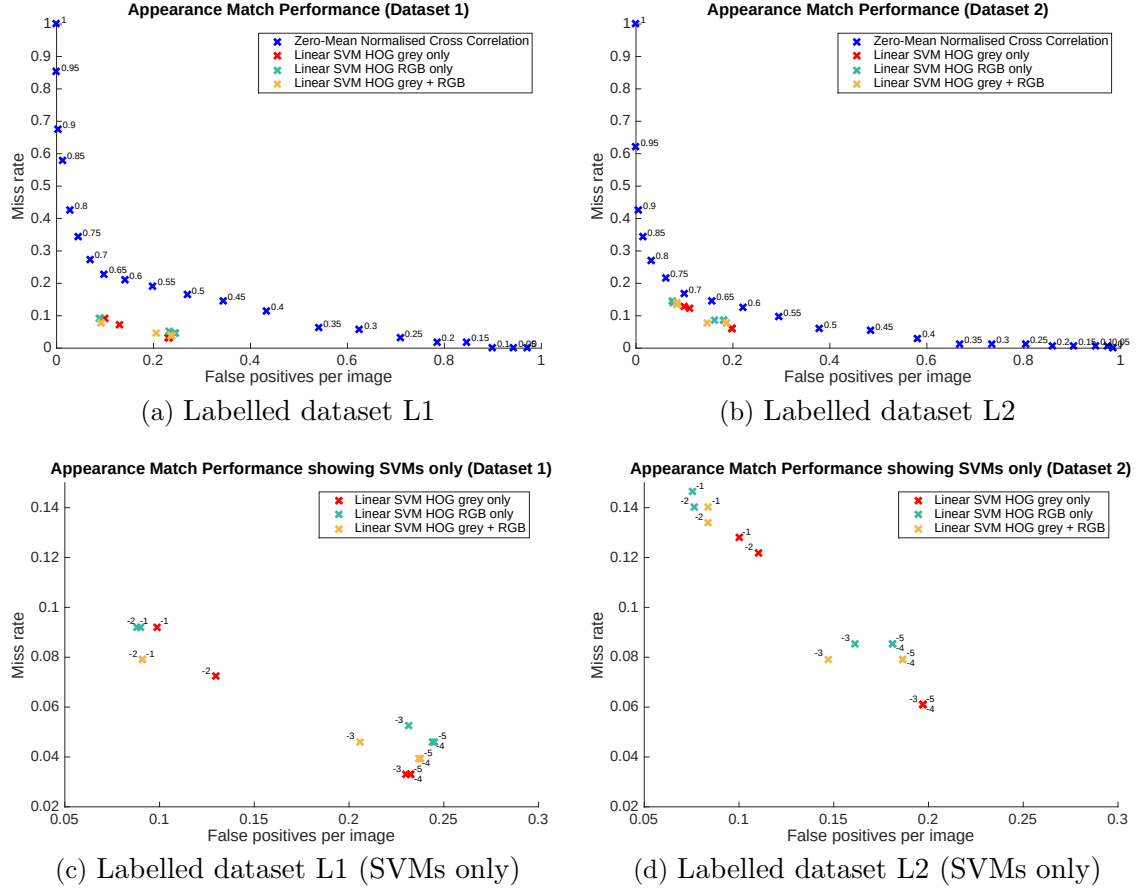


Figure 4.13: Appearance matching performance results showing miss rate against number of false positives per image. A ‘perfect’ classifier would appear in the bottom left corner – i.e. it misses no true positives and produces no false positives.

In the top figures, blue crosses indicate performance of the ZNCC matching function with the specified threshold T_{ZNCC} . Red, green, and orange crosses indicate performance of the SVM approaches with HOG descriptors computed from greyscale only, RGB channel only, and both grey and RGB, respectively.

For purposes of clarity, the SVM results are shown zoomed in in greater detail in the bottom figures. The number next to each point indicates the exponent of 10 of that SVM’s C parameter: e.g. -1 indicates $C = 10^{-1}$.

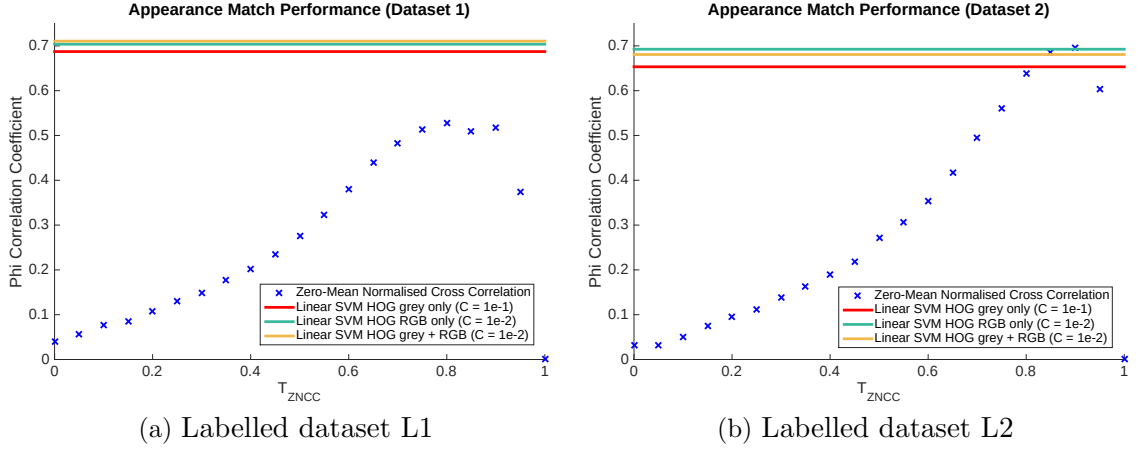


Figure 4.14: Appearance matching performance results showing phi correlation coefficient against T_{ZNCC} for the ZNCC matching approach. Red, green, and orange lines indicate the phi correlation coefficient scored by the SVMs with the best-performing values of C .

performs consistently ‘best’ on both datasets. This is likely because of its sensitivity to slight differences in exposure levels—recall the differences in exposure levels found between datasets in Section 4.8.2.2—which if compensated for with a lower T_{ZNCC} threshold, would allow unacceptably many false positives through. The SVMs appear to perform more consistently.

We assess each classifier using the *phi correlation coefficient*. This computes a score from -1 to 1 where values of -1 and 1 mean data are classified 100% incorrectly and 100% correctly, respectively. A value of 0 means data are classified no better than random prediction. It is calculated as:

$$\Phi = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (4.15)$$

where TP and TN are the number of true positives and negatives, respectively (i.e. *correctly* classified as a positive or negative), and FP and FN are the number of false positives and negatives, respectively (i.e. *incorrectly* classified). Figure 4.14 shows the phi correlation coefficients for varying values of T_{ZNCC} referenced against SVMs with the best-performing values of C (which happen to be the same on both

datasets). From this we can read that the SVMs generally have better discriminatory power than ZNCC except for one single value of $T_{ZNCC} = 0.9$ in dataset L2. Again we note that no single value for T_{ZNCC} performs best in both datasets, and the SVMs trained on colour data perform marginally better than those without.

Based on there results, we choose to use an SVM trained on HOG descriptors derived from both greyscale and colour patches, with $C = 10^{-2}$.

4.9.3 How Many Landmarks Are Sufficient?

It is trivial to show that solving unconstrained 6-DoF localisation using a camera, with known landmark positions, requires a minimum of three pixel-to-landmark correspondences in the non-degenerate case. By limiting or constraining the model, the number of degrees of freedom and therefore the minimum number of required correspondences can be reduced. For example, with a 3-DoF planar pose model, two pixel-to-landmark correspondences are sufficient. However, if the vehicle motion is not consistent with the restricted model, our pose estimates will quickly diverge. This was shown to occur in [75] in which planar and non-holonomic constraints are applied in order to parameterise relative motion with only one point correspondence.

For this reason we choose to use a 6-DoF pose model. Recall however from Equation 4.6 that we are able to incorporate prior knowledge into our least-squares estimation. By weighting the z , roll, and pitch components of our estimation solution, we can discourage fitting to these parameters whilst still remaining robust to small non-planar movements. In this section we evaluate in simulation the efficacy of this approach over using no prior information whatsoever.

4.9.3.1 Synthetic Data Generation

A set of synthetic scenes were generated using MATLAB. The virtual camera pose was set to $\mathbf{0}$, i.e. at the origin and pointing in the direction of the positive x -axis.

For each experiment, this base scene was modified in two ways:

1. The initial virtual camera pose is corrupted by some random zero-mean Gaussian noise, giving us a ‘true’ pose:

$$\bar{\mathbf{x}} = \mathbf{0} + \mathbf{v}, \quad \mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{V}) \quad (4.16)$$

where the covariance $\mathbf{V}$ of this noise was derived from a real-world set of over 10,000 frame to frame relative pose corrections that the VT&R localiser made to an initial estimate pose. This is done in order to simulate the uncertainty of frame to frame visual odometry motion estimates.

2. A constellation of n lights is randomly generated in a region around the camera. This region was chosen to roughly simulate urban scenes around Oxford, with lights within 5m from the ground, within 10m of the edge of the road, and up to 80m ahead or behind the vehicle. The landmark locations are back-projected into the virtual camera *at its perturbed location* $\bar{\mathbf{x}}$. Intrinsic camera parameters for our actual Ladybug2 camera were used.

Three different values for the prior $\mathbf{Q}^{-1}$ were tested:

1. No prior information ($\mathbf{Q}^{-1} = \mathbf{I}$).
2. Small weighted prior ($\mathbf{Q}^{-1} = \text{diag}(1, 1, 10, 10, 10, 1)$).
3. Large weighted prior ($\mathbf{Q}^{-1} = \text{diag}(1, 1, 10^3, 10^3, 10^3, 1)$).

The number of lights n was varied from 1 to 10. For each value of n , 200 simulations were run in which the camera pose was optimised in the usual way, using $\hat{\mathbf{x}} = \mathbf{0}$ as an initial guess. Note that no measurement noise was added and no data association outliers are considered; we purely wish to evaluate how each prior configuration interacts with n to produce a corrected pose estimate.

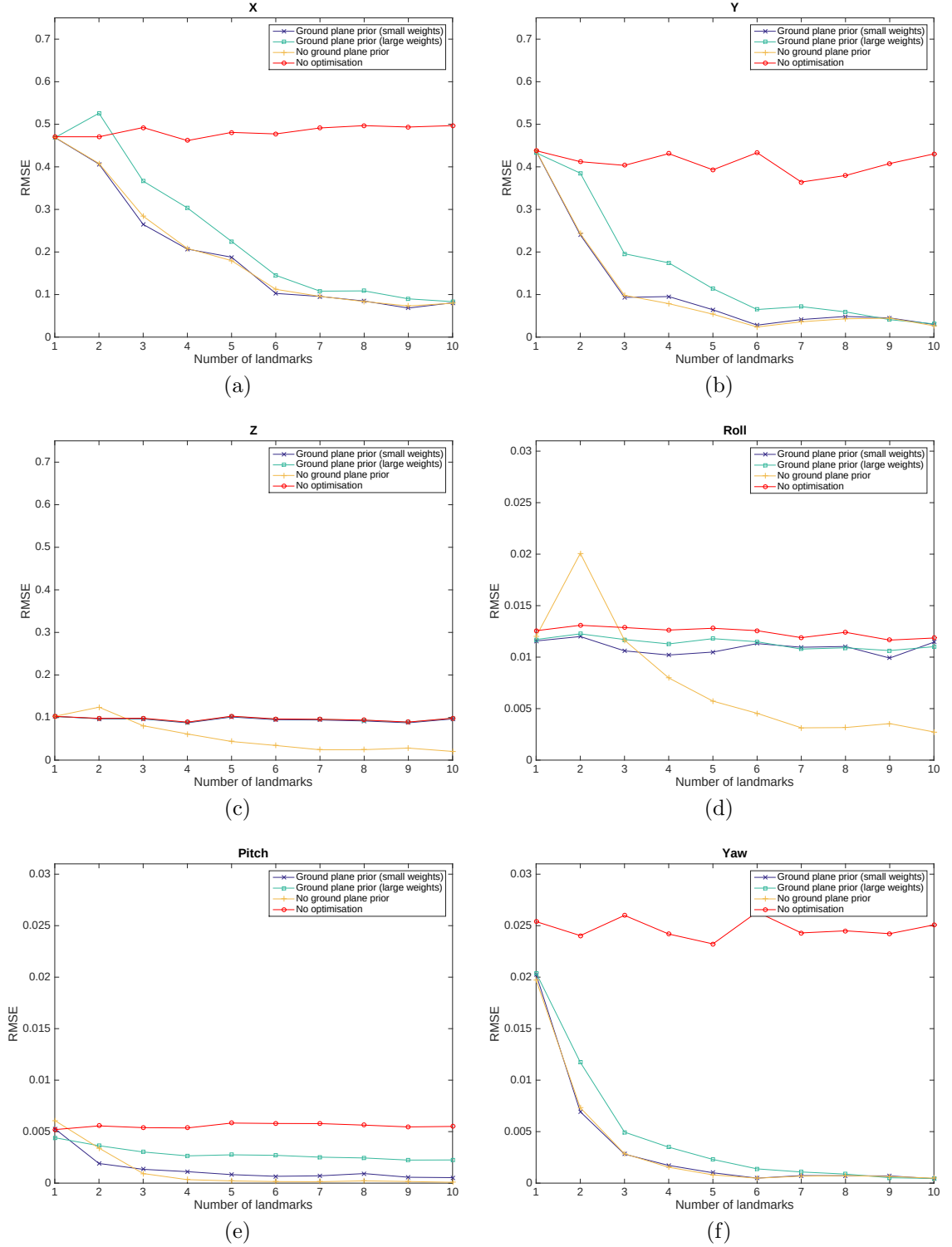


Figure 4.15: Results of the localisation simulations described in Section 4.9.3, comparing the effects of different strengths of soft kinematic constraints. Each graph shows the root mean square error of that pose component against the number of randomly generated synthetic measurements used in the least-squares solve, n . Unoptimised errors are shown in red and serve as a baseline against which to measure performance of the other configurations.

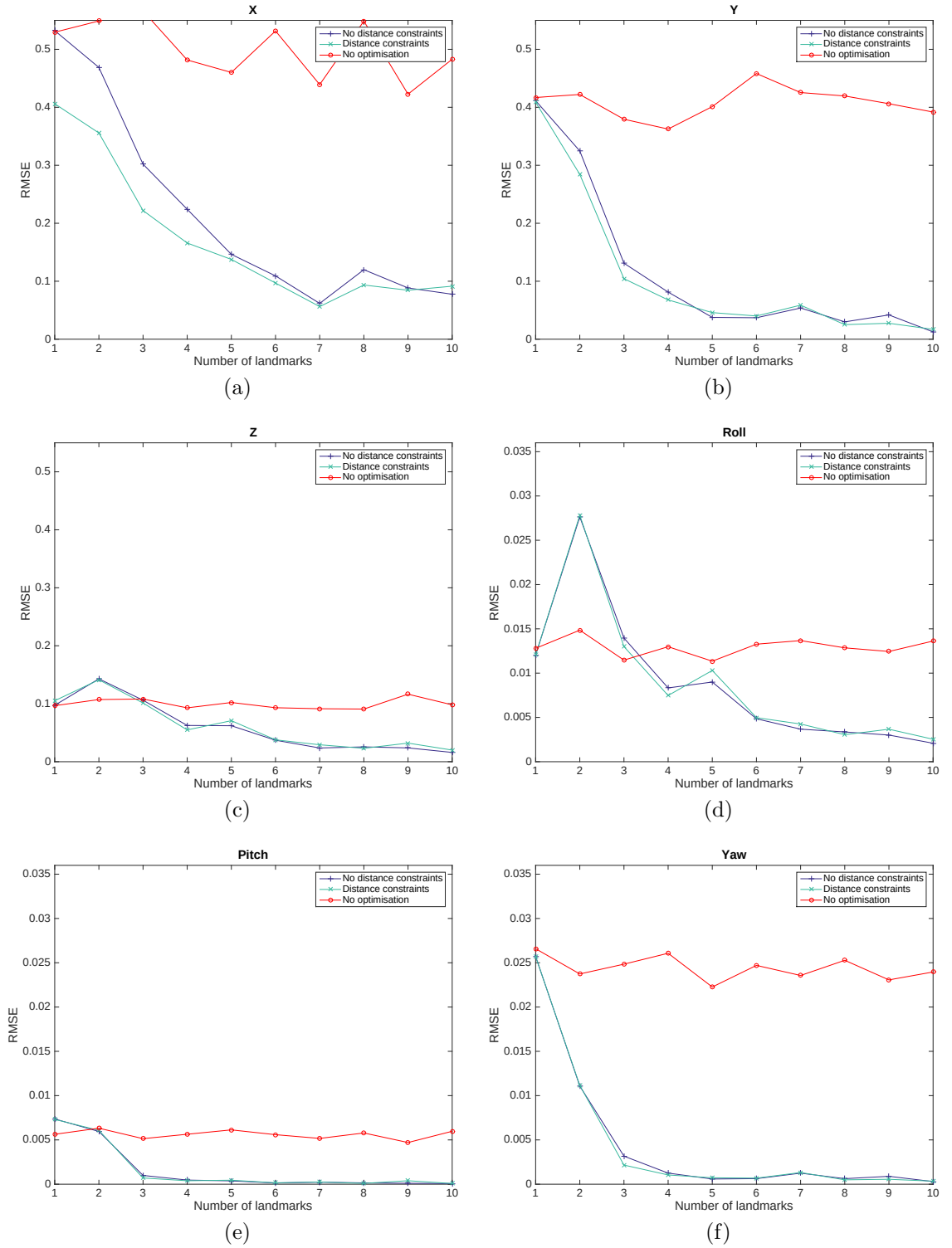


Figure 4.16: Results of the localisation simulations described in Section 4.9.3, comparing the utility of incorporating coarse depth measurements into cost function. Each graph shows the root mean square error of that pose component against the number of randomly generated synthetic measurements used in the least-squares solve, n . Unoptimised errors are shown in red and serve as a baseline against which to measure performance of the other configurations.

4.9.3.2 Simulation Results

The root mean square error (RMSE) between all the corrected pose estimates $\hat{\mathbf{x}}$ and the ‘true’ poses $\bar{\mathbf{x}}$ for each set of experiments were computed. These values are plotted against n in Figure 4.15. In addition, the RMSEs between the *uncorrected* pose estimate $\mathbf{0}$ and the ‘true’ poses are shown as a baseline – a configuration will not be of any use to us if it results in larger errors than doing no optimisation at all.

For $n \geq 3$, it appears that all three prior configurations result in some improvement over the baseline results. Two notable overall trends are evident. Firstly, the completely unconstrained configuration performs overall best, followed by the small weighted prior, followed by the large weighted prior. This would appear to confirm the aforementioned observations in [75]. Secondly, as n increases, our estimation errors decrease. This is not particularly surprising as it can be shown both in theory and in practice (as in for example [83]) that more feature measurements result in more accurate solution estimations.

However, we are most interested in $n < 3$. In general, it appears that every configuration with $n = 1$ performs no better than with no optimisation (except fractionally for yaw and roll). For $n = 2$, we see a notable reduction in estimation error for X, Y, yaw, and pitch components for both the unconstrained and small weighted prior configurations. Using a large weighted prior gives worse results. One interesting result is in the roll component, where the unconstrained configuration performs significantly worse than baseline and both constrained configurations, for $n = 2$. Intuitively we would expect at least one pose component to diverge: the movement is obviously sufficiently non-planar that two measurements are not enough on their own, without any additional constraints, to converge to a sensible solution.

Overall, these results suggest that it is worth optimising our pose estimate even when we have only two inlier measurements. It also appears to be beneficial to introduce a small soft kinematic constraint, biasing towards planar movement, par-

ticularly when $n = 2$.

4.9.3.3 Depth Measurement Simulations

To assess the utility of incorporating coarse depth measurements into the cost function, we ran another set of simulations in the same manner as above with the addition of a single depth measurement for each landmark. Zero-mean Gaussian distributed noise with a standard deviation of 1m was added to simulate some depth uncertainty.

Results for these simulations are shown in Figure 4.16. We compared RMSEs for the baseline configuration (no optimisation) against optimisation with and without the depth measurements. No other kinematic constraints were applied. It can be seen that errors in x are noticeably reduced when depth measurements are taken into account for $n < 5$, with little difference for the other pose components. We can therefore conclude that there is likely some benefit to including coarse depth measurements, particularly for small n .

4.9.4 How Many Appearances Are Sufficient?

In this section we investigate how the number of appearances we store for each landmark affects our system’s ability to reject outliers in the data association stage. As previously remarked upon, lights appear very different at varying distances, and so we would intuitively expect that limiting ourselves to only a subset of appearances would have negative consequences.

As a baseline, we ran our localiser over datasets D1g and D2d, keeping track of the total number of accepted matches P and rejected matches N returned by our appearance matching process, using the full set of appearances available to us for each light (for the purposes of this evaluation it is assumed that these are in fact ground truth inliers and outliers).

The localiser was then run again in several trials with increasingly limited ap-

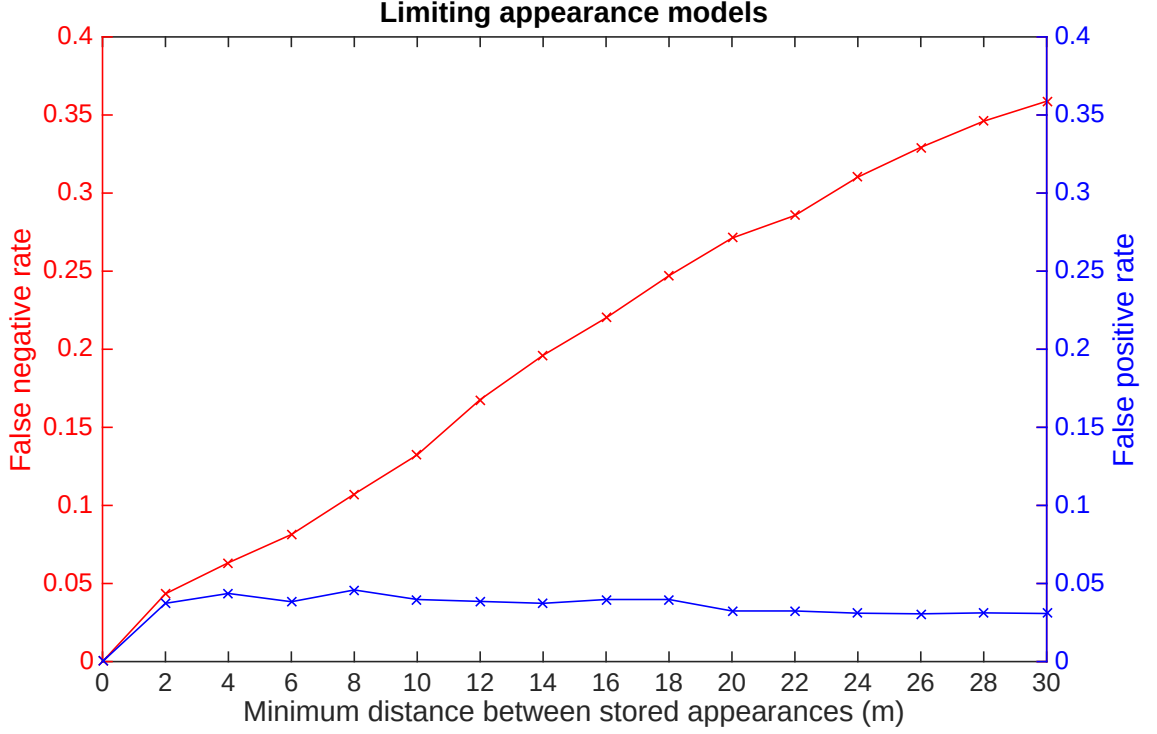


Figure 4.17: Graphs showing the relationship between the number of distinct image patches in our appearance models and the proportion of data associations that are incorrectly rejected (false negatives) or incorrectly accepted (false positives).

Along the x -axis we vary the minimum distance that the vehicle travels, while mapping, before a light’s appearance model is updated with a new image patch that describes what it currently looks like. For example, at $x = 30\text{m}$, we may only store 4 appearance patches to describe a particular light: captured when the vehicle is 0m, 30m, 60m, and 90m farther along its trajectory (from where the light was first observed). In contrast, at $x = 0\text{m}$, we may store more than 90 appearance patches to describe the same light – one for every camera frame in which it was observed.

The y -axis shows the proportion of false negatives (in red) and proportion of false positives (in blue) when matching new observations of lights against their stored appearance models. For example, when we store the appearance of a light only every 30m, we can expect to incorrectly reject around 36% of new observations of that light. This is in contrast to incorrectly rejecting only around 5% of new observations if we store an appearance every 2m. This makes sense as a light’s appearance will change much more drastically over a distance of 30m than over 2m. Similarly, at $x = 30\text{m}$, we incorrectly mistake one light for another, 3% more often than when using the full range of appearances.

pearance models. This was done by setting a minimum separating distance of X meters between each stored appearance (i.e. as if, during the map building process, we had only saved an appearance patch every X meters). For each trial, the appearance matching results were compared with our ‘ground truth’, and the total number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) were recorded. From these, the ratio of false positives to ground truth negatives (equivalently, *false positive rate*) and false negatives to ground truth positives (*false negative rate*) were computed:

$$\text{False positive rate} = \frac{FP}{N} \quad (4.17)$$

$$\text{False negative rate} = \frac{FN}{P} \quad (4.18)$$

These results are plotted against values of X ranging from 2m to 30m in Figure 4.17. As expected, as we make the appearance models more sparse, the false negative rate increases significantly. Interestingly, the false positive rate converges rapidly to a constant, however this is still undesirable.

Overall these results would appear to confirm our intuition that the richer our appearance model for each light, the better we are at verifying observations of that light in the future.

4.9.5 Other Parameters

Unless otherwise stated, appropriate values for the various thresholds and parameters used in our system were determined manually through experimentation. Table 4.3 provides a summary of each parameter and the values that were chosen.

Parameter	Description	Value	Reference
λ_B	Pixel intensity threshold for conversion from greyscale to binary image.	240	Page 109
λ_C	Light tracking cost function threshold.	100	Page 110
λ_D	Maximum distance of landmarks to consider for localisation	80m	Page 116
λ_J	Maximum number of measurements passed to JCBB for data association.	20	Page 117
λ_M	Minimum number of observations required for valid light track.	10	Page 111
λ_T	Maximum variance of candidate landmarks' pairwise location estimates.	300	Page 111
ω	Number of candidate appearances to compare against observation.	5	Page 119

Table 4.3:
Thresholds and parameters used in NightNav with their chosen values.

4.9.6 Overall Localisation Performance

The vehicle was localised in evaluation datasets using the method described in Section 4.6. This section evaluates the overall localisation performance in terms of how far the vehicle travels without a pose estimate from the localiser – that is, purely under the influence of VO (dead-reckoning). We compare our system to our in-house VT&R system operating in the same environment.

For NightNav, we consider a minimum of two matching lights, in addition to a weak kinematic constraint biasing towards planar movement to be sufficient for a successful localisation, as a result of our analysis in Section 4.9.3. For VT&R, a successful localisation requires a minimum of three feature correspondences. Note that both NightNav and VT&R were configured to consider themselves ‘lost’ if they travelled more than 30m by dead-reckoning. In these cases they would exhaustively search from the last known good location in their maps in an attempt to relocalise. In about 20% of cases, some manual intervention was required to reseed the pose – in a complete system this would likely be the job of an in-parallel loop closure or

visual place recognition algorithm, such as FABMAP [17] or SeqSLAM [63].

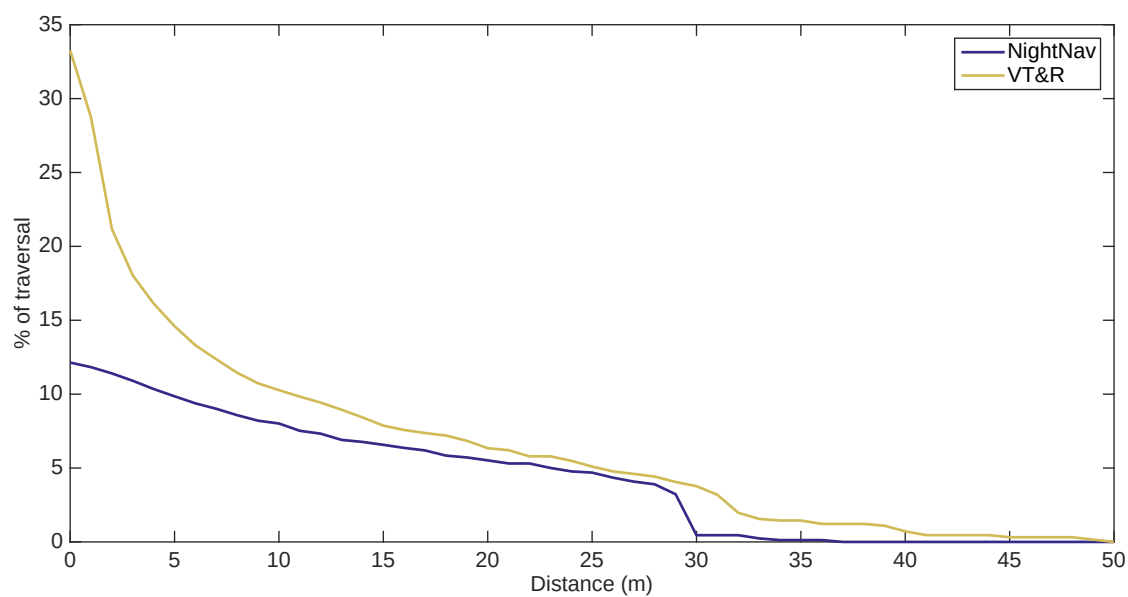
4.9.6.1 Results

Figure 4.18 shows the proportion of the overall total distance traversed during which the vehicle had travelled more than a particular distance without successfully localising. Figure 4.18a and 4.18b show this for both NightNav and VT&R for Route 1 and Route 2, respectively.

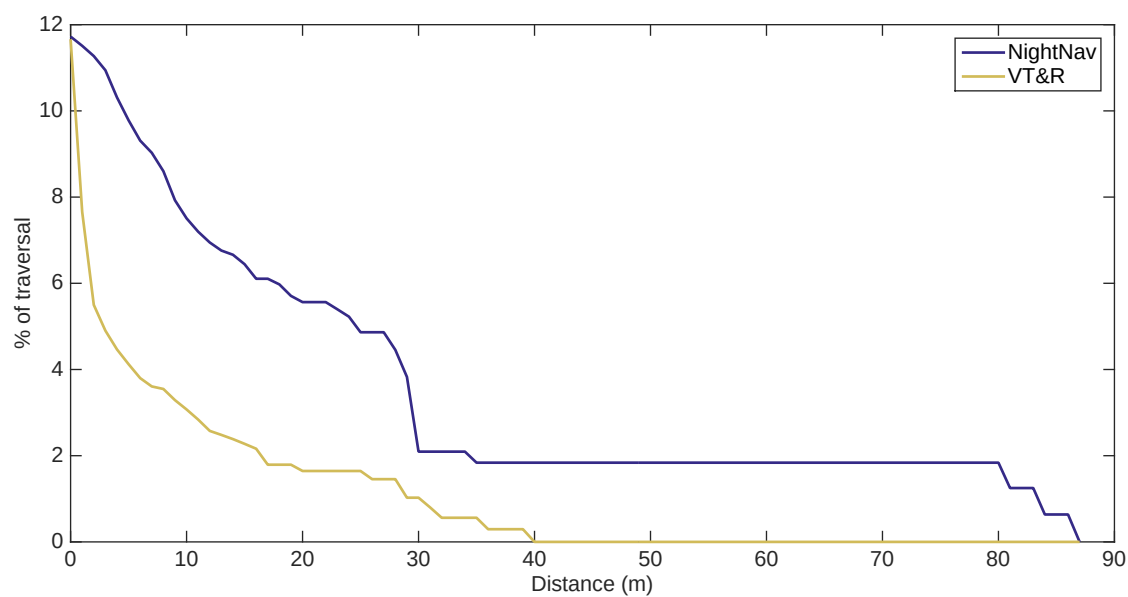
The most immediate implication of these graphs are that with the Route 1 datasets, NightNav appears to get lost less often and for shorter distances than the point feature based VT&R system. The converse, however, is true for the Route 2 datasets. In order to find the reason for this disparity, we must consider the spatial and temporal factors.

Recall from Section 4.8.2 that each route can be divided into regions that broadly correspond to different densities of lights. Figure 4.19 shows results from both routes decomposed by region. Overall, it is clear that a higher density of lights correlates with fewer and shorter localisation failures for both NightNav and VT&R. Similarly, we see the same pattern that NightNav out-performs VT&R in the Route 1 datasets (except interestingly in areas of medium light density), however VT&R does better in the Route 2 datasets. One particular region of note is an exceptionally dark stretch of road where NightNav is unable to localise for around 80m, due to lack of any landmarks. VT&R, however, is surprisingly able to localise using point features on the road surface, illuminated by the vehicle’s headlamps.

Now recall from Table 4.1 that our Route 1 datasets span a period of around five months – the earliest collected in January (from which both the NightNav and VT&R maps are built) and the latest collected in June. Figure 4.20 shows the Route 1 results decomposed by time: January (datasets D1a and D1c), May (dataset D1d), and June (datasets D1e-h). We can see that NightNav continues to out-perform

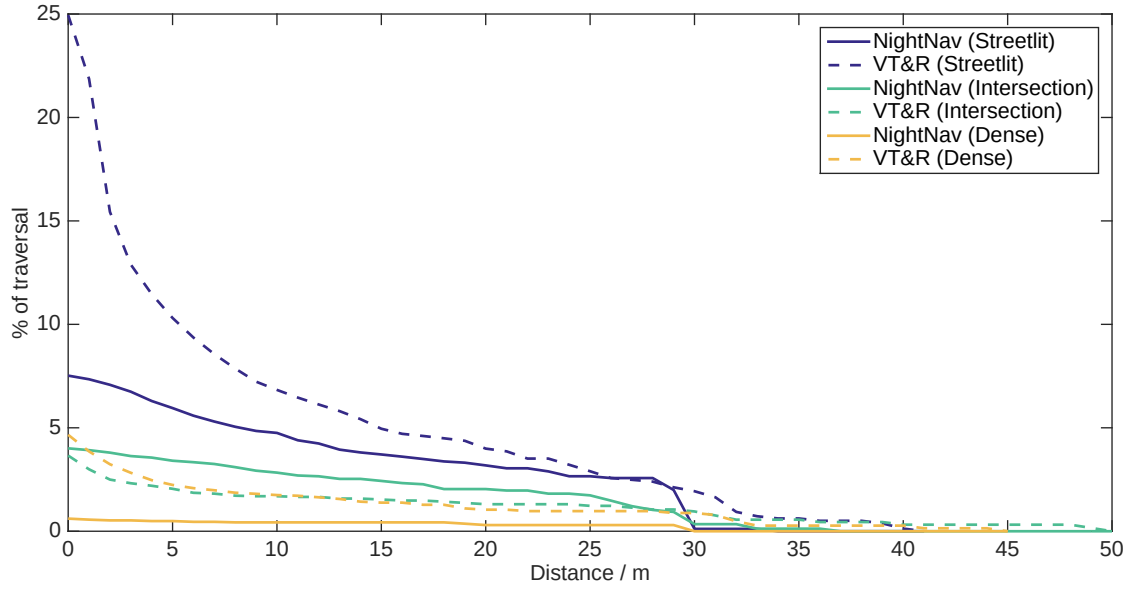


(a) Route 1

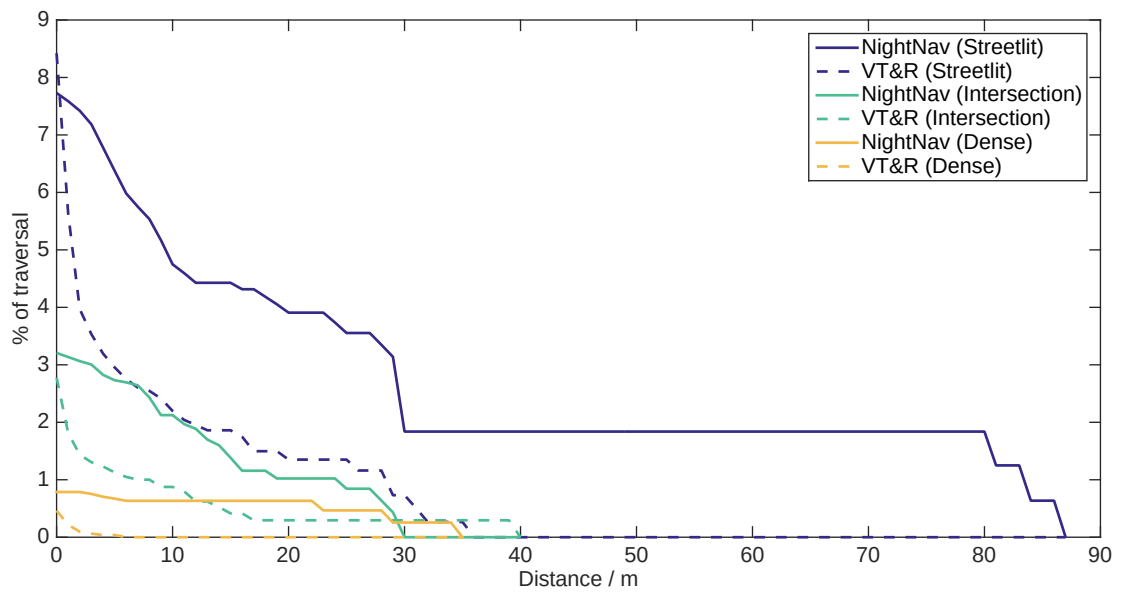


(b) Route 2

Figure 4.18: Graphs showing the proportion of the total trajectory during which each system has travelled more than a certain distance without successfully localising. For example in Figure (a), VT&R spends almost 33% of the total traversal in a state where it is lost for more than 0 metres. In contrast, NightNav spends only 12% of the total traversal in the same state.

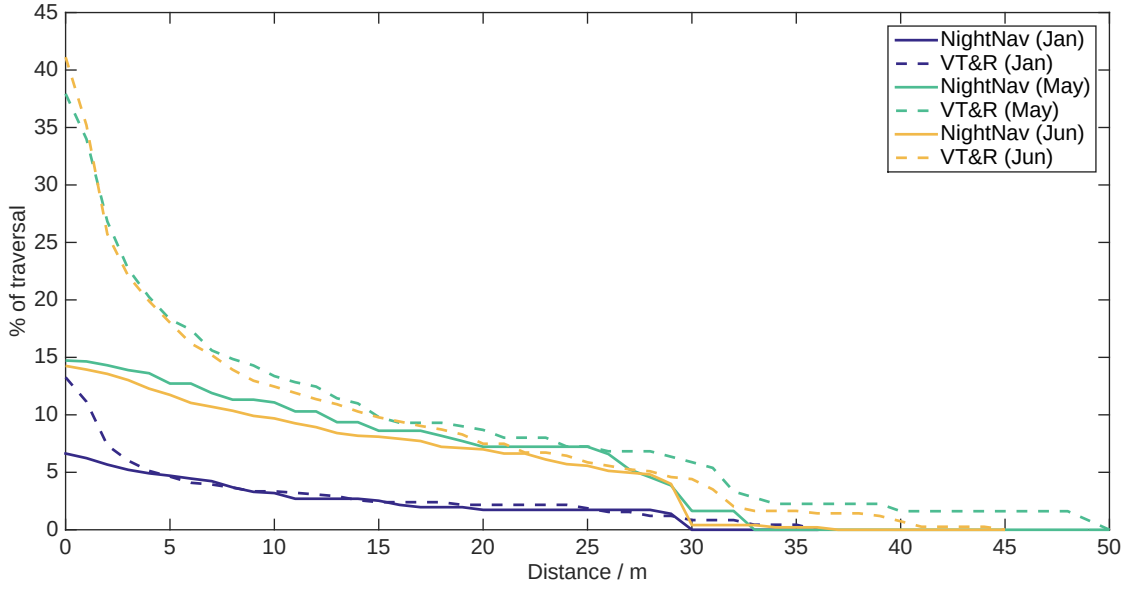


(a) Route 1



(b) Route 2

Figure 4.19: Graphs showing the proportion of the total trajectory during which each system has travelled more than a certain distance without successfully localising, decomposed by region type.



(a) Route 1

Figure 4.20: Graphs showing the proportion of the total trajectory during which each system has travelled more than a certain distance without successfully localising, decomposed by time the data were collected.

VT&R in each time period, however the difference is more pronounced with the May and June datasets. This is consistent with our understanding that point features are not as temporally stable as artificial light sources, hence VT&R is disproportionately negatively affected by five months worth of environmental change. The performance difference with the January datasets are much more marginal, however, therefore they are able to explain some of the performance disparity in the Route 2 results.

From observing the systems at runtime, we believe the rest of the disparity can be explained by time-of-day: January datasets D1a-c were collected relatively early in the evening – between 9-10pm. In contrast, due to the shorter hours of darkness inherent in summer, all of the Route 2 datasets were collected late in the evening – between 11:30pm-midnight, and therefore exhibit far less traffic than the former. It is likely that VT&R is more negatively impacted by traffic presence than NightNav as many of the point features on the road surface and at ground level will be obscured by passing vehicles.

Overall, we can conclude that given ‘almost perfect’ conditions—low vehicular presence, same time-of-day and time-of-year—point feature based VT&R can operate surprisingly well at night. However, under certain imperfect conditions—particularly after months of environmental change, or in the presence of more traffic—NightNav is capable of outperforming VT&R.

4.9.6.2 Cross-track Errors

Figure 4.21 shows the cross track error in x , y , and z , measured relative to the map trajectory. The median x , y , and z errors over all datasets are 0.35m, 0.36m, and 0.31m respectively. In computing these errors we assume that we precisely follow the map trajectory route, as we do not have access to ground truth data, and therefore they can be partly attributed to variations in position when driving in a lane. Large spikes in cross track error, however, occur when a lack of visible lights causes a localisation failure. Points at which NightNav gets lost are indicated by a vertical orange line. Figure 4.22 shows some example imagery from areas where the system gets lost most often.

4.10 Summary

In this chapter we have demonstrated a novel system that builds maps of artificial light sources and subsequently uses that map to localise at night using vision only. By modelling (via a database of appearances) the behaviour of the light sources at night, we are able deal with saturation, blurring, and distance-dependent lens flare in images. Using our database architecture from Chapter 3 we are able to store and access these map and appearance data in a consistent and efficient way. Operations like retrieving a window of stored appearances around an estimated location become very easy to perform. We show that this system is capable of outperforming a point feature based localiser over many kilometres of testing in a city at night.

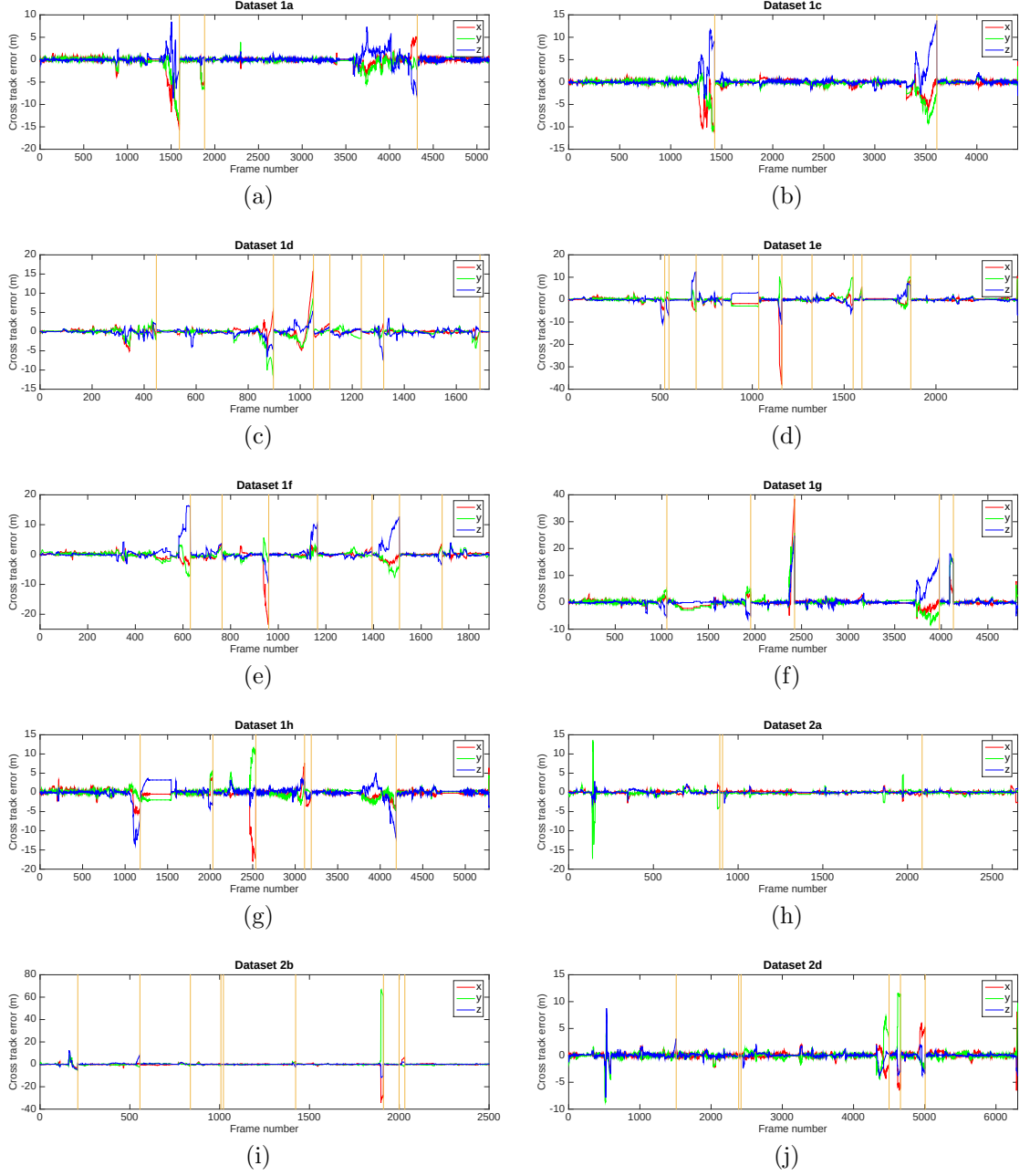
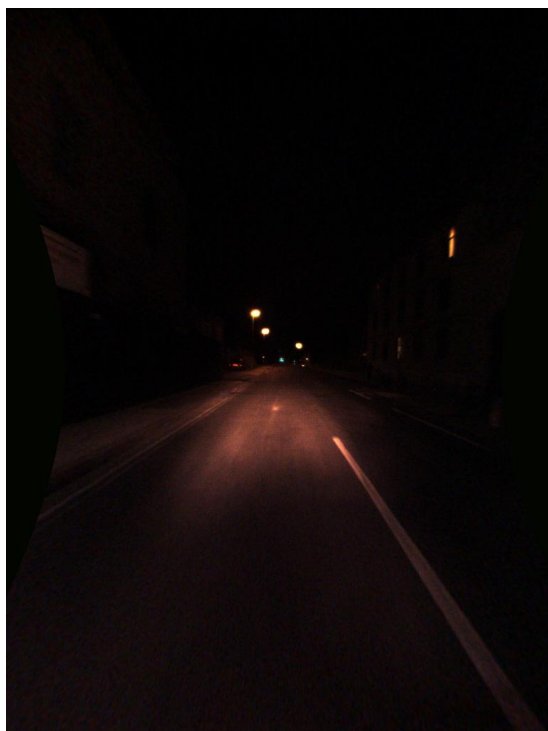
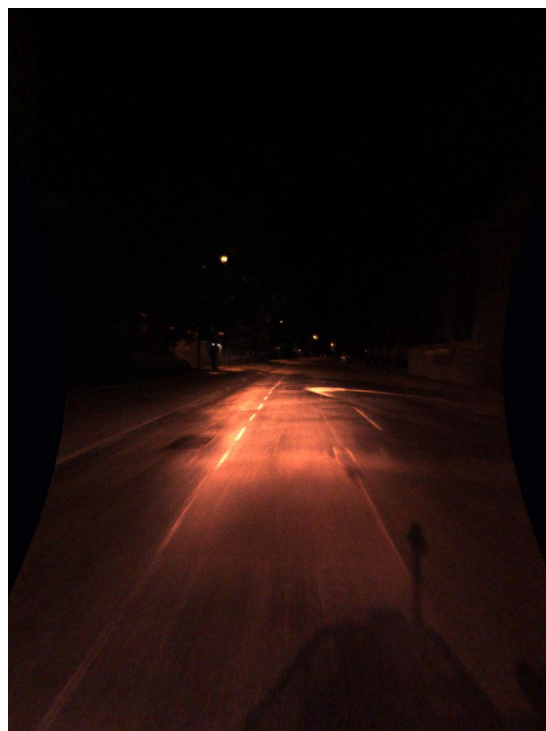


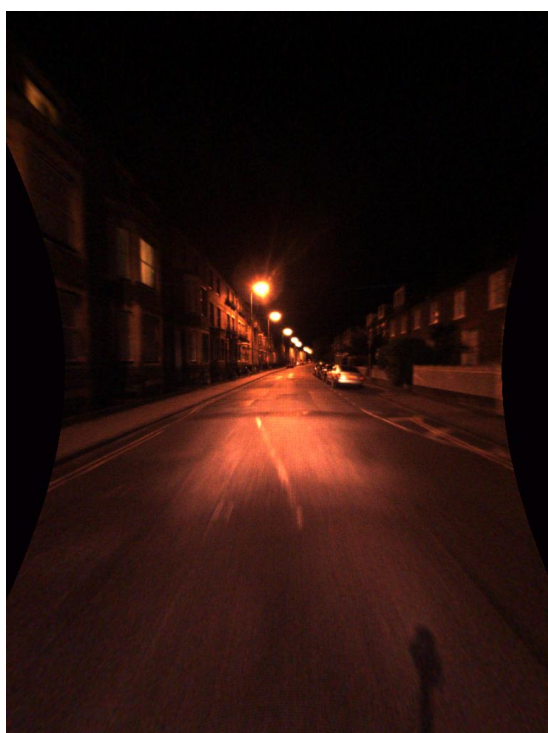
Figure 4.21: Cross track errors for every dataset. Each graph shows the errors in x, y, z for each frame. Vertical orange lines indicate places where the system became 'lost' i.e. no successful localisation for more than 30m. Some example camera images from these places are shown in Figure 4.22.



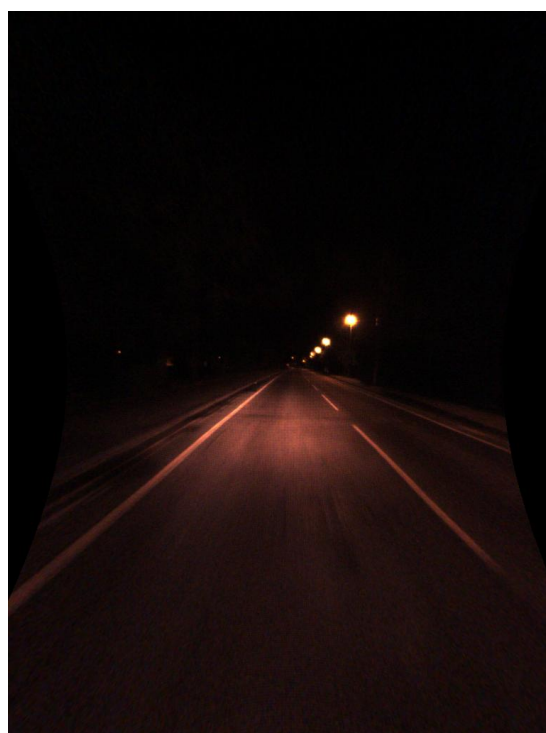
(a)



(b)



(c)



(d)

Figure 4.22: Imagery from areas where our navigation system fails most often. These areas are characterised by very sparse or no landmarks at all (top), or a degenerate configuration of collinearly occurring landmarks directly in front of the vehicle (bottom).

Chapter 5

Conclusions

“ *It’s still magic even if you know how it’s done.* ”

Terry Pratchett, *A Hat Full of Sky*, 2004

In this chapter we summarise the contributions of the thesis and discuss some opportunities for future work.

5.1 Summary of Contributions

The contributions of this thesis are as follows:

- **The design and implementation of a scalable data management framework for robotics applications.** In Chapter 3 we motivated the need to move away from ad-hoc, single-use, disorganised datasets, in order to make the most of the data we spent valuable time collecting and processing. Towards this end, we have designed and implemented a framework, built atop a relational database management system, that enables users to store and query

both raw sensor data and higher-level annotations and labels. Furthermore, data is by default linked together temporally, spatially, and according to the platform and sensors used for data collection, therefore covering a wide variety of query requirements. This, combined with the ability to annotate any and all data in a consistent manner, allows us to realise a rich, ever-growing data repository.

- **Optimisation of the database framework, and the design of a hierarchical architecture to facilitate retrieval of map data, online, on a robotic platform.** In Chapter 3 we demonstrated that different database back-ends were best suited to different tasks, and introduced our master-clone data architecture for graph-based map management. This architecture stipulates that we keep a centralised, master copy of *all* of our data ever recorded in a single PostgreSQL instance. For online tasks, small subsets of this master repository can be cloned to individual SQLite instances and deployed locally to robots as required. In order to meet real-time demands, we analysed and optimised the database system with indexes. Most importantly, we demonstrated its use in several localisation systems, including our own NightNav, managing many kilometres of real-world data. Table 5.1 summarises some real-world database usage statistics of the three navigation systems.

Navigation system	Total data (km)	Total data (hours)	Queries per second
NightNav	51	≈ 2	$O(100)$
LAPS	57	≈ 8	$O(1)$
EBN	1060	≈ 150	$O(10,000)$

Table 5.1: Summary of database usage statistics for the three navigation systems we assessed in this work: NightNav developed in Chapter 4, LAPS described in Section 3.12.1, and EBN described in Section 3.12.3. The total number of kilometres and hours worth of real-world data managed for these systems are shown, as well as the approximate number of queries each system executes per second.

- **A vision-based, teach and repeat, 6-DoF navigation system for autonomous vehicles, operating in urban environments, at night.** In Chapter 4 we demonstrated how navigation at night with cameras can be made possible by using bright, artificial light sources as landmarks. It is noted that light sources undergo drastic appearance change as their distance from the camera changes, providing us additional information for use in data association and pose optimisation. By storing these rich appearance models in our database system, the task of retrieving them becomes trivial. We showed, through the use of data-driven simulations, that surprisingly accurate 6-DoF localisation can be achieved with only a few landmarks: coarse depth measurements, derived from the stored appearance models of each light, and the addition of a soft kinematic constraint, biasing toward planar movement, can aid in convergence of our optimisation solution.

5.2 Future Work

Fully Versioned Database System

Although the database framework presented in Chapter 3 has, and continues to prove its utility, one feature that is missing is some form of *version control* capability. Currently, if a record in the database is modified, the old record will be overwritten and consequently completely forgotten. Although we keep backups, it would be undesirable to have to do a full restore for the sake of a few overwritten records. Document and source control systems such as Git¹ solve this problem by storing discrete, finely granular versions of data, meaning it is possible to revert individual files back to any previous version. Different parallel versions can be worked on by multiple people at the same time, and changes can be merged back into a master

¹<https://git-scm.com/>

repository. A similar distributed architecture and workflow could be imagined for our database system, greatly extending the master-clone architecture detailed in Section 3.8.2.1. How this could be achieved with a relational database management system, however, remains an open question.

Further GraphDB Performance Improvements

Recall from Section 3.7.3 that we encode a graph in a relational database using an edge-list representation in the Edges table. Traversing an edge in the graph using this table has an $O(\log(n))$ cost with a B-Tree index. In massive, highly-connected graphs (i.e. most nodes are connected to most others), this table gets very large and lookups can therefore become inefficient. Traversals of more than a few hops will quickly become impossible to do in real-time. Although we have not found this to be an issue yet, we may consider more efficient alternative approaches in the future. One particular alternative, used by some specialised graph databases like Neo4j, is to implement *index-free adjacency*. Effectively this means storing a list of physical pointers to neighbouring nodes directly alongside each node. The use of physical rather than logical pointers means a separate index does not need to be looked up: the physical pointer is simply dereferenced directly, resulting in $O(1)$ cost edge traversal. As relational databases only support logical pointers, we would need to write a custom back-end in order to implement this feature.

Additionally, recall that for geospatial queries in PostgreSQL we employ the PostGIS extension. SQLite has an equivalent extension named Spatialite². It is our intention to eventually integrate this into our SQLite-based implementation.

²<https://www.gaia-gis.it/gaia-sins/>

Consideration of Human Usage Factors

One of our main goals behind the database framework we developed in Chapter 3 was to make it easier for users to manage, store, and query their data. However, as is the case for any sufficiently sophisticated software framework, ours has a learning curve that users must overcome before they can take advantage of its benefits. This raises the question of how we best encourage users to use the system, and how we take their feedback into account in future iterations of its design. Additionally, when people do choose to use the system, how do we ensure they use it *correctly* and that the quality of data is maintained? An easy-to-use database is not of benefit to anyone if it becomes an unmanageable mess itself due to neglect and abuse. We believe that answers to these questions will become clearer over time and that good documentation, education, and tools will help to alleviate problems that arise.

Improve Localisation Runtime Performance

Many of the components of our nighttime localisation system have not yet been optimised for real-time performance. In particular, run time of the joint compatibility branch and bound (JCBB) algorithm is exponential in the number of measurements, hence why in Section 4.6.2 we limit the number of input measurements it is given. However, if most of these measurements are ambiguous, the JCBB interpretation tree can still grow intractably large. One avenue of future work would be to investigate alternatives to JCBB that do not search the entire hypothesis space. These include the randomised joint compatibility (RJC) algorithm by Paz et al. [71], and the 1-point RANSAC algorithm by Civera et al. [13]. Both of these claim performance commensurate with JCBB, but at a hugely reduced run time complexity, allowing them to meet real-time requirements.

Appearance Model Improvements

Recall that we store the appearance of lights from different locations in order to (1) reject outliers in our data association step, and (2) regularise our cost function by inferring coarse depth measurements via Equation (4.4). The latter quantitative model is, however, only really valid for point light sources like street lights. If we considered a variety of shape models, for example, one for planar light sources like windows, we could derive even more information from them, potentially further improving our pose estimates.

Mapping Improvements

Our map building approach, described in Section 4.5, uses a nearest-neighbour method to track lights independently through a sequence of camera frames. Although effective, it would be useful to further evaluate where this method fails, and whether maps could be improved by using a more sophisticated multi-target tracking approach. This may afford our system greater robustness, and provide the ability to handle more complicated clusters of lights.

5.3 Concluding Remarks

This thesis has demonstrated city-scale localisation of an autonomous vehicle in challenging nighttime conditions. Most importantly, it has been shown how a consistent and scalable approach to data management can be used to facilitate this task, and many others. We particularly believe that future navigation systems would benefit greatly from utilising a database framework like ours. We hope that the work presented here will inspire future roboticists to delve further into both of these areas.

Bibliography

- [1] T. Bailey, “Mobile robot localisation and mapping in extensive outdoor environments,” Ph.D. dissertation, Citeseer, 2002.
- [2] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded up robust features,” in *Computer vision–ECCV 2006*. Springer, 2006, pp. 404–417.
- [3] R. Bayer, “Organization and maintenance of large ordered indexes,” *Acta Informatica*, vol. 1, no. 3, pp. 173–189, 1972.
- [4] M. Bayes and M. Price, “An Essay towards solving a Problem in the Doctrine of Chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFRS,” *Philosophical Transactions (1683-1775)*, pp. 370–418, 1763.
- [5] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller, “An Atlas Framework for Scalable Mapping,” in *Robotics and Automation, 2003. Proceedings. ICRA’03. IEEE International Conference on*, vol. 2. IEEE, 2003, pp. 1899–1906. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1241872
- [6] R. Brooks *et al.*, “Visual map making for a mobile robot,” in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2. IEEE, 1985, pp. 824–829.
- [7] C. Brunner, T. Peynot, T. Vidal-Calleja, and J. Underwood, “Selective combination of visual and thermal imaging for resilient localization in adverse conditions: Day and night, smoke and fire,” *Journal of Field Robotics*, vol. 30, no. 4, pp. 641–666, 2013.
- [8] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “BRIEF: Binary robust independent elementary features,” in *Computer Vision–ECCV 2010*. Springer, 2010, pp. 778–792.
- [9] J. Castellanos, J. Montiel, J. Neira, J. D. Tardós *et al.*, “The SPmap: A probabilistic framework for simultaneous localization and map building,” *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 5, pp. 948–952, 1999.

-
- [10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: a distributed storage system for structured data,” in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1267308.1267323>
- [11] W. Churchill, “Experience based navigation: Theory, practice and implementation,” Ph.D. dissertation, University of Oxford, 2012.
- [12] W. Churchill and P. Newman, “Experience-based navigation for long-term localisation,” *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1645–1661, 2013.
- [13] J. Civera, O. G. Grasa, A. J. Davison, and J. Montiel, “1-point RANSAC for EKF-based structure from motion,” in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 3498–3504.
- [14] S. Clark and H. Durrant-Whyte, “Autonomous land vehicle navigation using millimeter wave radar,” in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 4. IEEE, 1998, pp. 3697–3702.
- [15] E. F. Codd, “A relational model of data for large shared data banks,” *Commun. ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970. [Online]. Available: <http://doi.acm.org/10.1145/362384.362685>
- [16] P. Corke, R. Paul, W. Churchill, and P. Newman, “Dealing with shadows: Capturing intrinsic scene appearance for image-based outdoor localisation,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 2085–2092.
- [17] M. Cummins and P. Newman, “Appearance-only SLAM at large scale with FAB-MAP 2.0,” *The International Journal of Robotics Research*, vol. 30, no. 9, pp. 1100–1123, 2011. [Online]. Available: <http://ijr.sagepub.com/content/30/9/1100.abstract>
- [18] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.
- [19] A. J. Davison and D. W. Murray, “Simultaneous localization and map-building using active vision,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 7, pp. 865–880, 2002.
- [20] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Voshall, and W. Vogels, “Dynamo: Amazon’s highly available key-value store,” in *Proceedings of twenty-first*

- ACM SIGOPS symposium on Operating systems principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007, pp. 205–220. [Online]. Available: <http://doi.acm.org/10.1145/1294261.1294281>
- [21] A. R. Eisenman, C. C. Liebe, and J. L. Joergensen, “New generation of autonomous star trackers,” in *Aerospace Remote Sensing'97*. International Society for Optics and Photonics, 1997, pp. 524–535.
 - [22] A. Elfes, “Sonar-based real-world mapping and navigation,” *Robotics and Automation, IEEE Journal of*, vol. 3, no. 3, pp. 249–265, 1987.
 - [23] H. J. S. Feder, J. J. Leonard, and C. M. Smith, “Adaptive mobile robot navigation and mapping,” *The International Journal of Robotics Research*, vol. 18, no. 7, pp. 650–668, 1999.
 - [24] P. Furgale and T. Barfoot, “Visual path following on a manifold in unstructured three-dimensional terrain,” in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 534–539.
 - [25] P. Furgale and T. D. Barfoot, “Visual teach and repeat for long-range rover autonomy,” *Journal of Field Robotics*, vol. 27, no. 5, pp. 534–560, 2010.
 - [26] P. Furgale, P. Carle, J. Enright, and T. D. Barfoot, “The Devon Island rover navigation dataset,” *Int. J. Rob. Res.*, vol. 31, no. 6, pp. 707–713, May 2012. [Online]. Available: <http://dx.doi.org/10.1177/0278364911433135>
 - [27] S. Gauglitz, T. Höllerer, and M. Turk, “Evaluation of interest point detectors and feature descriptors for visual tracking,” *International journal of computer vision*, vol. 94, no. 3, pp. 335–360, 2011.
 - [28] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The KITTI dataset,” *The International Journal of Robotics Research*, 2013.
 - [29] F. S. Grassia, “Practical parameterization of rotations using the exponential map,” *Journal of graphics tools*, vol. 3, no. 3, pp. 29–48, 1998.
 - [30] W. E. L. Grimson and T. Lozano-Perez, “Localizing overlapping parts by searching the interpretation tree,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 4, pp. 469–482, 1987.
 - [31] J. Guivant, E. Nebot, and S. Baiker, “Autonomous navigation and map building using laser range sensors in outdoor applications,” *Journal of robotic systems*, vol. 17, no. 10, pp. 565–583, 2000.
 - [32] A. Guttman, *R-trees: a dynamic index structure for spatial searching*. ACM, 1984, vol. 14, no. 2.
 - [33] M. Haklay and P. Weber, “OpenStreetMap: User-generated street maps,” *Pervasive Computing, IEEE*, vol. 7, no. 4, pp. 12–18, 2008.

- [34] W. R. Hamilton, “LXXVIII. on quaternions; or on a new system of imaginaries in algebra: To the editors of the Philosophical Magazine and Journal,” 1844.
- [35] C. Harris and M. Stephens, “A combined corner and edge detector.” in *Alvey vision conference*, vol. 15. Manchester, UK, 1988, p. 50.
- [36] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.
- [37] J. Hawke, C. Gurau, C. H. Tong, and I. Posner, “Wrong today, right tomorrow: Experience-based classification for robot perception,” in *Field and Service Robotics (FSR)*, Toronto, ON, Canada, June 2015.
- [38] A. Howard, “Multi-robot mapping using manifold representations,” in *Robotics and Automation, 2004. Proceedings. ICRA’04. 2004 IEEE International Conference on*, vol. 4. IEEE, 2004, pp. 4198–4203.
- [39] A. Howard and N. Roy, “The robotics data set repository (Radish),” 2003. [Online]. Available: <http://radish.sourceforge.net/>
- [40] S. Hrabar, P. Corke, and M. Bosse, “High dynamic range stereo vision for outdoor mobile robotics,” in *Proceedings IEEE International Conference on Robotics and Automation 2009 ICRA 09*. IEEE, 2009, pp. 430–435.
- [41] A. S. Huang, M. Antone, E. Olson, L. Fletcher, D. Moore, S. Teller, and J. Leonard, “A High-rate, Heterogeneous Data Set From The DARPA Urban Challenge,” *Int. J. Rob. Res.*, vol. 29, no. 13, pp. 1595–1601, Nov. 2010. [Online]. Available: <http://dx.doi.org/10.1177/0278364910384295>
- [42] P. J. Huber *et al.*, “Robust estimation of a location parameter,” *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.
- [43] K. Irie, T. Yoshida, and M. Tomono, “A high dynamic range vision approach to outdoor localization,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 5179–5184.
- [44] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Journal of Fluids Engineering*, vol. 82, no. 1, pp. 35–45, 1960.
- [45] J. J. Leonard and H. F. Durrant-Whyte, *Directed sonar sensing for mobile robot navigation*. Springer Science & Business Media, 1992, vol. 175.
- [46] K. Levenberg, “A method for the solution of certain non-linear problems in least squares,” 1944.
- [47] C. Linegar, “Vast-scale, vision-only localisation in changing environments,” Ph.D. dissertation, University of Oxford, Oxford, United Kingdom, 2016.

- [48] C. Linegar, W. Churchill, and P. Newman, “Work Smart, Not Hard: Recalling Relevant Experiences for Vast-Scale but Time-Constrained Localisation,” in *Proc. IEEE International Conference on Robotics and Automation (ICRA2015)*, 2015.
- [49] A. López, J. Hilgenstock, A. Busse, R. Baldrich, F. Lumbreras, and J. Ser-rat, “Nighttime vehicle detection for intelligent headlight control,” in *Advanced Concepts for Intelligent Vision Systems*. Springer, 2008, pp. 113–124.
- [50] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [51] F. Lu and E. Milios, “Robot pose estimation in unknown environments by matching 2D range scans,” *Journal of Intelligent and Robotic Systems*, vol. 18, no. 3, pp. 249–275, 1997.
- [52] O. Ludwig, D. Delgado, V. Goncalves, and U. Nunes, “Trainable classifier-fusion schemes: An application to pedestrian detection,” in *12th International IEEE Conference On Intelligent Transportation Systems*, ser. 432-437, vol. 1, St. Louis, 2009.
- [53] W. Maddern, A. D. Stewart, and P. Newman, “LAPS-II: 6-DoF day and night visual localisation with prior 3D structure for autonomous road vehicles,” in *Intelligent Vehicles Symposium, 2014 IEEE*, 2014.
- [54] W. Maddern and S. Vidas, “Towards robust night and day place recognition using visible and thermal imaging,” *RSS 2012: Beyond laser and vision: Alternative sensing techniques for robotic perception*, 2012.
- [55] P. C. Mahalanobis, “On the generalized distance in statistics,” *Proceedings of the National Institute of Sciences (Calcutta)*, vol. 2, pp. 49–55, 1936.
- [56] M. Maimone, Y. Cheng, and L. Matthies, “Two years of visual odometry on the Mars Exploration Rovers: Field reports,” *Journal of Field Robotics*, vol. 24, no. 3, pp. 169 – 186, March 2007. [Online]. Available: <http://dx.doi.org/10.1002/rob.v24:3>
- [57] C. McManus, “Learning place-dependant features for long-term vision-based localisation,” Ph.D. dissertation, University of Oxford, Oxford, United Kingdom, 2014.
- [58] C. McManus, W. Churchill, W. Maddern, A. D. Stewart, and P. Newman, “Shady dealings: Robust, long-term visual localisation using illumination invariance,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 901–906.
- [59] C. McManus, P. Furgale, B. Stenning, and T. D. Barfoot, “Visual teach and repeat using appearance-based lidar,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 389–396.

- [60] C. McManus, B. Upcroft, and P. Newman, “Scene signatures: localised and point-less features for localisation,” 2014.
- [61] C. McManus, B. Upcroft, and P. Newman, “Learning place-dependant features for long-term vision-based localisation,” *Autonomous Robots*, vol. 39, no. 3, pp. 363–387, 2015.
- [62] M. J. Milford, I. Turner, and P. Corke, “Long exposure localization in darkness using consumer cameras,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3755–3761.
- [63] M. J. Milford and G. F. Wyeth, “SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1643–1649.
- [64] P. Mühlfellner, P. Furgale, W. Derendarz, and R. Philippsen, “Designing a relational database for long-term visual mapping,” 2015.
- [65] J. Neira and J. D. Tardós, “Data association in stochastic mapping using the joint compatibility test,” *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 6, pp. 890–897, 2001.
- [66] P. Nelson, W. Churchill, I. Posner, and P. 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.
- [67] P. Nelson, C. Linegar, and P. Newman, “Building, Curating, and Querying Large-scale Data Repositories for Field Robotics Applications,” in *International Conference on Field and Service Robotics (FSR)*, Toronto, ON, Canada, June 2015.
- [68] D. Nistér, O. Naroditsky, and J. Bergen, “Visual odometry for ground vehicle applications,” *Journal of Field Robotics*, vol. 23, 2006.
- [69] S. Nuske, J. Roberts, and G. Wyeth, “Extending the dynamic range of robotic vision,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. IEEE, 2006, pp. 162–167.
- [70] R. O’Malley, E. Jones, and M. Glavin, “Rear-lamp vehicle detection and tracking in low-exposure color video for night conditions,” *Intelligent Transportation Systems, IEEE Transactions on*, vol. 11, no. 2, pp. 453–462, 2010.
- [71] L. M. Paz, J. E. Guivant, J. D. Tardós, and J. Neira, “Data association in O(n) for divide and conquer SLAM,” in *Robotics: Science and Systems*, vol. 3, 2007, p. 281.
- [72] W. H. Press, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

- [73] R. Ramakrishnan and J. Gehrke, *Database management systems*. Osborne/McGraw-Hill, 2000.
- [74] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” in *Computer Vision—ECCV 2006*. Springer, 2006, pp. 430–443.
- [75] D. Scaramuzza, F. Fraundorfer, and R. Siegwart, “Real-time monocular visual odometry for on-road vehicles with 1-point RANSAC,” in *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*. IEEE, 2009, pp. 4293–4299.
- [76] D. Scaramuzza, A. Martinelli, and R. Siegwart, “A toolbox for easily calibrating omnidirectional cameras,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 5695–5701.
- [77] J. Serra, *Image Analysis and Mathematical Morphology*. Academic Press, ISBN: 012637242X, 1984.
- [78] D. Sibley, C. Mei, I. Reid, and P. Newman, “Adaptive relative bundle adjustment,” in *Robotics: science and systems*, vol. 32, 2009, p. 33.
- [79] G. Sibley, C. Mei, P. Newman, and I. Reid, “A system for large-scale mapping in constant-time using stereo,” *International Journal of Robotics Research*, 2010.
- [80] G. Sibley, “Relative bundle adjustment,” *Department of Engineering Science, Oxford University, Tech. Rep*, vol. 2307, no. 09, 2009.
- [81] M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman, “The New College vision and laser data set,” *The International Journal of Robotics Research*, vol. 28, no. 5, pp. 595–599, May 2009. [Online]. Available: <http://www.robots.ox.ac.uk/NewCollegeData/>
- [82] A. Stewart, “Localisation using the appearance of prior structure,” Ph.D. dissertation, University of Oxford, Oxford, United Kingdom, 2014.
- [83] H. Strasdat, J. Montiel, and A. J. Davison, “Real-time monocular SLAM: Why filter?” in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2657–2664.
- [84] J. Stuelpnagel, “On the parametrization of the three-dimensional rotation group,” *SIAM review*, vol. 6, no. 4, pp. 422–430, 1964.
- [85] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, “Bundle adjustment—a modern synthesis,” in *Vision algorithms: theory and practice*. Springer, 2000, pp. 298–372.
- [86] C. Valgren and A. J. Lilienthal, “SIFT, SURF & seasons: Appearance-based long-term localization in outdoor environments,” *Robotics and Autonomous Systems*, vol. 58, no. 2, pp. 149–156, 2010. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0921889009001493>

- [87] V. Vapnik, “Pattern recognition using generalized portrait method,” *Automation and remote control*, vol. 24, pp. 774–780, 1963.
- [88] M. Waibel, M. Beetz, J. Civera, R. D’Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo *et al.*, “RoboEarth,” *Robotics & Automation Magazine, IEEE*, vol. 18, no. 2, pp. 69–82, 2011.
- [89] M. Warren, D. McKinnon, H. He, and B. Upcroft, “Unaided stereo vision based pose estimation,” in *Australasian Conference on Robotics and Automation*, G. Wyeth and B. Upcroft, Eds. Brisbane: Australian Robotics and Automation Association, 2010. [Online]. Available: <http://eprints.qut.edu.au/39881/>
- [90] M. Warren, D. McKinnon, and B. Upcroft, “Online calibration of stereo rigs for long-term autonomy,” in *International Conference on Robotics and Automation (ICRA)*, Karlsruhe, 2013.
- [91] Z. Zhang, “Parameter estimation techniques: A tutorial with application to conic fitting,” *Image and vision Computing*, vol. 15, no. 1, pp. 59–76, 1997.