# Large-Scale Urban Localisation with a Pushbroom LIDAR

Ian Baldwin

New College


Supervisor:

Professor Paul Newman

Mobile Robotics Group

Department of Engineering Science

University of Oxford

October 2012

Ian Baldwin
New College

Doctor of Philosophy
October 2012

**Large-Scale Urban Localisation with a Pushbroom LIDAR**

**Abstract**

Truly autonomous operation for any field robot relies on a well-defined pyramid of technical competencies. Consider the case of an autonomous car - we require the vehicle to be able to perceive its environment through noisy sensors, robustly fuse this information into an accurate representation of the world, and use this representation to plan and execute complex tasks - all the while dealing with the uncertainties inherent in real world operation.

Of fundamental importance to all these capabilities is localisation - we always need to know where we are, if we are to be able to plan where we are going (or how to get there). As road vehicles make the push towards becoming truly autonomous, the system's ability to stay accurately localised over its operating lifetime is of crucial importance - this is the core issue of lifelong localisation.

The goals in this thesis are threefold - to develop the hardware needed to reliably acquire data over large scales, to build a localisation framework that is robust enough to be used over the long-term, and to establish a method of adapting our framework when necessary such that we can accommodate the inevitable difficulties present when operating over city-scales.

We begin by developing the physical means to make large-scale localisation achievable, and affordable. This takes the form of a stand-alone, rugged sensor payload - incorporating a number of sensing modalities - that can be deployed in either a mapping or localisation role.

We then present a new technique for localisation in a prior map using an information-theoretic framework. The core idea is to build a dense retrospective sensor history, which is then matched statistically within a prior map. The underlying idea is to leverage the persistent structure in the environment, and we show that by doing so, it is possible to stay localised over the course of many months and kilometres.

The developed system relies on orthogonally-oriented ranging sensors, to infer both velocity and pose. However, operating in a complex, dynamic, setting (like a town centre) can often induce velocity errors, distorting our sensor history and resulting in localisation failure. The insight into dealing with this failure is to learn from sensor *context* - we learn a place-dependent sensor model and show that doing so is vital to prevent such failures.

The integration of these three competencies gives us the means to make inexpensive, lifelong localisation an achievable goal.

## Statement of Authorship

This thesis is submitted in fulfilment of the requirements for the degree of Doctor of Philosophy. This thesis is my own work, and except where otherwise identified to the contrary, describes my research.

Ian Baldwin, New College

**Funding** This work was kindly funded by the Office of Naval Research (ONR) N00014-08-1-0337 under Dr Kamgar-Parsi.

## Acknowledgements

My thanks go firstly to my supervisor, Professor Paul Newman. His wisdom and wit have been greatly appreciated over the past four years, and it was a privelege to be a part of the Mobile Robotics Group.

Thanks also to the members of MRG; it has been an invaluable experience, and I look forward to following the progress of the group in the years to come.

I would also like to thank my wonderful family for their unwavering, continued support throughout my time here.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Where am I? Lifelong localisation

The focus of this thesis is accurately localising a road vehicle in a map over its lifetime. Localisation is the keystone that supports the entire autonomous vehicle architecture - planning, navigating, and exploring. Almost every task that we would like an autonomous vehicle to accomplish will rely on these tasks, thus accurate localisation is a vital competency.

One can imagine the scope of such an endeavour when we consider the variable faces of the towns and cities we live in. Consider the stark differences in appearance between summer and winter, or the contrast between a previously empty lot and a new building site. Can we be sure that our experiences navigating the world a year ago will be just as relevant today?

The question of longevity in localisation is an important one to resolve if autonomous vehicles are to become a widespread reality. We cannot be tasked with repeatedly re-building a map of our surrounds; we require *persistence*.

The use of the Global Positioning System (GPS) has provided means for accurate localisation in areas unencumbered by foliage, buildings, and other urban

infrastructure - in short, the areas we most want to operate in. Urban centres induce multi-path errors, and in some cases cause complete signal drop-out. Should the absence of a GPS signal preclude us from confidently navigating a parking structure, or an underground tunnel?

Much focus has been given to the problem of simultaneously exploring and navigating - however, consider the usage profile of a typical driver. It is posited that the time spent navigating around a *known* environment will far exceed the number of times that it will be necessary to explore and map. We therefore make explicit the assumption that we are provided with a prior map, and develop the hardware and software to make localisation within this map - and within a budget - an achievable goal.

## 1.2 Large-scale urban localisation

In this thesis, we develop the means - in hardware and software - to provide a vehicle with the ability to stay accurately localised over the course of its operating life using an inexpensive sensor suite. This thesis is comprised of three core components:

1. Building systems for data acquisition and synthesis.

2. Building the inference framework necessary to localise in a map over extended periods.

3. Plastically adapting our sensor models given their surroundings, in order to adapt to a fluid, dynamic world.

Our hope is to provide a compelling narrative for the use of simple, inexpensive sensors for the express purpose of localisation within a map. Chapter 2 serves as a introduction to localisation, and Chapter 3 details some of the hardware and software requirements for operating on massive datasets - these can be skipped by readers

familiar with such concerns. The core of this thesis lies in the localisation strategy Chapter 4, which is extensively validated in Chapter 5, and the context-driven sensor adaptation in Chapter 6.

## 1.3 Contributions

The core contributions of this thesis are:

1. An information-theoretic localisation framework, validated extensively with real-world data collected over more than a year, that
   (a) is robust in the face of drastic scene change, and
   (b) leverages a *context*-specific sensor model, allowing for localisation over large scales.
2. A stand-alone localisation/mapping unit, complete with both hardware and software.
3. An implementation of a city-scale simulator, capable of providing large-scale realistic point cloud data from simulated ranging sensors.

## Thesis Structure

This thesis opens with a review of vehicle localisation techniques, and a background on localisation techniques for field robots. This chapter also includes a primer on the registration of sensor data, which will be used extensively in later chapters as part of a complete system.

The following chapter describes in depth the development of a simulator, which leverages a powerful procedural model generator to build simulated datasets of arbitrary cities, in real time. The synthesis of such data is useful for the exploration

and validation of algorithms, alleviating some of the logistical burden of data collection. This is followed by the description of the design and build of a stand-alone dual-purpose localisation/surveying system, intended to leverage the algorithms and concepts presented in this thesis.

Given the tools for rapidly acquiring (or synthesizing) sensor data over city scales, Chapter 4 develops the core strategy for localisation within a prior map. We focus here on the principles underlying the information-theoretic estimation approach, and also develop fall-back localisation techniques that leverage the advantages of the sensor modality chosen. The idea is to use a retrospective "swathe" of sensor data - built by fusing the output of multiple sensor modalities into a coherent whole - and matching the statistics of this sensor history with our prior map. This is then validated extensively over real-world data, spanning over a year, and traversing more than 100 kilometres.

Of course, operation over large scales has the potential to cause sensor degradation when confronted with highly dynamic scenarios, and we are not immune to this. The core idea in dealing with these situations is by *learning* a spatially-dependent sensor model that is capable of filtering the data such that the effect of these errors is drastically reduced, allowing for persistent localisation within a dynamic, complex workspace.

We take the position that it is not enough to rely on a single privileged model over the course of the vehicle operating life, and that doing so incorporates a brittleness that is easily dealt with in the proposed approach.

## 1.4 Publications

**Road vehicle localization with 2D push-broom LIDAR and 3D priors.** The core of Chapter 4 was published in the IEEE International Conference on

Robotics and Automation (ICRA) in Minneapolis, Minnesota in May, 2012.

**Laser-only road-vehicle localization with dual 2D pushbroom LIDARs and 3D priors.** This work was submitted, and accepted, for publication in the IEEE/RSJ International Conference on Intelligent Robots and Systems in Vilamoura, Portugal.

**The New College Vision and Laser Data Set.** The New College dataset was the first in a new track of data papers published in the International Journal of Robotics Research (IJRR) in May, 2009. Both the dataset, and stand-alone software tools for accessing and manipulating the data are available online [1].

[1] `http://www.robots.ox.ac.uk/NewCollegeData/`

# Chapter 2

# Background

## 2.1 Introduction

The idea of truly autonomous road-vehicles is fast becoming a reality. Initiatives like Google's Car project and the Defense Advanced Research Projects Agency (DARPA) Grand and Urban challenges have inspired great advances in autonomous driving technology. Figure 2.1 shows the podium finishers of the 2005 Grand Challenge:


(a) **1** Stanley (b) **2** Sandstorm (c) **3** H1ghlander

Figure 2.1: **Grand Challenge medallists.** (a) Stanley, Stanford (b) Sandstorm, CMU (c) H1ghlander, CMU.

The Grand Challenge was a showcase for the capabilities of some of the first autonomous vehicles, with 5 vehicles completing the entirety of the 212km desert course. The successor to the Grand Challenge was the 2007 Urban Challenge, in which the contestants were required to navigate a simulated urban setting, com-

plying with the rules of the road and dealing with simulated traffic, and other competitors. Figure 2.2 shows the final podium finishers of the challenge:

(a) **1**: Boss (b) **2**: Junior (c) **3**: Victor Tango

Figure 2.2: **Urban Challenge medallists.** (a) "Boss", Tartan Racing, CMU (b) "Junior", Stanford (c) "Victor Tango", Virginia Tech.

The "urban" nature of this challenge required teams to deal with other traffic, GPS dropout - mainly due to the "urban-canyon" effect, and signal degradation from overhanging foliage - and a host of other real-world difficulties.

The VisLab[1] Intercontinental Autonomous Challenge was an autonomous vehicle endurance-test, running from July to October of 2010. The challenge started in Parma and terminated in Shanghai, and was designed to validate the concept of an autonomously driven vehicle - albeit in a "follower" mode - over continental scales.

All of these challenges served to highlight the advancement in driverless technology across a broad spectrum of competencies - including navigation, path-planning, trajectory planning, and localisation. Localisation is of particular interest, as it forms a lower-level element on the technology pyramid that is required for successful autonomous operation. Consider Figure 2.3, a depiction of the information flow for "Boss":

[1] `http://viac.vislab.it/`

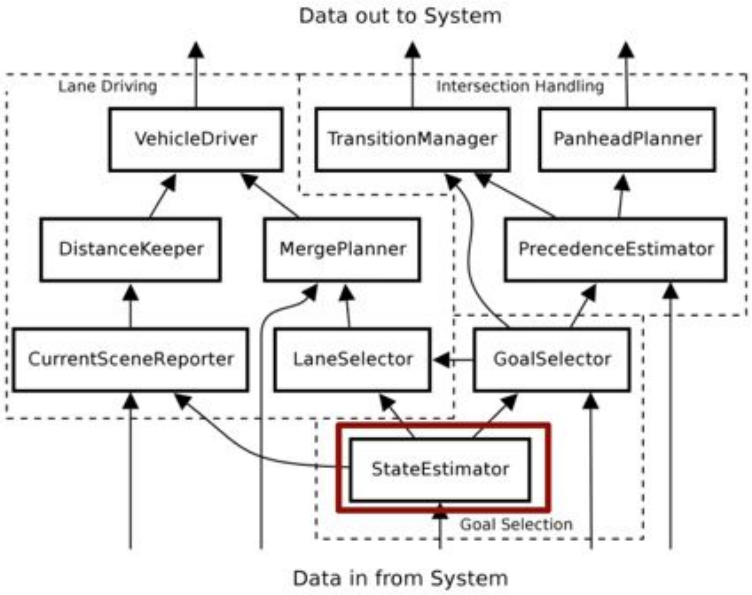



Figure 2.3: **Information flow for an autonomous vehicle.** The system diagram for the autonomous vehicle, "Boss". Note the core position of the estimation module (emphasis added).

Note the location of the state-estimator module at the root of the hierarchy - knowing the state of the vehicle is required for any higher-level behaviours.

In this thesis we are concerned with long-term, large-scale localisation of a road vehicle in an expansive workspace. The ability to localise against a given map over long periods of time is of considerable importance. We will make use of such a map (the definition of which will be formalised in Chapter 4), and develop systems to robustly estimate vehicle position across a range of conditions.

The vehicles in these challenges made use of a wide variety of sensor suites, incorporating a host of different sensing modalities, from *active* sensing - RADAR, LIDAR (light detection and ranging) - to *passive* sensors, predominantly vision-based. In the following section we examine the current state-of-the-art in terms of localisation for road vehicles, and the corresponding sensor modalities used.

## 2.2 Localisation on the road: an overview

This thesis is concerned with long-term, large-scale road vehicle localisation across weather conditions using low cost LIDAR. The core focus here is on providing robust estimates of vehicle state in a previously surveyed workspace, to facilitate higher level behaviours such as path and trajectory planning.

We do not consider the mapping problem - the number of times we will require a vehicle to explore unknown territory will be dwarfed by the number of times the vehicle will operate in a known environment. Therefore, we will assume that we have access to a previously obtained map of the workspace, and focus on the accurate localisation within this map over extended periods. The following sections partition localisation solutions into those which are predominantly LIDAR-based, versus those relying mainly on vision - although for any system, the final output will be a fused combination of these estimates.

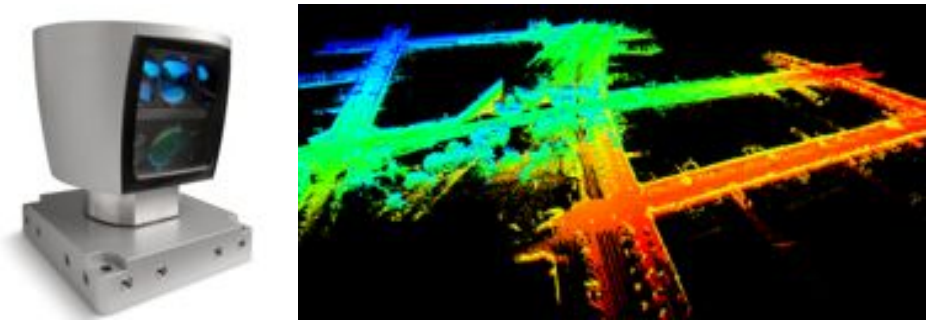
### LIDAR-based Localisation

Road-vehicle localisation in this context has been explored by a number of authors - and was crucial for all systems in the Urban Challenge. All of the challenge teams made use of a heterogeneous mixture of LIDAR sensors, both 2D and 3D. The dimensionality of the sensor here refers to how it perceives the world - if the observations are a full characterization of the environment, i.e. sensor readings are in $\mathbb{R}^3$, this is a 3D sensor. A statically mounted planar scanning-LIDAR sensor is a 2D sensor[2].

[2]Note that depending on the frame of reference (i.e. global vs sensor) these could also be called 2D and 1D sensors.

### 3D LIDAR

The Stanford entry - Junior - relied on a combination of reflectivity-based localisation and curb-based landmark matching. This reflectivity-based localisation by Levinson et. al [55],[56] is the most similar in spirit to this thesis. In terms of perception the authors make use of a 3D ranging sensor - the Velodyne HDL-64E - to perform both mapping and localisation [65]. The Velodyne is an *actuated* LIDAR sensor - range-readings are obtained by rotating a bank of 64 laser diodes at 10Hz - and was used extensively by teams during the challenge. Figure 2.4 shows the Velodyne LIDAR, and a map built using this sensor around the city of Karlsruhe:



(a) HDL-64E (b) 3D dataset

Figure 2.4: **Velodyne HDL-64E 3D LIDAR .** The Velodyne was a core component for mapping and localisation for many teams in the Urban challenge. (a) shows the sensor, and (b) a map acquired by [66].

The mapping component for Junior requires an offline relaxation step, which is then used at run-time in conjunction with a Sequential Monte Carlo - or *particle* - filter to estimate $\mathbb{SE}2$ pose. A particle filter can be thought of as a discrete approximation to a continuous probabilistic quantity - more about this in Section 2.3.1. The authors also made use of velocity data from the on-board INS during the prediction step, as well as a "sensor-resetting" trick - as proposed by [54] - in which a small number of particles are drawn from an "oracle", in this case, a Global Positioning System (GPS) sensor. Continually replenishing the particles in this fashion avoids

particle depletion - a known issue with particle-filter methods - although it does require access to a GPS.

While the results presented are impressive, the goal of this thesis is to develop a system that has no reliance on velocity estimates from an INS, global pose estimates, or an expensive 3D sensor like the Velodyne. We also seek to remove the requirement for the expensive offline relaxation step. The broad vision here is to augment vehicles with sensor payloads that are inexpensive, do not require modifications to the vehicle itself, and can be used in environments that present difficulties for GPS-based systems (urban canyons, high-latitudes, and so on).

In [51], the authors again make use of a Velodyne scanning LIDAR unit in a GPS-attenuated environment (a parking structure) to perform mapping and localisation. A representation of the workspace is built up by developing *multi-level surface maps* - a 2.5 dimensional-representation of the environment. To localise, the authors again make use of a particle filter to estimate the distribution over pose state. Particle depletion is avoided by employing techniques from [31].

Although 3D sensors like the Velodyne are useful for many tasks (obstacle detection, mapping and so on) we restrict ourselves in this thesis to only consider planar scanning LIDAR sensors. There are multiple reasons to eschew 3D sensors including high cost, and calibration challenges (both *extrinsic* and *intrinsic*) amongst others. In the next section, we consider localisation techniques that focus on 2D LIDAR.

### 2D LIDAR

The Carnegie Mellon Urban Challenge entry, [97] estimated vehicle position relative to the known (or estimated) road shape. This is an interesting concept, and one that will be pursued in this thesis; it is not necessary to have a globally metric map for the purpose of localisation, when all that is needed is the *relative* position with respect to some local representation of the world.

The vehicle had access to GPS data through the on-board Applanix POS-LV 220 INS, and while the authors note that the precision of this particular INS is on the order of $0.1m$, disruptions of the GPS signal from overhanging vegetation cause severe degeneracy - this is something that we encounter in this thesis, and is detailed in Chapter 4. The authors utilise declined SICK LIDARs to detect the passage of road markings, which are then used to register the perceived data within the map, providing continuous vehicle estimates (although this system was not actually active during the Challenge finals).

The Team Victor Tango entry [2] (a Hybrid Ford Escape) was equipped with a NovAtel Propak LB+ INS with OmniStar HP corrections, and used a bank of short-range SICK scanning LIDARs to perform curb-based localisation on the road-network. The authors note explicitly the effect of GPS "pop" - a noticeable change in position estimates when the vehicle transitions between areas of poor and good signal coverage - the resulting trajectory estimates are highly discontinuous (this effect is noticed in Chapter 4). To compensate for this behaviour, a separate "de-coupled" state-estimate is computed, using solely inertial and odometric data, and fused in a filtering framework.

Similar to the work presented in this thesis - in terms of sensing modality and scope - is that of Bosse et al. [13]. In this work, the authors utilise a rooftop-mounted 2D LIDAR for the purposes of large-scale outdoor mapping. Local maps - built from performing Iterative Closest-Point (ICP) scan-matching over successive LIDAR scans - are integrated into a global map by the Atlas framework [12]. ICP is a widely used technique in robotics for both mapping and localisation, and is detailed further in Section 2.3.2. The authors validate the system by mapping over 50 kilometres of urban road around Brisbane.

Whilst the scale of this endeavour is impressive, we eschew the use of 2D LIDAR maps in this thesis - our experiences using solely 2D LIDAR to stay localised within

a single map over the course of several months show that this is not a feasible representation for robust, long-term localisation performance - this is shown in Chapter 4. While we also make use of a 2D LIDAR, we will deploy it in such a way that we acquire **3D** maps. Such maps provide a much richer description of the world, and are better suited to long-term localisation tasks - a fact that is quantified and made explicit in section Chapter 4.

In the aerospace community, Carle et al. [19] make use of a prior in the form of an elevation map - a realistic assumption for a planetary rover. The authors develop a Multi-frame Odometry-compensated Global Alignment (MOGA) algorithm, that matches 3D features from the 3D prior with similar features observed from an on-board 3D LIDAR - this global alignment is typically conducted sparsely along the trajectory, with the inter-scan trajectory estimates obtained through Visual Odometry.

For completeness, we also consider work done in the field of Airborne Laser-Swathe Mapping (ALSM). Swathe-mapping arises naturally from the ground-mapping techniques employed by aerial vehicles, in which a powerful LIDAR is mounted perpendicular to the direction of travel of the aircraft, and "swathes" are built up from the ego-motion of the vehicle and the observed LIDAR data. In [95] the authors consider simulated LIDAR data taken of the ground from an aircraft overflight of a target environment, with the inter-swathe registration providing an estimate of the vehicle trajectory - however, as a proof of concept, no real-world experiments were presented. In [41] the authors build an offline post-processed database of planar patches, extracted from previous traversals of the target environment. Run-time data is then matched against this database by estimating the transformations between map and run-time planar surface estimates - this approach relies heavily on the decomposition of the scene into identifiable planar patches.

In this section, we have outlined representative LIDAR-based localisation sys-

tems, for both 2D and 3D LIDAR. In this thesis, we make exclusive use of 2D LIDARs - the active nature of the sensor, and its correspondingly robust, reliable performance is the prime motivating factor. In the next section, we explore the use of vision-based systems for localisation.

## Vision-based Localisation

Visual Odometry has had possibly the most engaging success story of all the aforementioned methods - the authors in [20] describe the VO system used aboard the Mars *Spirit* rover, as shown in Figure 2.5:

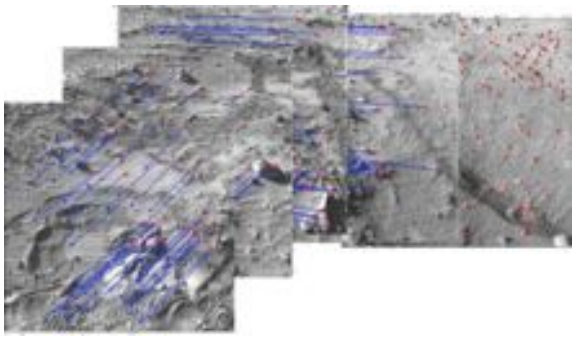


Figure 2.5: **Visual Odometry.** Consecutive frames from the Mars rover *Spirit*, detected features in red and inter-frame correspondences highlighted in blue - these correspondences are used to determine the camera ego-motion.

VO uses features identified across multiple image frames - highlighted in red in Figure 2.5 - and uses these correspondences to estimate the ego-motion of the camera. This is of course an *odometric* measure, subject to long-term drift.

Badino, Huber and Kanade [3] develop a hybrid topological-metric, or "topo-metric" vision-based localisation system that is capable of providing metric pose estimates and resolving topological localisation queries. Similar to our approach, a workspace is surveyed with a system equipped with a variety of sensors - cameras, laser range-finders, and GPS. A database of visual and point features is then built offline, with runtime localisation performed by a means of a Bayesian filter.

A localisation approach similar in spirit to the work in this thesis is known as Visual Teach and Repeat (VTR)[37]. In this work, a stereo-camera is used to build a manifold world representation consisting of previously-visited submaps (the *teach* phase), which is then used in subsequent revisits for localisation (*repeat*). VTR has been demonstrated to work robustly over large scales for path repetition. However, due to the limited view of the stereo camera used, this approach can suffer from relatively small convergence basin - this is a problem that is not encountered with our chosen sensing modality in this thesis.

The authors of "SeqSLAM" [62] present a method for dealing with vastly-varying appearance change over a given route, using image sequences - as opposed to image features - to perform topological localisation within a previous traversal.

In [85], the authors make use of a monocular camera as the sole means of generating full 6-DOF poses from a previously-surveyed workspace. In this approach, Normalised Information Distance is used as the metric in a multi-modal image and LIDAR framework to compare the run-time appearance of a scene with a prior map captured by a survey vehicle.

In [21] the authors again make use of a well-equipped survey vehicle to perform a survey run, recording landmark features from a stereo camera. Subsequent runs through the same environment attempt to localise within this canonical "experience" - a localisation failure prompts the system to save the run-time data as a *new* experience. In this Experience-Based Navigation (EBN) framework, by building up experiences over many traversals of a route, the authors show the system can remain localised across greatly varying seasonal conditions - however, the system still suffers from a fundamental limitation of stereo-cameras in that lateral deviations cause near-field feature drop-out, which is not the case with the LIDAR-based approach in this thesis.

In [68] the authors develop a real-time pose-estimation system for road-vehicles

by incorporating priors from overhead imagery with a state-of-the-art Relative Bundle-Adjustment (RBA) [82] approach to produce global pose estimates.

Although we do not make use of vision techniques for localisation in this thesis, they are an important facet of the final system - this is articulated in Chapter 3. Given that our focus is now exclusively on LIDAR, we turn to a brief synopsis of LIDAR-based localisation in a probabilistic framework, followed by a comparison of an alternate localisation approach - scan-based registration.

## 2.3 Localisation within a prior map

Localisation can be thought of as a process that explains the sensor observations of a vehicle as a function of its position in the world. As the platform moves through the environment, sensor readings are accumulated which we seek to explain through the trajectory of the vehicle in its surrounds.

For our localisation problem, we are concerned with estimating the *Special Euclidean* 2 ($\mathbb{SE}2$) pose of the vehicle, consisting of Cartesian $(x, y)$ position and orientation $(\theta)$, relative to some fixed coordinate system. This representation is sufficient for both path and trajectory planning for a road vehicle - i.e. we do not require the full 6 degrees-of-freedom (DOF) pose (similar arguments are posited in [55],[51]). Figure 2.6 shows an illustration of the coordinate conventions used throughout this thesis:

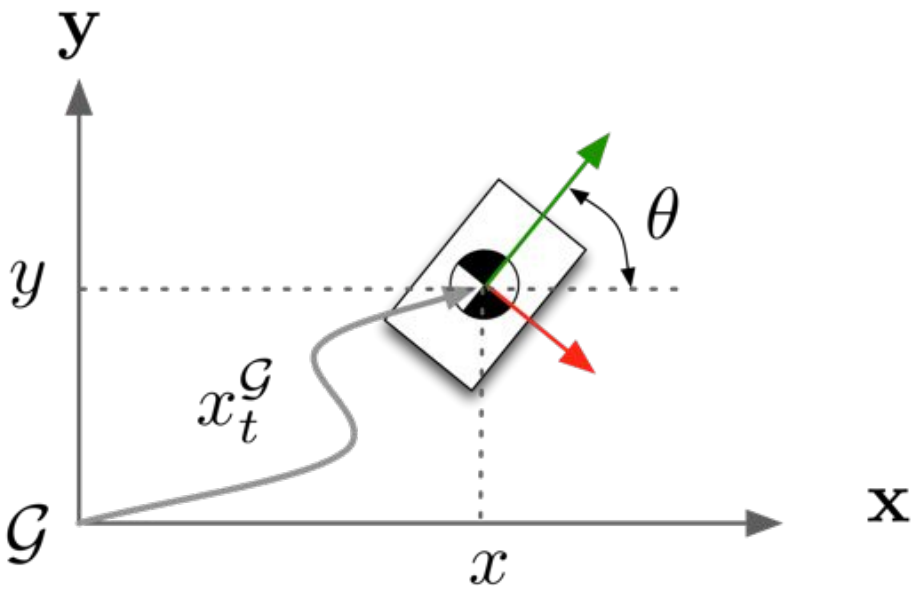



Figure 2.6: **Coordinate systems.** $x_t^{\mathcal{G}}$ is a pose at time $t$, consisting of both position and orientation with respect to some global fixed coordinate frame, $\mathcal{G}$. Vehicle-frame **x** and **y** axes are shown in red and green, respectively.

The pose of the robot $x_t$, consisting of Cartesian position $[x\ y]^T$ and orientation $[\theta]$, is expressed relative to a global fixed coordinate system, $\mathcal{G}$. We are particularly concerned with estimating this transformation relative to a given *map* - the goal here is to develop a localisation system for a road-vehicle that can be used *robustly* over the long-term - we are not concerned with *exploration*, or mapping in an unknown environment.

In the following section, we briefly outline the theoretical foundation of the scanning LIDAR, followed by a brief synopsis of probabilistic vehicle localisation.

### 2.3.1 Probabilistic localisation with ranging sensors

LIDAR - the optical equivalent of RADAR - makes use of a *collimated*[3], *coherent*[4] beam of light to perform ranging measurements. The LIDAR can be thought of as a time-of-flight sensor - a beam of light is emitted from a source, and the time

[3]Light in which the constituent rays have been made parallel
[4]Rays are in phase with each other

between emission and subsequent reception of the signal is used in conjunction with the speed of light to estimate the distance.

However, this direct method requires clocks that are accurate to the order of pico-seconds - therefore, most off-the-shelf ranging units estimate the distance by examining the phase-shift [69] of the returned light. Most surfaces will induce diffuse reflection of LIDAR light, with the notable exception of highly-polished (or retro-reflective surfaces), and materials that are transparent (e.g. glass). The LIDAR sensor emits light at a given frequency, and then measures the phase difference between the emission beam, and that returned by the target surface. Knowing this phase difference $\omega$ and the frequency of the beam $f$, the distance travelled by the beam is given by:

$$d = \frac{\omega}{2\pi}\lambda \qquad \text{where} \qquad \lambda = \frac{c}{f}$$

and $c$ is the speed of light. Figure 2.7 shows a commercially available scanning-LIDAR unit - manufactured by SICK - used extensively throughout this thesis:


Figure 2.7: **SICK scanning LIDAR.** An LMS-151 scanning-LIDAR unit, used throughout this thesis. The unit in this image is mounted upside-down in a custom, rapid-prototyped mount.

Scanning LIDARs similar to Figure 2.7 have been used extensively in indoor robotics for many years. An early success using 2D LIDAR for localisation and mapping over long-term was Minerva [89], a museum tour-guide robot that operated for several weeks in the Smithsonian National Museum of American History. The robot made use of both a horizontally-mounted scanning LIDAR and cameras to localise itself within an *occupancy-grid* [34] representation of the world. An occupancy grid is a discretized world view, in which grid-cell values correspond to the belief of occupancy:

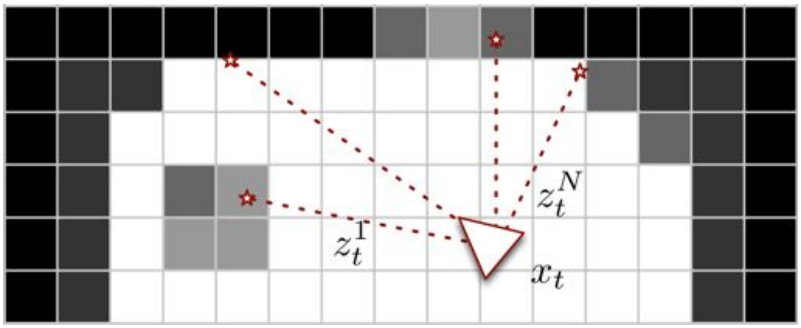



Figure 2.8: **2D occupancy grid.** A 2-dimensional occupancy grid of a toy environment. Cell values correspond to the belief of occupancy (darker cells indicate a higher belief). The occupancy grid is a discretized representation of a map, $\mathbf{m}$. $x_t$ corresponds to the *pose* of the robot at time $t$. Also shown are individual sensor measurements $\{z_t^1, \cdots, z_t^N\}$ from an on-board sensor.

The occupancy grid map, $\mathbf{m} = \{m_1, \cdots, m_2\}$ is a collection of cells, each with a *belief* over the occupied state of the cell. Occupancy grids are simple, discrete approximations of the underlying map, $\mathbf{m}$, although they can be expensive to store for large environments - a $100m$ x $100m$ grid representing an outdoor scene with granularity $.5m$ has $40k$ cells.

Visible in Figure 2.8 are sample measurements $\{z_t^1, \cdots z_t^N\}$ from an on-board sensor, which for a LIDAR consist of range and bearing readings. For each reading, we can construct an associated *likelihood model*, which is a **conditional** probability distribution given what we know about the physical measurement process of the

LIDAR, and the map in which we are operating. Figure 2.9 shows such a conditional probability for an exemplar $p(z_t^i \mid x_t, \mathbf{m})$ [5] :

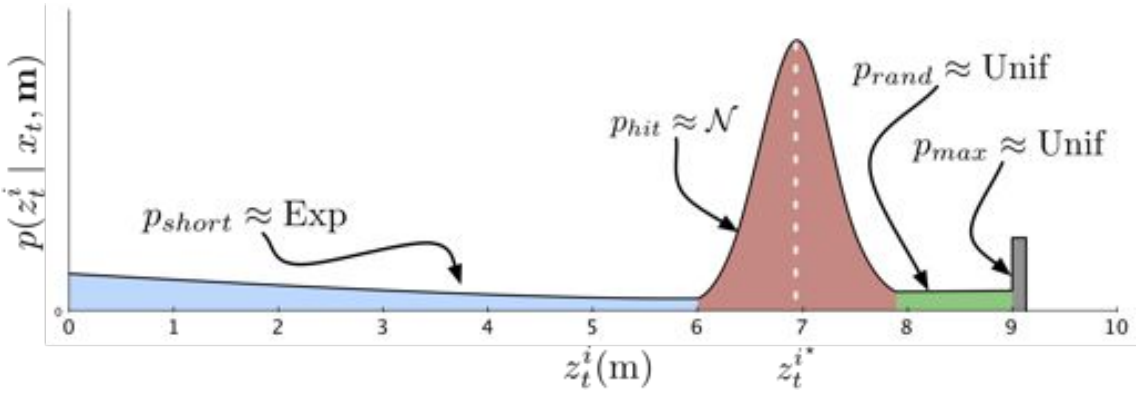



Figure 2.9: **LIDAR beam model.** A beam-model used in the measurement likelihood, inspired by a similar figure in [90]. The beam-model is comprised of a $p_{short}$ component, modelled by an exponential distribution, a $p_{hit}$ component (Gaussian), a "random" component (Uniform) and a max-range component (Uniform).

This conditional probability model is a *measurement likelihood* - it is a principled way of expressing how *likely* the observed reading is *given* the current state and the map. In Figure 2.9, $z_t^{i^\star}$ represents the *expected* reading from the sensor, obtained from ray-casting through the scene. The beam-model is comprised of multiple components - a "short" component $p_{short}$, modelling the probability of a premature return, $p_{hit}$ modelling the hit component, and two uniform components modelling noise and a max-range occurrence. The measurement likelihood is now defined as a product:

$$p(z_t \mid x_t, \mathbf{m}) = \prod_{i=1}^{N} P(z_t^i \mid x_t, \mathbf{m}) \tag{2.1}$$

Section 2.3.1 amounts to a strong *independence* assumption amongst individual beams that is typically not true in real-world LIDAR returns. However, an often-

[5] In a widely-accepted abuse of notation, we use $p(X)$ to represent $p(X = x)$ for legibility

used practice is to employ a *subset* of the LIDAR beams - this sub-sampling in beam-space makes this independence assumption more valid, and it tends to work well in practice.

Localisation using a map is expressed in Bayesian fashion, where - for any pose $x_t$ and associated sensor readings $z_t$ - we can formulate the resulting probability distribution over state as:

$$p(x_t \mid z_t, \mathbf{m}) = \frac{p(z_t \mid x_t, \mathbf{m}) \cdot p(x_t \mid \mathbf{m})}{\int p(z_t \mid x', \mathbf{m}) \cdot p(x' \mid \mathbf{m}) \, dx'} \tag{2.2}$$

where $p(x_t \mid z_t, \mathbf{m})$ is the *posterior* probability distribution over pose given the observed sensor data and map, $p(z_t \mid x_t, \mathbf{m})$ is the *likelihood* of the sensor data given the pose and map - and $p(x_t \mid \mathbf{m})$ is a *prior* probability distribution over the pose in the map.

Of course, a single observation $z_t$ will be insufficient to perform localisation in the map with any degree of certainty. We therefore consider the entire trajectory of the robot, consisting of a chain of poses and measurements. Given a sequence of poses, $x_{0:t-1}$, and the associated measurements $z_{1:t}$, the posterior distribution over the current pose $x_t$ is:

$$p(x_t \mid x_{0:t-1}, z_{1:t}, \mathbf{m}) = \eta \cdot p(z_t \mid x_{0:t}, z_{1:t-1}, \mathbf{m}) \cdot p(x_t \mid x_{0:t-1}, z_{1:t-1}, \mathbf{m}) \tag{2.3}$$

where $\eta$ is a normalisation factor. An important realization - and one that makes inference tractable - is that given our position in the world, the current sensor data is *conditionally independent* of all previous sensor data. It is of course not

completely independent, but all the information previously observed is encapsulated by knowledge of the pose - it is a *sufficient statistic* [88]. Given this conditional-independence assumption, We can rewrite the likelihood term as:

$$p(z_t \mid x_{0:t}, z_{1:t-1}, \mathbf{m}) = p(z_t \mid x_t, \mathbf{m}) \tag{2.4}$$

The right-most term in Section 2.3.1 is termed the transition - or *motion* model - and can be expressed as:

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, \mathbf{m}) = p(x_t \mid x_{t-1}, \mathbf{m}) \tag{2.5}$$

These conditional independence assumptions are at the core of *Markovian* localisation. As a shorthand - following [88], we now represent the posterior distribution over state as the *belief*:

$$bel(x_t) = p(x_t \mid z_{1:t}, \mathbf{m}) \tag{2.6}$$

and the pre-measurement update (i.e. the distribution over $x_t$ *before* observing $z_t$) as:

$$\overline{bel}(x_t) = p(x_t \mid z_{1:t-1}, \mathbf{m}) \tag{2.7}$$

Given the independence assumptions, and our shorthand for the probability distributions over $x_t$, we can write the localisation process as a Bayesian filter:

**Algorithm 1** Bayesian Filter

1: **procedure** FILTER($bel(x_{t-1}), z_t, \mathbf{m}$)
2: $\quad \overline{bel}(x_t) \leftarrow \int p(x_t \mid x_{t-1}, \mathbf{m}) \cdot bel(x_{t-1})$
3: $\quad bel(x_t) \leftarrow \eta \cdot p(z_t \mid x_t, \mathbf{m}) \cdot \overline{bel}(x_t)$
4: $\quad$ return $bel(x_t)$
5: **end procedure**

The filter takes, as input, the posterior distribution over the previous state, and the current measurements, and computes the posterior distribution over the current state. The recursive nature of this filter is a core part of probabilistic state inference in robotics.

We can represent the probability distribution - over poses and measurements - as a directed graphical model (also known as a Bayesian network):

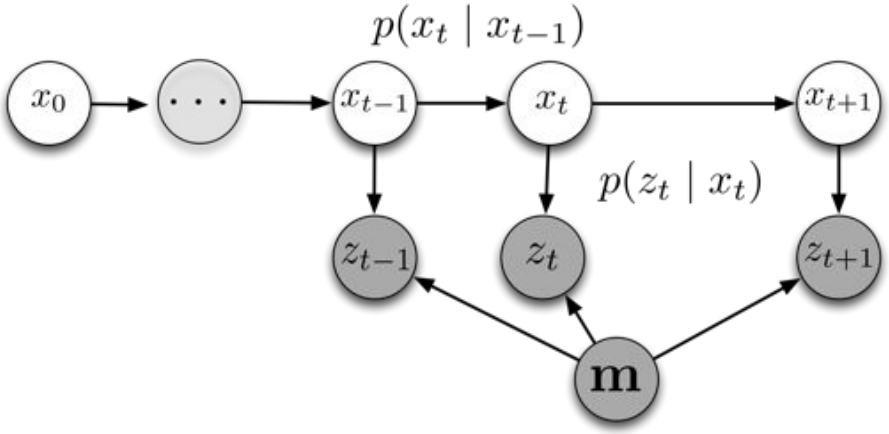



Figure 2.10: **Bayesian Network.** The posterior probability (represented as a graph) of the pose (and observation) chain of the robot as it moves through the environment. *Observed* quantities are shaded, unobserved - or *latent* quantities are not. Localisation consists of estimating the latent states $x_{0:n}$ from the observed data $z_{1:n}$ and the map, $\mathbf{m}$.

A graphical model can be thought of as a pictorial representation of a probability distribution - in this case, the joint probability distribution over poses and measurements. This representation is useful as it allows us to visualise immediately the conditional independence properties of the joint distribution.

Of course, this formulation makes no assumption on how to model these proba-

bility distributions. If the observations and transitions are both linear functions of the state, and system noise is normally distributed, then a Kalman Filter (KF) is a provably optimal Minimum Mean Square Error (MMSE) estimator [84].

However, given that this linearity assumption is unrealistic in real systems, a common approach is to perform linearisation of the system - this is the basis of the Extended Kalman Filter (EKF) [26], which performs such a linearisation of the non-linear observation and transition functions by means of a Taylor expansion. A significant improvement to the EKF is the Unscented Kalman Filter, as formulated by [46]. The Unscented Kalman Filter (UKF) relies on a deterministic sampling strategy known as sigma points - a form of stochastic linearisation. It has been proved to be a more robust, accurate estimator of the posterior distribution.

The underlying model of these filtering techniques is the Gaussian distribution, which is *unimodal* - this is problematic if we would like the model to consider various hypotheses over state. To remedy this, Multi-hypothesis Tracking (MHT) [76] uses a Gaussian Mixture Model (GMM) [27] to model multiple hypotheses over state.

An alternative *non-parametric*[6] approach is to approximate the posterior distribution in a *discrete* way. This was originally proposed by [92], and has had great success in the field of robotics. In this Sequential Monte-Carlo - or *particle* - filter, the posterior distribution over pose is represented as a set of weighted particles, as is conceptualised in Figure 2.11:

[6]It should be noted that non-parametric does not imply that the model is without parameters. Rather, the model does not correspond to any parametric family.

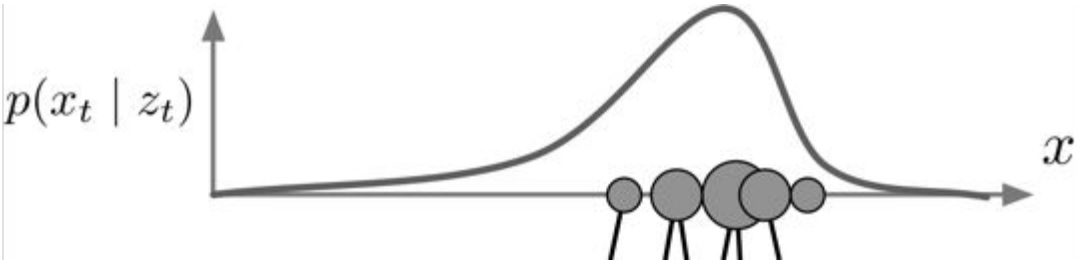



Figure 2.11: **Particle Filter: Approximation.** A weighted set of particles representing the underlying continuous probability distribution over pose.

At time $t$, these particles are then propagated using the transition model, giving rise to a new set of uniformly-weighted particles over pose-space:

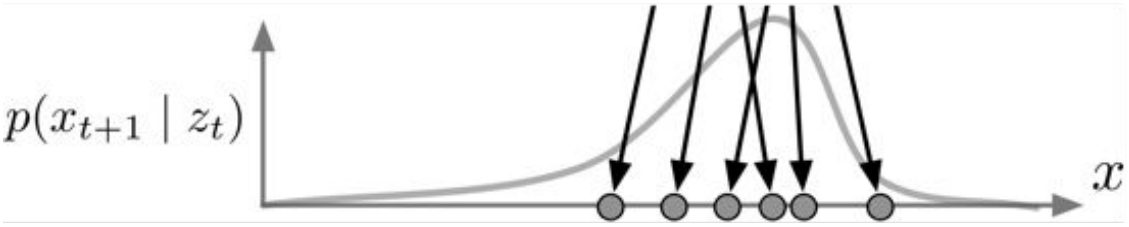



Figure 2.12: **Particle Filter: Propagation.** The set of propagated particles, using the motion model.

This is the *prediction* step. When new data are observed, the sensor-likelihood serves as a re-weighting function:

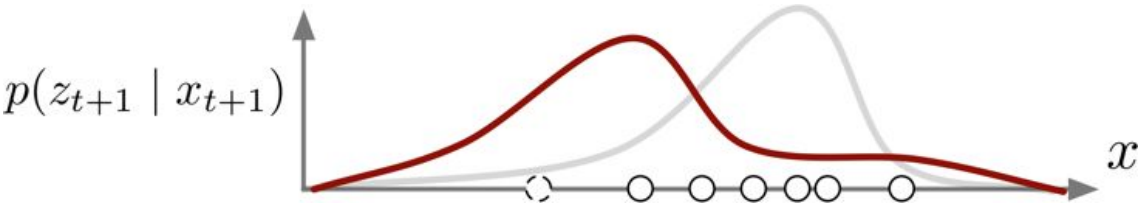



Figure 2.13: **Particle Filter: Re-weighting through the measurement likelihood.** Particle re-weighting through the observed data.

The above figure is a slight abuse of this pictorial representation, given the different domains plotted - however it serves well for this illustration. Given the relative weights assigned to each particle by the likelihood, we can now represent the posterior distribution over particles at $t + 1$:

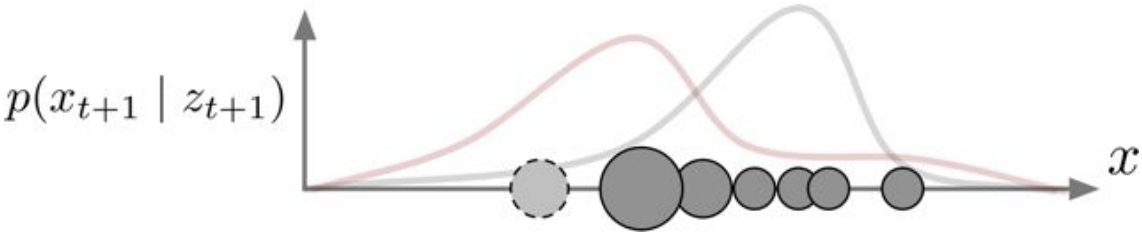



Figure 2.14: **Particle Filter: Approximation.** The new set of re-weighted particles now form the discrete approximation of the posterior distribution. The dashed circle represents a state-estimate added through external means, e.g. via GPS.

This figure palette was inspired by a similar sequence in [9]. As mentioned in the introductory sections, this approach can suffer from *particle-depletion*, where - due to a few unlucky random numbers - the posterior is not well approximated by the samples. Various re-sampling techniques are used, or in the case of [54], samples are drawn from an "oracle", which usually consists of a GPS unit.

Underlying this entire approach is a static world assumption - at each stage we are assuming sensor data is generated from static parts of the world, which is of course not true in a real environment. For our purposes, the map $\mathbf{m}$ is fixed - what is required is continual, robust estimates of the location of the vehicle within this map over the long term and, crucially, dealing with changes that occur in the real-world but are not reflected in the map.

Given that our target application is traversing a road-network, we do not expect to have to recover the global pose of the robot with no prior - the "global" localisation problem. We are explicitly focused on a *tracking* problem, and therefore it is not unreasonable to expect a vehicle to have good knowledge of its position at system-initialization.

We now turn to an overview of the use of point-based registration methods for localisation.

### 2.3.2 Localisation through registration

Given two sets of readings from a scanning LIDAR, the localisation problem can alternatively be framed as a *registration* problem. By aligning - or registering - these two sets of points (referred to as point clouds), we can estimate the inter-scan motion of the sensor. In an odometric or *open-loop* case, these two readings will be successive scans $z_t$ and $z_{t-1}$, but in the case of *localisation* will be $z_t$ and appropriate points from the map, $\mathbf{m}$. Scan-matching is often used in either open-loop or localisation modes, although global localisation techniques have been implemented [94].

To be consistent with the literature, we define a transformation $\mathcal{T}$ as a $3 \times 3$ homogeneous transformation, comprised of a rotational and translational component:

$$\mathcal{T} \in \left\{ \begin{pmatrix} R & \mathbf{t} \\ 0 & 1 \end{pmatrix} \middle| R \in \mathbb{SO}(2), \mathbf{t} \in \mathbb{R}^2 \right\} \tag{2.8}$$

where $R$ is the *Special-Orthogonal Group* $\mathbb{SO}(2)$ representing the rotation, and $\mathbf{t}$ is the Cartesian translational component - $\mathcal{T}$ is therefore a member of the *Special Euclidean Group*, $\mathbb{SE}(2)$.

As an example of the registration problem, Figure 2.15 shows two consecutive scans - separated by 1 second - from a SICK LMS-151 horizontally-mounted on a vehicle, passing through a typical urban scene:

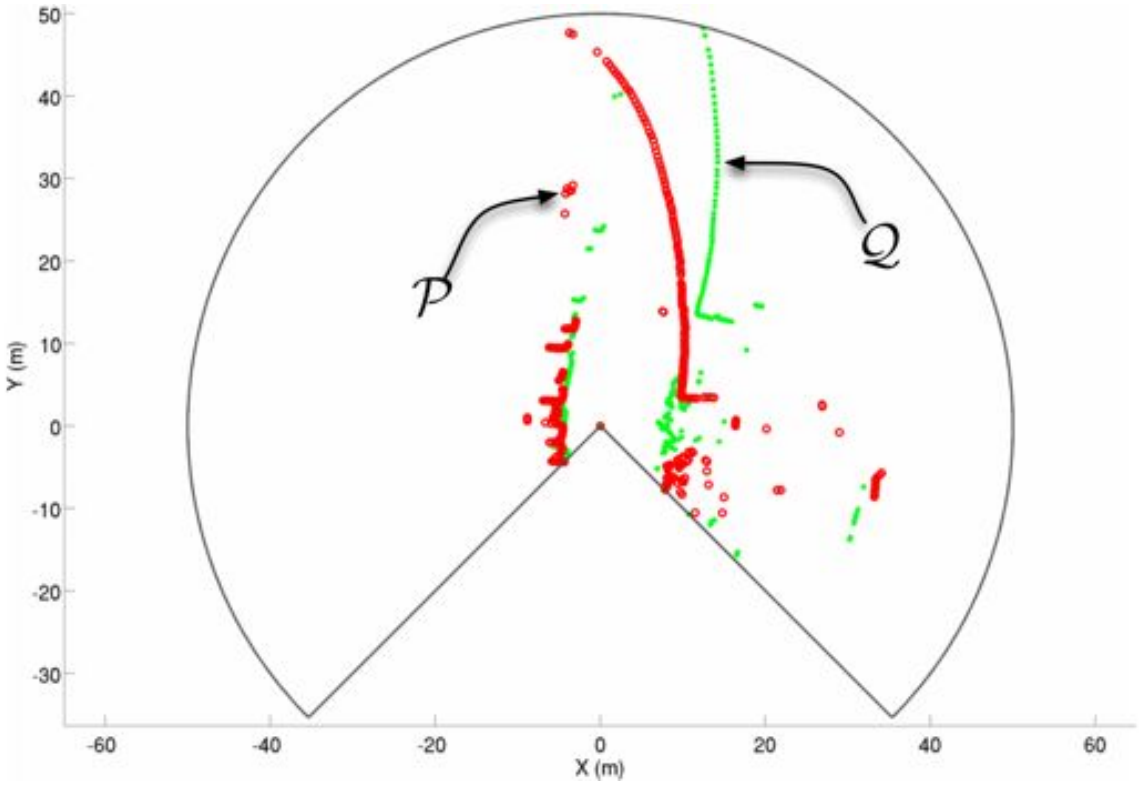



Figure 2.15: **Consecutive LIDAR scans: LIDAR frame.** Two consecutive sensor scans from a horizontally-mounted SICK LMS-151 rendered into the same frame. These are annotated $\mathcal{P}$ and $\mathcal{Q}$, which arise from projecting the range readings $z_t$ and $z_{t-1}$ into a Cartesian frame. Visible in both scans are similar scene elements - these will be used to estimate the intra-scan alignments. Visible also is the maximum-range scanning outline of the SICK LIDAR - this is approximately 50m.

Note that we have replaced $z_t$ and $z_{t-1}$ with $\mathcal{P}$ and $\mathcal{Q}$, which are the projections from range and bearing readings to Cartesian space.

Visible in both scans are similarities arising from the spatial co-location of these scans - a hedgerow (curved-section) is very visible in both LIDAR sweeps. The task is now to estimate the inter-scan transformation $\mathcal{T}' \in \mathbb{SE}2$ that will align $\mathcal{P}$ with $\mathcal{Q}$ - this resulting transformation will correspond to the motion of the sensor over the intervening time period. Figure 2.16 shows the results of applying the known - or *ground-truth* - transformation to scan $\mathcal{P}$:

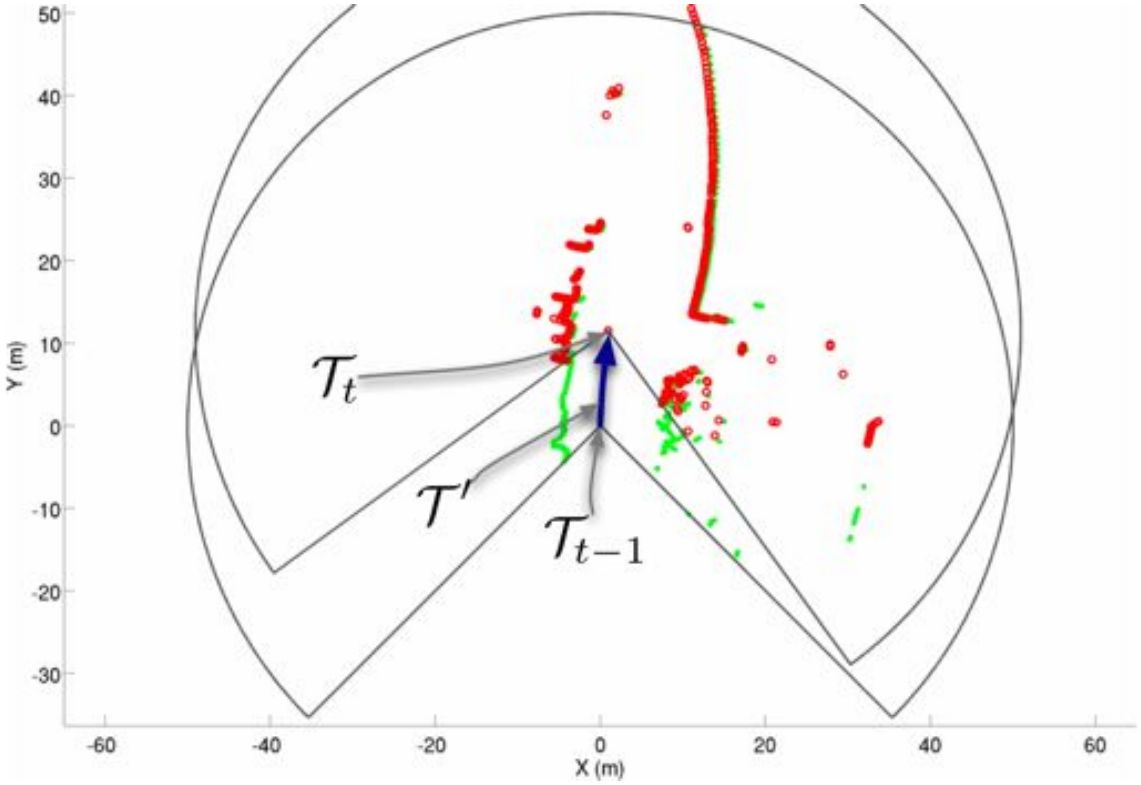



Figure 2.16: **Consecutive LIDAR scans: Global frame.** The same two scans as in Figure 2.15, with the corresponding inter-scan alignment transform applied. As can be seen, common elements in the scene now align well.

Given the appropriate transform $\mathcal{T}'$, we see that the scans align well - salient features in each scan match up well in the projection. This particular alignment of scans was obtained by using the pose estimates supplied by a very capable Inertial Navigation System (INS) - the capabilities, and in particular the shortcomings, of this system are detailed further in Chapter 4. Of course, we would like to eliminate our reliance on such a pose source and therefore must estimate $\mathcal{T}'$ through other means.

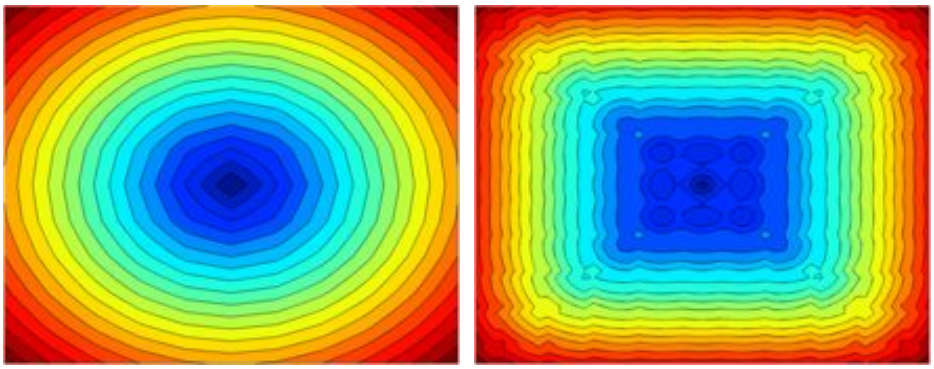
Given the two point clouds $\mathcal{P}$ and $\mathcal{Q}$, we seek a rigid transformation $\mathcal{T}$ that will bring the data into alignment. We can define a distance-based error-function between associated points in each scan in a least-squares sense as:

$$E = \sum_{i=1}^{|\mathcal{Q}|} \|p_i - \mathcal{T} \cdot q_i\|^2 \tag{2.9}$$

where $|\mathcal{Q}|$ is the cardinality of $\mathcal{Q}$, and $p_i$ is the point in $\mathcal{P}$ that is associated with $q_i$. The required transformation is a minimisation over this sum-of-squares error function:

$$\hat{\mathcal{T}} = \underset{\mathcal{T}}{\operatorname{argmin}} \; E(\mathcal{T}) \tag{2.10}$$

which can be done in closed form using quaternions, [42], or a Singular-Value Decomposition (SVD) approach [1]. This formulation is closed-form if the correspondences are known - unfortunately they are usually not. Figure 2.17 compares the difference in the error function for known and estimated (by means of a nearest-neighbour approximation) correspondences for a set of gridded points:



(a) Cost surface: known correspondences

(b) Cost surface: nearest-neighbour estimated correspondences

Figure 2.17: **Cost surface: known vs. unknown correspondences.** The difference in the cost-surface, comparing the squared-error of $E(\mathcal{T})$ with known (true) correspondences (a), and correspondences estimated with a nearest-neighbour approximation. (Dark-blue corresponds to the lowest cost, red to the highest)

As can be seen in Figure 2.17, the cost-surface - when using correspondences estimated in a nearest-neighbour sense - exhibits multiple local minima, which is a challenging optimisation problem. We can generalise Section 2.3.2 by introducing an *association* function, $w(\cdot)$, which determines the appropriate weighting between neighbouring points in $\mathcal{P}$ and $\mathcal{Q}$:

$$E(\mathcal{T}) = \sum_{j=1}^{|\mathcal{P}|} \sum_{i=1}^{|\mathcal{Q}|} w_{ij} \|p_j - \mathcal{T} \cdot q_i\|^2 \qquad q \in \mathcal{Q},\ p \in \mathcal{P} \tag{2.11}$$

The Iterative Closest Point (ICP) algorithm, as formulated by [7][100], has been extensively used to solve Section 2.3.2 [77]. The original algorithm uses a "point-to-point" metric - the distance function is defined over point-to-point distances, and the weighting function is:

$$w_{ij} = \begin{cases} 1, & \text{if } p_j = f_{NN}(\mathcal{P}, q_i) \\ 0, & \text{otherwise} \end{cases} \tag{2.12}$$

where $f_{NN}(q_i)$ is a search function that returns the nearest neighbour for an input point cloud and point query. ICP alternates between an optimisation step and a correspondence-estimation step, as is outlined in Algorithm 2.

It has been shown that using kd-trees - a tree-based spacial-decomposition technique - in the association step gives the complexity of ICP as $\mathcal{O}(M \log N)$ where $M$ and $N$ are the cardinalities of $P$ and $Q$ respectively.

Iterative Dual Correspondence, as formulated by [60] is a dual-step method where the translational component is estimated via the Euclidean distance, with the rotational component estimated through the polar distance. Polar Scan Matching

**Algorithm 2** IterativeClosestPoint

**procedure** EstimateTransformation($\mathcal{P}, \mathcal{Q}, \hat{\mathcal{T}}$,tol)
  **while** $\delta >$ tol **do**
    $\mathcal{Q} \leftarrow$ **ApplyTransform**$(\mathcal{Q}, \hat{\mathcal{T}})$
    **for** $q_i = 1$ to $|\mathcal{Q}|$ **do** ▷ Nearest-neighbour association
      $p_i \leftarrow f_N N(q_i)$
    **end for**
    $\mathcal{T}' = \underset{\mathcal{T}}{\text{argmin}} \sum_{i=1}^{|\mathcal{Q}|} \|p_i - \mathcal{T} \cdot q_i\|^2$ ▷ Minimisation
    $\delta \leftarrow \Delta(\mathcal{T}', \mathcal{T})$
    $\hat{\mathcal{T}} \leftarrow \mathcal{T}'$
  **end while**
  return $\hat{\mathcal{T}}$
**end procedure**

(PSM) [30] is another algorithm that utilizes a polar-representation of the scan measurements in the registration step.

An improved version of ICP using a point-to-plane metric was proposed by [100], in which the underlying surface for the target - or model - point cloud is approximated by estimating the surface normals at each point, and then minimising:

$$E(\mathcal{T}) = \sum_{i=1}^{|\mathcal{Q}|} w_i \|(p_i - \mathcal{T} \cdot q_i) \cdot \vec{n}_i\|^2 \tag{2.13}$$

where $\vec{n}_i$ is the estimated normal to the surface, which will be referred to as Iterative Closest Surface (ICS). This metric better approximates the relationship between the candidate point-sets, and will be used later in Chapter 6 in an open-loop fashion to robustly estimate vehicle velocity. A further generalisation is the plane-to-plane metric, otherwise known as Generalised ICP [80]. This is a probabilistic formulation of the ICP process, of which ICP and ICS can be shown to be special cases.

The core difficulty encountered by all of the aforementioned ICP-based methods is the correspondence estimation. If we *knew* the exact correspondences for any putative scan pair, we could easily evaluate the optimal alignment in one step - as shown in Figure 2.17. In [74], the authors propose *learning* correspondences a-priori, and use this learned model to better estimate correspondences matches at run-time. This is accomplished by modelling successive LIDAR scans as a *Conditional Random Field*, and inferring the marginal association probabilities of each LIDAR beam of one scan with each of the beams in the successive scan - essentially learning the weight vector $\mathbf{w}$.

Although ICP methods have been used extensively for localisation, we show in Chapter 4 how such methods are insufficiently robust for truly long-term localisation. We now turn to a brief overview of feature-based registration.

### 2.3.3 Feature-based methods

Registration methods, as discussed, make use of the entirety of each of the scans. The alternative is to develop methods that look for *salient* regions in each of the point clouds - this greatly alleviates the correspondence problem, at a cost of developing feature detectors that are sufficiently robust *and* descriptive enough to model the salient properties of all the possible data that is expected to be seen.

For example, a corner-detector will work well in an indoor setting, but will be of no use when navigating outdoors in unstructured environments. Although we do not make use of feature-based registration in this thesis, the following sections outline some well-known 2D and 3D interest-region detectors:

#### Spin-images

Spin-images [44] were originally developed for surface-modelling, but have been used extensively on point clouds. The starting assumption is that each point in the cloud

is *oriented* - i.e. has both position and surface normal. To generate a spin-image, the plane containing the normal vector is spun about the normal axis - all the surface points that intersect with this plane are binned, forming the image. Spin images are particularly useful in registration tasks as they are rotation, scale and pose invariant [29].

#### Integral Volume Descriptors

In [38], the authors develop the Integral Volume Descriptor (IVD), a local descriptor that is computed from the volume integral of the point cloud, for a given radius, at each point. Registration is done through a branch-and-bound search for correspondences, which provides an initial coarse alignment.

#### Feature histograms

In [78], the authors develop *persistent* feature histograms, using a 16-dimensional feature vector to represent the geometry of each individual point, using a number of heuristics. The algorithm then uses the resulting features to register a good initial guess, that can subsequently be used by ICP-based methods for a refined alignment.

#### Robust corner-detection

In [57] the authors develop a general purpose feature-detector by adapting the well-known Kanade-Tomasi corner-detection algorithm, and show repeatability experiments over the "Intel" dataset, and sections of the MIT DARPA dataset.

#### 2D Interest-region detectors

In [93] the authors develop a multi-scale interest-region detector explicitly for the purpose of 2D scan-matching, citing the lack of equivalent 2D feature descriptors. The descriptor incorporates a number of detectors and encodes the scan structure

in an area proportional to the scale of the detected feature. Specific to planar point clouds, the authors in [58] isolate pertinent features in the polar-space of incoming scans, and compute the optimal alignment between identified features in a least-squares sense.

To employ feature-based registration techniques as the prime means of data-registration (and therefore, localisation within a map), we are required to make two assumptions:

1. The descriptors are rich - and robust - enough to identify interest regions that exist throughout the operating environment of an autonomous vehicle - an exceptionally broad range.

2. These features are *persistent* enough to allow for long-term localisation.

Although (1) is difficult to guarantee, (2) is more so given the fluid, dynamic nature of the real-world. Therefore when we make use of registration techniques in this thesis it will be exclusively done through the ICS framework - this is discussed at length in Chapter 6.

## 2.4 Summary

This section has presented a background of the fundamentals of localisation within a given map, using probabilistic and geometric approaches. The concept of scan-registration has been introduced, along with the complexities inherent in correspondence-estimation and feature-robustness.

In Chapter 4 we develop an information-theoretic approach to registration with a particular focus on the task of vehicle localisation, and show that by doing so we are not subject to the constraints outlined in this section.

As a precursor to this, in the following chapter we establish means - both simulated and in hardware - to capture high-fidelity point cloud data in both real-world and simulated conditions.

# Chapter 3

# Development and Testing Tools

## 3.1 Introduction

For the robotics research tasks that we are concerned with, we will require both simulated and real-world data. In this chapter, we explore both the synthesis of simulated city-scale sensor data, as well the design and build of a stand-alone sensor suite. The following section details the generation of artificial point cloud data, followed by a description of the design process and subsequent deployment of a stand-alone surveying unit.

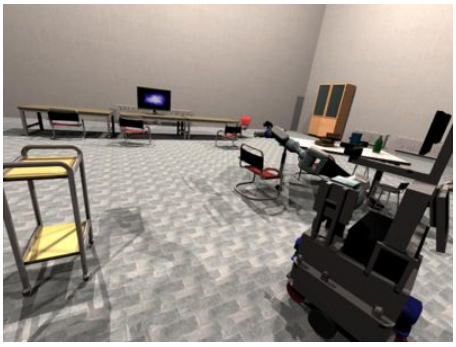
## 3.2 Simulating 3D surveys

Given the complex nature of robotic platforms and the logistical effort required to collect data, it is often easier to design and validate algorithms on synthetic data before field trials. This can be problematic, as we would like this data to mimic the real-world arbitrarily closely; however, such realism is often difficult to obtain. In the following section we review some of the existing open-source solutions, followed by a description of the implemented approach.

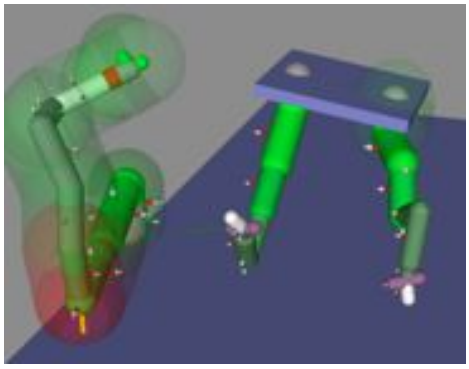
### 3.2.1 Existing simulators

Various open-source solutions exist for representing robotic vehicles, and environments, for varying degrees of accuracy and modelling effort. Figure 3.1 shows screenshots of some of the more widely-known implementations:

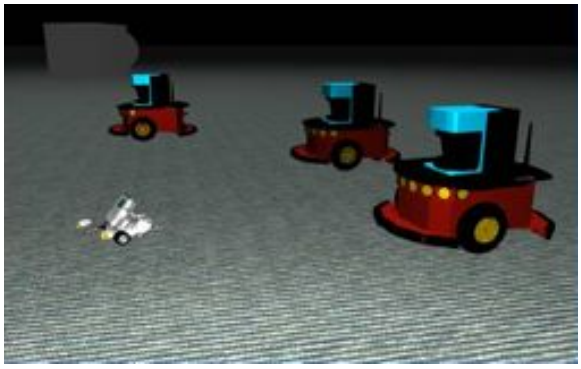
(a) Gazebo



(b) Morse



(c) OpenSim



(d) Microsoft Robotics Studio

Figure 3.1: **Existing simulation software.** (a) Gazebo, (b) Morse, (c) OpenSim, and (d) Microsoft Robotics Studio.

Gazebo[1] is an open-source software suite (formerly under the umbrella of the Player/Stage[2] software library), and is capable of simulating a variety of platforms and sensors. Morse[3] is an open-source library that allows for single and multi-robot interaction with a variety of indoor and outdoor environments. OpenSim and Microsoft Robotics Studio are both Windows-centric frameworks, with similar goals

[1] `http://gazebosim.org/`
[2] `playerstage.sourceforge.net/`
[3] `http://www.openrobots.org/wiki/morse/`

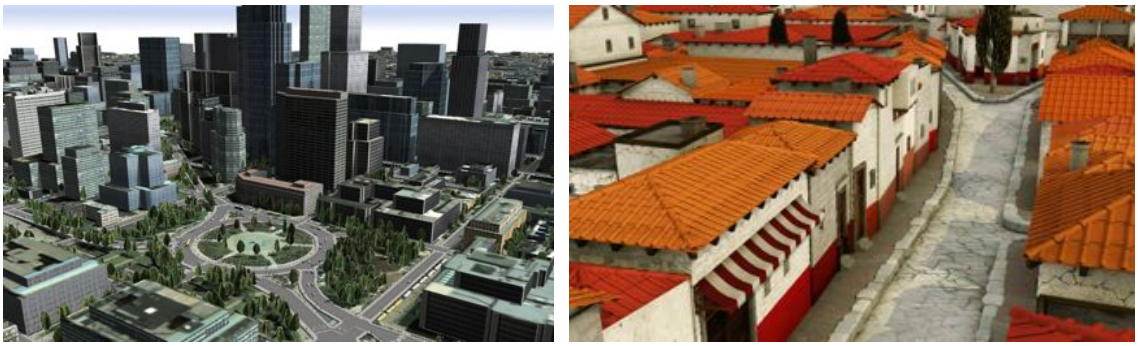
as Gazebo and Morse. However, none of these platforms provide the means to easily generate and investigate the arbitrarily complex, expansive scenes that we require - specifically, surveys of realistic urban environments.

### 3.2.2 Obtaining realistic world models

A key goal of our simulator is to provide as rich a world representation as possible, without requiring large pre-processing phases, or inducing degenerate run-time performance. We would like to have access to expansive, realistic terrain and city models, while still being able to load the model into the memory of a typical workstation. CityEngine[4] is a software package that is capable of providing such models.

#### CityEngine

CityEngine is developed specifically for the generation of arbitrarily complex urban scenarios. Figure 3.2 shows a few examples of cities generated by CityEngine and subsequently rendered post-hoc:



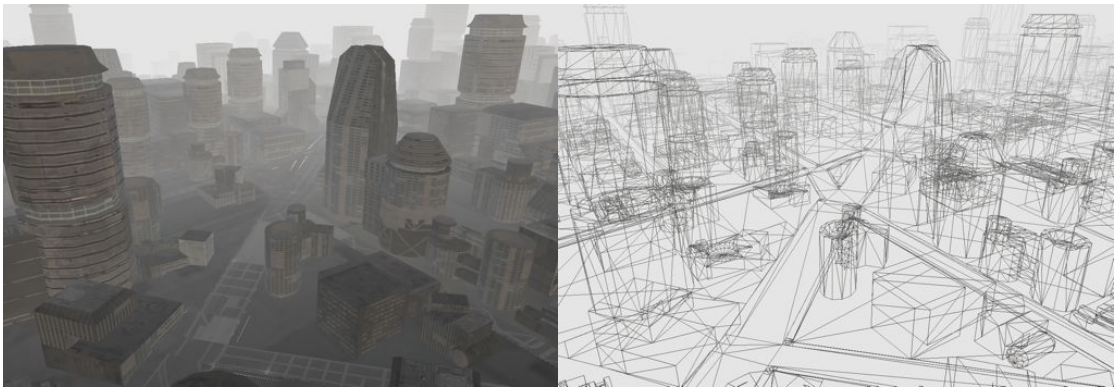
(a) (b)

Figure 3.2: **Example CityEngine cities.** (a) A city-centre, and (b) A town street.

CityEngine is a powerful rule-based procedural model generator, that is capable of constructing highly complex environments with very realistic terrain and building

[4] `http://www.esri.com/software/cityengine/index.html`

models. Figure 3.3 shows an example CityEngine model, both textured and showing the constituent model primitives:



(a) A textured virtual city (b) The polygon mesh of the same city

Figure 3.3: **Example city and mesh.** An example of one of the cities created by CityEngine that will be used to produce 3D surveys. (a) Shows the basic textured city, and (b) the constituent triangle mesh.

Figure 3.3 constitutes an example city that we may wish to survey. The geometry of the environment is represented as a "polygon soup" - many thousands of triangles mapped with predefined textures. In our surveying task, we would like to produce 3D point clouds that capture the underlying complexity of this model.

In this thesis, we are primarily concerned with LIDAR ranging sensors. Figure 3.4 shows the outline of a simulated scanning LIDAR (with a 270° field-of-view) in an environment consisting of simple polygons:

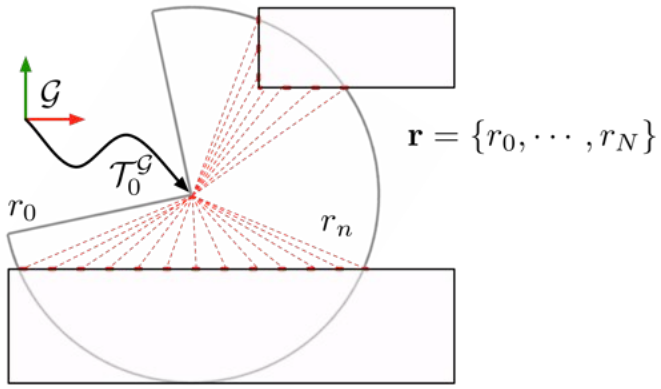



Figure 3.4: **Scanning LIDAR in a simple environment.** A simple scene, showing a planar view of a 2D LIDAR in an environment consisting of simple polygons. Producing point clouds requires calculating the intersection of the sensor beams with the world geometry.

Given the current transformation of the sensor in some reference frame, denoted here by $\mathcal{T}_0^{\mathcal{G}}$, we would like to establish the point-of-impact of each beam $r \in \mathbf{r}$ in the world - the general terminology for this is *ray-casting*. Then, by moving the sensor through the world, we will obtain a representative sample point cloud of the scene. We define a ray $x(t)$ with origin $x_0$ and direction $\vec{n}$ to be:

$$x(t) = x_0 + t \cdot \vec{n} \qquad x, \vec{n} \in \mathbb{R}^3, \quad t \in \mathbb{R} \tag{3.1}$$

where $(t)$ parameterises distance along the ray from a given start-point. Given an arbitrary plane in the model we can solve for $t$, fully specifying the intersection in world-coordinates. With a plane described by normal $\vec{n}_p$ and point $p_0$, $t$ is:

$$t = \frac{p_0 - (x_0 \cdot \vec{n}_P)}{(\vec{n} \cdot \vec{n}_P)} \tag{3.2}$$

where $(a \cdot b)$ denotes the dot-product of two vectors $a$ and $b$. Of course it is not enough to determine the planar intersection - we also need to know if the resulting point lies within the polygon vertices (in the models we explore, these are composed exclusively of triangles). A natural way of parameterizing this constraint is by using the *barycentric* co-ordinates, shown in Figure 3.5:

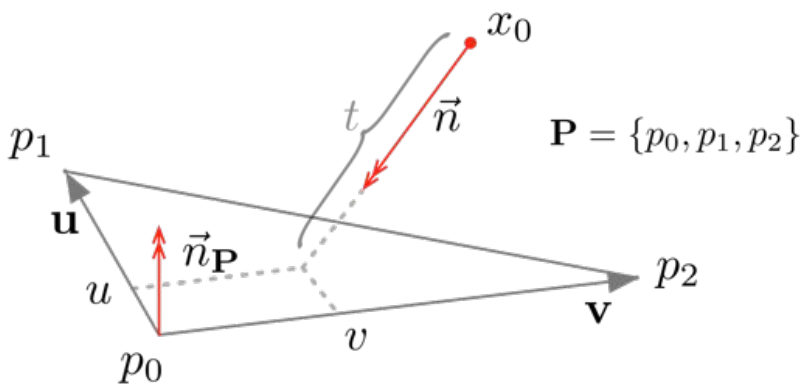



Figure 3.5: **Ray/triangle intersection using barycentric co-ordinates.** The barycentric co-ordinates $(u, v)$ are a weighted sum of the polygon vertices $\mathbf{P}$, and have natural constraints when considering the putative intersection of ray $x(t)$.

The barycentric representation is a linear combination of reference vertices, and provides a convenient representation for the intersection test. Using this *parametric* representation, the coordinates must obey the following constraints:

$$u >= 0, \quad v >= 0, \quad u + v <= 1$$

if the ray $x(t)$ has an intersection with the triangle formed by $\mathbf{P}$. The location of the intersection is expressed with respect to the vertices:

$$x_{u,v} = (1 - u - v) \cdot p_0 + u \cdot p_1 + v \cdot p_2 \tag{3.3}$$

A useful property of this formulation is that the barycentric representation is analogous to the co-ordinate system used in texture mapping - therefore, solving for $(u, v)$ will be informative both geometrically and appearance-wise. For example, if the underlying texture of a triangle is known, it is possible to index into a pre-computed remission table to determine the reflectiveness of the underlying surface.

A well-established method of solving for this intersection is through the Möller-Trumbore [64] ray intersection test. Setting Equation (3.1) to be equal to Equation (3.3) gives rise to a system of linear equations that can be solved efficiently:

$$[-\vec{n}, p_1 - p_0, p_2 - p_0] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = x_0 - p_0 \tag{3.4}$$

There are a multitude of algorithms for calculating such an intersection, as noted by [59], although the Möller-Trumbore and Badoeul [4] are most often employed due to their ease of implementation. The algorithm in [4] also works in a barycentric coordinate system, but requires the computation of the normals - however, given that the majority of the scene is static, it would be possible to pre-compute these in a pre-processing phase.

By applying Cramer's rule that, for any $Ax = b$ where $A$ has a non-zero determinant and a unique solution, we can solve for the individual unknowns in Section 3.2.2 as follows:

$$x_i = \frac{|\ A_i\ |}{|\ A\ |} \tag{3.5}$$

where $A_i$ is formed by replacing column $i$ with $b$. Section 3.2.2 can now be

rewritten as:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\mid -\vec{n}, p_1 - p_0, p_2 - p_0 \mid} \begin{bmatrix} \mid x_0 - p_0, p_1 - p_0, p_2 - p_0 \mid \\ \mid -\vec{n}, x_0 - p_0, p_2 - p_0 \mid \\ \mid -\vec{n}, p_1 - p_0, x_0 - p_0 \mid \end{bmatrix} \tag{3.6}$$

which we can use to solve for $[t, u, v]^T$ directly. With the advent of more powerful instruction sets, such as Streaming SIMD[5] Extensions 4 (SSE4) [43], fundamental native geometric operations (such as the dot-product) allow for correspondingly faster ray-queries [40]. Figure 3.6 shows an example scan using this ray query test in a complex city scenario:

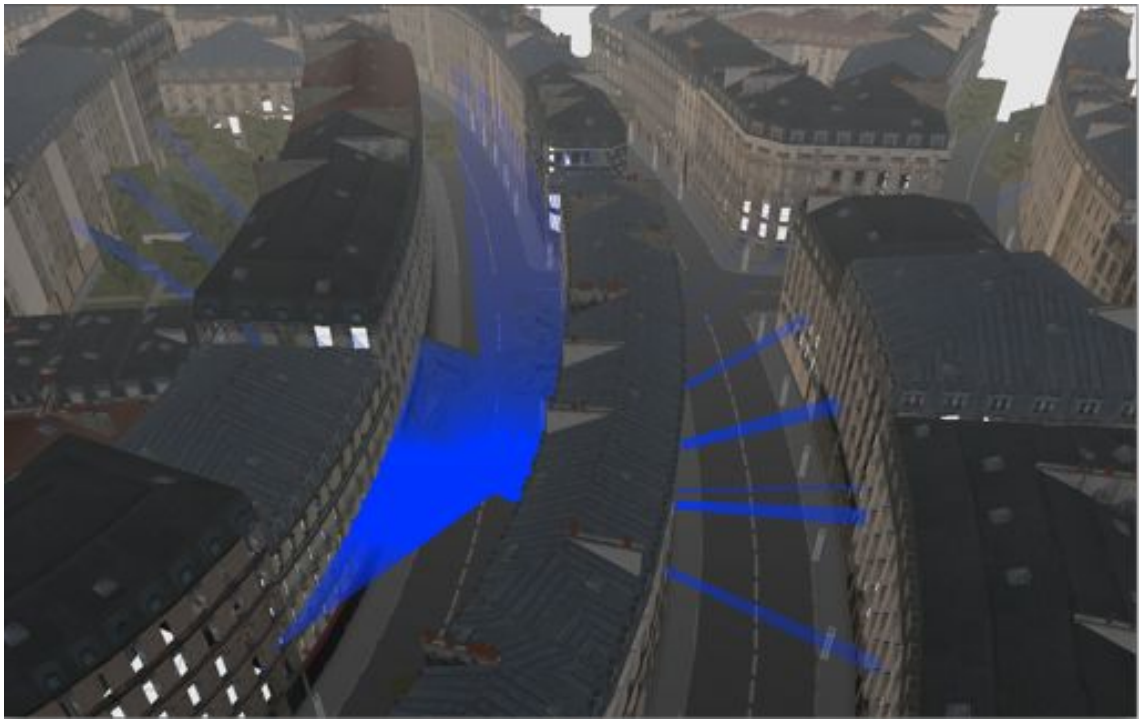


Figure 3.6: **Synthetic LIDAR scan.** A synthetic LIDAR scan in a virtual city, generated by CityEngine. Triangle-intersections for several hundred ray-queries were established exhaustively using the Möller-Trumbore method.

[5]Single instruction, multiple data

Visible in Figure 3.6 is the scan-centre (in the lower-left) and the corresponding beams - simulating a planar LIDAR scanner - traced to their intersection in the world. This figure was generated in a naive fashion - each of the triangles in this scene (approximately $700,000$) was checked for an intersection with each of the rays launched from the simulated sensor.

Although easy to implement, the overall approach is *linear* - $O(n)$ - in terms of number of triangles in the scene. Whilst this is not problematic for smaller models, it becomes a limitation for scenes of any useful size.

It should be noted that this does not preclude the use of such a method in an offline mode - however, we seek a representation that will allow us to perform real-time ranging queries in an arbitrary model. For our task - generating point cloud surveys - we have a number of sensors we would like to simulate. Table 3.1 compares the theoretical range-query requirements for different sensor types.

Table 3.1: Sensor queries per second

| Sensor | LMS-151 | Flash-LIDAR | Velodyne |
|---|---|---|---|
| Queries/second | 50 | 500k | 1.3M |

From Table 3.1 it is apparent that the Velodyne LIDAR is the most taxing in terms of query requirements[6] (although the performance gap to flash-LIDAR[7] is closing). Although we may not be able to simulate the full capabilities of the Velodyne in software (and real-time), we require performance such that we can realistically model 3D ranging sensors that produce comparable point clouds. Fortunately, there are well-known data-structures we can leverage to make real-time sensor simulation tractable - these are outlined in the following section.

[6] `http://velodynelidar.com/lidar/products/brochure/HDL-64E%20S2%20datasheet_2010_lowres.pdf`

[7] `http://www.advancedscientificconcepts.com/products/tigereye.html`

### 3.2.3 Scalable range-sensor models

The approach taken in complex rendering or intersection/query tasks is to separate the problem into two phases. There is a pre-processing phase, followed by a subsequent query (or *set* of queries).

The query operation is similarly split into separate phases - there is a *broad* phase, which restricts - at a high-level - the search space to be explored, followed by a *narrow* phase, which consists of the low-level search (or intersection test) over geometry primitives. This is a form of the general divide-and-conquer strategy, where a complex problem is broken down into constituent sub-problems and solved.

The core idea in any intersection problem is to exploit the underlying spatial relationships of the geometry in the scene - there is no need to test triangles for intersection that are distantly removed from the final query site - therefore, we seek a *spatial* decomposition that will allow us to discard regions of the search space early-on that will be uninformative. The most coarse representation of this spatial decomposition would be a discrete voxelisation approach, where the environment is decomposed into voxels (or *volume pixels*).

A more sophisticated approach would be to build a hierarchical model of the scene - this partitioning approach is known as a *bounding hierarchical volume*, where sub-regions are encapsulated in so called *bounding volumes*. Figure 3.7 shows different approaches for generating bounding volumes:

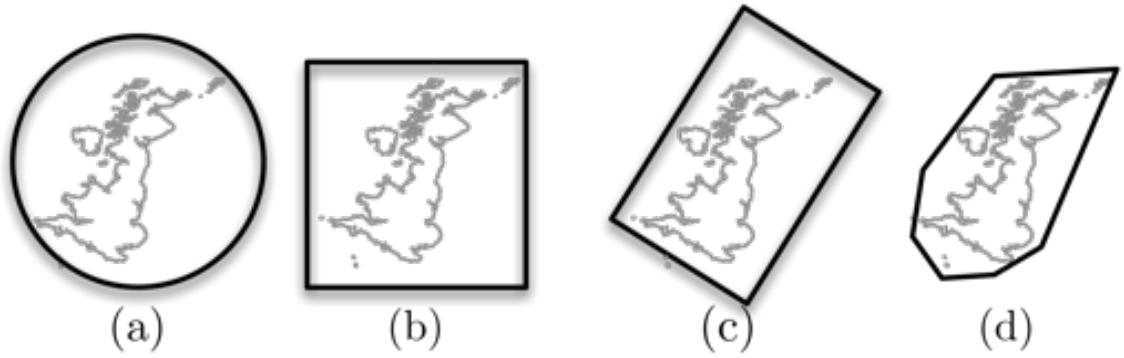



Figure 3.7: **Bounding volume construction.** Different ways of generating bounding volumes for arbitrary geometry: (a) spherical, (b) axis-aligned bounding box (AABB), (c) oriented bounding box (OBB), and (d) convex-hull. The more complex the bounding volume (i.e. the more representative of the underlying geometry), the fewer high-level queries have to be performed, at a cost of significant increases in memory requirements.

Axis-Aligned Bounding-Box (AABB) trees provide a good trade-off between model-processing and query times, and are described further in the following section.

### AABB trees

Simply bounding elements in the scene by a volume (an axis-aligned box, in this case), will not result in a speed increase. The core idea is to organise these boxes into a tree-structure - this will reduce the linear behaviour exhibited by the exhaustive search, to logarithmic ($O(\log(N))$) as a function of polygon count. The tree will provide a "bail-out" capability - it is not necessary to test child nodes if there is no intersection of the query ray with the parent node.

Figure 3.8 shows an example scene with arbitrary primitives, and an associated AABB tree:

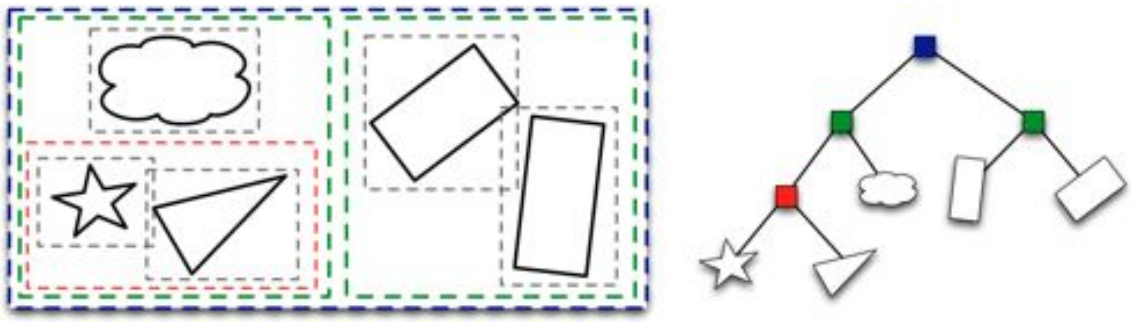


Figure 3.8: **Example AABB tree.** Shown are example primitives (left) along with their associated bounding boxes (grey, dashed), and the resulting AABB tree (right). The node colours in the tree correspond to the bounding-box colours in the spatial layout.

Note that in Figure 3.8, some of the bounding-boxes overlap when the underlying geometry does not - this is a disadvantage of using such a simplified representation.

Building an AABB tree over $n$ primitives results in a tree with $n-1$ internal splits, if the tree is binary. Although there is no *requirement* for the tree to have a branching degree of 2, binary trees are easy to build and traverse [35, p.239].

Given that the final tree representation is only one of a number of possible tree combinations (this number is exponential in the number of input entities), heuristics are used to construct the tree. There are three commonly used methods of tree construction - top-down (nodes are constructed recursively from a root node), bottom-up (the tree is constructed from aggregating leaf-nodes and sub-trees) and insertion, where nodes are inserted one-at-a-time into the tree. In the top-down approach, each step consists of partitioning the input set into subsets, encapsulating the subsets with the chosen bounding strategy, and recursing - top-down methods are generally easier to implement.

Shown in Figure 3.9 is an AABB tree built over an example 3D model, with boxes encapsulating the individual model triangles:

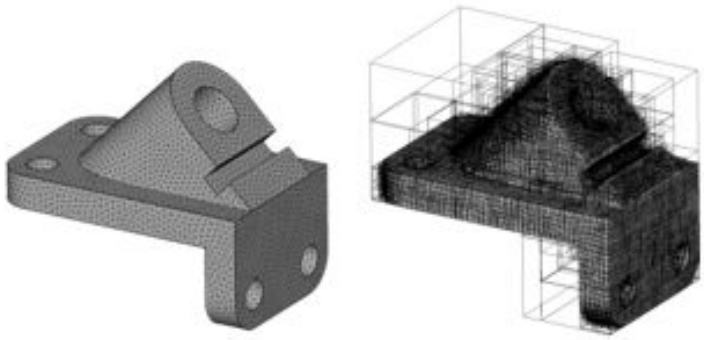


Figure 3.9: **AABB tree representation of a CAD model.** A representation of an Axis-Aligned Bounding Box tree (left) over the adjacent model. The AABB tree allows for rapid ray-intersection queries, which is a requirement for any simulated ranging sensor. Image courtesy of [87].

Tree construction is done as a pre-processing step - however, if the scene contains dynamic components, the tree nodes will have to be updated as the geometry in the scene changes. Given the very simple box representation, updating the box is very easy - the disadvantage (as compared to a more descriptive bounding volume, like convex hulls) is that more low-level checks will be required, as the box volumes will often intersect, even if the underlying geometry does not. Given the AABB tree, it is relatively simple to calculate the mesh intersection for any given ray. Figure 3.10 shows an example in 2-d:

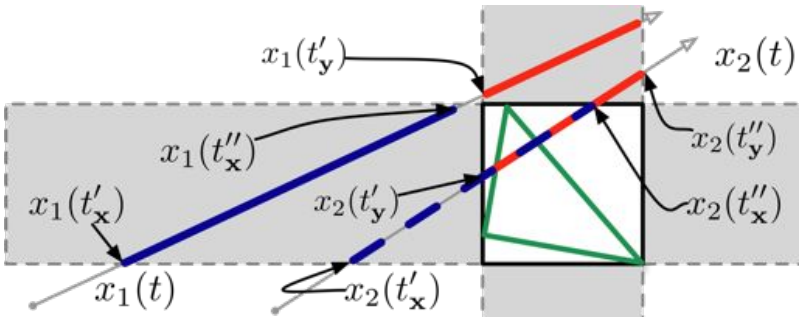



Figure 3.10: **Computing the ray/AABB intersection.** Computing the intersection of a ray is a matter of examining the point-plane intersections with the constituent *slabs* of the box (shown in shaded grey). Slabs are axis-aligned planes, the intersection of which form the bounding-box. Overlapping entry/exit points for slabs indicates a potential collision.

Figure 3.10 was inspired by a similar figure in [35, p.180]. To establish whether a ray intersects a bounding box, all that is required is to calculate the ray/plane intersections with each of the constituent *slabs* (axis-aligned planes) defining the box. The entry/exit parameter values for aligned planes $\{\mathbf{x}, \mathbf{y}\}$ are denoted as $\{t', t''\}$, respectively. Any ray that intersects with the bounding box must have entry/exit $t$ values that *overlap* - for example, $x_2(t'_{\mathbf{y}}) \in [x_2(t'_{\mathbf{x}}), x_2(t''_{\mathbf{x}})]$. Any ray that does not obey this condition cannot be an intersection candidate. Once we have queried down to a leaf-node of the AABB tree, we can solve for the intersection of the ray with the underlying polygon using the Möller-Trumbore triangle-intersection test.

We now turn to a comparison of the AABB tree with the exhaustive-search, and show the huge performance gain with this hierarchical method.

### 3.2.4 Query comparison

To illustrate the performance of the AABB tree over the naive linear search, we present the following experiment. Figure 3.11 shows an environment with a small number of box meshes in a flat grid-world:

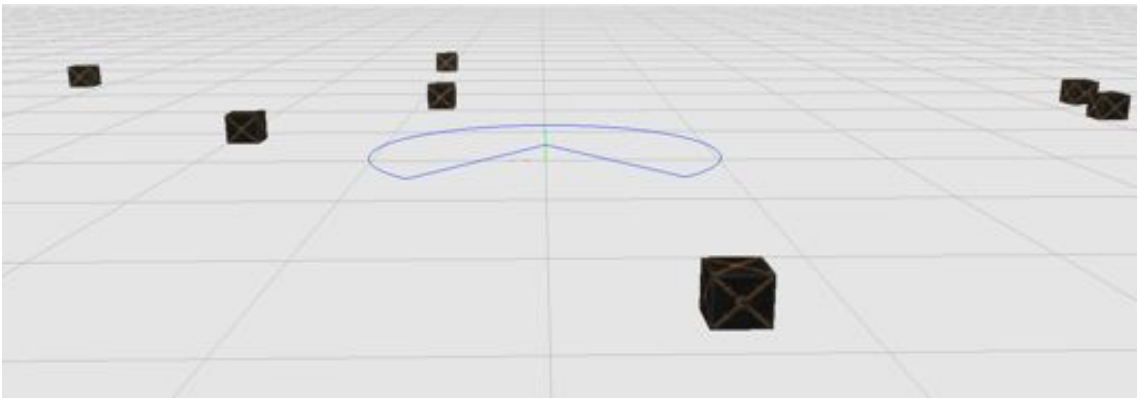


Figure 3.11: **Small mesh query.** The synthetic environment used for a simulated LIDAR query against a small number of box meshes. Visible in the figure is the scanning outline of a hypothetical SICK LMS-151.

Visible in Figure 3.11 is the LIDAR outline (a simulated LMS-151 in this case),

as well as the meshes used in the experiment. Given the small number of meshes, it is reasonable to expect that the exhaustive search will be comparable to the AABB tree. Figure 3.12, however, shows a more testing situation with many thousands of obstacles:

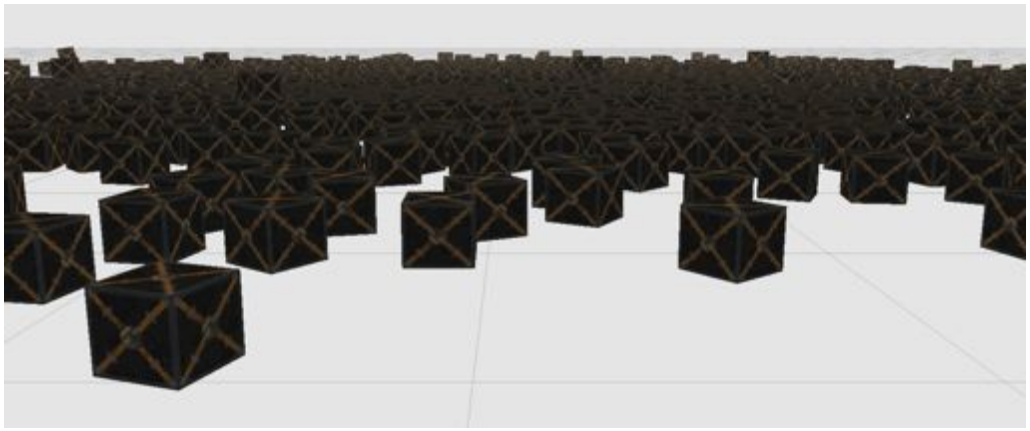


Figure 3.12: **Large mesh query.** A similar scene to Figure 3.11, with substantially more meshes. Building a hierarchical spatial representation of the scene allows for ray queries in the order of $O(\log(N))$.

Given the massive increase in polygon count, we expect the AABB tree to substantially outperform the exhaustive search. To quantify this performance, each approach was instantiated in an environment with increasingly more meshes, and timed across one full scan of the hypothetical LIDAR (541 individual beams). Figure 3.13 shows the comparison of these sensor query times:

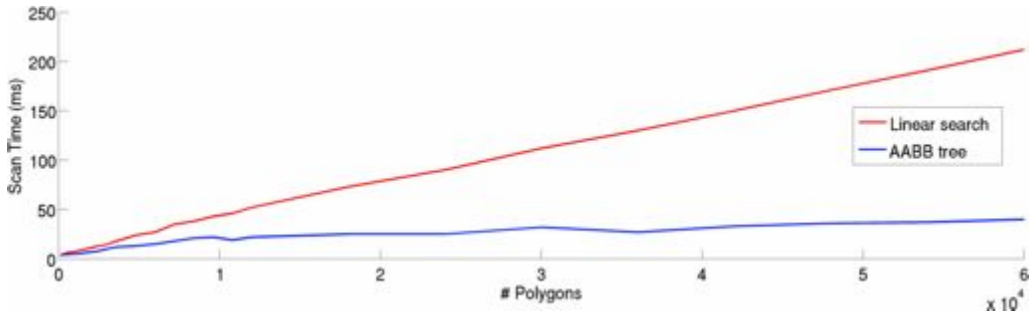



Figure 3.13: **Ray query comparisons.** A comparison of the query times for both the exhaustive approach (shown in red), and the AABB tree queries (shown in blue). Very apparent in this figure is the logarithmic performance of the AABB tree - this is a necessary property for large models.

Very apparent in Figure 3.13 is the linear behaviour of the naive approach, against the logarithmic performance of the hierarchical tree. Given this method of rapidly evaluating intersections, we can now explore expansive, complex environments and generate realistic point clouds in real-time. Figure 3.14 is an example of such a model generated by CityEngine:

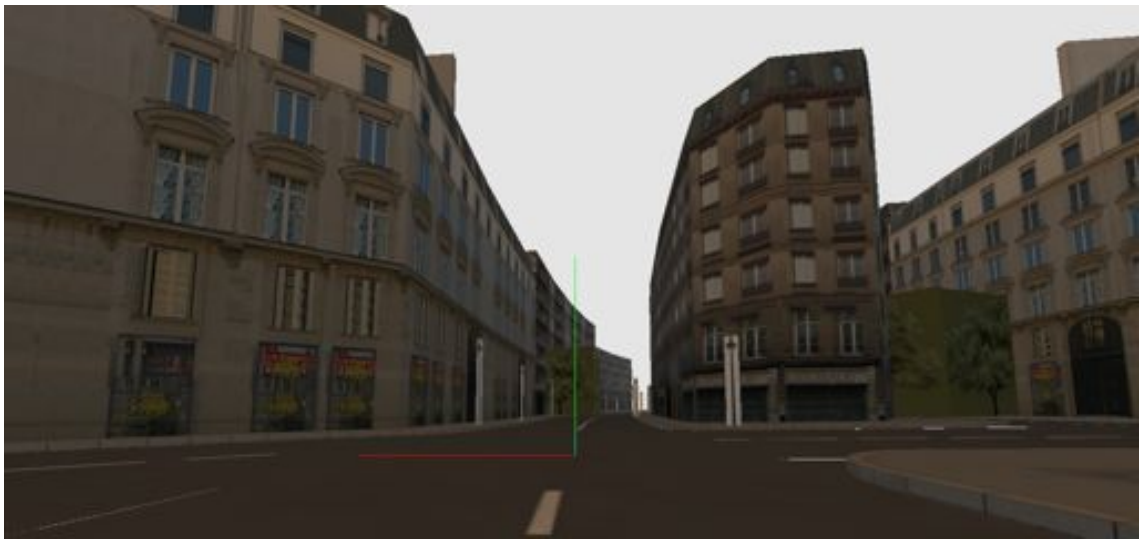


Figure 3.14: **$18^{th}$ century Paris.** A model generated by CityEngine, corresponding to a hypothesized $18^{th}$ century Paris, with approximately $700k$ texture-mapped polygons.

Figure 3.15 shows the resulting point cloud $\mathbf{X} = \{x, y, z\}_{i=1}^{N}$ obtained by arbitrary motion of a hypothetical LMS-151 through the scene:

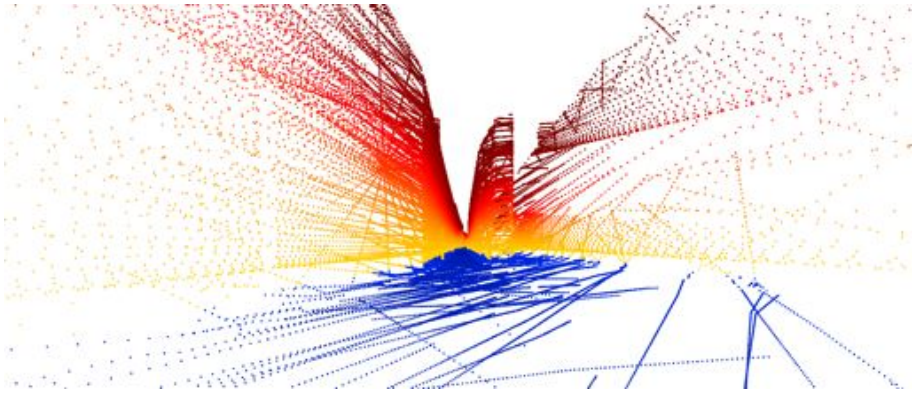


Figure 3.15: **Paris point cloud.** The point cloud representation of the scene illustrated in Figure 3.14. The interleaved scan-lines visible in the figure arise through the arbitrary motion induced through the scene in order to capture the data (points are coloured by height above ground).

The cardinality $N$ of this cloud $\mathbf{X}$ is approximately 1.2M, sampled at frame-rate (60Hz). The point cloud does not incorporate any noise model, giving rise to the very regular structure - such a model can of course be incorporated into the ray-casting process trivially. Given the point cloud, we can perform the reverse process - i.e. perform triangulation on this point cloud to recover a mesh. This is done through the *ball-pivoting* algorithm, and is shown in Figure 3.16:

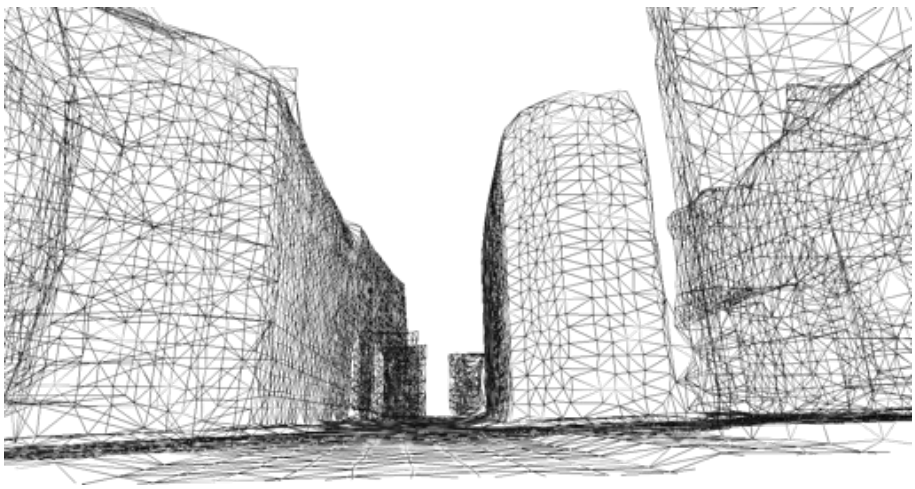


Figure 3.16: **Paris re-meshing.** A mesh generated from the point cloud data shown in Figure 3.15. This mesh was generate by sub-sampling the point data and then producing facets with the ball-pivoting algorithm.

Ball-pivoting [5] is a relatively simple algorithm that produces a surface mesh from an unstructured point cloud. The central idea of the algorithm is - given a point cloud with estimated normals, and a "seed" triangle - to roll a $d$-dimensional ball of a pre-specified diameter $\rho$ around the edges of the mesh, incorporating points that fall within the diameter. A realisation of the algorithm over a cross-section of a 3-dimensional ($d = 3$) point-set is shown in Figure 3.17:

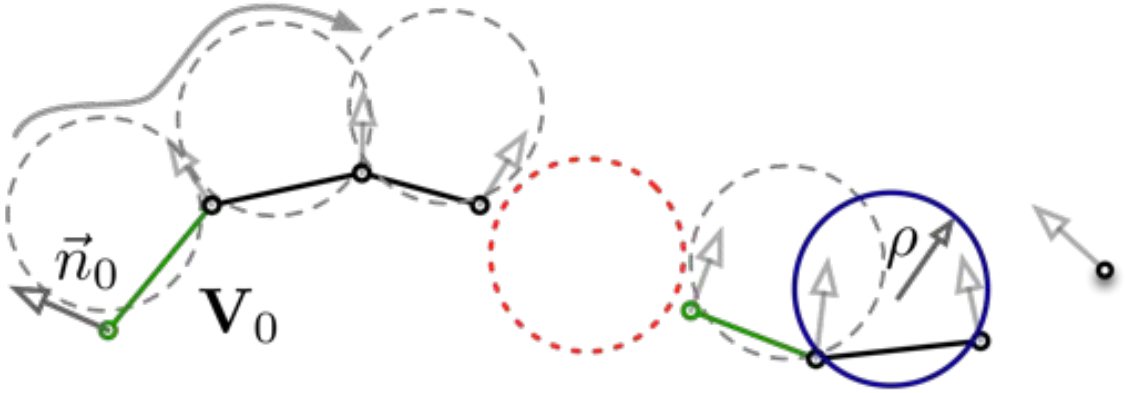



Figure 3.17: **Ball-pivot mesh construction.** The ball-pivoting algorithm is given a set of unordered points, $\mathbf{X}$ and their associated normals, $\vec{n}$. The algorithm then estimates a seed face ($\mathbf{V}_s$) and pivots on a facet edge, incorporating a point if it falls within the predefined radius $\rho$. When the inter-point distance exceeds $\rho$, the algorithm is re-initialised.

The ball-pivoting algorithm provides a simple, robust method for generating a triangulated mesh from the point clouds obtained from either the simulator, or real LIDAR data. As further illustration, Figure 3.18 shows another example city, generated via CityEngine:

Figure 3.18: **$12^{th}$ century Medieval city.** An unlikely scenario for any robot - a $12^{th}$ century village. The point illustrated here is that we are unrestricted - in terms of input meshes - that we can generate realistic LIDAR data from. The mesh extracted from a point cloud sampling of this city is shown in Figure 3.19.

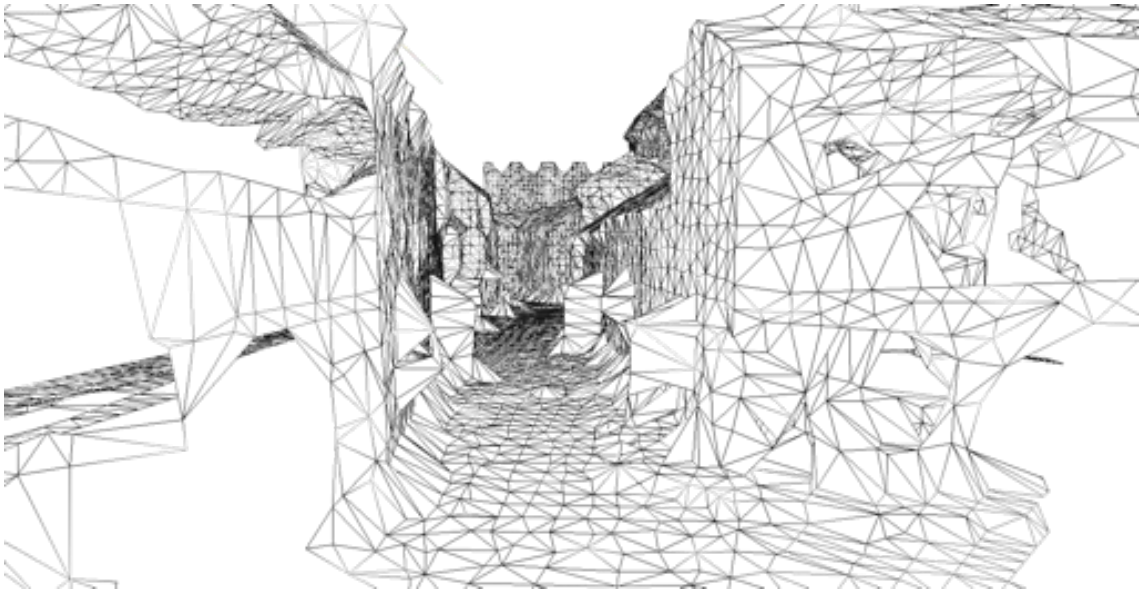


Figure 3.19: **$12^{th}$ century Medieval city (mesh).** The triangulated mesh, obtained from point cloud data generated from sweeping a simulated LIDAR sensor through the environment shown in Figure 3.18. Again, the mesh is a faithful representation of our initial model.

In both cases, the resulting mesh corresponds very well to the original model

- as should be expected. This ray-casting approach, combined with the modelling powers of CityEngine and incorporating various other well-known approaches for dealing with expansive terrains (paging scene managers, for instance, allowing for theoretically infinite worlds) allows for the rapid synthesis of realistic training data, a very useful tool for robotics research.

We now present an interesting consequence of this - leveraging large Computer-Aided Design (CAD) databases online to further expand our simulator capabilities.

### 3.2.5 Leveraging the web

The web is replete with 3D CAD models, ranging from professional to purely amateur[8] (Trimble's SketchUp Warehouse, for instance[9]). We would like to incorporate this vast repository of high-fidelity models into a simulation environment, to generate sensor data that is more like its real-world counterpart.

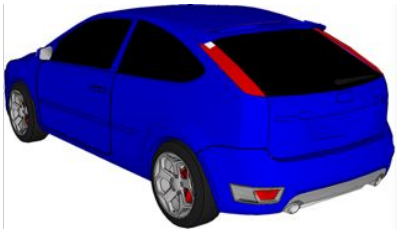
The quality of these models is often good enough - when sampled using our virtual LIDAR - to faithfully represent what is observed in real-world data. Figure 3.20 shows an illustration of various makes of cars observed around Woodstock in Oxfordshire, and correspondingly matched with CAD models on Sketchup:

[8] "Amateur" in this sense meaning unpaid, as opposed to lacking in quality.
[9] `http://sketchup.google.com/3dwarehouse/`

(a) Volvo X90

(b) Volvo X90: CAD

(c) Ford Focus

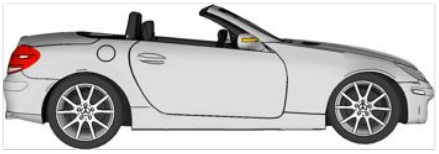


(d) Ford Focus: CAD

(e) Smart Car

(f) Smart Car: CAD

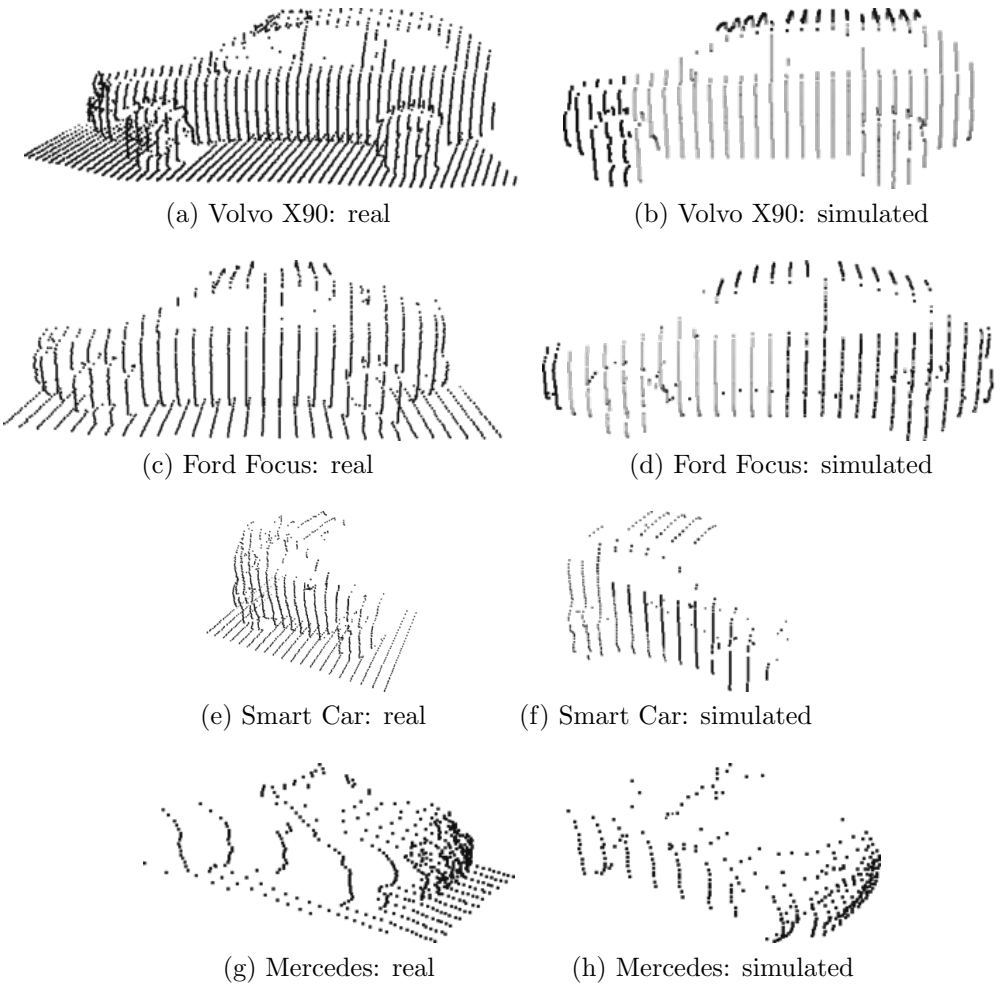
(g) Mercedes-Benz



(h) Mercedes-Benz: CAD

Figure 3.20: **A sampling of vehicles around Woodstock, Oxfordshire.** A collection of cars encountered on a typical run around Woodstock - (a) a Volvo X90, (c) a Ford Focus, (e) a Smart Car, and (g) a Mercedes-Benz SL class.

We can easily incorporate these models into our simulation framework, and the following images contrast the real-world data with the data generated by the simulator (using similar sensor configurations and orientations):



(a) Volvo X90: real (b) Volvo X90: simulated

(c) Ford Focus: real (d) Ford Focus: simulated

(e) Smart Car: real (f) Smart Car: simulated

(g) Mercedes: real (h) Mercedes: simulated

Figure 3.21: **A comparison of LIDAR data.** Real LIDAR data (left-column) compared against simulated LIDAR data (right-column) for the vehicles in Figure 3.20: (a) Volvo X90, (c) Ford Focus, (e) the Smart Car, and the (g) Mercedes.

Figure 3.21 illustrates the similarity between the real and simulated LIDAR data - in fact, the ground-plane was explicitly kept in the real LIDAR data, to provide some measure of differentiation.

Cars are of course the most obvious example - people tend to spend large amounts of time faithfully recreating their favourite vehicles. However the Google/Trimble warehouse exhibits a broad range of object classes - Figure 3.22 through Figure 3.24 show several commonly-encountered features of small-town England, along with their digital counterparts:

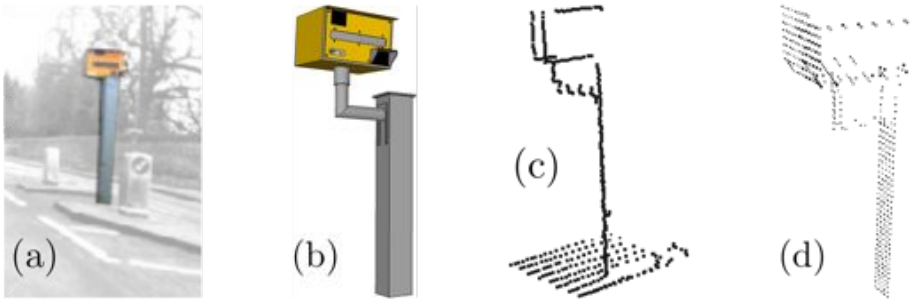



Figure 3.22: **Commonly encountered: Traffic cameras.** A traffic camera in Woodstock (a), its corresponding CAD model (b), the real-world LIDAR data (c), and the corresponding simulated data (d).

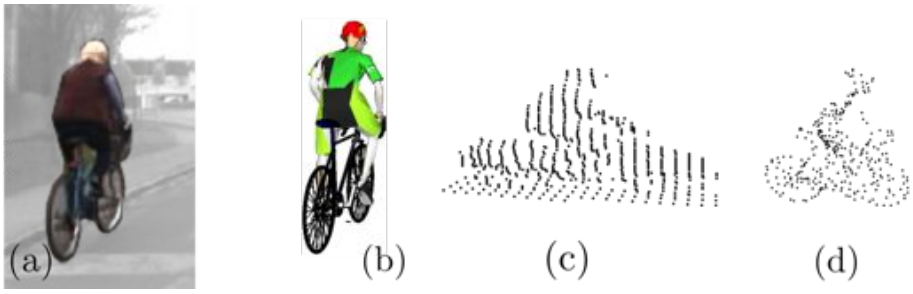



Figure 3.23: **Commonly encountered: Cyclists.** A cyclist in Woodstock (a), a somewhat similar CAD model (b), the real-world LIDAR data (c), and the simulated data (d). In some sense the CAD model is *too* accurate - however, this data does not incorporate a noise model, nor has it been dithered.


Figure 3.24: **Commonly encountered: Postboxes.** A regular sighting around English towns - free-standing postboxes (a), the CAD model (b), the real-world LIDAR data (c), and the simulated data (d).

As is evident from Figure 3.22 through Figure 3.24, objects that are not articulated or deformable have real and simulated point clouds that correspond to a much

greater degree than those that are - the cyclist, for instance, is not particularly well-represented.

An interesting point to note is that the cyclist point cloud - as extracted from a vertically-oriented planar LIDAR sweeping through the environment - has artificially elongated the point cloud along the direction of motion - a natural consequence of the relative velocities of the cyclist and the capture vehicle - these problems will arise again in Chapter 6 in a different context - vehicle localisation.

The entire framework is approximately $18k$ Lines-Of-Code (LOC), and makes use of the open-source rendering engine, OGRE in addition to components from the physics Engine, Bullet. Given that we now have the tools to generate arbitrary synthetic data, we move on to the description of the design and build of a sensor-package intended for producing the real-world equivalent cheaply, and reliably.

## 3.3 Producing real-world 3D surveys

We now turn our attention to producing equivalent, high-quality real-world surveys using inexpensive sensors. The grand vision here is to develop a stand-alone unit that could be employed in either a mapping, or localisation role. Given that the primary focus of this thesis is localisation, the following section articulates the design and build of the Navigation Base Unit (NABU).

### 3.3.1 Navigation Base Unit ("NABU")

The NABU is a first-generation sensor payload that is designed to be used in either a data-gathering ("mapping") or exploration ("localisation") role. Figure 3.25 shows a rendering of the NABU:

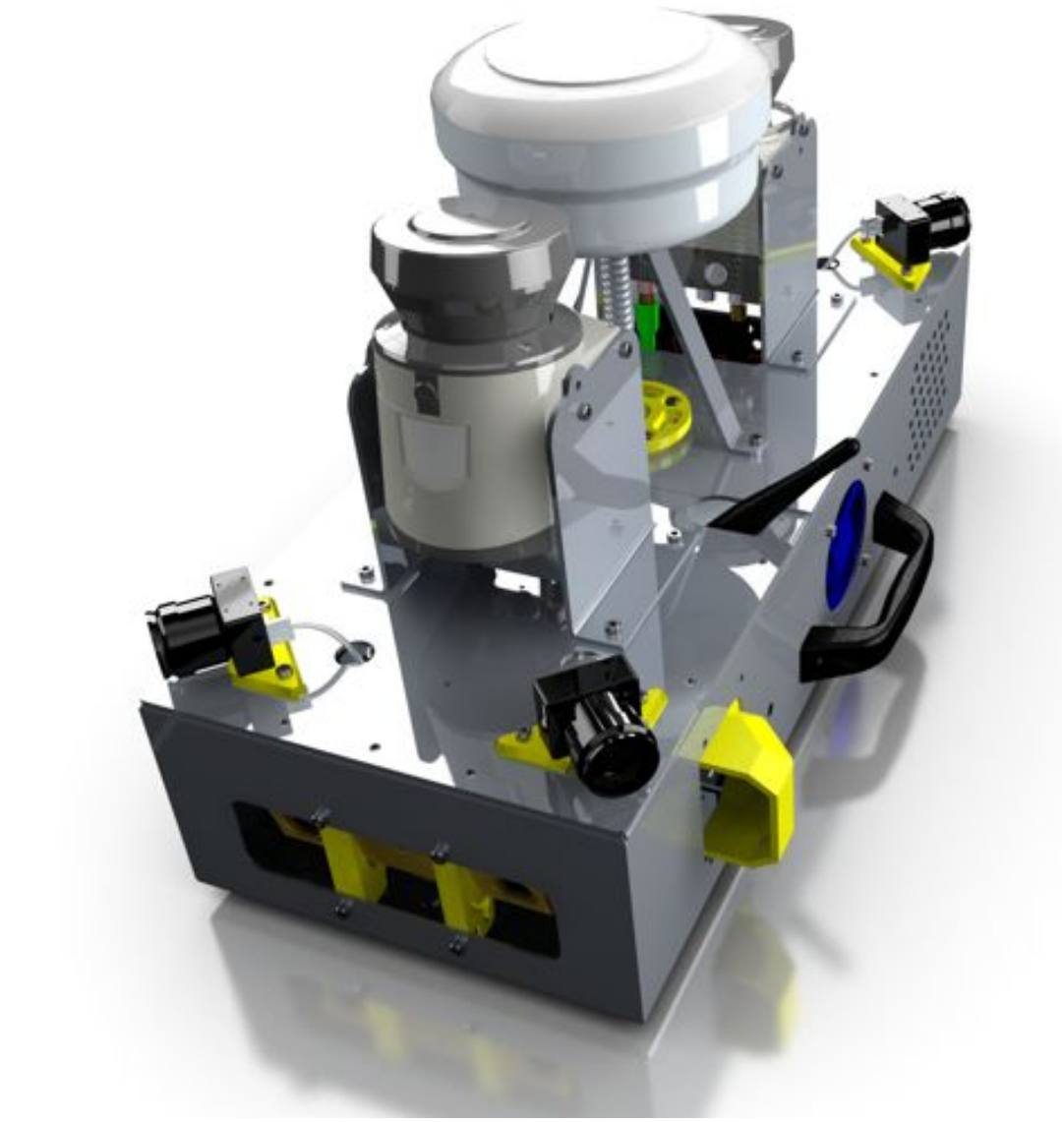


Figure 3.25: **NABU: CAD rendering.** A Computer-Aided Drawing (CAD) of the NABU. The NABU is a vehicle-agnostic stand-alone sensor suite that is designed to be used for mapping and/or localisation tasks.

The NABU was designed to incorporate a variety of sensing modalities - from vision (PointGrey Bumblebee2 and Firefly cameras) to scanning-LIDAR sensors (the SICK LMS-151) and a Trimble Global Positioning System (GPS) unit. Figure 3.26 shows an overview of the general payload proportions:

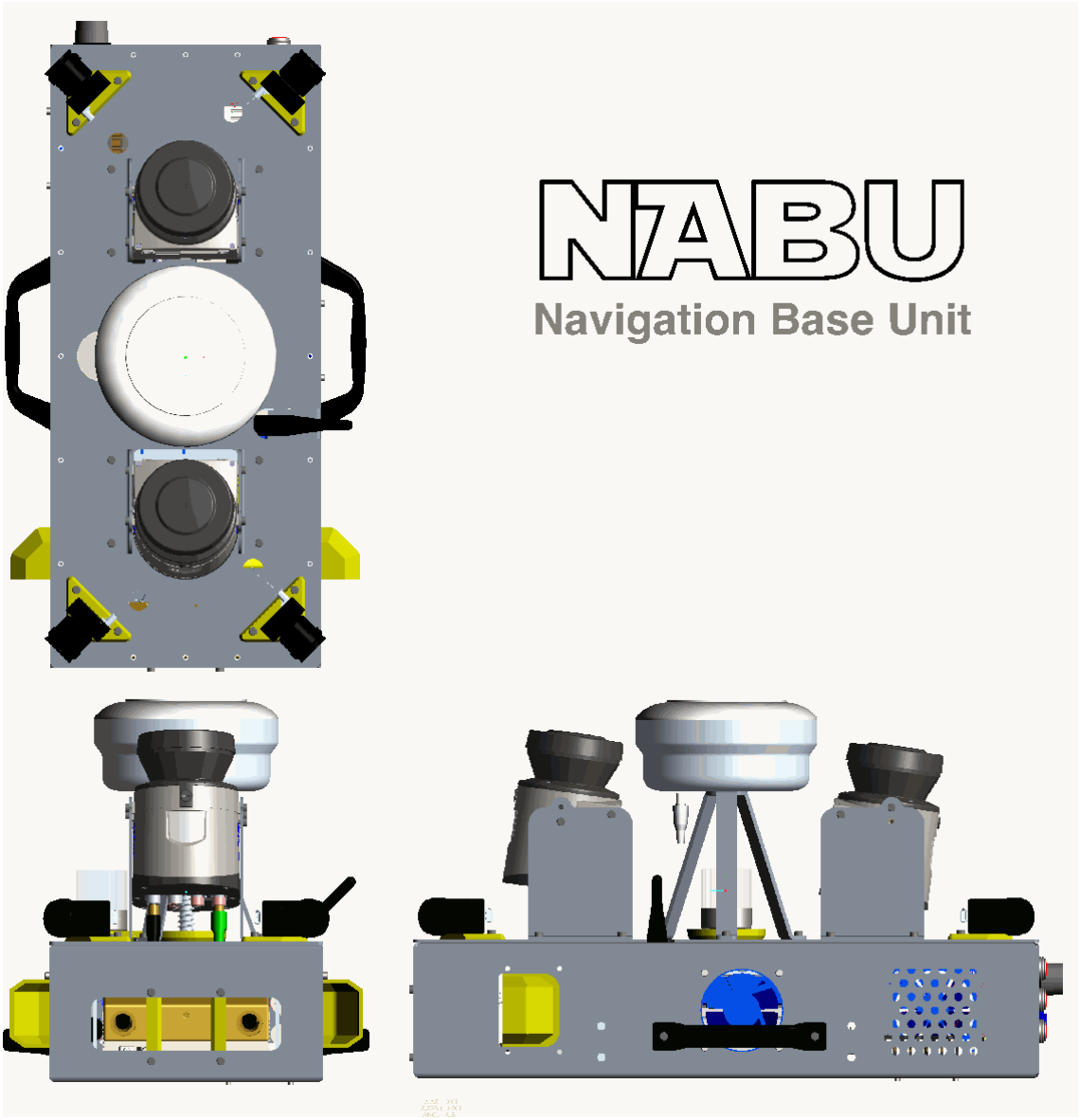
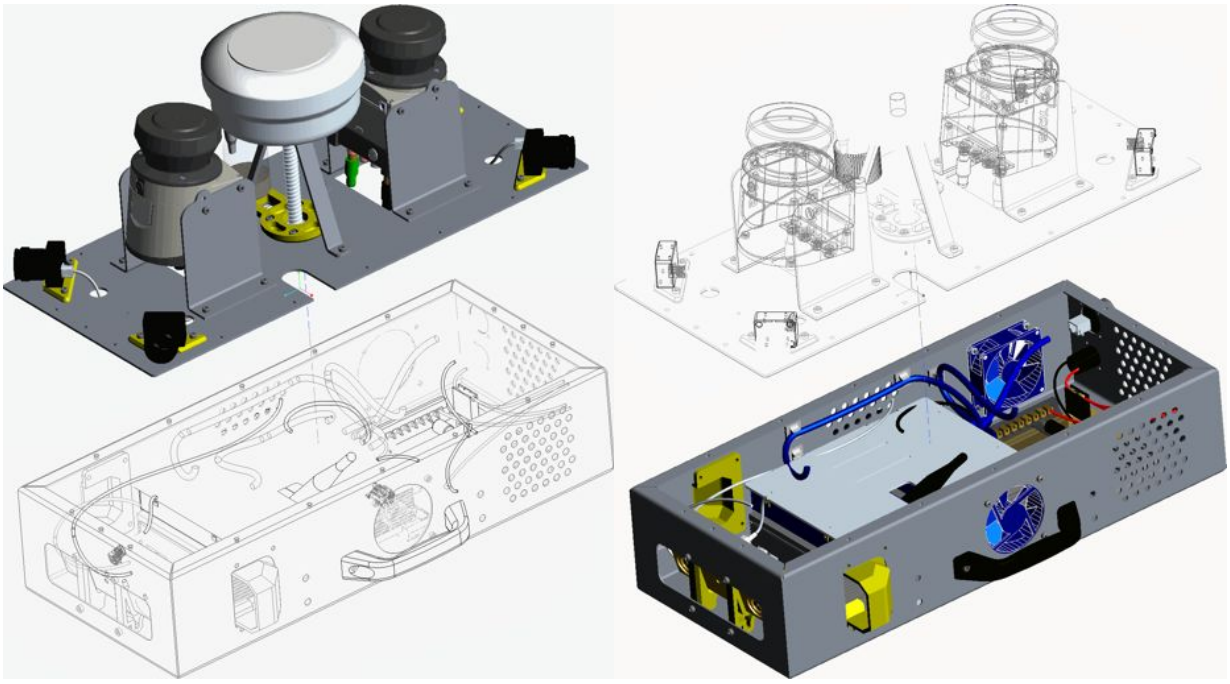



Figure 3.26: **NABU: Top, front and right-side views.** The top, front, and right views of the NABU. The unit was designed to be easily handled and mounted, and to allow for simple sensor re-configuration.

The NABU was designed to be easily transported, mounted, and interfaced with. The unit is mostly comprised of a 6061-grade aluminium, with supporting mounts and brackets manufactured through rapid-prototyping - these are rendered as yellow in all the CAD figures.

The design comprises of two major sub-assemblies, as is shown in Figure 3.27:

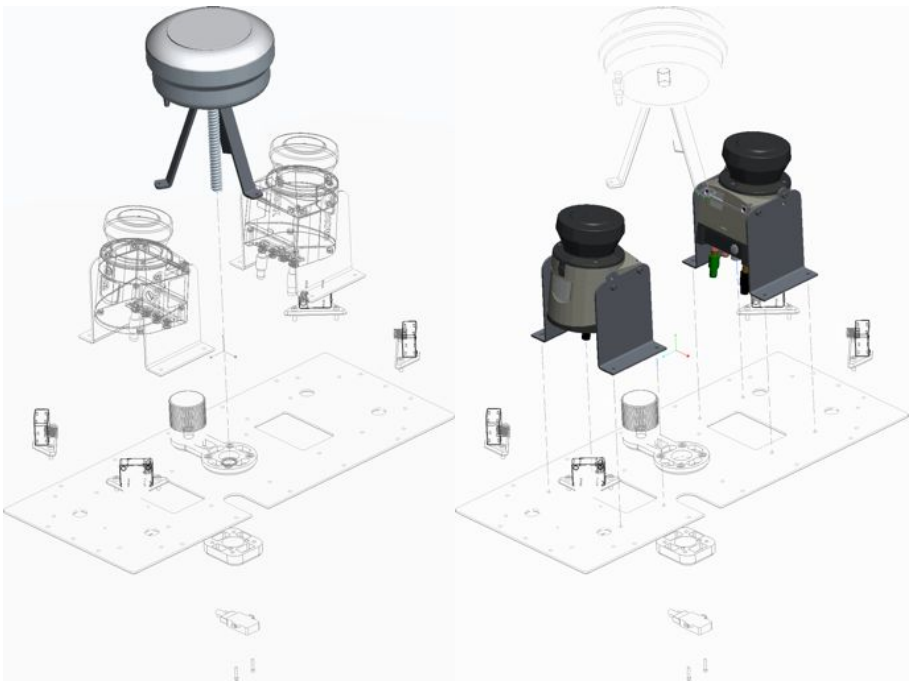


(a) NABU: Upper sub-assembly (b) NABU: Lower sub-assembly

Figure 3.27: **NABU: Sub-assemblies.** The two major sub-assemblies of the NABU. (a) Shows the upper sub-assembly (shaded) which contains the majority of the sensors, while (b) is the lower sub-assembly (shaded), which contains the computing and power-management components.

The upper sub-assembly in Figure 3.27 houses two SICK LIDARS, a Trimble R8 GPS unit, four PointGrey FireFly cameras, and a MicroStrain IMU. Each of the LIDARS has configurable mounts, allowing for either a declined, or inclined attitude (all of these images render the front LIDAR at an 8° incline, with the rear LIDAR at an 8° decline). The Firefly cameras are oriented around the periphery of the unit to cover as much of the surrounds as possible - however, the mount locations allow for rapid re-configuration if desired.

The bottom sub-assembly houses the on-board PC, a PointGrey Bumblebee2 camera, and the supporting electronics and cooling mechanisms to ensure reliable operation. The following sections detail the technical characteristics of each of the sensors.



(a) Trimble R8 (b) LMS-151

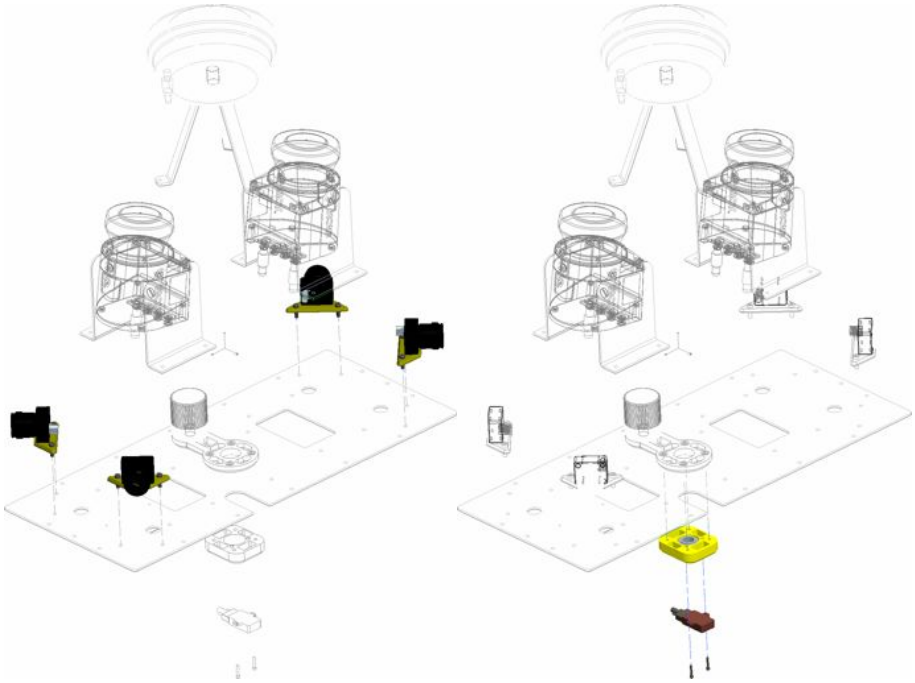
Figure 3.28: **NABU sensors: GPS and LIDAR.** Two of the sensor modalities of the NABU: (a) a Trimble R8 GPS, and (b) dual SICK LMS-151 scanning LIDARs.

**GPS** The Trimble R8 is a GPS/GLONASS compatible system that features a built-in UHF radio, allowing for either rover (mobile) or base-station (stationary) operation. Specifications from the Trimble Site[10] state that the accuracy of the unit is typically less than 5m Distance Root Mean Squared (DRMS), depending on ionospheric and multi-path conditions - multi-path conditions typically arise from "urban-canyon" environments, where the signal is reflected off of buildings, introducing error into the timing information.

**LIDAR** SICK manufacture a large number of scanning LIDAR devices - in this thesis we utilize the SICK LMS-151 series exclusively, which has a scanning range of up to 50m if the target has a remission value of 75% or greater. 100% remission

[10] `http://trl.trimble.com/docushare/dsweb/Get/Document-449956/R8-R6-5800Receivers_4A_UG_7440.pdf`

is defined to be the return from a diffuse reflecting white surface - therefore retro-reflective surfaces can have a remission value greater than 100%.



(a) Point Grey Firefly (b) Microstrain IMU

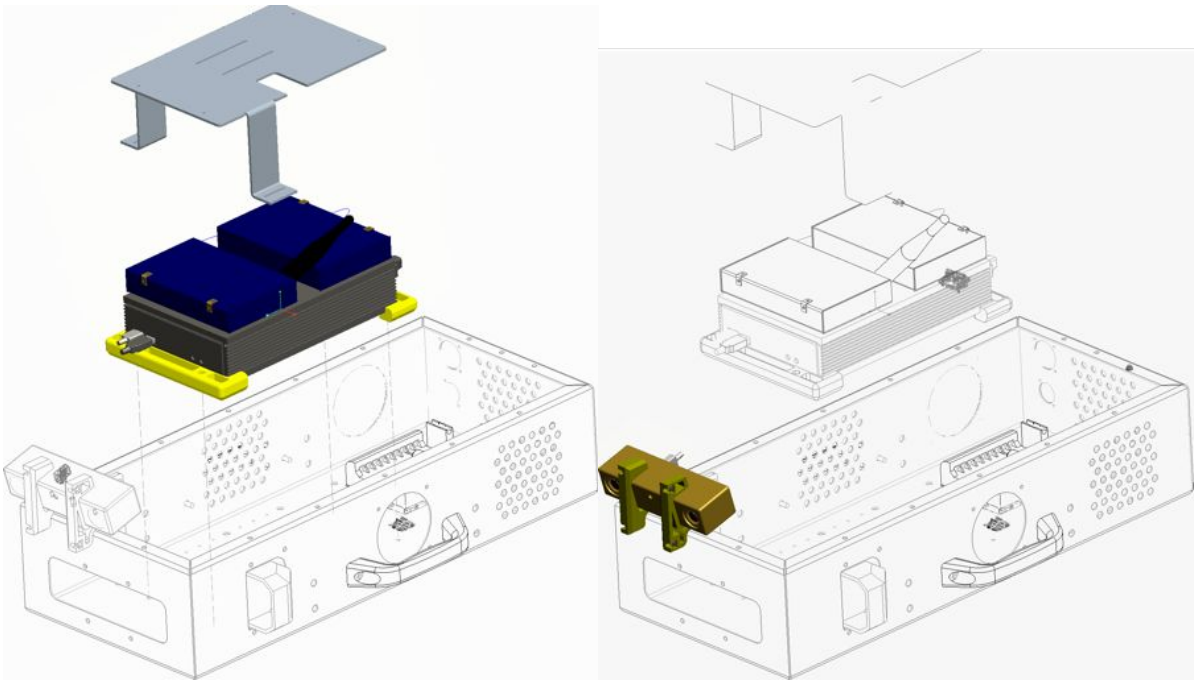
Figure 3.29: **NABU sensors: Firefly cameras, and IMU.** The remaining sensor packages on the top sub-assembly are the (a) four Point Grey Firefly monocular cameras, and (b) a MicroStrain Inertial Measurement Unit (IMU).

**Firefly cameras** The NABU has mount-points for four Firefly FireWire cameras from Point Grey (rendered in solid in Figure 3.29(a)) - these are monocular cameras, producing 752x480 images at 60 frames-per-second (fps).

**IMU** The Inertial Measurement Unit (IMU) is a MicroStrain 3DM-GX3-25, a Micro-Electro Mechanical System (MEMS)-based sensor that is capable of providing tri-axial angular rate and acceleration data at 100Hz (in addition to magnetometer and temperature readings).

The upper sub-assembly contains the majority of the sensor hardware, while the base sub-assembly contains the on-board computer, networking equipment and

power electronics as detailed in the following sections.

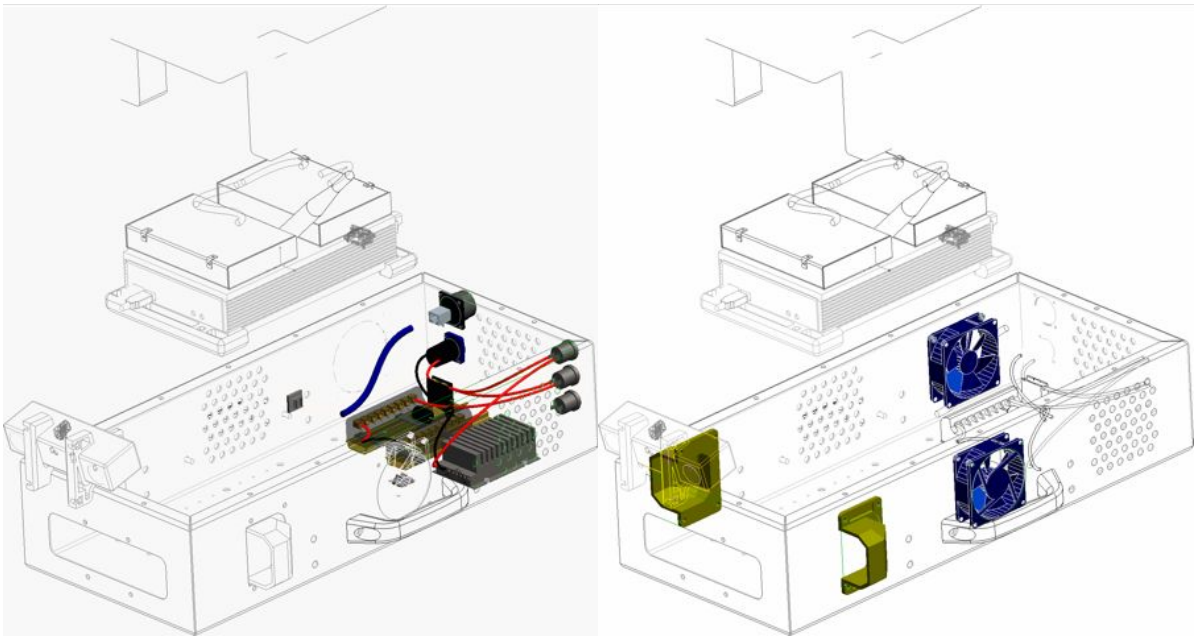


(a) Onboard PC (b) Bumblebee2

Figure 3.30: **NABU: On-board PC and Bumblebee2 camera.** Shown in (a) is the combined PC/Networking stack, consisting of a fan-less low-power PC, a router, and a wireless access point (WAP). (b) is a Point Grey Bumblebee2 stereo camera.

**Computing and Networking** The on-board PC consists of a fan-less dual-core Atom-based computer with 2GB of RAM, and a variety of I/O connections. Given the requirement of being able to connect to the unit wirelessly, the NABU also has a full networking stack with a dedicated Netgear access point. Also incorporated is a router, both for interfacing with the SICK LIDARs and to provide a gigabit external connection for rapid data offload.

**Bumblebee2** The PointGrey Bumblebee2 is a stereo-vision camera that is widely used for vision tasks - for example, Visual Odometry and Dense stereo - this is discussed further in section Chapter 4. The Bumblebee2 used in the NABU produces a 512x384 stereo pair at 20Hz.



(a) Electronics (b) Cooling

Figure 3.31: **NABU: Power electronics and cooling units.** The electronics shown in (a) provide the necessary voltage rails and power-regulation for the different sensor types. (b) shows the fans (blue) and air ducts (yellow) responsible for cooling the unit when stationary and moving, respectively.

**Electronics and cooling** The heterogeneous mixture of sensors on the NABU require dual voltage rails. The NABU takes - as sole power input - a 12V source, which is then transformed (by means of a DC-DC converter) to 24V to provide power for the LIDARs. The PC and fans all run off of the 12V rail - total power consumption for the NABU in logging mode is less than 60W - and can run easily off of a typical cigarette-lighter or auxiliary power-source.

As the PC is of the fan-less variety, ensuring an active airflow over the cooling fins is paramount. When the unit is stationary, cooling is provided for by two 12V PC fans, mounted in opposing directions (see Figure 3.31(b)). This allows for cooling of both the PC and the electronics, by ensuring a constantly circulating air mass. When in motion, the front air-scoops and rear-vents ensure a consistent airflow over

the PC and electronics.

Given the wide array of sensing modalities, the NABU can be used for a number of tasks, including localisation via scan-matching (using the LIDARs), Visual-Odometry (Bumblebee2 camera), metric mapping (using the LIDARs in conjunction with the VO), and appearance-based topological mapping (with the FireFly cameras). In the following section, the build and deployment of the NABU into its target role is discussed.

### 3.3.2 Construction and deployment

Figure 3.32 shows the NABU during construction. Highlighted are the locations of the IMU, the Bumblebee2 and the on-board PC. Visible in the front are air-scoops to ensure cooling from the oncoming air-flow when mounted on top of a vehicle.

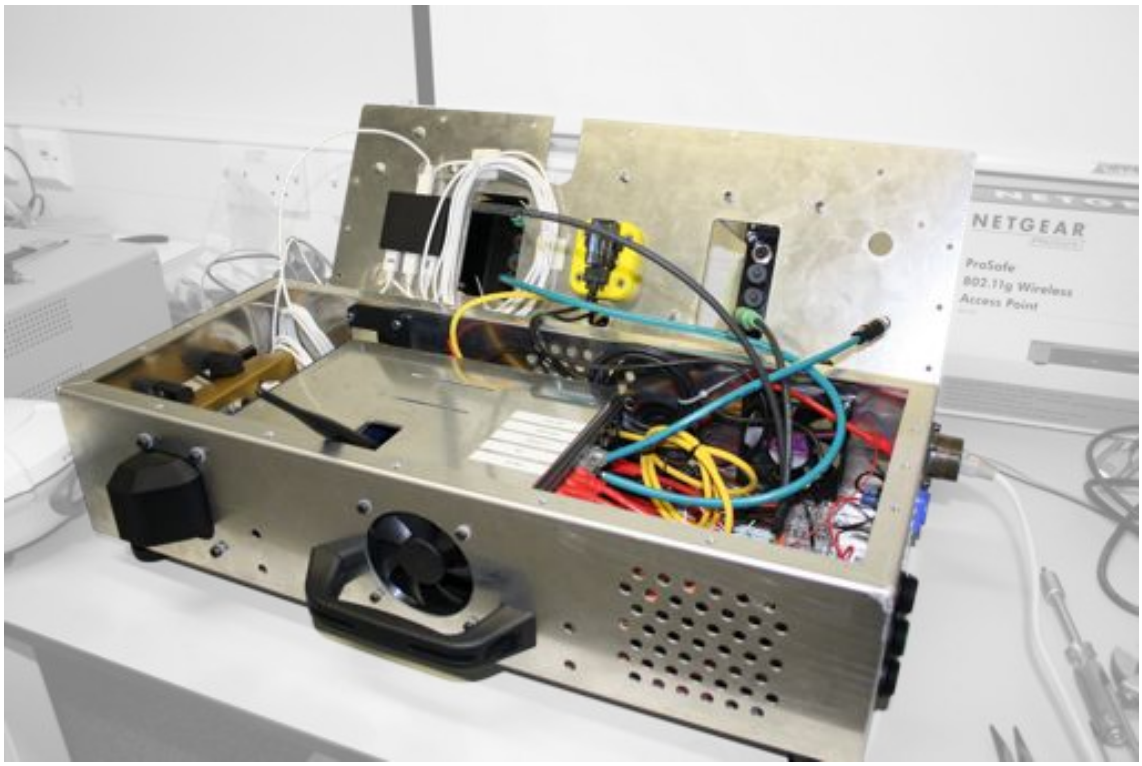



Figure 3.32: **NABU: Construction.** Visible in this image is the Bumblebee2 stereo camera from PointGrey (left side of the image), the MicroStrain IMU (top) and the air-scoops and fans for cooling.

Clearly visible in this image is the IMU (mounted centrally on the top subassembly), the FireWire relay-box, necessary for running multiple cameras off of one bus (adjacent), the aerial of the WAP of the networking stack (lower-left), and the input ports and control switches (right-hand side). Figure 3.33 shows the NABU fully assembled and mounted atop the Wildcat:

Figure 3.33: **NABU: Mounted on the Wildcat.** The NABU mounted on the MRG Wildcat. Visible are the PointGrey FireFly cameras (front-left, front-right), the SICK LIDARs (both inclined and declined) and the Trimble R8 GPS unit.

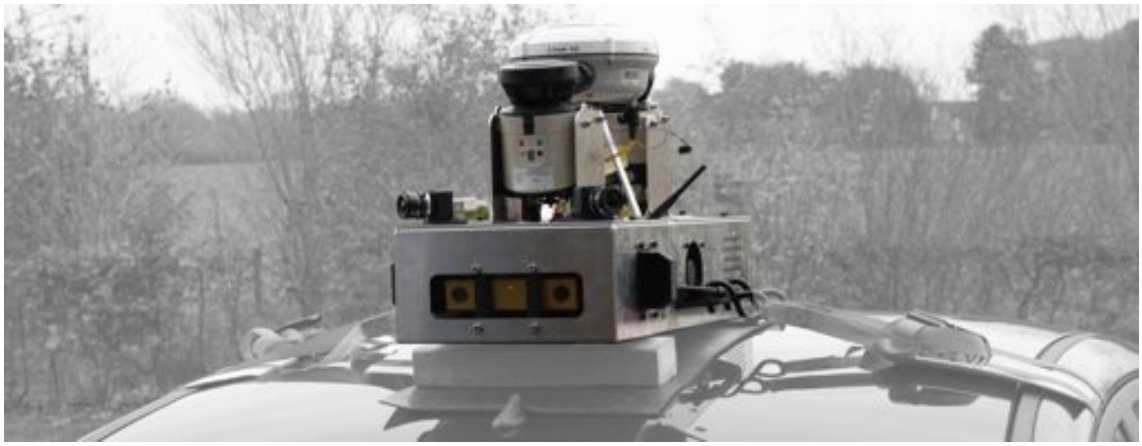


Figure 3.34: **NABU: Dataset collection.** The NABU mounted on a Ford Focus, about to collect data around Piccadilly, London.

The ad-hoc mounting system visible in Figure 3.34 is not the preferred state - the unit is designed to interface with 80/20 vehicle-mounting brackets, however these were not available at the time of this particular mission. To date, the unit has collected in excess of 600GB of data, over more than 80 km of trajectory around Begbroke and London.

## 3.4 Summary

This chapter has detailed the design and development of systems to aid the goal of outdoor vehicle localisation. Tools for both the modelling and simulation of real-world environments and the means to capture - easily and reliably - such representations using real-hardware have been developed.

In the following chapter, we explore the theory and application necessary to perform long-term, large scale localisation using simple sensors.

Figure 3.35: **NABU dataset: Centre of London.** An exemplar dataset collected by the NABU, driving past the Houses of Parliament - note the excellent reflectivity map. (Thanks to Ashley Napier for the point cloud data from this run)

# Chapter 4

# Large-Scale Urban Localisation with a Pushbroom LIDAR

## 4.1 Introduction

As discussed in Chapter 2, a core competency for any autonomous vehicle is the ability to stay localised in a prior map over the long term. The goal of this thesis is to produce a system that is capable of providing such estimates relying solely on cheap, readily available, 2D LIDAR sensors.

In contrast to SLAM - where the robot has no a-priori information about its surrounds - we utilise a heavily-instrumented survey vehicle which is responsible for collecting data in the target workspace (a town center, for instance).

We will refer to this survey as an "experience", consisting of laser-sweeps, poses, and velocities (both angular and linear). We adopt this terminology to emphasize that we do not need a globally correct metric map for our road localisation task. The end-goal is then to have fleets of vehicles leveraging this data utilising inexpensive sensor suites, such as the "NABU", described in Chapter 3.

Given the constraints of our proposed sensor payload, we require our localisation

scheme to have no reliance on 3D laser, differential GPS or integrated inertial systems. Figure 4.1 shows an exemplar survey vehicle, the MRG "Wildcat":


Figure 4.1: **The MRG "Wildcat".** The Wildcat is a heavily-instrumented platform initially developed by BAE systems. The vehicle is equipped with an Oxford Technical Solutions Inertial Navigation System and a host of sensing modalities including LIDAR and camera units.

This heavily modified Bowler Wildcat was developed by BAE systems, and has powerful on-board processing capabilities in addition to an Oxford Technical Solutions RT3000 Inertial Navigation System (INS). Visible in front of both headlights are rigidly mounted SICK scanning LIDARS for the express purpose of localisation.

As an illustration of a typical target environment, consider Figure 4.2. This shows the Begbroke Science Park in Oxford overlaid with 26 km of INS trajectory data collected around the road network over a 3 month period:

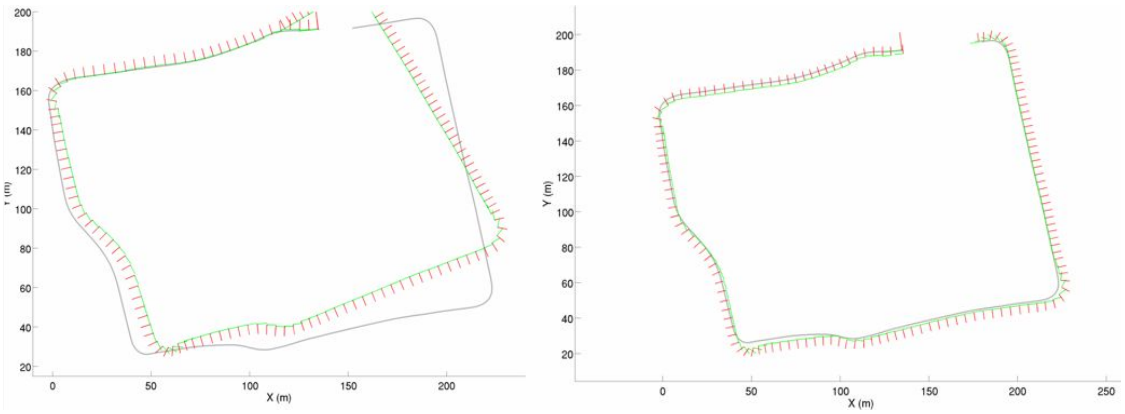

Figure 4.2: **26 kilometres of trajectory data around the Begbroke Science Park.** A substantial dataset comprising of 3 months of INS data (26km total distance) around the Begbroke site in Oxford, contrasting areas of good GPS reception with poorer areas (predominantly the northern and southern sections). As can be seen in the lower section of the image, the trajectories exhibit a large variation, caused mainly by GPS signal degeneracy due to foliage. The majority of the trajectories (black, solid) are counter-clockwise, with a single clockwise loop (red, dotted).

Shown in Figure 4.2 are the indicated trajectories from the Wildcat INS as it traverses the science park. Note that in the sections unobstructed by foliage and buildings (predominantly the north-east and western sections), the performance is consistent and the estimated vehicle path follows the road shape - however in the southern section a small copse of trees inhibits DGPS corrections and the INS-estimated path diverges wildly from the road. This effect, noted in Chapter 2

amongst the DARPA teams, is highly detrimental to any form of autonomous vehicle operation. Relying on these erroneous INS pose estimates for higher-level tasks (such as trajectory planning) will result in system failure.

Given that we are restricted to a 2D LIDAR, an initial attempt at resolving the trajectory ambiguities in Figure 4.2 might be to mount the LIDAR horizontally and build a 2D representation of the Begbroke site - similar to the approach in [13] - this would then be used subsequently to localise against.

To validate whether this concept is feasible, a single map instance was generated around Begbroke and then subsequently revisited over several months, using an ICS scan-matcher to localise the run-time data with the pre-recorded map. Figure 4.3 shows the results of the ICS-matcher running in both *open-loop* and *localisation* mode:



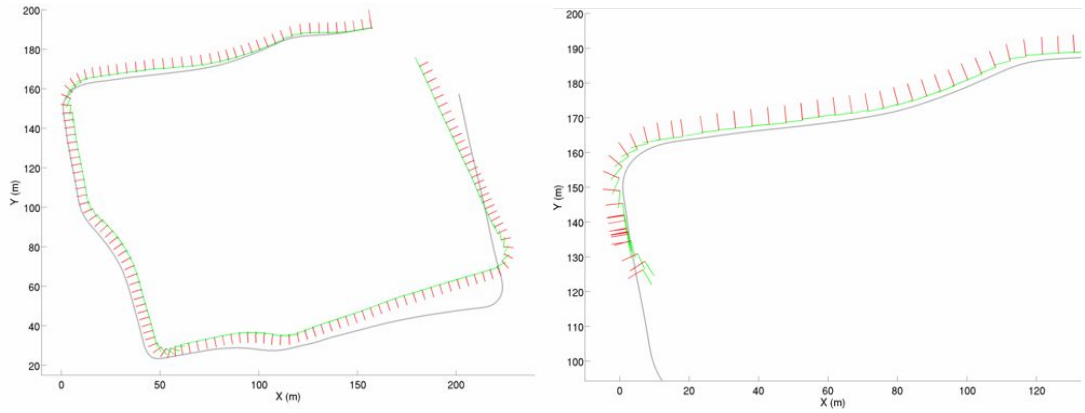

(a) ICS: Open-loop performance (b) ICS: Localisation performance

Figure 4.3: **Scan-matching comparison: Open-loop vs. Localisation.** The left image is the open-loop (no map) performance of the scan-matching algorithm, and the right image is the *localisation* performance of the algorithm - with the same data - given a reference map, around a single loop of the Begbroke Science Park. The INS trajectory is shown in grey, the (sub-sampled) ICS pose estimates in colour.

The left plot of Figure 4.3 - and the following figures - serve as control experiments; we want to know if (or when) the scan-matcher fails, whether the failure is due to the run-time data, or degeneracies in the map. The reference map - used

during *localisation* in Figure 4.3(b) - was from a loop earlier on the same day, and as expected the localisation works well.

Figure 4.4 shows a similar plot for data collected 63 days after the map was collected, but using the **same** map as Figure 4.3(b):




(a) ICS: Open-loop performance (b) ICS: Localisation performance

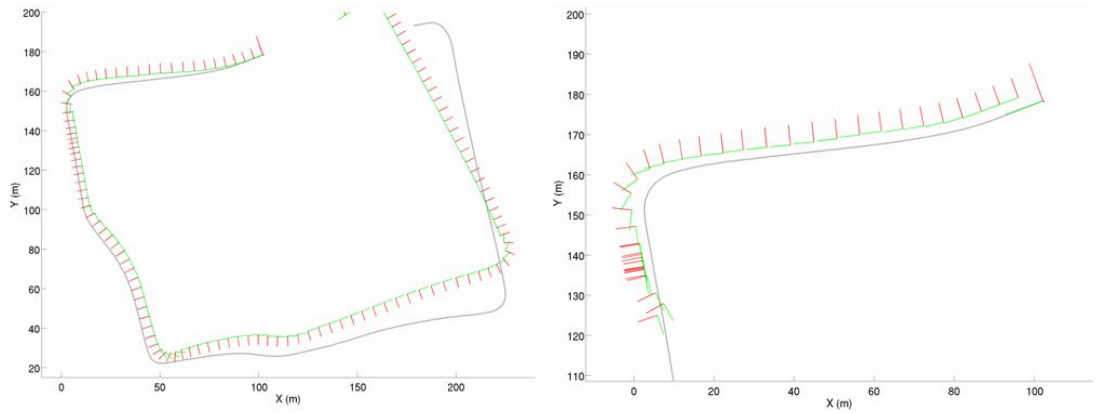
Figure 4.4: **Scan-matching comparison: Open-loop vs. Localisation, 63 days later.** Open-loop (a) and localisation (b) performance for an ICS scan-matcher, using a map from 63 days prior. Notice the localisation failure at $(x = 0, y = 120)$ - this is due to distinct scene change in the intervening months.

Very apparent from Figure 4.3(a) and Figure 4.4(a) is that the open-loop performance is consistent; however, we notice failures occurring at grid coordinates $\{x = 0, y = 120\}$. The following image sequence is a comparison of images acquired from a bumper-mounted camera for these datasets:

(a) Map Acquisition (b) Acquisition+63

Figure 4.5: **Scene change over 63 days.** Distinct scene change over a 63 day period, both in terms of weather and geometry. Noticeable in (a) but missing in (b) are fence-boards, whose absence causes a distinct change in appearance (in LIDAR space) of the run-time vs. map data.

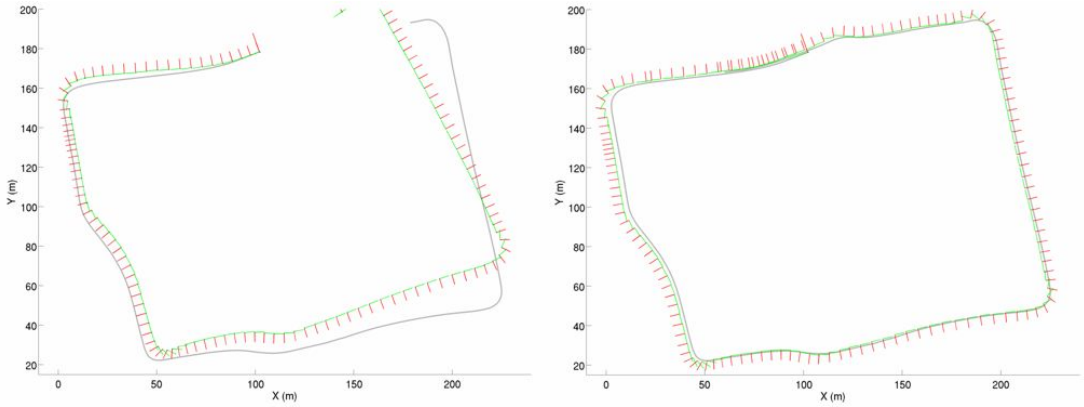
To verify that this performance was consistent, a similar experiment was run for data collected 64 days after map acquisition:




(a) ICS: Open-loop performance (b) ICS: Localisation performance

Figure 4.6: **Scan-matching with a fixed-map over the long-term, 64 days later.** Open-loop (a) and localisation (b) performance for the ICS matcher for a dataset taken 64 days after map acquisition. Notice the similar failure location to Figure 4.4(b).

As a final check, we can compare a map from +63 days with runtime data from +64 days:




(a) ICS: Open-loop performance (b) ICS: Localisation performance

Figure 4.7: **Scan-matching comparison: Open-loop vs. Localisation, 1 day difference.** With 1 day separating the "map" and runtime data, we can see that the open-loop performance is consistent (a), **and** that the localisation performance is consistent (b).

In Figure 4.7, we can see the consistent performance of the scan-matching algorithm in open-loop **and** localisation mode - there has not been much scene change in the intervening day, and therefore localisation works as expected.

The underlying issue here is that there is noticeable scene change - from the point of view of the horizontal 2D LIDAR - from map acquisition to runtime, 63 days later. Of course the majority of the scene has not changed, which is reflected in a cursory visual appraisal of the scene - building walls are still in the same place. Given that the perception of the world consists solely of a 2D "slice", it is understandable that failures occur when the world changes.

It is worth pausing here to reflect on what these graphs represent. Begbroke is a benign, constrained environment in terms of both traffic (vehicle and pedestrian) and scene change (one slowly changing building site). Given that - even in this constrained environment - we cannot rely on 2D localisation as a basis for long-term localisation, we seek to develop a system that is robust to scene-change, both in the long, and short, term.

The core idea is to use the 2D LIDAR to acquire 3D data - this is done by

deploying the LIDAR in a "pushbroom " configuration, and this is detailed in the following sections.

## 4.2 Synthesising 3D data with a pushbroom LIDAR

In this section, we develop the notion of the run-time exploitation of prior 3D maps using a single 2D pushbroom laser - a fixed scanning LIDAR sensor which generates data from being "pushed" through the world by a vehicle, as illustrated by Figure 4.8:

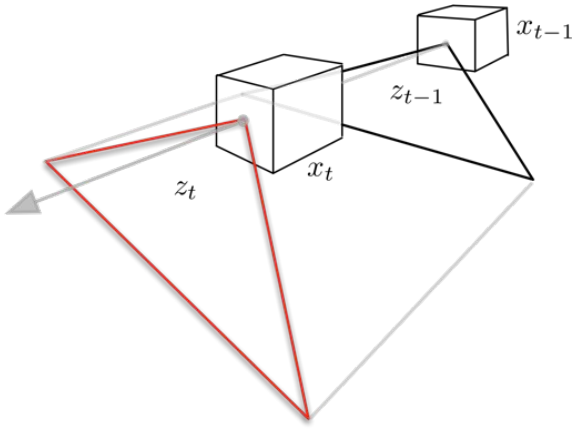



Figure 4.8: **Pushbroom LIDAR.** An illustration of the "pushbroom" LIDAR approach. 3D maps are acquired from a 2D LIDAR by inducing out-of-plane motion through the LIDAR fan. Shown here are two consecutive poses, $x_{t-1:t}$ and their associated measurements, $z_{t-1:t}$.

Conventional approaches for generating 3D point data from a scanning planar LIDAR are shown in Figure 4.9, including a static "angel-wing" configuration (a), mounting the unit on a mechanism rotating about the primary axis (b), or oscillating the LIDAR on its secondary axis (c). In the actuated configurations, very accurate timing is needed between the LIDAR timestamps and the rotation encoder in order to ensure accurate point projection. (In all cases, accurate timing correspondences between the on-board data-logging machine and the LIDAR are required).


(a) Lisa (b) Bobcat (c) Homer

Figure 4.9: **Generating 3D point clouds from 2D LIDARs.** (a) The Lisa platform, based on a Segway RMP. (b) A Bobcat vehicle, with a rotating 2D LIDAR, as developed by [14] (c) The Homer research platform with a "nodding" LIDAR.

In the proposed system, we intentionally orient our single laser downwards - seeking out ground strike. In this way, we convert our 2D LIDAR sensor into an intrinsically 3D sensor, by inducing out-of-plane motion through the sensor fan. Unfortunately, we can no longer directly infer translational and rotational changes through 2D scan-matching.

However, by integrating velocity estimates of the vehicle motion over a window, we can generate a retrospective swathe of 3D laser data. This dense 3D "recent history" can then be continually aligned within our survey 3D point cloud by considering and matching the population statistics of the swathe and survey patch. We now consider in detail this generative process and various methods for estimating the vehicle velocities.

## 4.3 Localisation with a pushbroom LIDAR

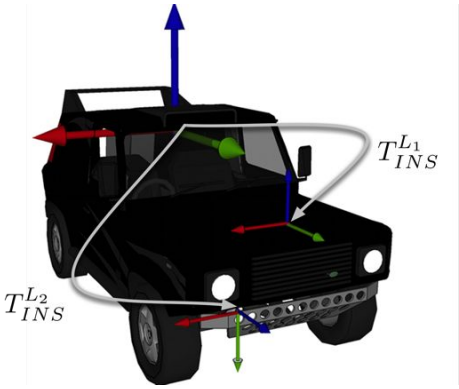



Figure 4.10: **Coordinate frames and conventions.** The coordinate frames of the Wildcat and the constituent sensor systems: $T_{INS}^{L_1}$ from the INS to the horizontal LIDAR, and $T_{INS}^{L_2}$ to the declined LIDAR. All coordinate systems are right-handed, and the $X$, $Y$ and $Z$ axes are in red, green, and blue respectively.

Figure 4.10 shows the coordinate conventions used throughout this section. Figure 4.11 illustrates an example swathe obtained from the declined LIDAR mounted on the Wildcat (Figure 4.10). Indicated in the figure is the motion of the vehicle through the scene:

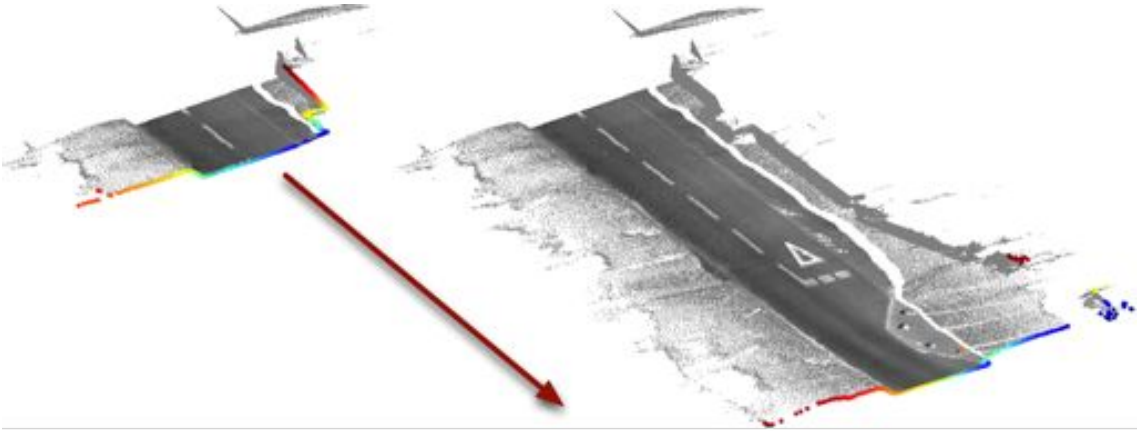


Figure 4.11: **Example 3D data from a 2D pushbroom LIDAR.** The process of generating 3D point clouds from the egomotion of the declined LIDAR through the scene, highlighted by the arrow. If we observe the rotational and linear velocities of the sensor, we can integrate these to produce this 3D cloud. Particularly visible are the retro-reflective road-markings and lane demarcations.

We define the generation of a run-time point cloud $\mathcal{Q}$ as:

$$\mathcal{Q} \leftarrow \Phi(\mathcal{Z}_{t-N:t}, x_{t-N:t}) \tag{4.1}$$

where $\mathcal{Z}_{t-N:t}$ are raw LIDAR scans, $x_{t-N:t}$ is a retrospective relative *Special Euclidean* 2 ($\mathbb{SE}2$) rigid pose chain, $N$ is the window length in seconds, and $\Phi$ is a function that projects LIDAR scans into $\mathbb{R}^3$. A laser-scan at time $t$ is defined as:

$$\mathcal{Z}_t = \{r_1, \ldots, r_{541}, i_1 \ldots i_{541}\} \tag{4.2}$$

where $r_n$ denotes the laser range reading (in meters) for beam $n$ of scan $\mathcal{Z}_t$ and $i_n$ is the intensity of each of the 541 beams in the SICK LMS-151. To obtain the relative pose-chain, we utilise both the observed rotational velocities $\Omega(t)$ around the three ordinate axes of the vehicle (roll, pitch, yaw), and the linear velocities given by some extrinsic source. Given these velocities, we can integrate the state equation:

$$\dot{x}(t) = \begin{bmatrix} v(t) \cdot \begin{bmatrix} cos(\int_{t_0}^{t} \Omega_z(t))\mathrm{dt} \\ sin(\int_{t_0}^{t} \Omega_z(t))\mathrm{dt} \end{bmatrix} \\ \Omega_z(t) \end{bmatrix} \tag{4.3}$$

to produce the $\mathbb{SE}2$ pose-chain, $x_{t-N:t}$ (consisting of Cartesian position, and orientation). We can then project the LIDAR ranges using $\Phi(\cdot)$ thereby generating the swathe $\mathcal{Q}$. Figure 4.12 shows an overhead view of the 3D point cloud $\mathcal{P}$ - the survey point cloud - with an exemplar swathe, $\mathcal{Q}$:

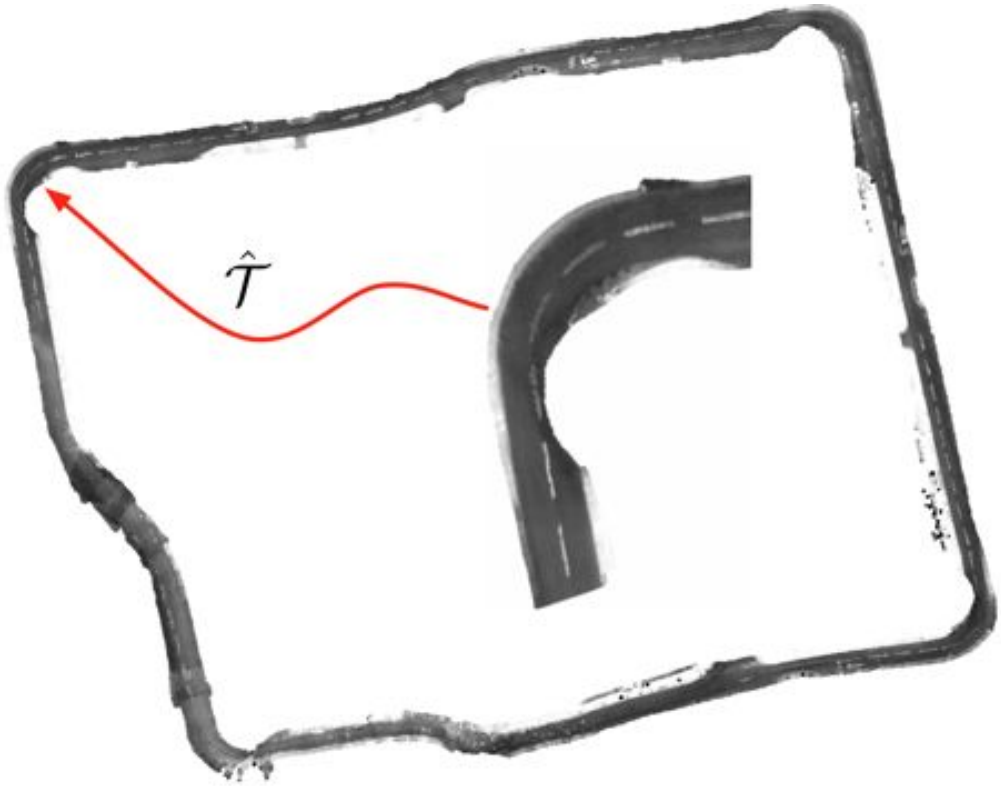



Figure 4.12: **Run-time swathe and 3D prior.** Overhead view of the 3D point cloud $\mathcal{P}$ developed during the original experience (outer image), coloured by reflectance. Shown in the centre is an example swathe, $\mathcal{Q}$. Our task is to determine the transformation $\hat{\mathcal{T}}$ that best aligns $\mathcal{Q}$ with $\mathcal{P}$. Note that the scales across these images differ for the purposes of illustration.

The tracking problem is - given the survey point cloud $\mathcal{P}$, and the swathe developed during runtime, $\mathcal{Q}$ - to establish a transformation $\hat{\mathcal{T}}$ that best aligns the clouds. We restrict the transformation space to be contained within the Special Euclidean 2 group ($\mathbb{SE}2$) - again we posit that for a large number of the tasks required of an autonomous vehicle (path and trajectory planning) the $\mathbb{SE}2$ pose is sufficiently descriptive. However, we have assumed so far that the velocities over the window are given - a valid question now is how we obtain these estimates from sensor data.

### 4.3.1 Ego-motion estimators

The Wildcat has access to velocity estimates through the INS - an Oxford Technical Solutions RT3000 unit. This system is equipped with a Satellite-Based Augmentation System (SBAS), which - in the case of the UK - is supplied by the European Geostationary Navigation Overlay Service (EGNOS). The unit is quoted as having .2 metre Circular Error Probable (CEP), although as shown by Figure 4.2, we can see that this value is regularly violated with adversarial satellite locations and ionospheric conditions, and the presence of buildings and foliage.

The INS provides pose estimates, velocity estimates (both linear and angular) as well as tri-axial accelerations. We cannot, of course, expect fleets of vehicles to have access to such a system - although we will utilise the INS velocities estimates as a baseline for performance when comparing other methods. Two other methods for estimating velocity are LIDAR Odometry (LO) and Visual Odometry (VO).

#### Visual Odometry

We discussed the use of VO previously in Chapter 2 as a state-estimator - we now seek to make use of it for velocity estimation. (We are not interested in matching features of the visual system into a pre-defined map for localisation purposes - rather, we seek to use the relative pose estimates to generate both linear and angular velocity feeds to generate our LIDAR swathe)

Figure 4.13 and Figure 4.14 shows the linear and rotational velocities obtained from the numerical differentiation of pose estimates from a VO system (implemented by Winston Churchill of the Mobile Robotics Group [21]) over 80 seconds of data in the Woodstock area of Oxfordshire:

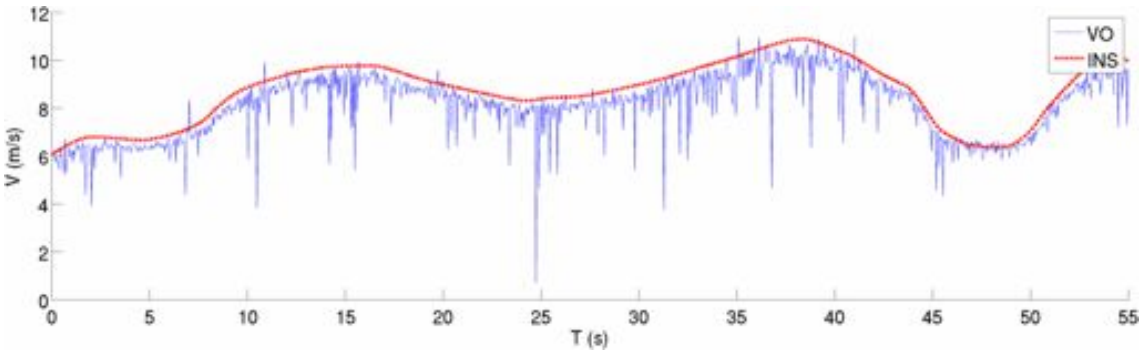



Figure 4.13: **Visual Odometry: Linear velocity estimates.** Linear velocity estimates over 80 seconds around Woodstock in Oxfordshire, using the VO system. The ground-truth (INS) velocity is shown in red, with the instantaneous VO velocity estimates shown in blue.

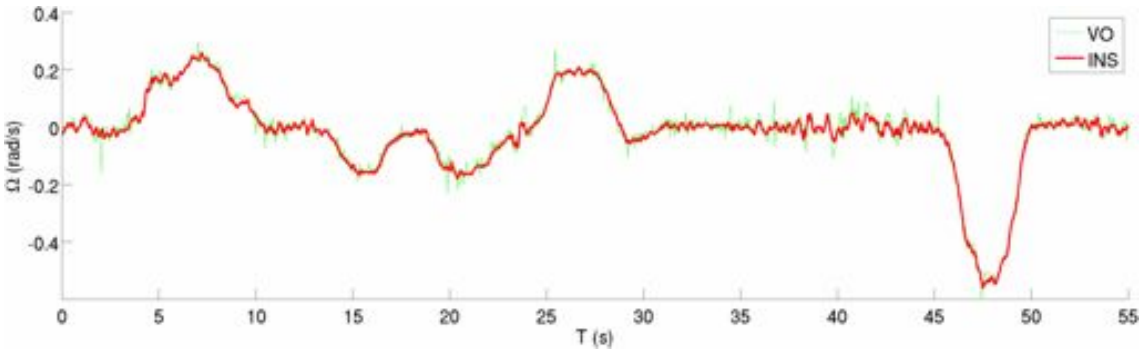



Figure 4.14: **Visual Odometry: Rotational velocity estimates.** Rotational velocity estimates (yaw) over the same 80 seconds using the VO system. The ground-truth (INS) yaw is shown in red.

As can be seen from the figures, the rotational velocity tracks the INS estimates particularly well. However the linear velocity estimates exhibit a slight bias, which is particularly detrimental to our swathe generation process - the consistent under-estimate leads to "compressed" point clouds and correspondingly erroneous pose estimates. This stereo-bias is a known issue when estimating motion from far-field features - as is detailed in [75] - and as such, we move on to estimating relative motion using LIDAR.

### LIDAR odometry

LIDAR Odometry (LO) utilises a horizontally-mounted 2D LIDAR scanner and scan-matching to produce pose-estimates in an open-loop fashion, as discussed in Chapter 2, and earlier in this chapter.

LO makes assumptions about the planarity of the world and is subject to error induced by real-world factors such as ground-strike, dynamic obstacles in the scene, and so on. However, as can be seen from Figure 4.15 and Figure 4.16 - which are the corresponding LO estimates of the linear and rotational velocity of the data shown in Figure 4.13 and Figure 4.14 - LO serves as a good (if noisy) estimator of both linear and rotational velocities:

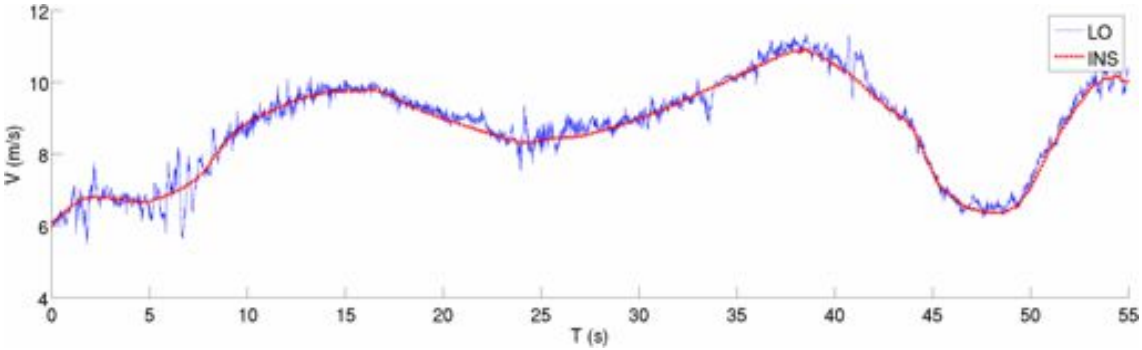



Figure 4.15: **LIDAR Odometry: Linear velocity estimates.** Linear velocity estimates over the same 80 seconds as Figure 4.13 using the LO system. The ground-truth (INS) velocity is shown in red, with the instantaneous LO velocity estimates shown in blue.

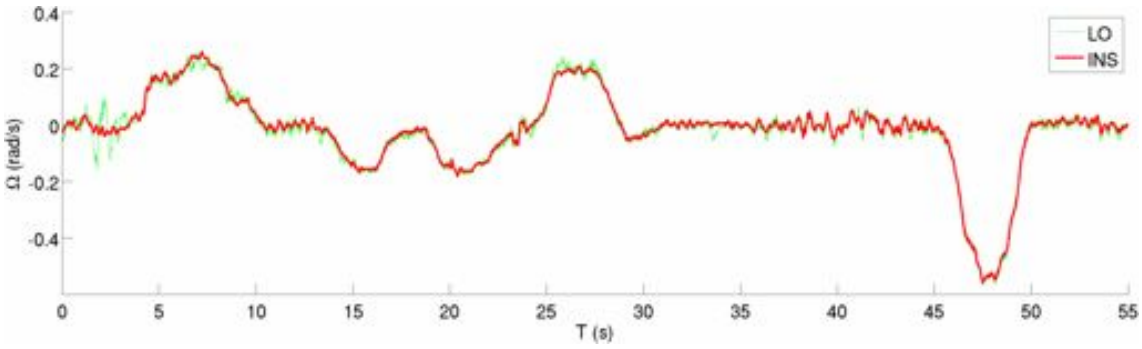



Figure 4.16: **LIDAR Odometry: Rotational velocity estimates.** Rotational velocity estimates over the same 80 seconds as Figure 4.14 using the LO system. The ground-truth (INS) yaw is shown in red.

If we measure the Root Mean Square (RMS) error for each of the two odometry systems (as measured against the INS estimates), we see that on average the LIDAR has a lower RMS value ($0.48m/s$ vs $1.74m/s$ for LO vs. VO in the above example). This motivates us to make use of LIDAR as the source of rotational and linear velocity estimates.

However, due to the numerical differentiation and corresponding noise increase, we require some form of smoothing over the velocities. Figure 4.17 through Figure 4.19 show the effects of different smoothing algorithms over scan-match velocity estimates around the Begbroke site:

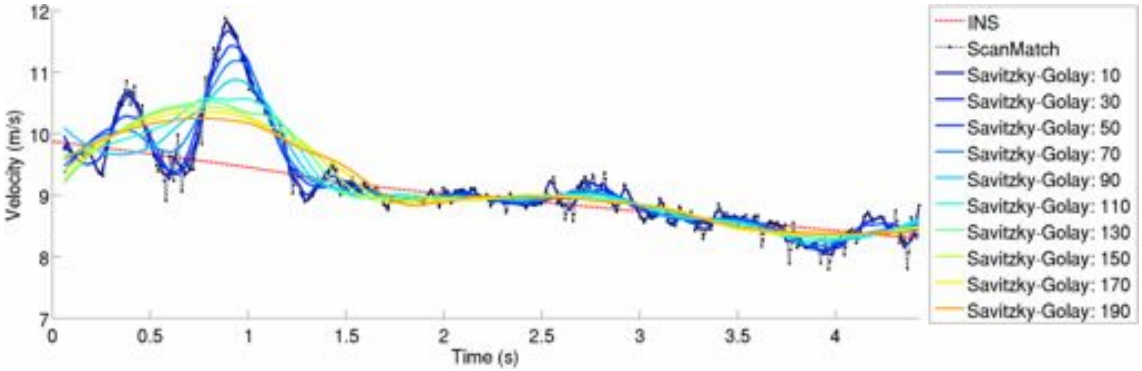



Figure 4.17: **Velocity Filtering: Savitzky-Golay.** Velocity filtering using the Savitzky-Golay algorithm for varying window sizes over scan-match velocity estimates.

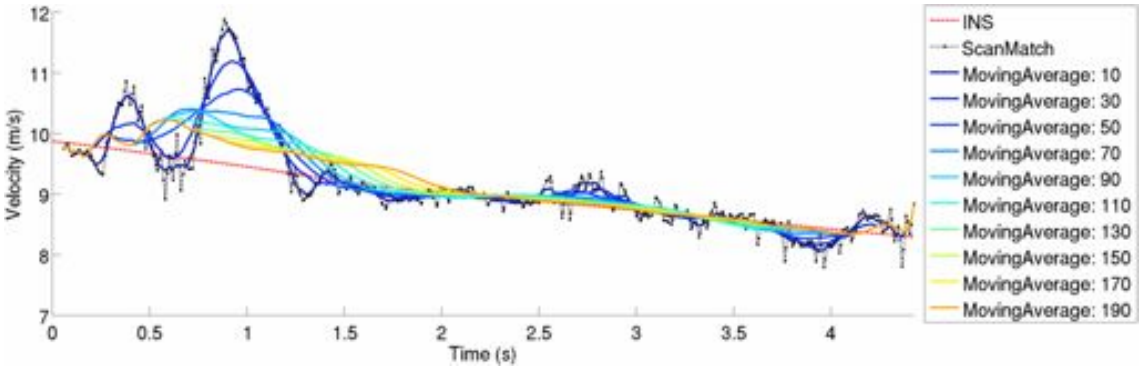



Figure 4.18: **Velocity Filtering: Moving-average.** Velocity filtering using a moving-average filter of varying window sizes over scan-match velocity estimates.

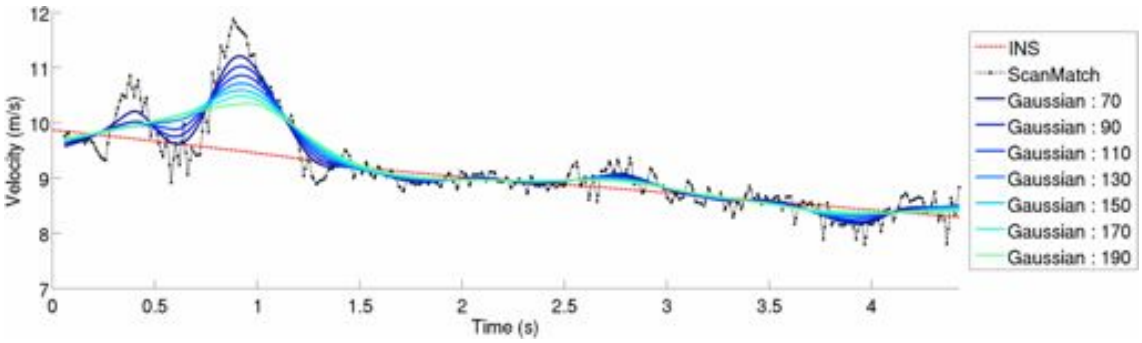



Figure 4.19: **Velocity Filtering: Kernel-smoothing.** Velocity filtering using a Gaussian kernel-based smoothing algorithm of varying kernel-widths over scan-match velocity estimates.

From these comparisons, it can be seen that the best performing algorithm is the moving-average filter - this approach makes minimal assumptions about the underlying signal and works well for damping the velocity oscillations (visible in Figure 4.17 through Figure 4.19).

Using LO to estimate for velocity estimation works particularly well in scenes which are mostly static. However, as can be seen from Figure 4.17 through Figure 4.19, there are certain areas that are sources of systemic error. This effect, and its resolution, is more challenging and is discussed at length in Chapter 6.

Given that we now have a method to estimate the vehicle velocity over a given window period, we now are able to fully generate the retrospective swathe. The focus, given this swathe, is to produce localisation estimates by matching it within the 3D prior. We seek to minimize some objective function:

$$\hat{\mathcal{T}} = \underset{\mathcal{T}}{\operatorname{argmin}}\ f(\mathcal{P}, \mathcal{Z}_{t-N:t}, \mathbf{x}_{t-N:t}, \mathcal{T}) \tag{4.4}$$

which is a function of our prior experience, observed LIDAR data, and the integrated pose chain. We now turn our attention to ways of accomplishing this.

## 4.4 Existing methods for 3D point cloud alignment

The localisation problem is fundamentally one of point cloud registration; if we can accurately register a run-time cloud to a prior, this will give us an accurate estimate of the current vehicle location.

As outlined in Chapter 2, many methods exist for the alignment of point cloud data, which can be broadly separated into those that perform alignment on the entire point clouds and those which identify salient regions or features. We now contrast the performance of two widely-used registration algorithms using our "swathe" data - Generalised ICP and the Normals Distribution Transform.

### Generalised ICP

Generalised ICP was discussed previously in Chapter 2 as a principled generalisation of both ICP and ICS. Generalised ICP (GICP) [80] utilises a probabilistic plane-to-plane matching approach, as opposed to the point-to-point or point-to-plane metrics used by ICP and ICS, respectively. GICP assumes the underlying points $P$ and $Q$ generate observations $\hat{P}_i \sim \mathcal{N}(P_i, C_i^P)$ and $\hat{Q}_i \sim \mathcal{N}(Q_i, C_i^Q)$ , and then optimises:

$$\hat{\mathcal{T}} = \underset{\mathcal{T}}{\operatorname{argmax}} \sum_i \log(p(d_i^T)) \tag{4.5}$$

where $d_i^T$ corresponds to the distribution of the transformed points as a function of $\mathcal{T}$:

$$d_i^T \sim \mathcal{N}(\hat{P}_i - \mathcal{T} \cdot \hat{Q}_i, C_i^P + \mathcal{T} \cdot C_i^{\hat{Q}} \cdot \mathcal{T}^T) \tag{4.6}$$

This is fully detailed in [80]. Figure 4.20 and Figure 4.21 show examples of both

good and failure cases of the GICP algorithm over real-world data acquired by the 2D LIDARs on the Wildcat:

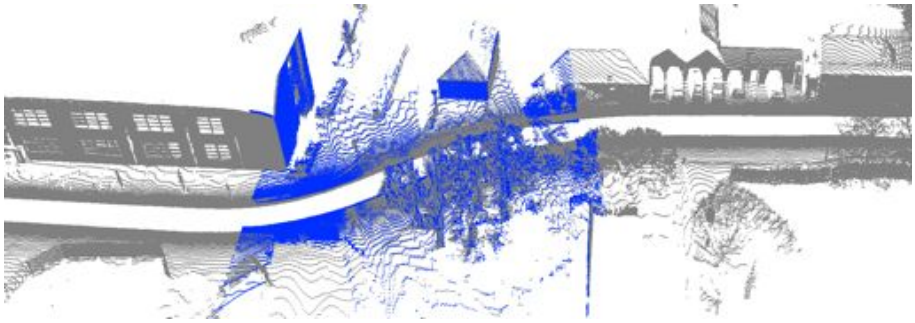


Figure 4.20: **GICP: Good match.** A successful match of a run-time swathe (blue) against the survey (grey), using velocities from the INS. This location in Begbroke (along the western section) is used as a control point as there exists substantial planar-structure visible from the LIDAR, which is highly beneficial for point-based registration algorithms. (Swathe data is in blue, the map in grey).

As expected, GICP works well when we have good estimates for the velocities (obtained from the INS). However, when we substitute velocities inferred from the horizontal LIDAR, the resulting point cloud becomes "warped", leading to misalignments. Figure 4.21 highlights a particular example of this in the town centre of nearby Woodstock:

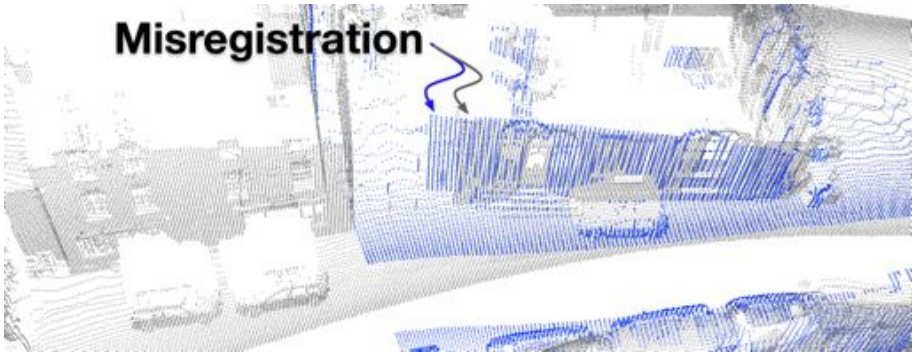



Figure 4.21: **GICP: Matching failure.** A location in the nearby town of Woodstock that causes GICP to fail to find a good match. This swathe data was generated using the velocities inferred from scan-matching through the horizontal LIDAR. Point-to-point matching algorithms (and their variants) perform worse when the velocity estimates exhibit any degree of compression/expansion. A comparison of the trajectories obtained using GICP is shown in Chapter 5.

## The Normals Distribution Transform

Biber in [8] utilises a novel approach to 2D scan registration based on Gaussian mixtures, which was subsequently extended to 3D [86]. The input space is discretized into cells - or *voxels* - with the density of points $P$ falling into each voxel modelled by a Gaussian distribution. The advantage is a differentiable probability density, which is useful for gradient-based optimisation routines. Then, given a point cloud $\mathcal{Q}$, NDT optimises the *score* function:

$$\hat{\mathcal{T}} = \underset{\mathcal{T}}{\operatorname{argmin}} \left\{ -\sum_{i=1}^{|\mathcal{Q}|} \log f(\Gamma(\mathcal{T}, \mathcal{Q}_i)) \right\} \tag{4.7}$$

where $\Gamma$ is a transformation operation that transforms $\mathcal{Q}_i$ by $\mathcal{T}$ and $f(\cdot)$ for a given point is defined as:

$$f(\mathcal{Q}_i) \propto -\exp\left((\mathcal{Q}_i - \mu_k)^T \Sigma_k^{-1} (\mathcal{Q}_i - \mu_k)\right) \tag{4.8}$$

where $\mu_k$ and $\Sigma_k$ denote the mean and covariance of the Gaussian approximation for the corresponding voxel. Similarly to the GICP results, Figure 4.22 and Figure 4.23 show a successful match at the "control" point, as well as a failure in Woodstock centre:

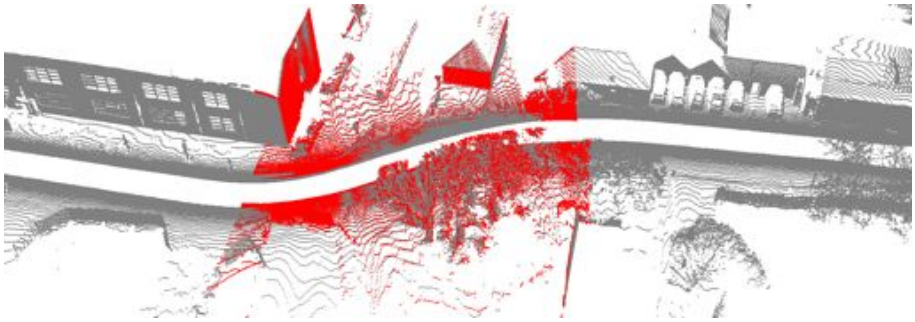


Figure 4.22: **NDT: Good match.** A successful match from a run-time point cloud (coloured red) against the survey (grey).

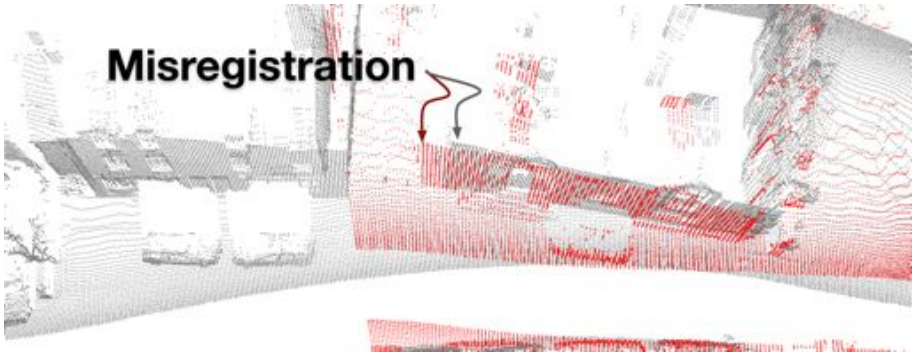



Figure 4.23: **NDT: Matching failure.** A matching failure for a similar point cloud, in the same location as in Figure 4.21.

Notice the similar failure location of Figure 4.23 to Figure 4.21 - this is attributable to the "warping" introduced into the swathe by degenerate velocity readings at this location arising from ground-strike, traffic and so on. In Chapter 5 we illustrate the degenerate effects that these mis-registrations have on the resulting trajectory estimates.

In the following sections, we build up the concept of information-theoretic alignment in the case of 3D swathes and in Chapter 5 show how this formulation results in superior localisation performance.

## 4.5 Point cloud alignment by Maximum Likelihood

The underlying assumption made here is that the true pose of the vehicle will be the transformation that best aligns the swathe and prior point cloud. Given that we *have* a model of the environment - in the form of the prior, $\mathcal{P}$ - we can estimate the $\mathbb{SE}2$ alignment in a maximum-likelihood setting as a function of pose and this prior:

$$\hat{\mathcal{T}} = \underset{\mathcal{T}}{\operatorname{argmax}}\, p(\mathcal{Z} \mid \mathcal{T}, \mathcal{P}) \tag{4.9}$$

where $\mathcal{Z}$ are the observations and $\mathcal{T}$ is the vehicle pose. We define another projection function $\Psi$:

$$\mathcal{Q}' = \Psi(\Phi(\mathcal{Z}_{t-N:t}, \mathbf{x}_{t-N:t}), \mathcal{T}) \tag{4.10}$$

which projects a relative point cloud into the global frame as a function of pose $\mathcal{T}$. Through this projection we can generate a *likelihood-field* of our observed data with respect to a model, which we can use for sequential pose estimation - the question now concerns the choice, and form, of such a model.

If we perform a histogramming operation such that we obtain bin counts $z_i$ corresponding to the counts of the points of $\mathcal{Q}'$ falling into bin $i$ of a 2D grid on the XY plane, then we can formulate the measurement model as a multinomial distribution:

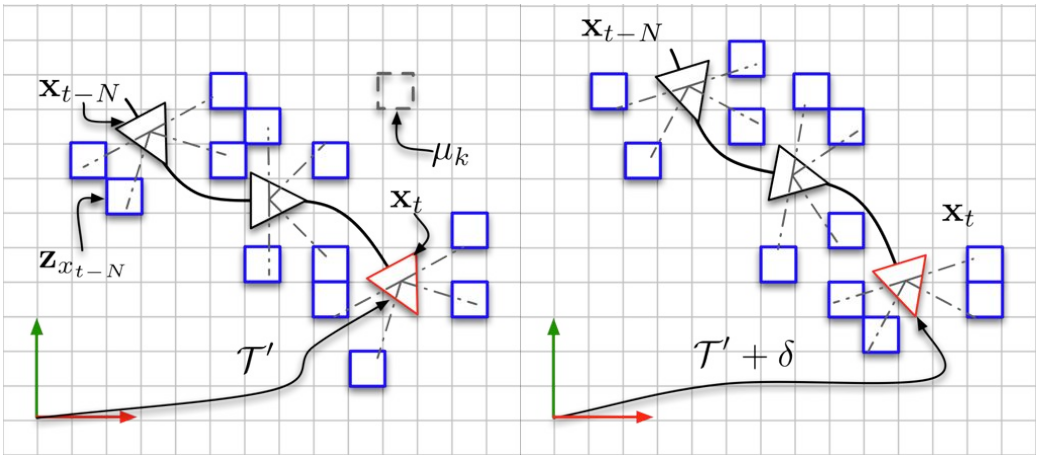
$$Mult(z_1, \ldots, z_k \mid \mu, N) = \frac{N!}{z_1!, \ldots, z_k!} \prod_{k=1}^{K} \mu_k^{z_k} \tag{4.11}$$

which is parameterized by observations of cardinality $N$, with cell probabilities $\mu$ and observed counts $\mathbf{z} = \{z_1, \cdots, z_k\}$, so that $N = \sum_{i=1}^{K} z_i$. The multinomial distribution expresses the likelihood of seeing the bin-counts $\mathbf{z}$ under the model of cell probabilities $\mu$. As noted in [63], the samples observed are not equal to their counts as the sampling appeared in a particular order - however, we do not sum over orderings as the data had one ordering (the ordering observed). Therefore, Section 4.5 is equivalent to:

$$Mult(z_1, \ldots, z_k \mid \mu) = \prod_{k=1}^{K} \mu_k^{z_k} \tag{4.12}$$

Of course, this assumes that the variables are independent and identically distributed ($i.i.d$), despite the spatial dependencies that will exist in the data - this is a common assumption [102] and is used here with no adverse effect.

We perform this histogramming operation by projecting the observed points onto the $XY$ plane. In this fashion, regular prismatic structure manifests as higher bin counts; this makes sense as if we see a high wall during the survey, we have high expectations of seeing it in subsequent runs. Points on the ground have a naturally lower density, but are still informative about the shape and structure of the road. Therefore, representing the point clouds by their projected probability distributions is a natural way of capturing the structure of the environment. An illustration of this process is shown in Figure 4.24:




(a) Observations at $\mathcal{T}'$ (b) Observations at $\mathcal{T}' + \delta$

Figure 4.24: **Observations as a function of pose.** Figure (a) shows the distribution of observations at global pose $\mathcal{T}'$, with rigid pose-chain $\mathbf{x}_{t-N:t}$ obtained from the linear and rotational velocity estimates, while figure (b) shows the distribution of the same observations and pose-chain, at a different global-pose $\mathcal{T}'+\delta$ (Blue-squares represent observations, triangles represent poses).

Of vital concern is how to estimate the cell probabilities of the model, $\mu$. These values will obviously correlate with the data from our prior, $\mathcal{P}$, and we could therefore build an empirical estimate of $\mu$ by counts obtained from this prior - which we will term $\mathbf{z}^m$ (Note that this corresponds to a similar histogramming operation on the prior point cloud).

However, this empirical estimate of $\mathbf{z}^m$ would likely give zero-probability to combinations of run-time data not observed in the map. As such, we require some form of smoothing over $\mu$, which we achieve by adding a diffuse prior to the multinomial in the form of its natural conjugate - the Dirichlet distribution:

$$Dir(\mu_1, \ldots, \mu_k \mid \alpha_1, \ldots, \alpha_k) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} \mu_k^{\alpha_k - 1} \tag{4.13}$$

The Dirichlet distribution is a multivariate distribution over K categories, parameterized by "shape" parameters $\alpha$ - the Dirichlet is known as a "distribution over

distributions" as the $\mu$ variables are constrained to sum to unity, and is therefore a natural prior for the cell probabilities of the multinomial.

The $\alpha$ parameters also have a natural interpretation in terms of pseudo-counts or "virtual observations". An illustration of the effect of increasing these alpha parameters over a tri-variate distribution are shown in Figure 4.25:

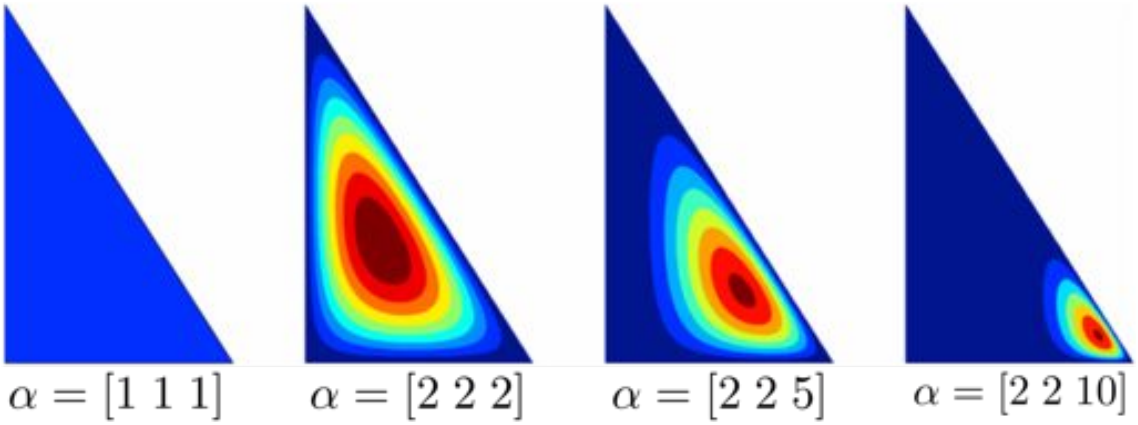



Figure 4.25: **Dirichlet "shape" parameters.** An illustration of the growth of a trivariate Dirichlet distribution with increasing variable counts. On the left, the $\alpha$ parameters are equal and unary - each variable has equal probability. As we observe more "counts" of one variable, the posterior mass begins to shift - with a corresponding shrink in variance - to that variable (Red corresponds to high probability, blue to low probability).

Figure 4.25 shows how increasing the observations of a variable biases the posterior distribution, but crucially still maintains some probability mass on the remaining variables. We require - for the probability parameters of the multinomial distribution - a posterior distribution over the cell probabilities. Applying Bayes rule, we have:

$$p(\mu \mid \mathbf{z}^m, \alpha) \propto p(\mathbf{z}^m \mid \mu)\, p(\mu \mid \alpha) \tag{4.14}$$

where the posterior over parameters $\mu$ is proportional to the likelihood of the data, multiplied by a prior over said parameters, given by the Dirichlet. The conju-

gacy property of the Dirichlet provides a simple update rule:

$$p(\mu \mid \mathbf{z}^m, \alpha) = \prod_{k=1}^{K} \mu_k^{\mathbf{z}_k^m} \prod_{k=1}^{K} \mu_k^{\alpha_k - 1} \tag{4.15}$$
$$\propto \prod_{k=1}^{K} \mu_k^{\alpha_k + \mathbf{z}_k^m - 1}$$
$$p(\mu \mid \mathbf{z}^m, \alpha) = Dir(\mu \mid \alpha + \mathbf{z}^m) \tag{4.16}$$

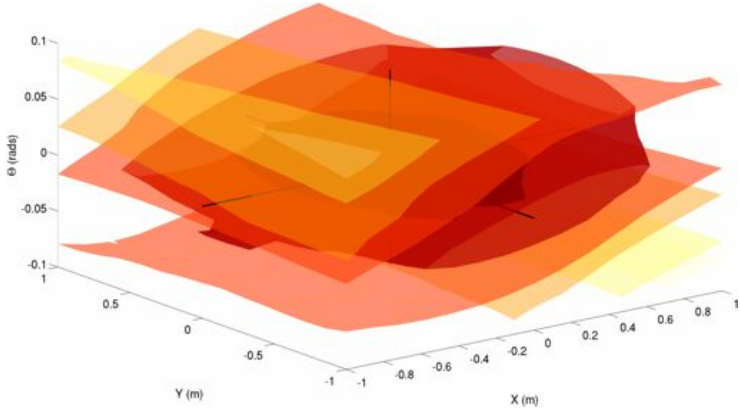
We can see that the resulting distribution is a Dirichlet, smoothed by our initial prior counts. A common approach is to set all $\alpha$ values to be 1 as an uninformative prior - this is shown as the leftmost image in Figure 4.25. This is often referred to as Add-One smoothing [83], or Laplacian smoothing. Jeffreys forwards a different interpretation - the Jeffreys prior is the square root of the determinant of the Fisher Information matrix and for the Dirichlet takes the value of 1/2 [101]. In the following experiments, we make use of the uniform prior.

This model now gives us the likelihood of a certain configuration of bin counts, given the pseudo-counts combined with the data from our prior - these are the parameters of our multinomial model. Our goal now is to estimate the pose that maximizes the likelihood of the binned points of the run-time swathe, given our prior:

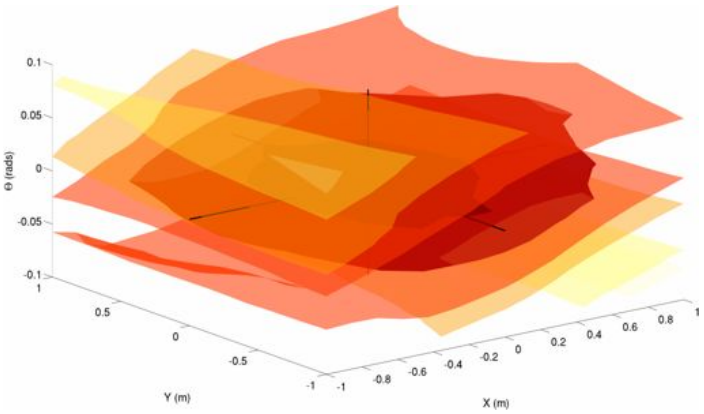
$$\hat{\mathcal{T}} = \underset{\mathcal{T}}{\operatorname{argmax}}\, p(\mathbf{z} \mid \mathcal{T}, \mu) \tag{4.17}$$

To investigate the properties of this likelihood, we can generate the "objective" volume in $\mathbb{SE}2$ space around some known ground-truth value, and then measure the likelihood as a function of induced error. An important parameter in this approach

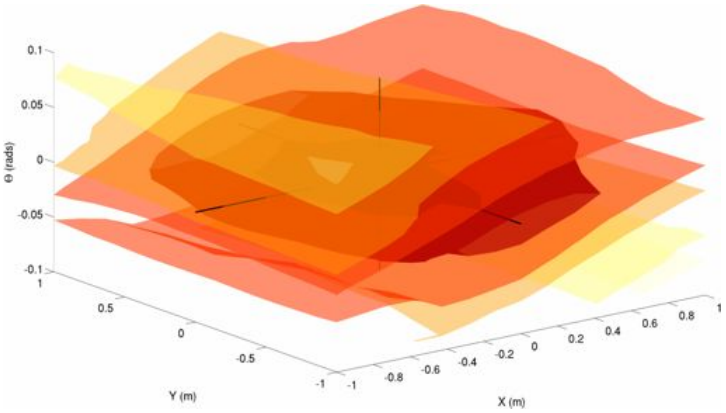
is the histogramming parameter - Figure 4.26 shows the evolution of this likelihood for increasing granularity:




(a) Granularity: low



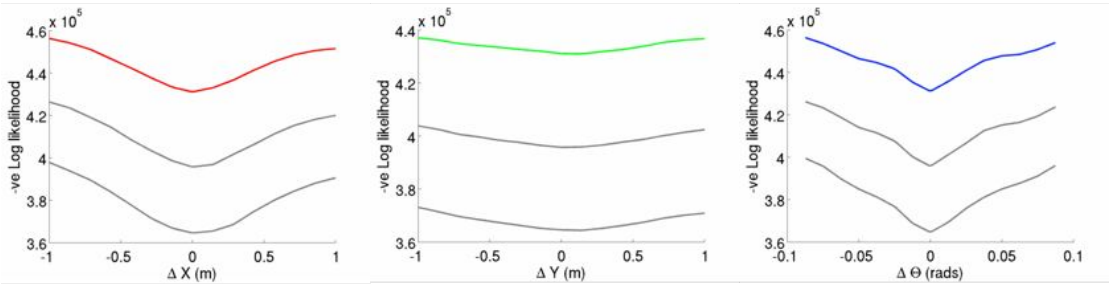

(b) Granularity: medium




(c) Granularity: high

Figure 4.26: **Likelihood sensitivity as a function of histogram granularity.** These figures show the evolution of the objective (i.e. likelihood) surface for increasing histogram granularity, from smallest granularity (a), to largest (c).

**How to interpret volume plots:** Figure 4.26 shows the evolution of the likelihood as a function of increasing histogram granularity - i.e. *decreasing* bin size. These iso-surface plots represent surfaces of equal-likelihood in the objective volume. Darker, redder colours correspond to a *lower* objective; lighter, yellow ones to a higher value. Objective, in this sense is the negative log-likelihood - this is to frame the objective as a minimisation problem, which we work with exclusively in the following sections. The following figures show how the likelihood changes along the cardinal axes:




(a) Likelihood sensitivity: X (b) Likelihood sensitivity: Y (c) Likelihood sensitivity: Θ

Figure 4.27: **Likelihood sensitivity over $\mathbb{SE}2$.** An enumeration of the objective function (likelihood) along the cardinal axes of $\mathbb{SE}2$ space: (a) X-axis, (b) Y-axis and Θ-axis, (c) for various histogram granularities.

**How to interpret axes plots:** Figure 4.27 shows the evolution - along each of the cardinal axes - of the objective function around some known "ground-truth" alignment. The grey plots beneath the coloured plots show the objective function for previous histogram values - in the case of the likelihood, with increasing granularity we see a corresponding *increase* in the *negative* log-likelihood; i.e. a *decrease* in the log-likelihood. This is intuitive - as we add more bins, correspondingly fewer points fall into each bin, and the model expresses a lower likelihood.

Note in particular that the *minimum* is invariant, despite changing granularity. We can also see that the likelihood is sensitive to errors in rotation and lateral deviation, but not particularly to those in the forward direction. This is an artefact of using the out-of-plane LIDAR data to estimate forward motion - however, this

was not a limiting factor in the data evaluated in this thesis, which is demonstrated over more than 100km in Chapter 5.

Also, given the multiplicative nature of the multinomial likelihood, the resulting numeric value becomes rapidly smaller given more observed data[1]. This becomes an issue for example when we observe a large amount of LIDAR data in our swathe - we would like our model to be invariant to the cardinality of the run-time data observed.

An answer to this is to make use of the *average* likelihood - a value that is invariant to the sample size. This proof is given in [81] and is outlined here for continuity. The average log-likelihood is defined as:

$$\bar{L} = \log L(\mathbf{z} \mid \mu)^{|\mathbf{z}|^{-1}} \tag{4.18}$$

where $L$ is shorthand for *likelihood*. For ease of notation, we let $N = |\mathbf{z}|$. Substituting the form of the multinomial (Section 4.5), we obtain:

$$\begin{aligned} \bar{L} &= \frac{1}{N} \log \frac{N!}{\prod_i \mathbf{z}_i!} \prod_i \mu_i^{\mathbf{z}_i} \\ &= \frac{1}{N} \log N! - \frac{1}{N} \sum_i \log \mathbf{z}_i! + \sum_i \frac{\mathbf{z}_i}{N} \log \mu_i \end{aligned} \tag{4.19}$$

It is then possible to make use of Stirling's approximation for large factorials:

$$\log n! \approx n \log n - n$$

By plugging this approximation into Equation (4.19) and collecting terms, we

[1] Although we evaluate the log-likelihood, it is still an issue with increasing bin number.

obtain:

$$\bar{L} = -\sum_i \frac{\mathbf{z}_i}{N} \log \frac{\mathbf{z}_i}{N} + \sum_i \frac{\mathbf{z}_i}{N} \log \mu_i \tag{4.20}$$

As $n \to \infty$, the normalized histogram is interpreted as a probability distribution itself, and substituted into Equation (4.20):

$$\begin{aligned} \beta_i &= \frac{\mathbf{z}_i}{N} \\ \hat{L} &= -\sum_i \beta_i \log \beta_i + \sum_i \beta_i \log \mu_i \end{aligned} \tag{4.21}$$

where Equation (4.21) is known as the Kullback-Leibler divergence [49]. This divergence, also known as the *relative entropy*, is therefore the negative logarithm of the average multinomial log-likelihood:

$$\mathcal{D}_{KL}(P \| Q) = \lim_{N \to \infty} -\frac{1}{N} \log \bar{L}(\mathbf{z} \mid \mu)$$

hence, identical distributions will have zero-divergence or unity average likelihood as we observe infinite data. Motivated by this relationship, in the next section we explore the use of the KL divergence as an objective function.

## 4.6 Point cloud alignment using Relative Entropy

The Kullback-Leibler divergence belongs to the family of *f-divergences*. If we define a function $f(t)$ to be convex for $t > 0$, the divergence of distribution $P$ from distribution $Q$ is:

$$D_f(P||Q) = \int_x Q(x)\ f\left(\frac{P(x)}{Q(x)}\right) dx \tag{4.22}$$

over the domain $x$. Various well-known distance measures are defined for various choices of $f(\cdot)$ [24], and these are listed in Table 4.1. These include non-parametric measures such as the $\chi^2$ divergence, and also information-theoretic distances - such as the KL divergence:

Table 4.1: Common f-divergence functions

| Function | Measure |
|---|---|
| $t \log t$ | Kullback-Liebler |
| $(t-1)^2$ | $\chi^2$ |
| $2(1-\sqrt{t})$ | Hellinger |
| $\mid t - \ 1 \mid$ | Variational |

We now focus on the use of the Kullback-Liebler divergence or *relative entropy* of the two distributions (as a function of pose) as an objective function to be minimized. The KL-divergence for any two distributions $\mathbf{P}$ and $\mathbf{Q}$ is:

$$\mathcal{D}_{KL}(\mathbf{P}\|\mathbf{Q}) = \int_{-\infty}^{\infty} \mathbf{P}(x)\ \ln\frac{\mathbf{P}(x)}{\mathbf{Q}(x)} dx$$

This integral becomes the following summation:

$$\mathcal{D}_{KL}(\mathbf{P}\|\mathbf{Q}) = \sum_i \mathbf{P}(x_i) \ln\ \frac{\mathbf{P}(x_i)}{\mathbf{Q}(x_i)}$$

for the discrete case. The KL divergence is an effective representation of the information-distance between these distributions, provided that they are well defined - and this is a detail that we now must consider. Consider Figure 4.28, which shows

estimates of two (very similar) probability distributions obtained by counts:

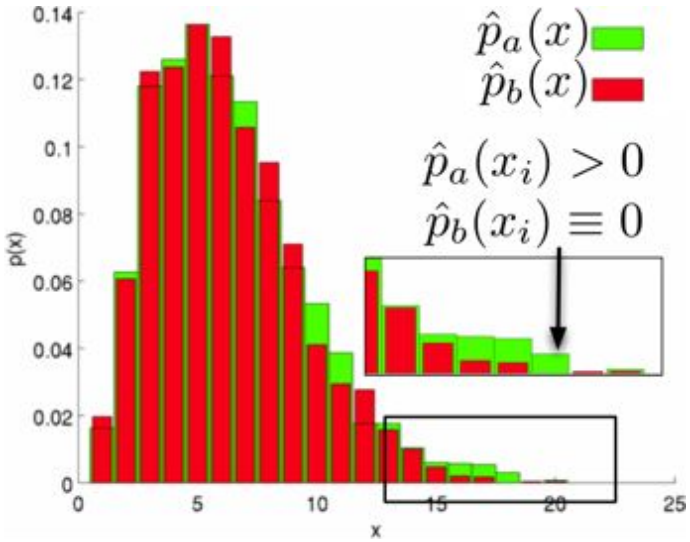



Figure 4.28: **Estimating the KL-divergence from histograms.** An issue arising from estimating the KL-divergence for probability distributions obtained from counts.

In the indicated bin, we have a non-zero number of counts under distribution $\hat{p}_a$, but zero counts under $\hat{p}_b$. Technically, the divergence between these two distributions is infinite - which is clearly only a symptom of limited data, rather than the true probabilistic distance.

We could perform a similar technique as in Section 4.5 by adding a non-informative (Dirichlet) prior to these distributions - however, the Dirichlet ignores *correlations* in the dataset - we know that a large bin count in bin $k$ should admit the possibility of larger counts in the neighbourhood $\mathbf{k} \in \mathrm{Ne}(k)$.

Driven by our experience in optimizing this objective function, we apply a discrete Gaussian convolution to the raw bin counts observed for both the prior $\mathcal{P}$ and swathe $\mathcal{Q}$. If we denote the counts observed in bin $(ij)$ for $\mathcal{P}$ as $\mathcal{P}_{ij}$, then the convolution is:

$$(\mathcal{P}' \star f)(i,j) = \sum_{(a,b)\in A} f(a,b)\ P(i+a, j+b) \quad \{\forall (a,b) \mid f(a,b) \neq 0\} \tag{4.23}$$

where $\star$ denotes the convolution operator of the observed bin counts with a function $f(\cdot)$ - in this case a discrete Gaussian kernel, $\mathcal{N}(0,\sigma)$. This operation contributes significantly to the smooth nature of the objective function and captures the *correlations* of the data.

As a final step, to prevent infinite divergences, we apply absolute discounting to the probability distributions as follows. For any two probability distributions $\mathbf{P}$ and $\mathbf{Q}$ obtained by counts, with the sets of non-zero bins defined as $S_{\mathbf{P}}$ and $S_{\mathbf{Q}}$ respectively, we define the smoothed probability distribution $\mathbf{Q}$ to be:

$$\mathbf{Q}(i) = \begin{cases} \mathbf{Q}(i) - \epsilon & \text{if i} \in S_{\mathbf{Q}} \setminus S_{\mathbf{P}} \quad (4.24) \\ \epsilon & \text{f } i \in S_{\mathbf{P}} \setminus S_{\mathbf{Q}} \quad (4.25) \end{cases}$$

Absolute discounting reduces the probability mass in distribution $\mathbf{Q}$ in all the non-zero bins that do not intersect with $\mathbf{P}$, and this mass is reapportioned into bins that have mass under $\mathbf{P}$ but not $\mathbf{Q}$ (which is a set difference, denoted by the $\setminus$ operator). This is necessary in order for the divergence measure to be properly defined. Figure 4.29 shows a representative objective function, using the same data as shown in Figure 4.26:

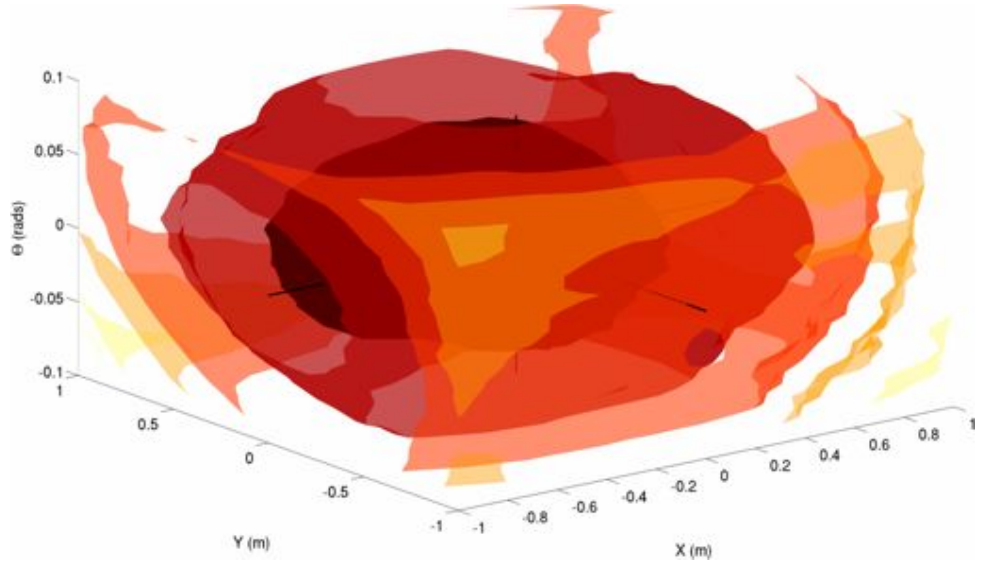
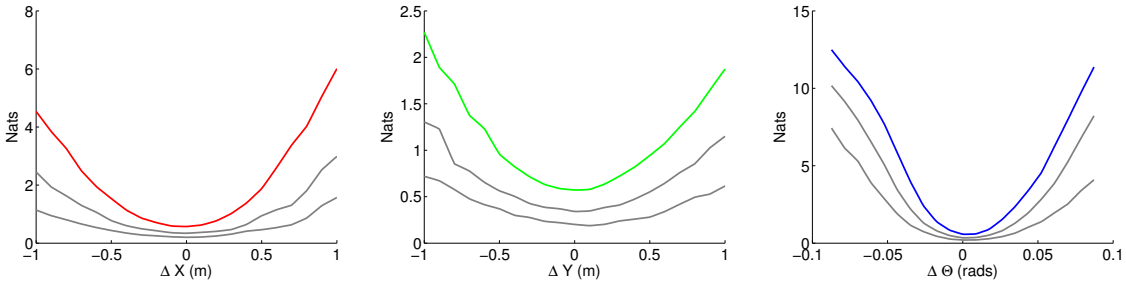



Figure 4.29: **Relative Entropy objective "volume".** Representative iso-surfaces of the relative entropy, exhaustively evaluated over the input domain using the same data as shown in Figure 4.26.

Similarly, Figure 4.30 shows the enumeration along the cardinal axes of $\mathbb{SE}2$ space:




(a) Sensitivity: X (b) Sensitivity: Y (c) Sensitivity: $\Theta$

Figure 4.30: **Relative Entropy sensitivity over $\mathbb{SE}2$.** An enumeration of the objective function along the cardinal axes of $\mathbb{SE}2$ space: (a) X-axis, (b) Y-axis and $\Theta$-axis, (c).

After the smoothing process, we see that the KL divergence is now *more* sensitive to changes along the cardinal axes. This is necessary for the estimation process to prevent accrued pose error (and ultimately tracking failure). The relative lack of sensitivity along the direction of travel (as compared to the transverse direction) is

both a function of the environment and the motion of the vehicle - similar concerns are discussed in [91]. However, we have not found this to be a limiting concern in the data - over a year and 100km collected in widely varying environments - that has been evaluated.

### 4.6.1 Objective function optimisation

There are many different optimisation approaches for any given objective function, both gradient-based and derivative free. We propose Algorithm 3, which exhibits good performance across all the data acquired:

**Algorithm 3** Objective function optimisation

**procedure** EstimateTransformation($\mathcal{P}, \mathcal{Q}, \hat{\mathcal{T}}$, TOL)
  $\mathrm{g} \leftarrow \mathrm{g}_{init}$ ▷ Initialise histogram granularity
  $\delta \leftarrow \infty$ ▷ Initialise objective function change
  $\mathbf{P} \leftarrow \mathbb{H}(\mathcal{P}, \mathrm{g})$ ▷ Histogram
  $\mathbf{P}' \leftarrow \mathbf{P} \star \mathcal{N}(0, \sigma)$ ▷ Apply convolution
  **while** $\delta >$ TOL **do**
    $\mathcal{T} \leftarrow \hat{\mathcal{T}}$
    $\hat{\mathcal{T}}_{(x,y)} \leftarrow \underset{(x,y)\in XY}{\operatorname{argmin}} \mathbf{F}(\Pi(x, y, \hat{\mathcal{T}}_\theta), \mathcal{Q}, g, \mathbf{P}')$ ▷ Grid search
    $\hat{\mathcal{T}}_\theta \leftarrow \underset{\theta}{\operatorname{argmin}} \mathbf{F}(\Pi(\hat{\mathcal{T}}_{(x,y)}, \theta), \mathcal{Q}, g, \mathbf{P}')$ ▷ Line-search
    $\delta \leftarrow \Delta(\hat{\mathcal{T}} - \mathcal{T})$
    $\mathrm{g} \leftarrow \mathrm{g} + g_{delta}$
  **end while**
  $return(\hat{\mathcal{T}})$
**end procedure**

**procedure** $\mathbf{F}(\mathcal{T}, \mathcal{Q}, g, \mathbf{P})$
  return $\mathrm{D}_{kl}(\mathbb{H}(\Psi(\mathcal{Q}, \mathcal{T}), \mathrm{g}), \mathbf{P})$ ▷ Objective function
**end procedure**

The algorithm takes, as input, the survey experience point cloud data, a candidate swathe, an initial estimate of the desired transformation $\hat{\mathcal{T}}$ (an $\mathbb{SE}2$ pose), and

the desired exit tolerance.

In Line 2, the histogram granularity, g, is initialized to a default value, and the objective function change value set to infinity in Line 3.

In Line 4 we define the operator $\mathbb{H}(.)$ to represent the histogramming operation that produces a discrete probability density function (pdf) of the input point cloud data, with the number of bins determined by the second parameter.

Algorithm 3 works in two stages - discrete grid search over $XY$, followed by a line-search for minima using Brent's method [17] over $\Theta$. Line 16 defines the objective function used in the optimisation procedure, which takes as input a pose, the swathe, histogram granularity, and the prior distribution and returns the KL-divergence between them. Again, $\Psi$ is a projection operator, transforming an input point cloud $\mathcal{Q}$ by a transformation $\mathcal{T}$. The $\Pi$ operator (Line 8 and Line 9) takes a $\{x, y, \theta\}$ tuple and returns a transform, $\mathcal{T}$.

The granularity is increased by a quantity $\mathrm{g}_{delta}$ at every iteration (Line 11), to provide the annealing effect visible in Figure 4.30. Simultaneously, the domains for both the grid and line searches are contracted, providing increasingly finer estimates.

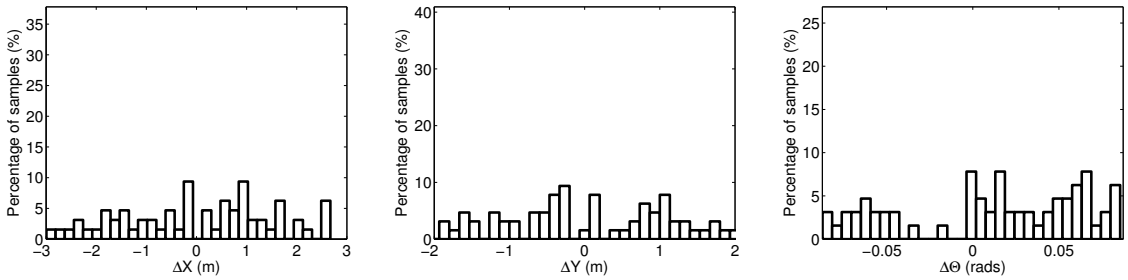
The halting measure, $\delta$, is the difference between the previous $\mathbb{SE}2$ estimate and the current estimate, and the optimisation halts once this measure has reached a predefined value. The difference between the two $\mathbb{SE}2$ poses is measured as given by the metric in [53], in which the orientation in a $\mathbb{SE}2$ pose is expressed with a complex number representation, giving:

$$\mathcal{T} \rightarrow (x_t, y_t, a, b) \in \mathbb{R}^4 \tag{4.26}$$

where $a$ and $b$ are the complex components of the angle - the euclidean metric is now valid for comparing two poses in $\mathbb{SE}2$. At the next discrete interval, we will

have observed more rotational and linear velocity data, and require a pose seed for Algorithm 3 to initiate the search procedure.

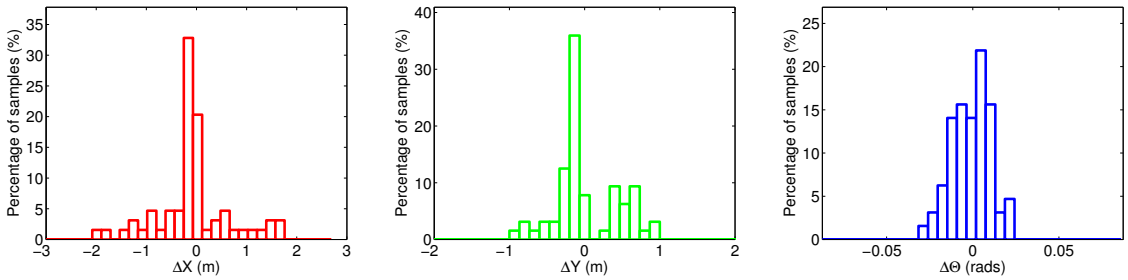
To evaluate the convergence of this algorithm, Figure 4.31 shows the distributions over pose-estimates around some known ground-truth registration that were provided to the algorithm:




(a) $\Delta X$ (b) $\Delta Y$ (c) $\Delta\Theta$

Figure 4.31: **Algorithm convergence test.** The input distributions over pose for $X$ (a), $Y$ (b), and $\Theta$(c) provided to the algorithm.

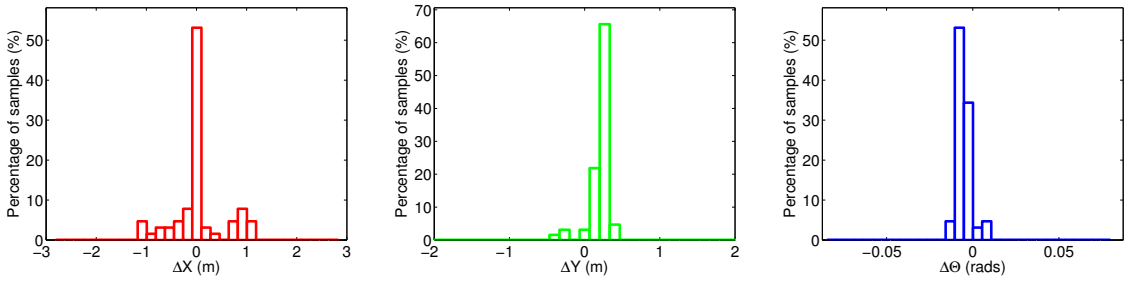
Given these initial estimates, we would hope to see the final residual errors cluster close to 0. Given that Algorithm 3 is iterative, we can plot the residuals as a function of iteration number - Figure 4.32 shows this distribution for 1 iteration:




(a) $\Delta X$ (b) $\Delta Y$ (c) $\Delta\Theta$

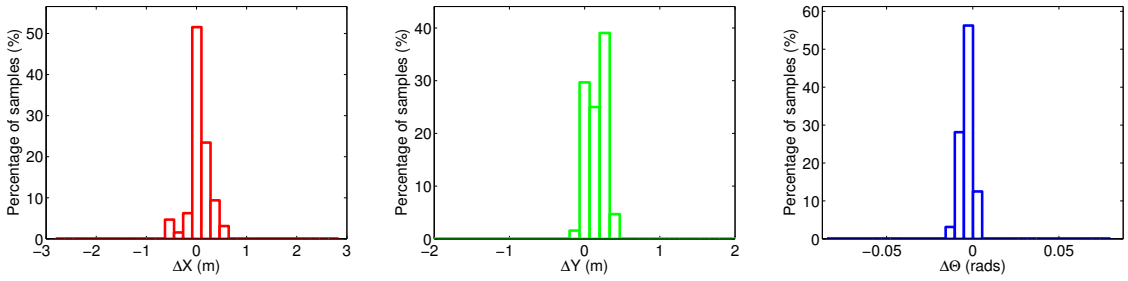
Figure 4.32: **Relative Entropy optimisation: 1 iteration.** Residual error after running Algorithm 3 for 1 iteration: (a) $X$, (b) $Y$, (c) $\Theta$.

Figure 4.33 through Figure 4.35 show the evolution of the residuals for increasing iteration number:



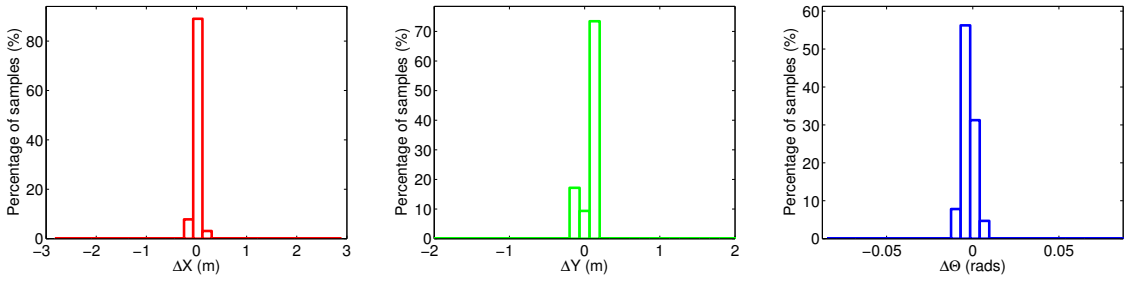

(a) $\Delta X$ (b) $\Delta Y$ (c) $\Delta\Theta$

Figure 4.33: **Relative Entropy optimisation: 2 iterations.** Residual errors after 2 iterations of Algorithm 3 over: (a) $X$, (b) $Y$ and (c) $\Theta$.




(a) $\Delta X$ (b) $\Delta Y$ (c) $\Delta\Theta$

Figure 4.34: **Relative Entropy optimisation: 3 iterations.** Residual errors after 3 iterations of Algorithm 3 over: (a) $X$, (b) $Y$ and (c) $\Theta$.




(a) $\Delta X$ (b) $\Delta Y$ (c) $\Delta\Theta$

Figure 4.35: **Relative Entropy optimisation: 4 iterations.** The final residual errors, after 4 iterations of Algorithm 3 over: (a) $X$, (b) $Y$ and (c) $\Theta$.

As can be seen from the residual plots, this approach performs well for this optimisation task. This algorithm, using Relative Entropy as an objective function,

has been tested extensively with real-world data, which is discussed in Chapter 5, however there exist cases where the performance is degenerate - particularly in areas devoid of structure. We now turn to means of addressing this deficiency.

## 4.7 Leveraging remission structure

Figure 4.36: **Featureless road section.** A relatively featureless section near the Begbroke Science Park. The lack of informative structure in this region requires us to utilise the reflective properties of the LIDAR data.

An important requirement of this approach (as with all 3D point cloud registration methods) is the presence of distinct structure - it is unlikely, in any given urban scenario, to have a completely featureless expanse. However, as is shown in Figure 4.36, there are sections (highway, mostly) that are largely featureless.

We therefore seek to leverage alternate sources of information. Given that the LIDAR sensor can report reflectance, we can match run-time swathes and survey patches by maximizing the *mutual information* (MI) between the distributions over reflectance.

Using MI as an alignment criterion, proposed initially by [98], has a long history

in multi-modal medical image registration [22], and image-registration in general [72][98]. The idea is to estimate the joint-probability of remission occurrences for two images $A$ and $B$ - that is induced by transformation $\mathcal{T}$ - and therefore estimate the MI by:

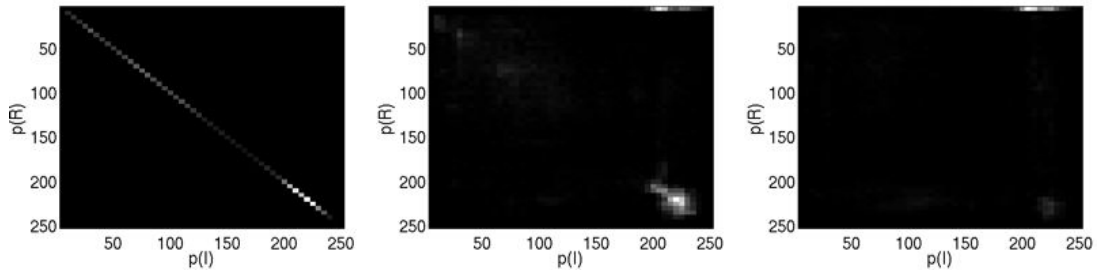
$$\mathcal{I}(A, B) = \mathcal{H}(A) + \mathcal{H}(B) - \mathcal{H}(A, B) \tag{4.27}$$

where $\mathcal{H}(\cdot)$ denotes the Shannon entropy. Consider the following example, showing an image of the Wildcat rotated about the centre axis of the image:

(a) Wildcat : reference image (b) Wildcat : 10° (c) Wildcat : 40°

Figure 4.37: **Using Mutual Information for alignment.** Shown in (a) is a *reference* image, corrupted by varying degrees of rotation - (b )10° and (c) 40°.

We would like, given the original Wildcat image $\mathcal{I}$ and a rotated version $\mathcal{R}$, to estimate the rotation that best re-aligns the two. For the image rotations shown in Figure 4.37, the corresponding joint probabilities over intensity are plotted:




(a) $p(\mathcal{I}, \mathcal{R}_0)$ (b) $p(\mathcal{I}, \mathcal{R}_{10})$ (c) $p(\mathcal{I}, \mathcal{R}_{40})$

Figure 4.38: **Estimating the joint distribution over intensity.** The joint distribution over intensity, $p(\mathcal{I}, \mathcal{R}_\alpha)$, for the images shown in Figure 4.37. Note in (a) the strong diagonal, indicating exact alignment (this is expected, they are the same images). For increasing rotation, the distribution becomes more diffuse, with a correspondingly higher entropy, resulting in a *lower* mutual information.

The mutual information for varying degrees of rotation ($\alpha$) is shown in Figure 4.39:

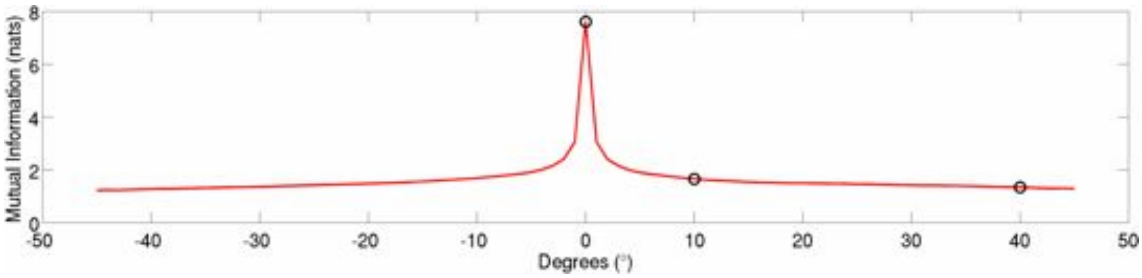



Figure 4.39: **Mutual Information: Registration example.** The MI for varying degrees of rotation ($\alpha$). Note the sharp peak at 0°, which is to be expected - these images are maximally informative about each other at this angle. As the rotation is increased, the probability mass of the joint distribution becomes more diffuse, meaning the two distributions are progressively less informative. Highlighted are the rotation values for the images shown.

Figure 4.40 shows an example reflectivity map for one of the locations in Figure 4.36:

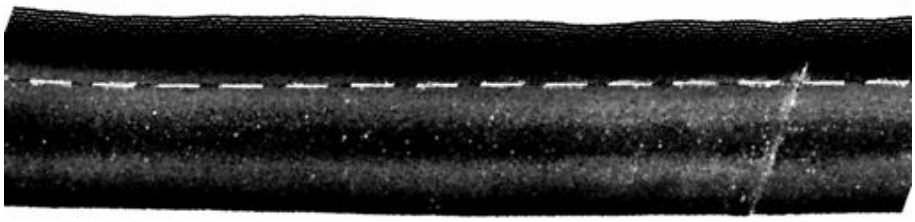


Figure 4.40: **Road reflectance from the declined 2D LIDAR.** Road reflectance from the 2D LIDAR over the area shown in Figure 4.36. Clearly visible are the lane dividers.

We seek to perform a similar registration of such a run-time reflectivity swathe with our prior. Of course, the registration problem for the swathe is slightly different from that of the image-registration case. For images, there is a discrete grid specified a-priori - the image pixel locations. Also, when aligning images, there is only one sample per grid-point, given by the intensity of the pixel at that location. For our registration problem, depending on the histogramming parameters chosen, we may have many points - and their associated reflectances - to evaluate at each location. As a simplifying assumption, for all points in a bin, we take the resulting reflectivity to be the average of these points.

Consider Figure 4.41, which again shows the objective function, but using a MI metric over joint reflectance data of the swathe and point clouds:

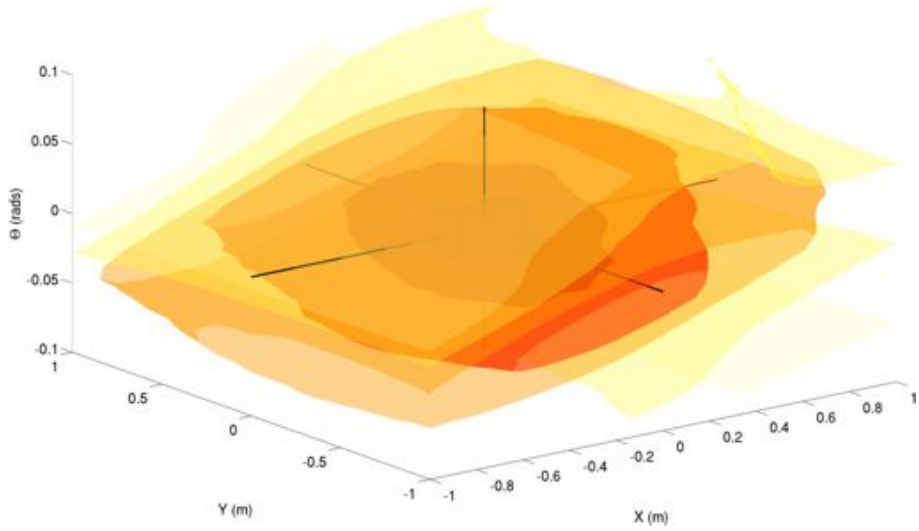



Figure 4.41: **Localisation through reflectance matching.** The objective function over $\mathbb{SE}2$, using the mutual-information over the joint reflectances of the swathe and map.

Note the colour variation of Figure 4.41 as compared to Figure 4.29 - the lower cost is much more "concentrated" in this objective function.

In the image-registration literature, optimisation of such an objective function is often done with derivative-free optimisation functions, such as the Nelder-Mead Simplex algorithm, or a *direction-set* algorithm, such as Powell's method [36].

Nelder-Mead [70] evaluates the objective function over a simplex of points - in the case of the 2D example shown in Figure 4.42, this is a triangle - and evaluates the descent direction based on the gradient information obtained by these evaluations:

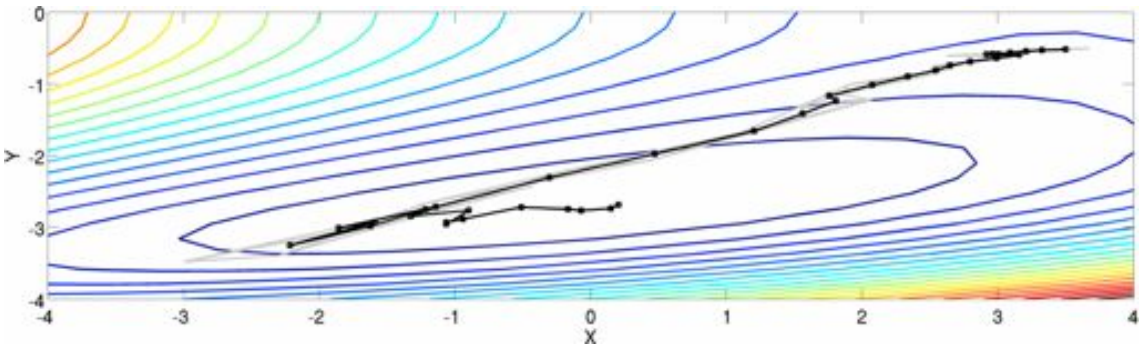



Figure 4.42: **Optimisation illustration: Simplex algorithm.** The Nelder-Mead simplex method over an example objective function (Himmelblau's function). Simplex evaluations - and their associated centroid - are shown in grey and black, respectively.

Direction-set methods use a set of search vectors, and minimise the objective function by means of a bi-directional search along the direction of these vectors. These vectors are updated at each step depending on their contribution to the overall decrease of the objective function. This is a generalisation of "taxi-cab" search, or coordinate descent, another direction-set method:

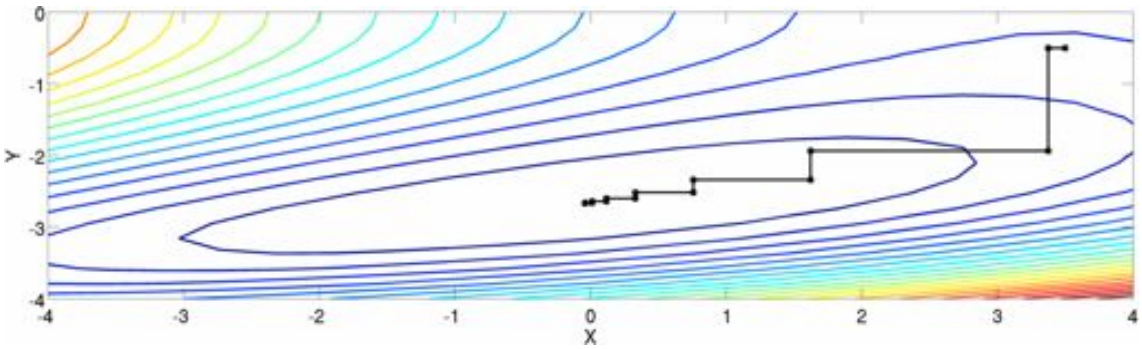
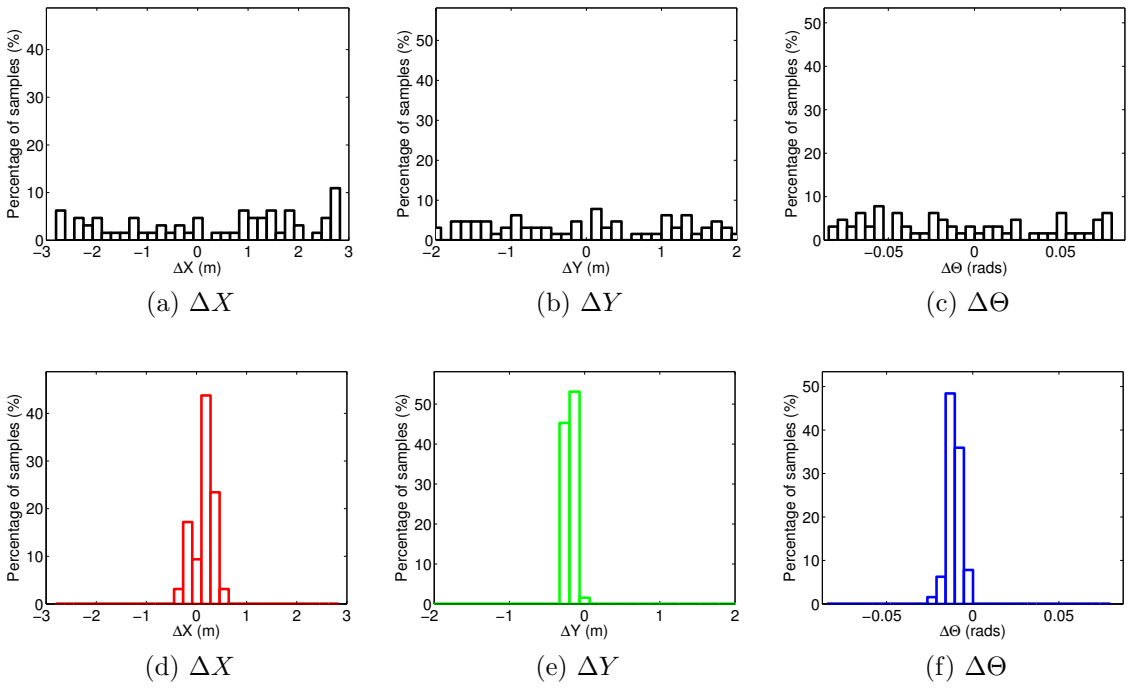



Figure 4.43: **Optimisation illustration: Co-ordinate descent.** Optimisation (function minimisation) using repeated application of coordinate descent, with minima found using Brent's method.

For coordinate descent, no modification is made to the search directions, and the objective function is minimised along each in turn until some halting criteria is met. Although simple, coordinate-descent - with minimisation along each search coordinate done using Brent's method - works well for optimising the mutual-information

objective function. Figure 4.44 shows the resulting distribution of the residual error after termination of the optimisation:




(a) $\Delta X$ (b) $\Delta Y$ (c) $\Delta\Theta$

(d) $\Delta X$ (e) $\Delta Y$ (f) $\Delta\Theta$

Figure 4.44: **Mutual Information optimisation: Residual error.** The residual error distribution at termination of the optimisation over (a) $X$, (b) $Y$, and (c) $\Theta$.

Again, the residual plots give a good indication that this algorithm - in conjunction with the MI objective function - will lead to robust localisation estimates - results demonstrating the validity of this localisation strategy are given in Chapter 5. Of course, there are parallels between the optimisation based on perceived reflectance, and that based on projected point density. In fact, we can consider the projected data to be an intensity plot of point density, and then optimise:

$$\mathcal{I}(\mathbf{z}, \mathbf{z}^m) = \mathcal{H}(\mathbf{z}) + \mathcal{H}(\mathbf{z}^m) - \mathcal{H}(\mathbf{z}, \mathbf{z}^m) \quad \text{where} \tag{4.28}$$

$$\mathbf{z} = \mathbb{H}(\Psi(\mathcal{Q}, \mathcal{T}))$$

$$\mathbf{z}^m = \mathbb{H}(\mathcal{P})$$

The advantage here is that the MI is not as sensitive to the choice of smoothing (as compared to the KL divergence), and we can easily incorporate this as a new objective function into Algorithm 3.

A crucial aspect to the entire system, and to the localisation approaches outlined, is an accurate time mapping between individual sensors - this is something we must address.

## 4.8 The importance of timing

Of vital importance is the timing calibration between the time as perceived by the sensor clocks and those of the on-board computer. Disagreement between these clocks will result in point clouds that exhibit "smear". Thinking of the information content of $\mathbf{Q}$, this smearing or blurring will flatten the objective function making optimisation harder.

We employ the TICSync [39] timing algorithm which learns a probabilistic mapping between clocks, and is able to recover both skew and offset. Shown in Figure 4.45 are the TICSync-corrected data (left), and the same data with a normally distributed 50ms error in timing (right). Visible in the right figure is the ghosting in laser-space that increases with increased timing error:

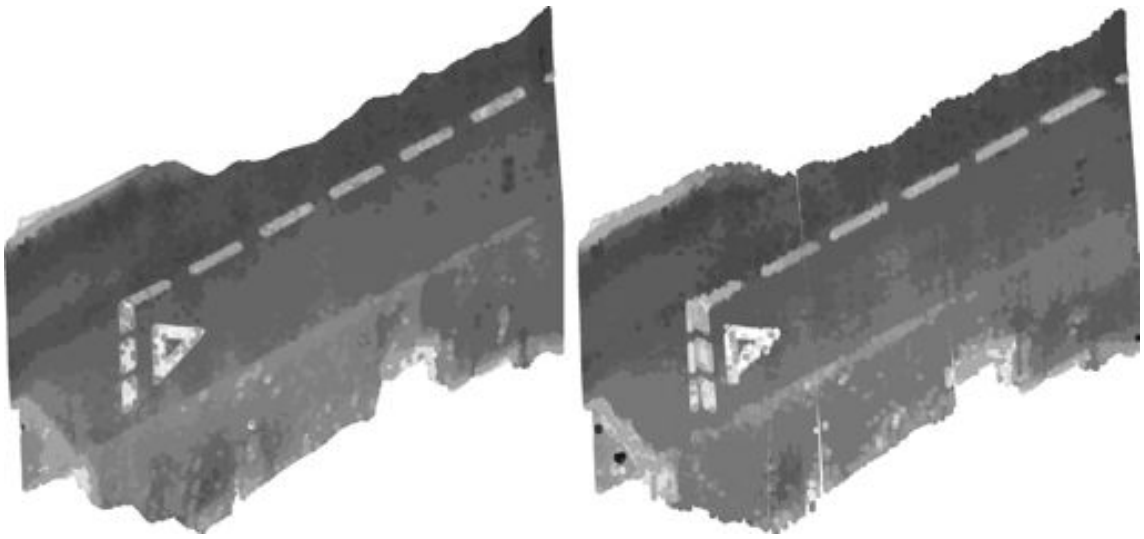


Figure 4.45: **Timing illustration.** Overhead view of a set of road markings around the Begbroke site. The point cloud generated with TICSync-corrected timings is shown on the left, as compared against a similar point cloud with normally-distributed 50ms error. The corrected point cloud does not exhibit the ghosting (visible in the right image).

Use of TICSync is important in any multi-sensor payload, and we use it extensively through this thesis.

## 4.9 Summary

This section has presented and analysed methods for point cloud alignment from 2D-LIDAR for localising a road vehicle within a prior map. Various information-theoretic approaches for estimating pose were formulated using both the geometric and intensity appearance of a scene, and the following chapter presents results using these techniques over extensive real-world data, spanning the course of a year and over 100km of driven distance.

# Chapter 5

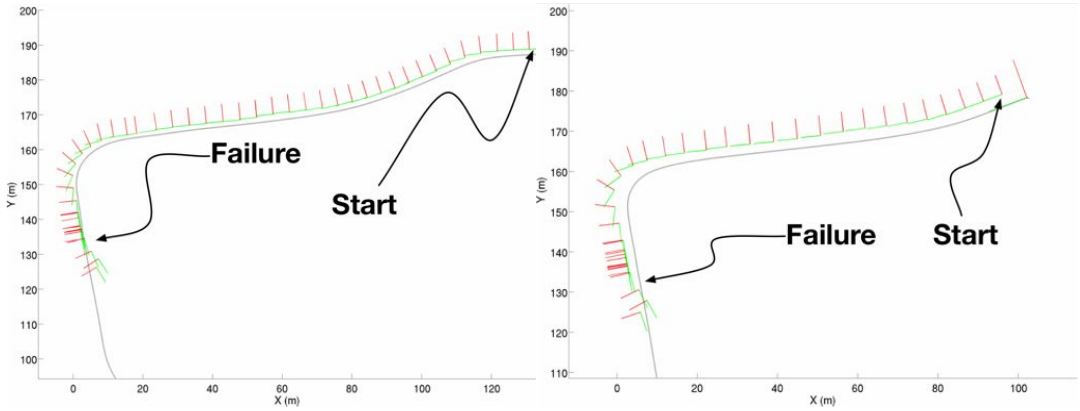
# Validation

## 5.1 Introduction

In this chapter we establish a method for comparing the performance of our map-based localisation procedure - which we call Lifelong Localisation with LIDAR ( or $L^3$) for short - with the estimates from the INS. Establishing that the performance of this approach is comparable to the state-of-the-art INS will validate the concept of inexpensive, LIDAR-based, stand-alone localisation systems.

## 5.2 Performance comparison

As a first step towards this validation, recall Figure 4.3 through Figure 4.6 from Chapter 4 that showed the failure cases using scan-matching over the long-term - and were a motivating factor for the algorithms developed in this thesis. In these figures, we showed that using a simple 2D map is insufficient for robust long-term localisation performance. These localisation failures are repeated here in Figure 5.1:




(a) Localisation performance: +63 days (b) Localisation performance: +64 days

Figure 5.1: **ICS-localisation failures.** The failures using a scan-matching approach with a static map from two months prior. These are the same figures as shown in Chapter 4, repeated here for continuity.

Now consider Figure 5.2, which shows the trajectory estimates *using the same data* with $L^3$:

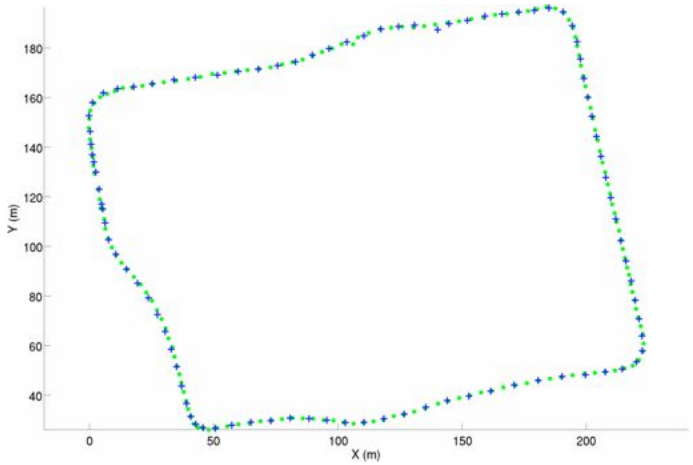



Figure 5.2: $L^3$ **vs. ICS over 1.5km (and 2 months).** The same run-time data - as used in Figure 5.1 - with the same map, captured two months prior. This figure shows that the localisation performance of $L^3$ - which is explicitly leveraging the *persistent* 3D structure of the environment - is superior to that of scan-matching. (Note: Trajectory estimates have been down-sampled and plotted as points for easier visible differentiation.)

As can be seen from Figure 5.2, thanks to the persistence of the 3D structure in the environment, $L^3$ does not suffer from the same brittleness as the conventional ICS

localisation approach does. Note that although these are only two examples, they are representative of a broader catalogue of failures and serve well as an illustration.

As further validation, Figure 5.3 contrasts the trajectories obtained from the INS against the estimated trajectories using $L^3$ for the 26km dataset collection (originally shown in Chapter 4, Figure 4.2)

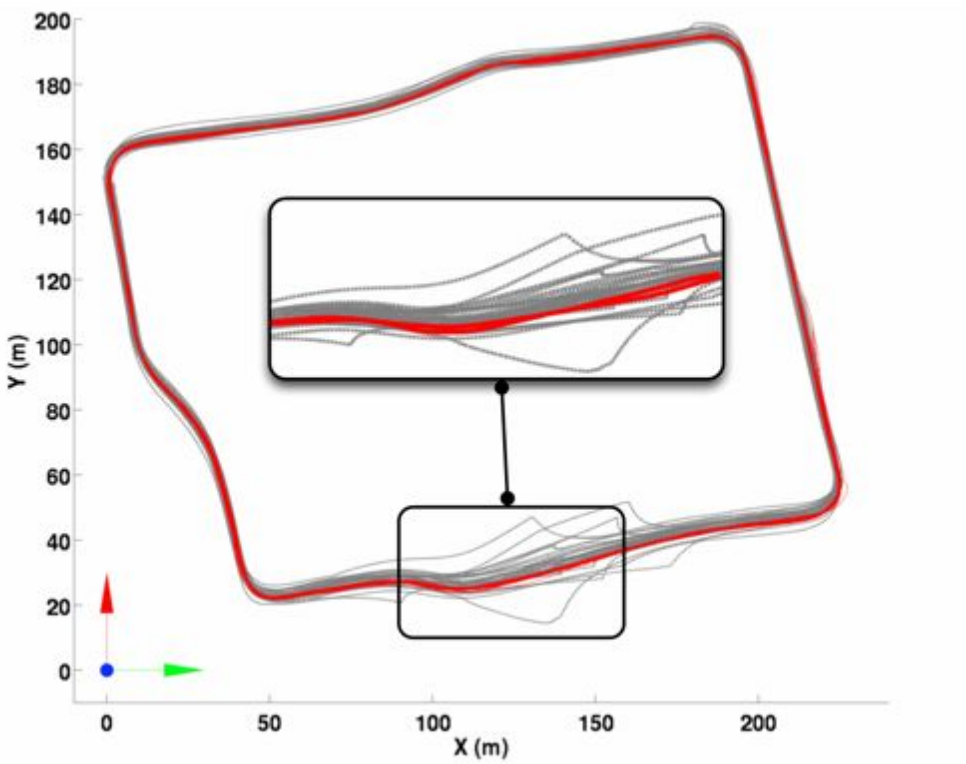



Figure 5.3: $L^3$ **vs. INS over 26km (and 3 months).** A comparison of the INS trajectories (grey, dotted) vs. the $L^3$ trajectories (red, solid). As is very apparent from the image, the INS data drifts substantially in areas of poor reception (predominantly the northern and southern sections), while $L^3$ does not exhibit the same degenerate performance.

Table 5.1: $L^3$ performance

| **Time To Map** (TTM) (days) | **Distance** (km) | **Cumulative Distance** (km) |
|---|---|---|
| 91 | 26 | 26 |

Table 5.1 highlights the important statistics of the performance over this dataset.

The **Time To Map** is the time that elapsed between map acquisition and the run-time data. As the datasets in Figure 5.3 were recorded over a 3 month period, the maximum time difference between the canonical map, and the final traversal is 3 months. We will also keep a tally on the total cumulative distance that the algorithm has successfully traversed - this is the cumulative distance column. However, aside from the striking visual contrast of Figure 5.3, we would like some quantifiable way of comparing the trajectory performance.

Given that we are localizing relative to a given map - and we know the trajectory that it was generated *from* - one method to compare the different estimators would be to measure the *relative* displacement to this **canonical** trajectory for both the INS and $L^3$.

We show that - using this metric - we can outperform the INS over long-term, large-scale localisation tasks. An illustration of this displacement metric is shown in Figure 5.4 below:

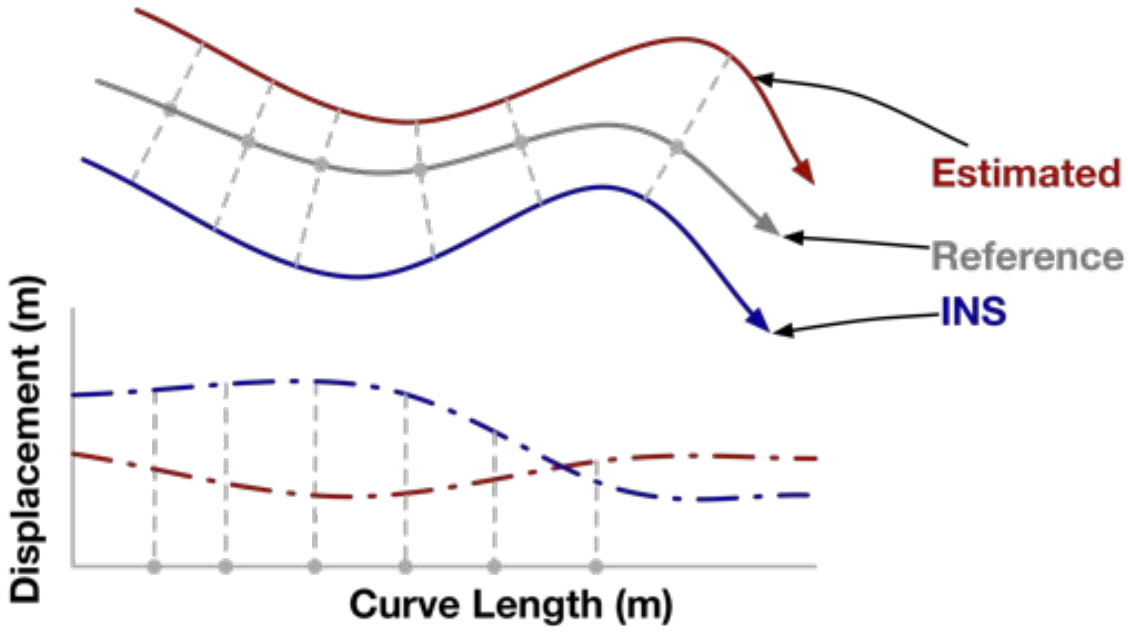



Figure 5.4: **Relative displacement illustration.** The distance between each pose (in the *test* trajectory, estimated using either INS, or $L^3$) to the closest pose in the *reference* trajectory is accumulated as a function of arc length. Better performance means a lower total displacement to the reference trajectory.

The distance from each pose in each trajectory is measured to the closest pose in the reference trajectory, and accumulated as a function of arc-length along the trajectory. We define this displacement measure, which - for pose $x_\tau$ - is:

$$E_\tau := \|x_\tau - \hat{x}_\tau^{\mathcal{S}}\| \tag{5.1}$$

where $x_\tau$ is the $\mathbb{SE}2$ pose at arc length $\tau$ along the trajectory curve, as given by either the INS or $L^3$, and $\hat{x}_\tau^{\mathcal{S}}$ is the closest pose in the survey trajectory to $x_\tau$. Note that the representation of the angle in each $\mathbb{SE}2$ pose here is actually *complex* - using this formulation means that the Euclidean distance is a metric over $\mathbb{SE}2$ [53].

It must be emphasised that the *relative* aspect of this metric is in reference to the trajectory used to build the original *map*. Other relative error metrics compare the ratio of pose deltas in successive links of a pose chain - we explicitly do not use this formulation, as we are more interested in the performance relative to the trajectory used to construct the map, rather than the relative error of sequential pose estimates.

Using this relative metric, we expect to do better - in terms of *lower* accumulated distance to the reference trajectory - than the INS, where this accumulation is given as the integral over the length of the trajectory:

$$E = \int_{\tau_0}^{\tau_N} E_\tau \quad d\tau$$

which is approximated by:

$$E = \sum_{i=1}^{N} \|x_t - \hat{x}_t^{\mathcal{S}}\|$$

for a trajectory consisting of $N$ discrete poses. Figure 5.5 shows the mean and standard deviation using this metric for the 26 kilometres of data collected around Begbroke:

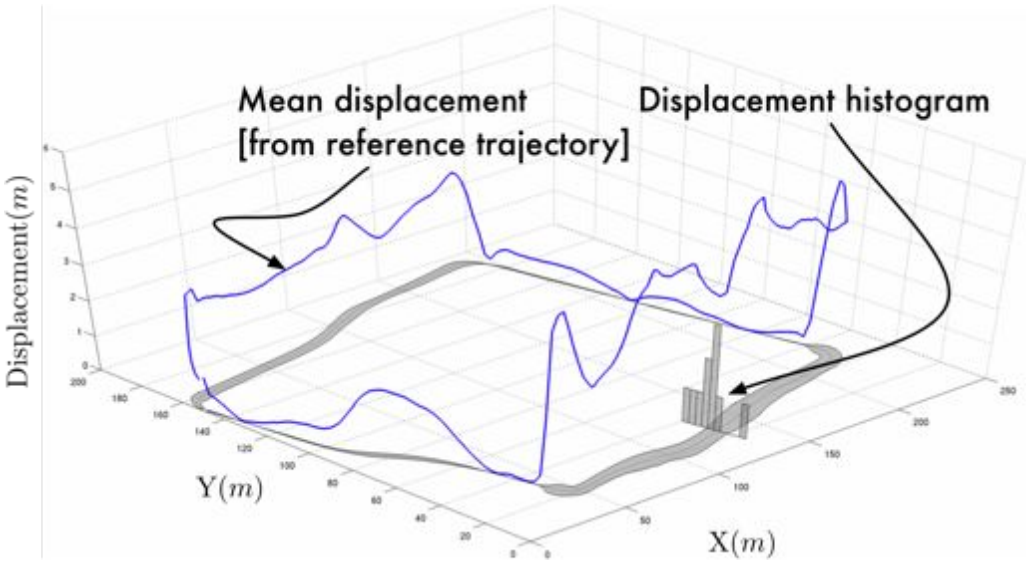



Figure 5.5: **INS cumulative relative displacement over 26km.** 26 kilometres of accumulated relative displacement data - using the INS - around the Begbroke site. By measuring the distance from any pose in a given trajectory to the closest pose (in an Euclidean sense) in a reference trajectory, we obtain this graph. Indicated are the mean ($z$-axis) and standard deviation (shaded,grey) for the entire set, as well as an example histogram for a given pose. The degraded INS performance is highly visible in the northern and southern sections.

Figure 5.5 was constructed by measuring the relative displacement - using Section 5.2 - for all of the indicated INS trajectories of the 26km dataset, and then performing a histogramming operation along a spline representation of the canonical route. Highly visible in the figure are the large deviations in the northern and southern sections.

It is important to note that this displacement will capture both the *true* deviation

from a trajectory to the reference trajectory **in addition** to the localisation error. Hypothetically, if we had traversed **exactly** the same route as the survey vehicle, we would expect this displacement to be zero. As we never traverse precisely the same route twice, we expect this displacement measure to be governed by some bound.

Of course, there are constraints on this bound imposed by the way roads are constructed. In Oxfordshire, for example, major access roads have a minimum specified carriageway of $6.0m$ [23]. This means that we should observe a deviation - using the displacement metric - substantially lower than this.

Figure 5.6 contrasts the displacement for the INS and $L^3$ for one loop of the Science park, as measured against a reference trajectory:

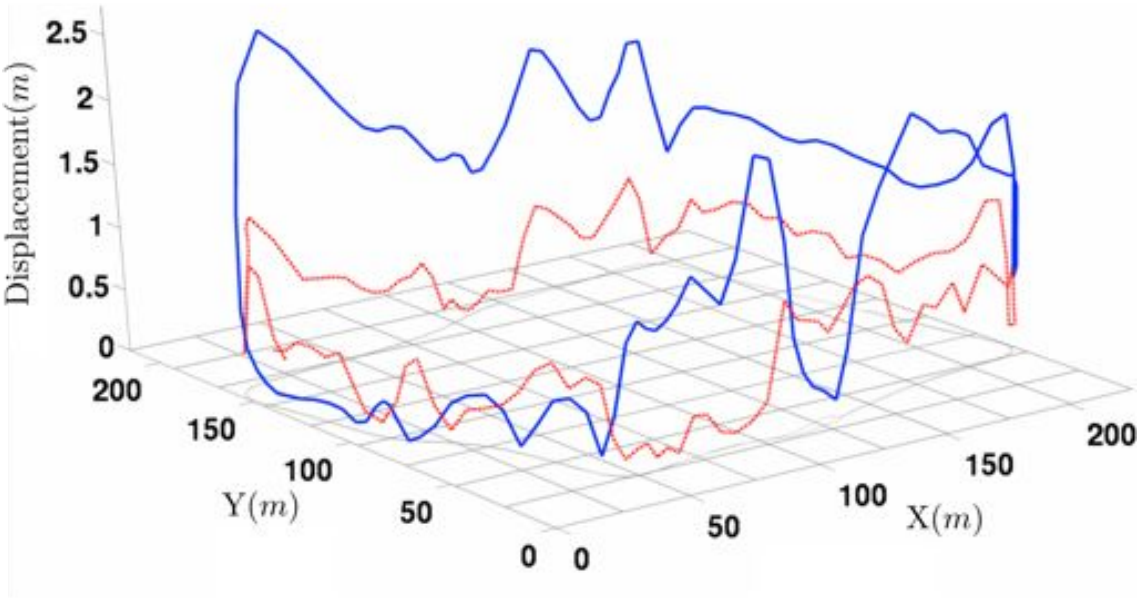



Figure 5.6: **Relative displacement: $L^3$ vs. INS over 700m.** An example relative displacement graph of both $L^3$ (shown in red, dotted) and the INS (blue, solid) as measured against a map of the Begbroke site. The INS exhibits a high displacement in areas of poor GPS signal quality, whereas $L^3$ is relatively constant.

As we can see from the Figure 5.6, the displacement for both the INS and $L^3$ are bounded - however, on average, the relative displacement over the route is substantially lower for $L^3$.

One of the reasons for this is the GPS signal-dropout experienced by the INS at

various locations around the Begbroke site - this dropout is problematic if the system is to be relied upon for any form of autonomous operation. Around Begbroke, the areas in which the INS exhibits a particularly large deviation are the northern and southern sections, which are characterized by tall buildings and foliage, respectively. In contrast, the displacement of $L^3$ is consistently lower over the site.

Figure 5.7 shows a comparison of the mean cumulative displacement per trajectory, collected over all 26km. Depicted are the median, $25^{th}$ and $75^{th}$ percentiles of the mean displacement per trajectory, for all the trajectories considered:

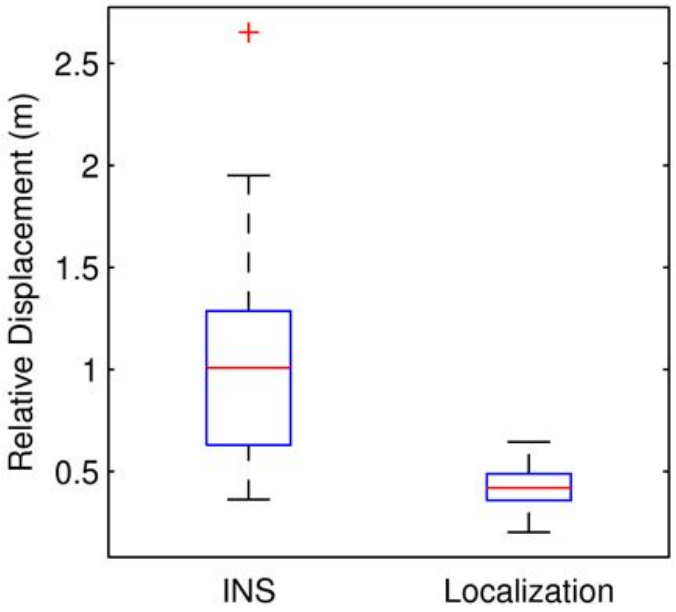



Figure 5.7: **Cumulative relative displacement comparisons.** A comparison of the mean displacement from the experience map for both the INS and $L^3$ over 26 km worth of trajectory data spanning three months. The box plot shows the median, $25^{th}$ and $75^{th}$ percentiles, with outliers plotted individually as points. As can be seen from the figure, the standard deviation of this displacement is substantially lower for $L^3$.

This plot was obtained by averaging the relative displacement for the INS and for $L^3$ (using the INS velocity estimates) for each loop of the 26km dataset (more than 50). Not only is the median displacement distinctly lower for the $L^3$ approach, but outliers have been eliminated and the variance of the displacement substantially

reduced. This is compelling evidence that localisation using this method is far more reliable and repeatable over the long term than relying on estimates from the INS.

An advantage of the active sensing modality is that the system can often remain localised in the face of drastic weather-induced scene change, which results in high variability in appearance space. Consider Figure 5.8 which shows an image sequence of three of the many appearances of the Science Park across the seasons:

(a) Snow (b) Rain (c) Sunshine

Figure 5.8: **Appearance variation over the seasons.** The northern road of the Begbroke Science Park after a snow-storm, (a), a rain-shower (b) and in bright sunshine (c). The active sensing modality is impervious to the drastic change in appearance (although performance can be severely degraded *during* precipitation - see Appendix A for an example).

Drastic scene change can be problematic for vision-based localisation systems. However, Figure 5.9 through Figure 5.11 show the LIDAR data across the same datasets as Figure 5.8:

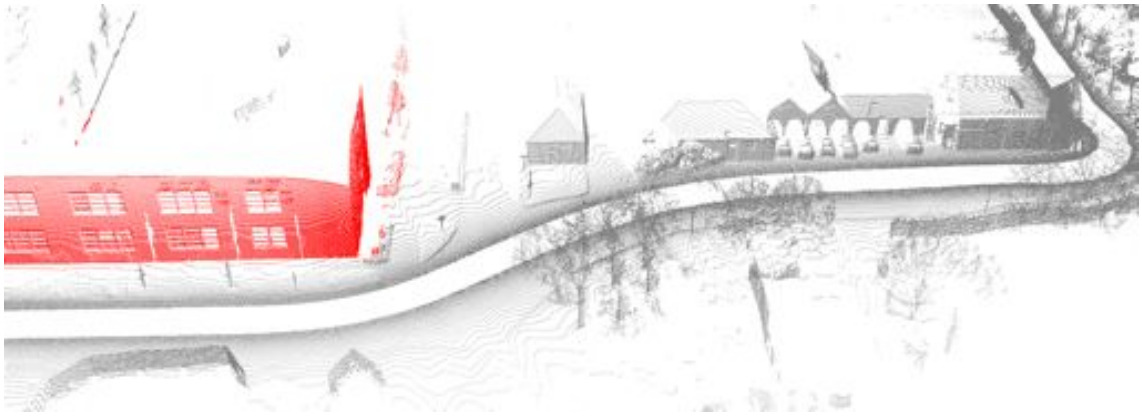


Figure 5.9: **LIDAR data: Snow.** The 3D LIDAR data collected from the declined LIDAR, corresponding to Figure 5.8(a).

Notice particularly in Figure 5.8(a) the obstruction of the road surface by accumulated snow - for this reason, road-reflectance matching (leveraged to good effect by [55]) is not reliable across seasons - whereas the prismatic structure of the environment is practically unchanged. Figure 5.10 and Figure 5.11 show the LIDAR data over the remaining scenes of Figure 5.8:

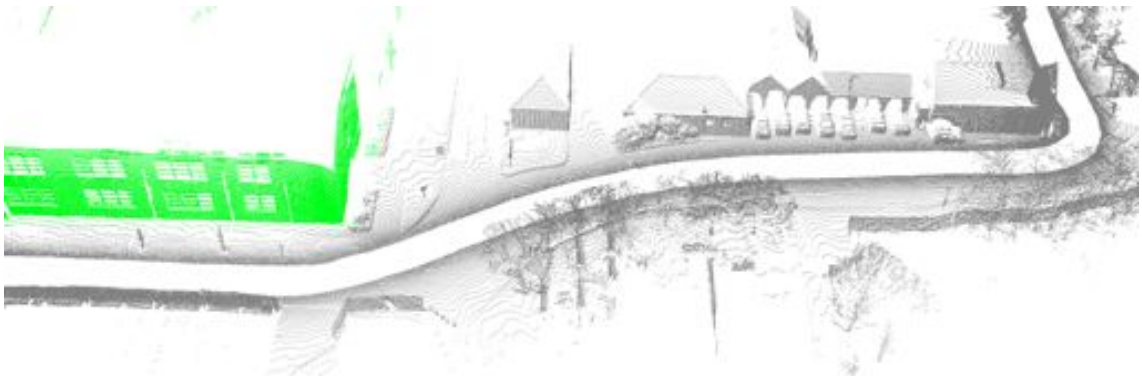


Figure 5.10: **LIDAR data: Rain.** The dataset corresponding to Figure 5.8(b). Note again that - although the road surface is different to both Figure 5.8(a) and Figure 5.8(b), the prismatic structure of the environment - buildings, walls - is unchanged.

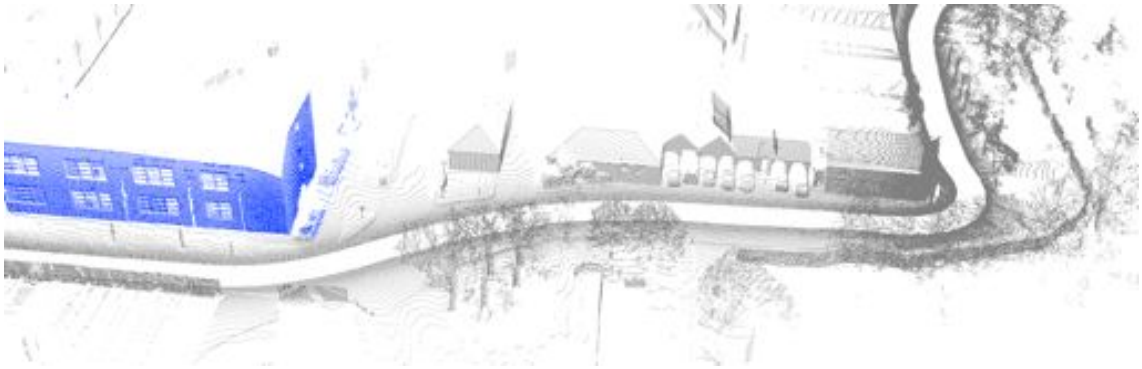


Figure 5.11: **LIDAR data: Sunshine.** The dataset corresponding to Figure 5.8(c).

The same building is highlighted in each of the images to serve as a visual anchor. This qualitative evaluation highlights the self-similarity of the scene - from the perspective of the declined 2D LIDAR - across the varying environmental conditions. This is the core reason for leveraging LIDAR - the ability to provide robust ranging measurements through seasonal short-term, and long-term, change.

## 5.3 Map management

Given the scale of the maps evaluated in this section, it is not possible to store a single monolithic map instance in the memory of a computer. The map is therefore partitioned into a *graph*, using equally-spaced nodes in the reference trajectory. Then, given the vehicles current location within the map, the next map-node can be found by a simple traversal:

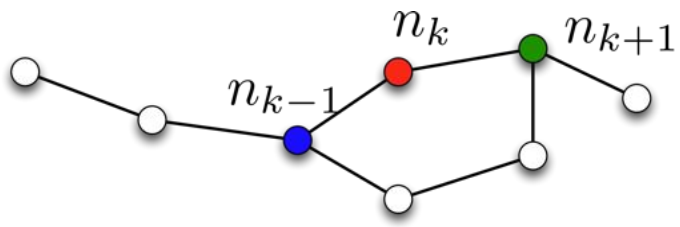



Figure 5.12: **Graph-based map-management.** Due to the sizes of the map for large-scale localisation tasks, map retrieval is done by traversing a graph of anchor nodes embedded in the map trajectory. This allows for sequential access, and alleviates the need to load the entire map into memory. Shown here is the current node $n_k$, and the next and previous nodes, $n_{k+1}$ and $n_{k-1}$ respectively.

The map is sectioned into sub-maps by constructing the Voronoi tessellation - another space-partitioning technique - of the map given equally-spaced poses in the reference trajectory. The map-equivalent of Figure 5.12 is shown in Figure 5.13:

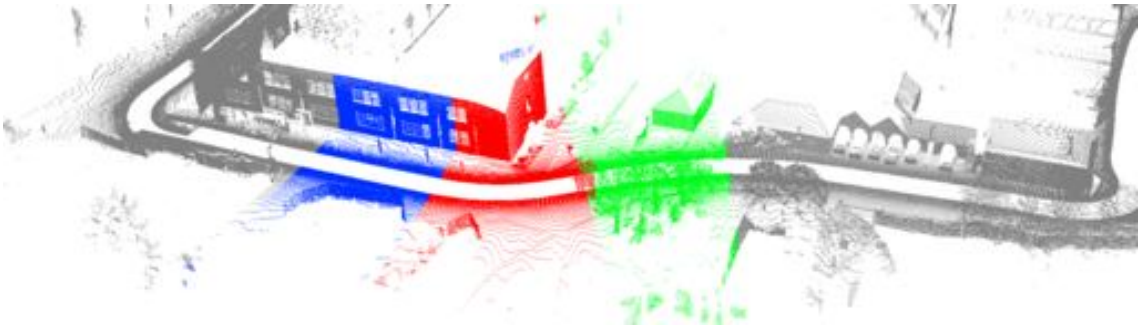


Figure 5.13: **Map-management: Begbroke nodes.** The partitioned map of Begbroke, illustrated by Figure 5.12. The red section corresponds to the vehicles current position, with the next and previous sections highlighted in green and blue, respectively. (The background map is rendered grey).

Figure 5.13 shows the metric maps associated with the current, previous and

next nodes in the graph. This representation is necessary to avoid attempting to load entire city-scale maps into memory. The next section presents results obtained over the nearby bustling town of Woodstock.

## 5.4 Large-scale urban localisation

The Science park is an ideal testing ground for LIDAR-based localisation - the site has relatively low long- and short-term scene change. However, we would like the system to be robust enough to use in testing real-world situations.

Further experiments were therefore conducted in and around the neighbouring towns of Woodstock and Kidlington - both busy urban centres in Oxfordshire. Figure 5.14 shows an overhead view of both these towns, in relation to the Begbroke Science Park:

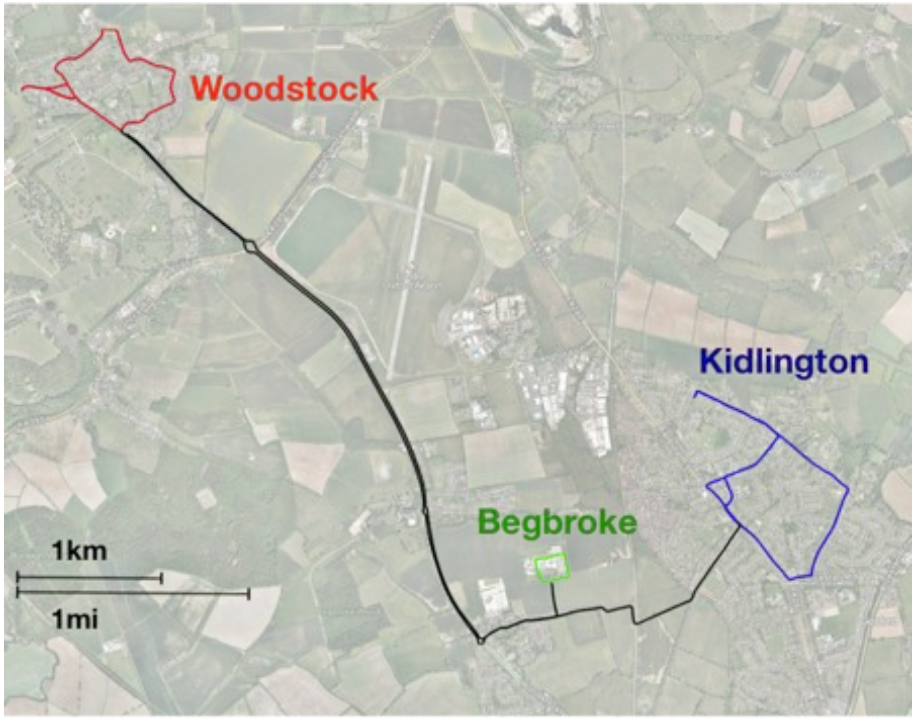



Figure 5.14: **Oxfordshire test sites.** An overview of the target workspaces for the $L^3$ approach. Highlighted in green is the Begbroke site; in red, the town of Woodstock, and in blue, Kidlington. (Overhead imagery courtesy of Google Maps).

## Woodstock

In order to establish a baseline around Woodstock, a dataset consisting of multiple-loops in and around the town centre was taken at 8pm, ensuring minimal vehicle and pedestrian traffic - again we note the advantage of the LIDAR sensing modality, allowing operation at all times of the day.

This dataset consists of 10 kilometres around Woodstock and surrounds. We now compare the $L^3$ estimator (using velocity estimates from both the INS *and* the horizontal LIDAR via scan-matching) over this dataset to obtain baseline performance. Figure 5.15 shows the estimated trajectory for this evening dataset using two velocity sources - the INS, and the LO system:

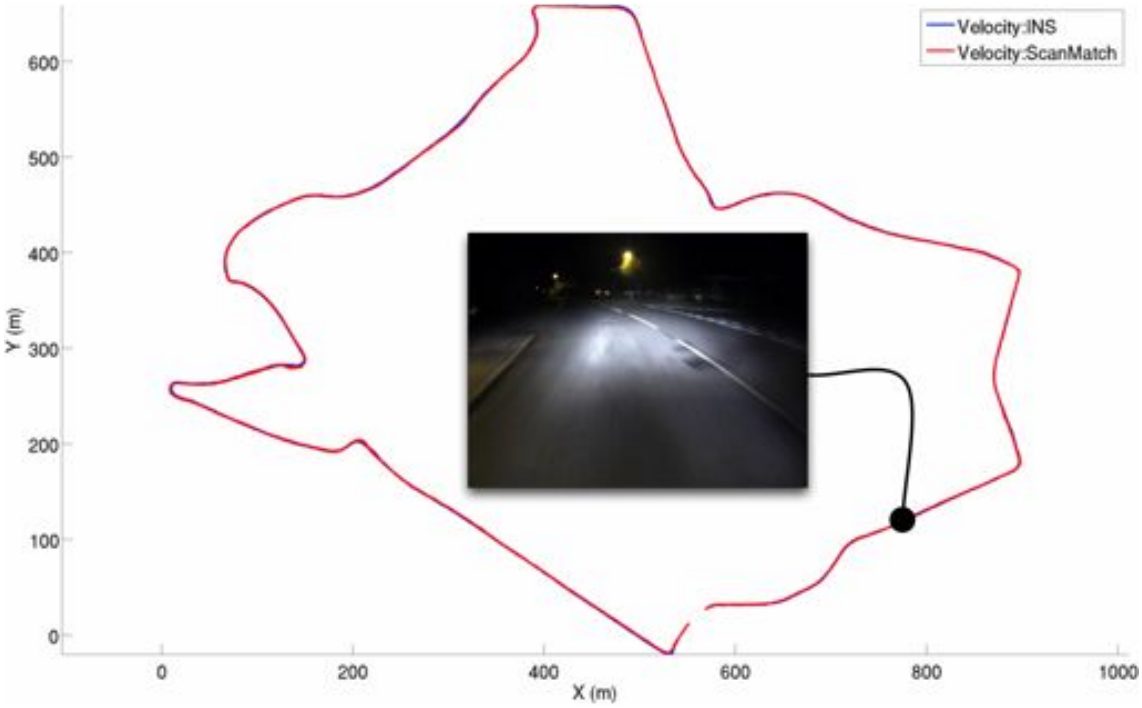
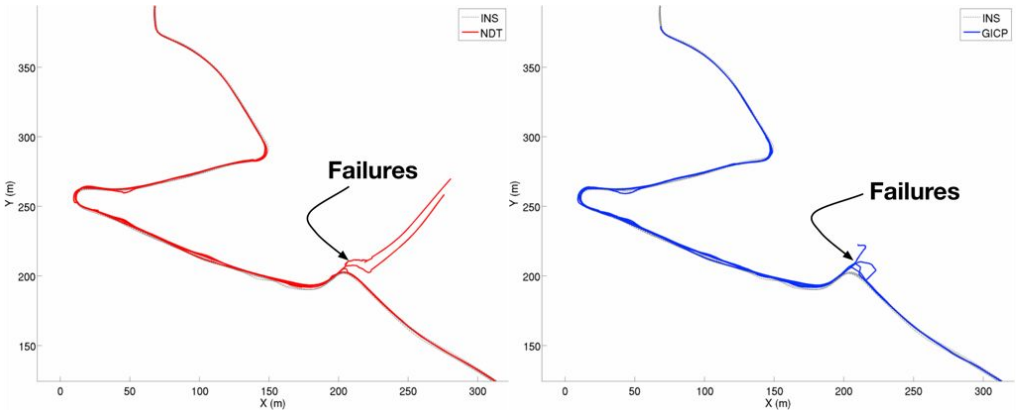



Figure 5.15: **Woodstock evening results.** An overview of the Woodstock evening dataset trajectory estimates. Two velocity sources are used: (1) INS velocity estimates as a control (blue), LO estimates (red). As can be seen from the figure, they are indistinguishable. Also shown is an example image from the forward-facing stereo camera - this is a testing situation for VO.

Figure 5.15 is a useful "control" experiment - with minimal activity around the

town at 8pm, the results provide an excellent baseline for performance comparison. As can be seen from Figure 5.15, it is impossible to distinguish between the localisation estimates using the INS as a velocity source, and the estimates using LO.

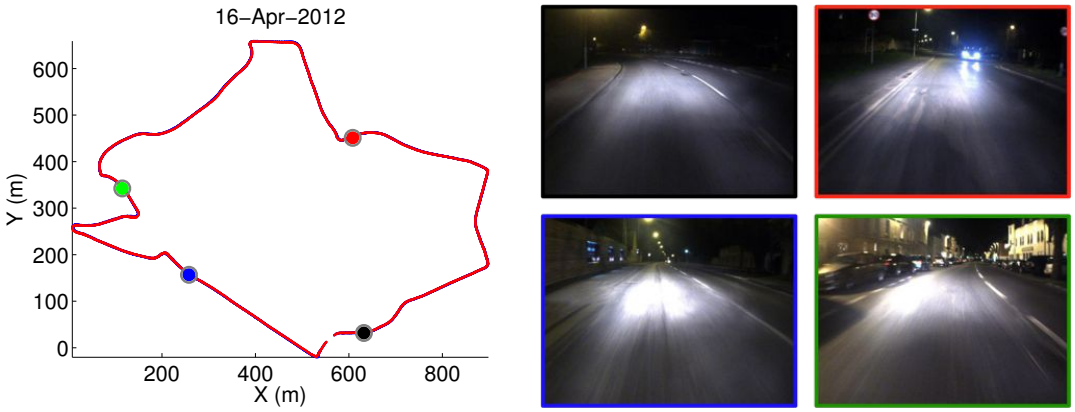
Here we take a brief diversion to analyse other methods - in Chapter 4 we discussed the use of GICP and the Normal Distributions Transform for registration, and hence localisation. However, as can be seen from Figure 5.16, these algorithms have similar failure patterns when the LO is used as a velocity source:




(a) NDT Failure: Evening dataset (b) GICP failure: Evening dataset

Figure 5.16: **NDT/GICP failure cases.** NDT (a) and GICP (b) have similar failure cases when using LO as a velocity source - slight velocity errors result in matching (and subsequently tracking) failure. This catastrophic failure is not suffered by $L^3$. (These comparisons were made over the same data used by $L^3$ in Figure 5.15).

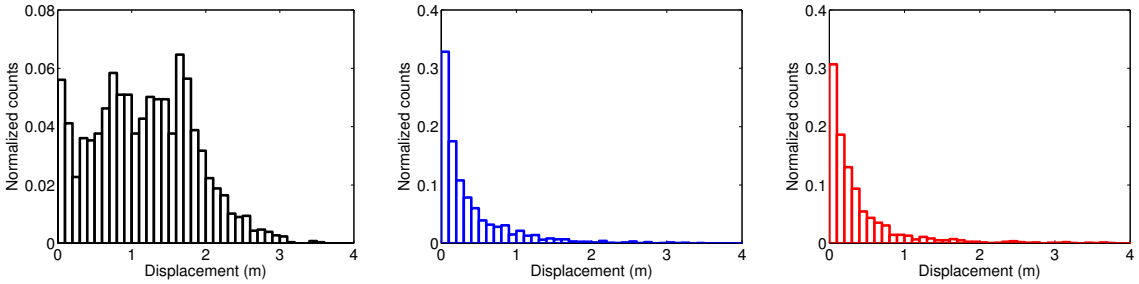
This is an issue if we are seeking to use LO as the source of our velocity estimates - as such we discount the use of such approaches for localisation. The following image sequences show an illustrative comparison between the $L^3$ systems, and the INS. Figure 5.17(a) shows an overview of the route, and (b) shows representative images recorded during the run:




(a) Woodstock route (b) Representative images

Figure 5.17: **Relative displacement: INS (16/04/2012).** An overview of the route taken around Woodstock (a), representative images from the dataset, (b).

In Figure 5.17(a), the $L^3$ trajectories estimated using the INS velocity feed are indicated in blue, with those using the LO velocity feed in red. The Woodstock route is approximately 2.6km long, which is traversed multiple times in any given dataset. We now compare the relative displacement histograms obtained from the INS, with equivalent histograms from $L^3$:

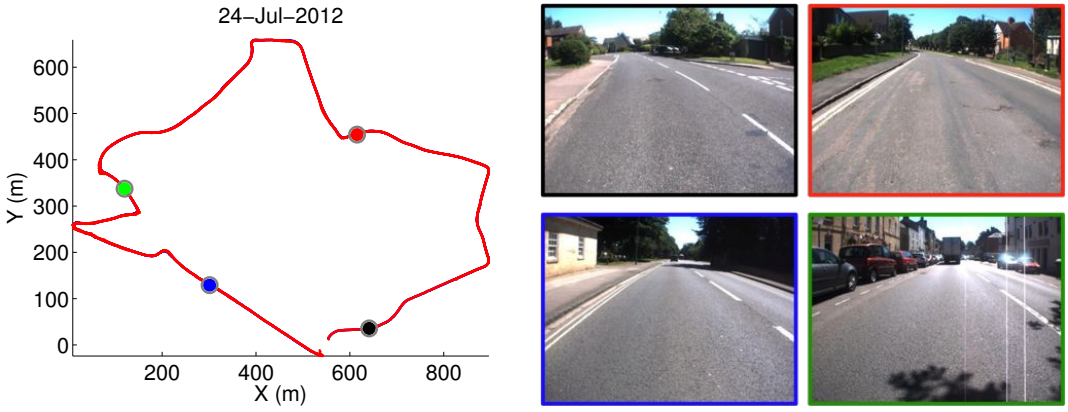



(a) INS (b) $L^3$ (INS velocity) (c) $L^3$ (LO velocity)

Figure 5.18: **Relative displacement: $L^3$ (16/04/2012).** Histograms of the relative displacement of the INS (a) against: (b) $L^3$ with INS velocity, and (c) $L^3$ with LO velocity.

As is clear from Figure 5.18, the relative displacement statistics are more consistent - i.e. closer to the canonical trajectory - for $L^3$ as compared to the INS, which exhibits a broad range of displacements. Consider again the constraints imposed by

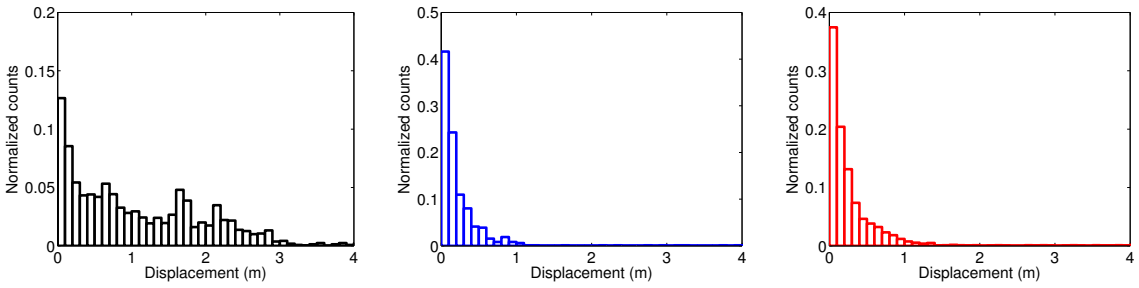
the road network - an average lateral deviation of 2 metres is unlikely. Figure 5.19 shows a similar plot over the route, two weeks later:




(a) Woodstock route (b) Representative Images

Figure 5.19: **Relative displacement: INS (24/07/2012).** A similar plot to Figure 5.17, showing the route around Woodstock (a), and representative images from the dataset, (b).

Consider the striking appearance change between Figure 5.17(b) and Figure 5.19(b) - our active sensing modality again ensures we can stay localised irrespective of the time of day. Figure 5.20 again shows the equivalent displacement statistics for $L^3$ against the INS:

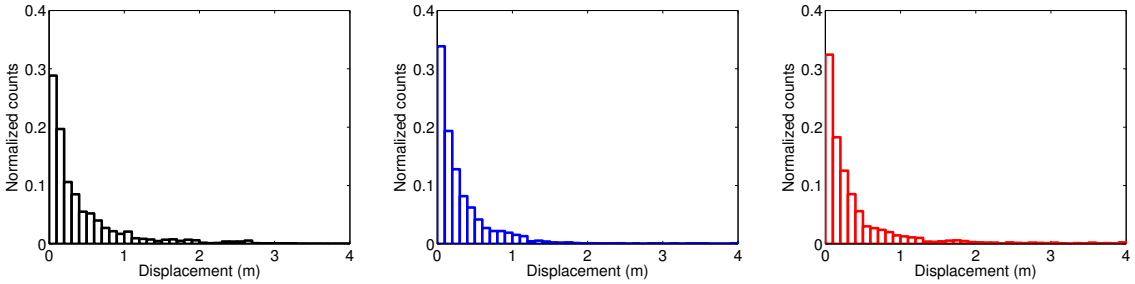



(a) INS (b) $L^3$ (INS velocity) (c) $L^3$ (LO velocity)

Figure 5.20: **Relative displacement: $L^3$ (24/07/2012).** Histograms of the relative displacement for $L^3$ over the trajectories shown in Figure 5.19(a), showing (a) the INS performance, (b) $L^3$ with INS velocity, and (c) $L^3$ with LO velocity.

To illustrate that this comparison is a valid one, Figure 5.21 again shows the same

route, but with Differential GPS (DGPS) corrections supplied by a *base-station* in Begbroke:




(a) INS (b) $L^3$ (INS velocity) (c) $L^3$ (LO velocity)

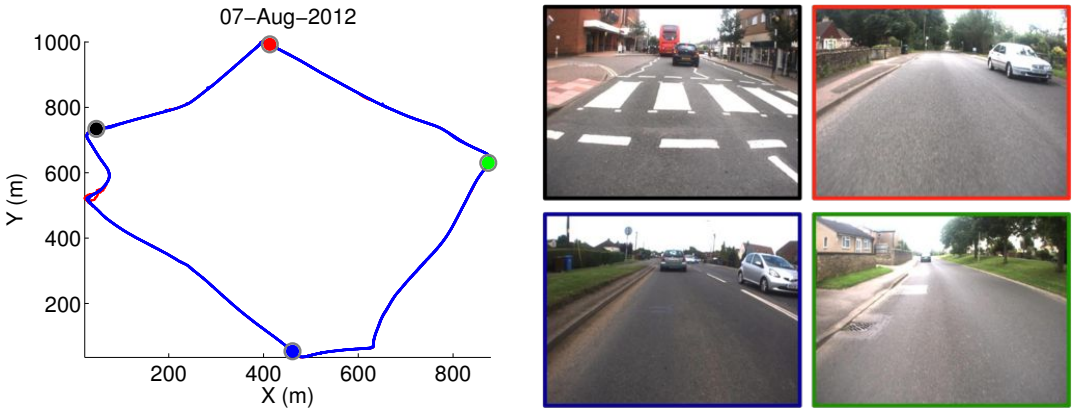
Figure 5.21: **Relative displacement: $L^3$ (23/08/2012).** Histograms of the relative displacement using a DGPS base station, showing (a) the INS performance, (b) $L^3$ with INS velocity, and (c) $L^3$ with LO velocity. Notice the substantially improved INS estimate.

DGPS uses a fixed base-station, allowing the system to estimate, and correct for, timing errors (typically from ionospheric effects) [67]. This results in substantially improved INS estimates, as is evidenced by Figure 5.21(a) - these similar deviation statistics using the higher precision DGPS validate this comparison.

Note that the results shown are a subset of all of the data collected around Woodstock, although they serve to highlight the salient points. We now consider a different route around the nearby town of Kidlington.

## Kidlington

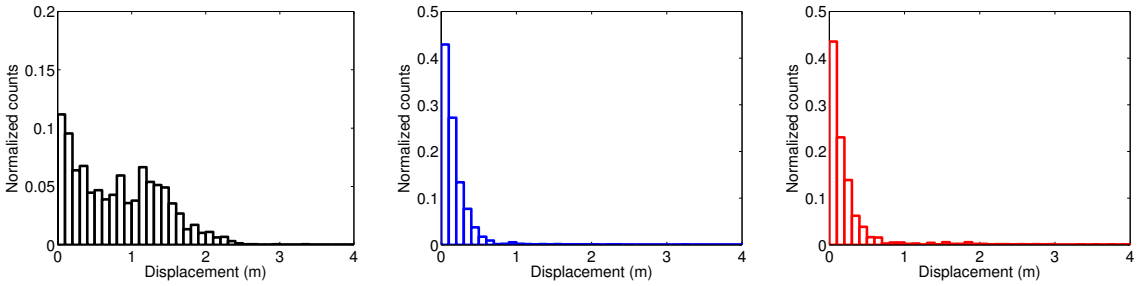
As final validation of $L^3$, we present results gathered around Kidlington. Figure 5.22 shows several loops around the 2.8km site:




(a) Kidlington route (b) Representative images

Figure 5.22: **Relative displacement: INS (7/08/2012).** The route around the 2.7km test site (a), and a sampling of images (b). Kidlington provides a very different scene as compared to Woodstock with longer, sparser sections (in the south, predominantly).

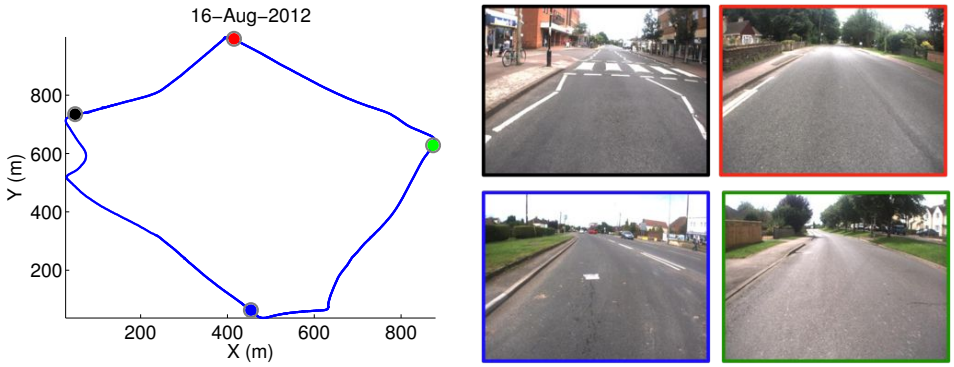
The Kidlington route is very different to the Woodstock route - in particular, the southern section is much "sparser" in terms of surrounding structure. Even so, $L^3$ works well as shown in Figure 5.23:




(a) INS (b) $L^3$ (INS velocity) (c) $L^3$ (LO velocity)

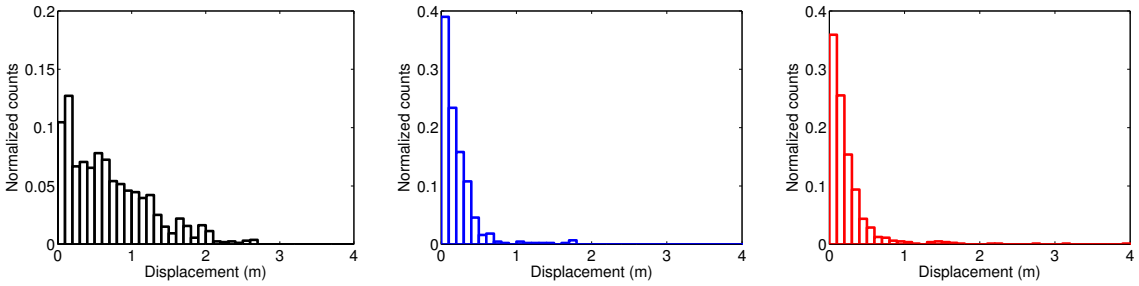
Figure 5.23: **Relative displacement: $L^3$ (7/08/2012).** Histograms of the relative displacement for $L^3$ over the trajectories shown in Figure 5.22(a), showing (a) the INS performance, (b) $L^3$ with INS velocity, and (c) $L^3$ with LO velocity.

Again clearly visible is the superior performance of the $L^3$ approaches (both LO and INS-based). Figure 5.24 shows another Kidlington dataset, one week later:




(a) Kidlington route (b) Representative images

Figure 5.24: **Relative displacement: INS (16/08/2012).** The route around Kidlington (a), and associated images from the dataset, (b).




(a) INS (b) $L^3$ (INS velocity) (c) $L^3$ (LO velocity)

Figure 5.25: **Relative displacement: $L^3$ (16/08/2012).** Histograms of the relative displacement for $L^3$ over the trajectories shown in Figure 5.24(a), showing (a) the INS performance, (b) $L^3$ with INS velocity, and (c) $L^3$ with LO velocity.

Given the similarity of the results produced - over Kidlington and Woodstock - we can conclude that using $L^3$ is a robust, reliable way of localising a vehicle over the long term. The next section presents cumulative results for all the data analysed.

### 5.4.1 Cumulative results

The results for the entire localisation comparison are encapsulated by Figure 5.26:

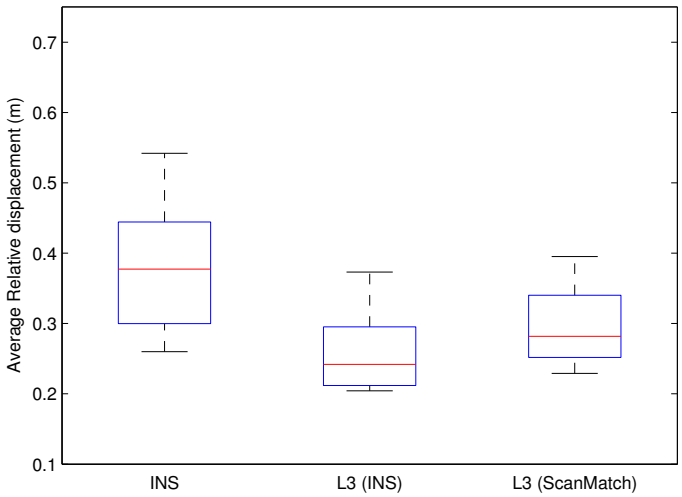



Figure 5.26: **Average relative displacement over 50 kilometres.** A plot showing the average relative displacement - the average displacement to a reference trajectory, averaged over more than 50km of trajectory data. This is shown for the INS (left), the $L^3$ approach with INS velocity estimates (centre), and (right) $L^3$ with LO velocity estimates.

This figure shows the **average** relative displacement to a constant reference trajectory for each of the modalities considered, collected over more than 50km. Figure 5.26(left) shows the estimates for the INS itself - note that over the Woodstock site we don't see quite the same variation as we do over Begbroke - this is because the vast majority of the route is free of tall buildings and foliage.

Even so, we see that - using $L^3$ with the INS velocity estimates - we are *still* performing better (in terms of the distance to the reference trajectory) than the INS. If we instead substitute the LO system as the velocity source, our performance degrades - but crucially is *still* better than the INS.

An example route highlighting this deviation around Woodstock is shown in Figure 5.27:

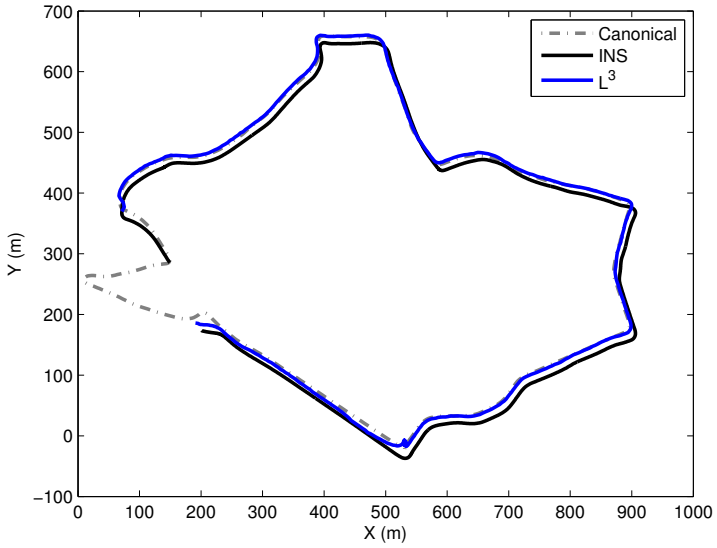



Figure 5.27: **Trajectory comparison around Woodstock: INS vs $L^3$.** Comparisons of estimated trajectories around the canonical Woodstock route (grey, dashed) using $L^3$ (blue) and the INS (black). Note the vertical shift present in the INS estimate.

By plotting the relative displacements of these trajectories to the canonical route, we obtain Figure 5.28:

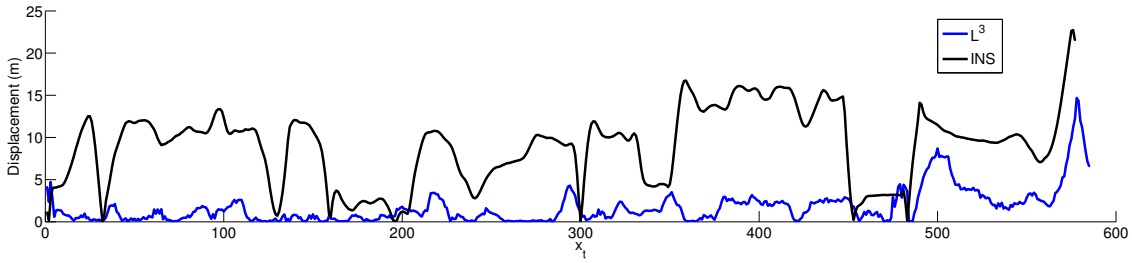



Figure 5.28: **Displacement comparison around Woodstock: INS vs $L^3$.** The displacement comparison for the trajectories shown in Figure 5.27, showing the larger averaged displacement of the INS to the reference trajectory.

Figure 5.28 highlights why the INS often accrues larger error. Often, the INS estimated trajectories are shifted vertically, increasing the average distance to the reference trajectory - this is something we wish to avoid.

Table 5.2 shows the cumulative results accrued around Begbroke, Woodstock and

Kidlington over a period of more than a year using $L^3$, and is offered as compelling evidence of the validity of this approach, and the seriousness with which we take long-term operation:

| **TTM** (days) | **Distance** (km) | **Cumulative** $L^3$ (INS) (km) | $L^3$ (LO) (km) | **Location** |
|---|---|---|---|---|
| 0-31 | 26.00 | 26.00 | - | Begbroke |
| 0 | 10.56 | 36.56 | 10.56 | Woodstock |
| 49 | 10.47 | 47.03 | 21.03 | Woodstock |
| 114 | 5.22 | 52.25 | X | Woodstock |
| 119 | 2.23 | 54.48 | X | Woodstock |
| 136 | 10.05 | 64.53 | 31.08 | Woodstock |
| 141 | 12.93 | 77.46 | 44.01 | Woodstock |
| 148 | 4.91 | 82.37 | 48.92 | Woodstock |
| 178 | 5.51 | 87.88 | X | Woodstock |
| 0 | 7.86 | 95.74 | 56.78 | Kidlington |
| 23 | 13.19 | 108.93 | X | Kidlington |

Table 5.2: Over 100km of successful localisation around Oxfordshire, with 50km of LO-only localisation.

Unfortunately, the rows marked $X$ in Table 5.2 indicate that the LO-based $L^3$ system failed - lost tracking - during the course of the dataset. This cutoff is a harsh measure, as in some cases, the system had stayed localised for multiple previous loops, only to fail on the last loop - in these cases, the entire dataset is marked as a failure.

Given that we are inferring vehicle velocity from sensor data, it is unsurprising that we encounter - over the course of a year - situations that cause our localisation algorithm to fail. Figure 5.29 shows the location of 3 failures, on 3 separate loops over the course of 6 months:

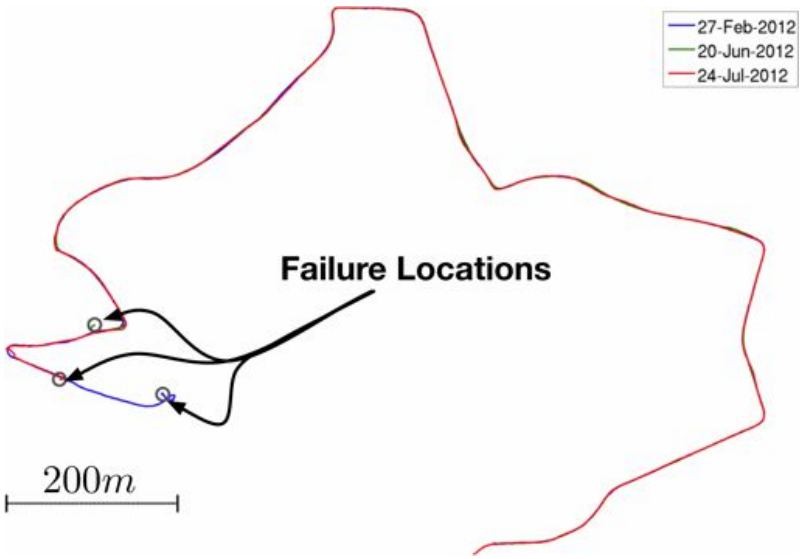



Figure 5.29: **Woodstock failure locations.** Localisation failures for 3 loops from 3 different datasets over a 6 month period. In the next section, we develop means for addressing the root cause of these failures.

Note that these are failures of LO-based $L^3$ - using $L^3$ with velocity estimates from the INS does not suffer from these degeneracies. Also note the locations of these failures - all in close proximity to the town centre. The primary cause of these failures are biased velocity data from the LO - this can come from a number of sources, but the majority arises from traffic. In Chapter 6 we explore means to combat this degeneracy, but first we briefly address a previous concern - staying localised in featureless sections.

## 5.5 Localisation on the open road

In Chapter 4, we discussed the use of Mutual Information as a criterion to match run-time data against the map prior - this was to enable us to stay localised in the event that the surrounds contained little distinct structure. Figure 5.30 again shows the problematic section of Begbroke:

Figure 5.30: **Featureless road section.** A relatively featureless road section outside of Begbroke, described in Chapter 4. We use remission-based matching to stay localised in these areas.

Figure 5.31 shows the localisation results over this section, using MI with remission as an objective function:

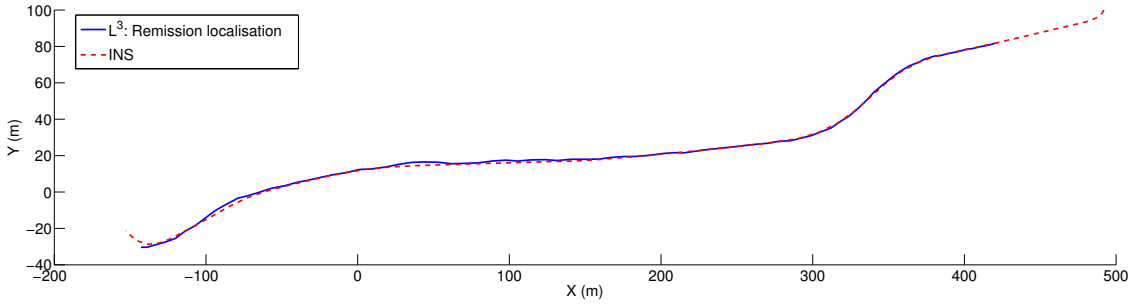



Figure 5.31: **Featureless road section: Localisation results.** Localisation results over the section shown in Figure 5.30 by matching remission using Mutual Information.

As we can see from Figure 5.31, we are able to stay localised on the road by making use of the reflective properties of the world. At any given location, we can have a reasonable expectation of being able to use either remission-based or structure-based localisation, and therefore have the means to stay localised regardless of the environment, and across a wide range of weather conditions.

## 5.6 Summary

In this chapter, we have validated the $L^3$ approach over more than 100km of real-world mileage, accrued over a year across a wide variety of environments. We have presented a relative metric for comparison, and show that - using this metric - we can outperform the repeat capabilities of the $\$100,000$ INS.

We have also highlighted various points of failure in the system, and with this in mind, turn to the next chapter where we characterise these failures, and develop means of addressing them.

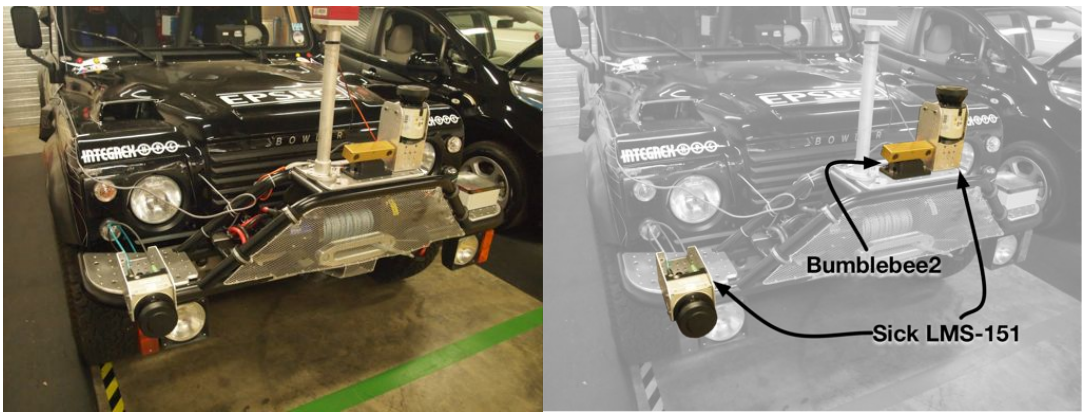
# Chapter 6

# Leveraging Experience For Robust Long-Term Localisation

## 6.1 Introduction

In the previous chapter, the $L^3$ approach was validated over a large amount of real-world data. However, we saw the deleterious effect of using scan-matching for velocity estimation in areas with heavy traffic. This velocity error directly degraded the performance of the localisation estimates, and must be removed - this is the focus of this chapter.

In the following sections, we seek to demonstrate the viability of using 2D LIDAR data as the *sole* means for accurate, robust, long-term road-vehicle localisation within a prior map in a complex, dynamic real-world setting. Estimation errors induced by passing vehicles, pedestrians, ground-strike etc., will be accommodated by learning a positional-dependent sensor model - that is, a *contextual* sensor-model that varies spatially - and it will be shown that learning such a model for LIDAR is necessary to deal gracefully with the complexities of real-world ranging data - this is explored in Section 6.3. Figure 6.1 shows the LIDAR configuration:




(a) Wildcat sensor payload (b) Dual LIDARs, and a Bumblebee2 camera

Figure 6.1: **Wildcat sensor configuration.** The dual-LIDAR setup on the Wildcat. Note the horizontally-oriented LIDAR adjacent to the Bumblebee camera, and the vertically-oriented LIDAR on the front bumper. The velocity estimates from the horizontal LIDAR and the range-data from the declined LIDAR allow us to build the run-time swathe, as discussed in Chapter 4.

With this arrangement, it is possible to generate the run-time swathe (as discussed in Chapter 4), an example of which is shown in Figure 6.2:

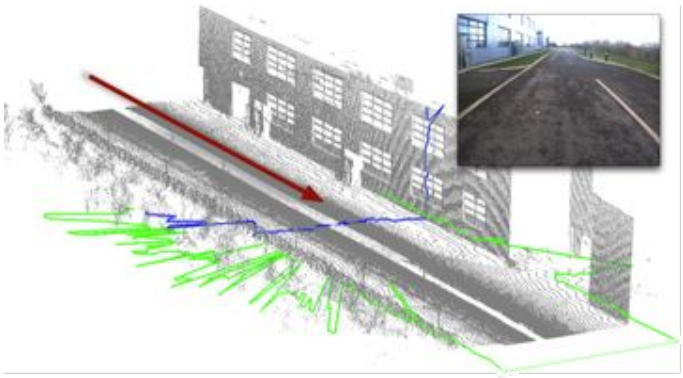


Figure 6.2: **Example point clouds and illustrative scans.** A perspective view of a typical run-time generated point cloud, with the vertical and horizontal lasers highlighted in blue and green respectively. The motion of the vehicle - generating the swathe data as it moves through the environment - is indicated by the arrow. Clearly visible in the swathe are the window frames and building edges. The inset image shows the view of the scene from a camera on the front bumper of the vehicle.

This chapter shows how probabilistically modelling the noisy velocity estimates from the horizontal laser feed and fusing these estimates with data from the declined

LIDAR to form the dense 3D swathe will provide the means for long-term pose estimation.

Crucially important to this method is the accommodation of velocity errors arising from transient objects in the horizontal LIDAR fan - to this end, a non-stationary probabilistic filter for point-data is learned and we show that by doing so, we are able to negate the velocity error induced from vehicles, ground-strike and so on.

The core thesis here is that the *context* of the sensor is greatly informative given the observed data in the LIDAR fan. In the next section we examine some of the difficulties incurred from estimating vehicle velocities from the horizontal LIDAR in a testing real-world environment, and the need for such a filter.

## 6.2 LIDAR-only localisation

The high-level localisation procedure is again outlined in Algorithm 4 for continuity:

**Algorithm 4** Localisation Procedure

1: **procedure** RunLocalisation($\mathcal{P}, \mathcal{T}$)
2: $\hat{\mathcal{T}} \leftarrow \mathcal{T}$ ▷ Initialise pose
3: **loop**
4: $\{\mathcal{Z}_h, \mathcal{Z}_v\} \leftarrow (r_1^h, \ldots, r_n^h), (r_1^v, \ldots, r_n^v)$ ▷ Acquire LIDAR Data
5: $\mathbf{V}, \Omega \leftarrow \text{EstimateVelocities}(\mathcal{Z}_h)$
6: $Q \leftarrow \text{BuildSubmap}(\mathbf{V}, \Omega, \mathcal{Z}_v)$
7: $\mathcal{T}' \leftarrow \text{Predict}(\hat{\mathcal{T}}, \mathbf{V}, \Omega)$ ▷ Predict new pose
8: $\hat{\mathcal{T}} \leftarrow \text{Update}(\mathcal{P}, \mathcal{Q}, \mathcal{T}')$ ▷ Optimise estimate using $L^3$
9: **end loop**
10: **end procedure**

At run-time, the algorithm is seeded with an initial pose guess, $\mathcal{T}$. It is then run continuously, taking in new horizontal and vertical scan data $(\mathcal{Z}_h, \mathcal{Z}_v)$ from the dual-LIDAR system. The horizontal scan data is used to estimate the linear and rotational velocities $\mathbf{V}$ and $\Omega$ by running an ICS-based scan-matcher, which are then subsequently fed into the map-building procedure in order to construct the run-time

swathe. Once the swathe has been generated, it is utilised in the pose estimation step to solve for the current best pose estimate $\hat{\mathcal{T}}$ using $L^3$.

This algorithm was shown to work well extensively over large amounts of real-world data, as demonstrated in Chapter 5 - however, we noticed localisation failures in areas with particularly high traffic volumes, and this is something we seek to address. Figure 6.3 shows an overhead view of the Woodstock route, decomposed into various regions:


Figure 6.3: **Woodstock, Oxfordshire.** An overview of the route driven in Woodstock, Oxfordshire. The route passes through a number of very different environments - suburban housing (highlighted in red, eastern section), a region with dense foliage and steep gradients (green, north) and a town centre with a large number of pedestrians and vehicles (blue, western and southern section). Accurate localisation in these very different settings is a difficult task. (Imagery courtesy of Google Maps).

In addition to the error induced by traffic (predominantly in the blue highlighted section in Figure 6.3) the route around Woodstock has a large altitude gradient, as is shown by Figure 6.4:

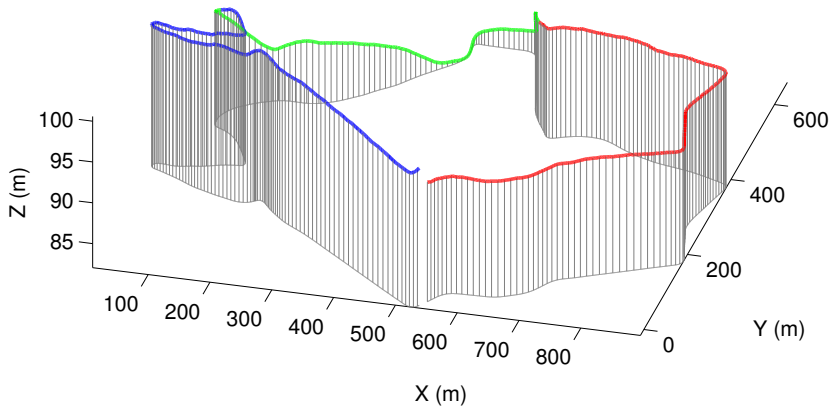



Figure 6.4: **Woodstock height map.** A height plot of the trajectory around Woodstock, obtained from the INS. The $(x, y)$ projected trajectory is plotted in grey. As can be seen, there is a large gradient change in the northern section of Woodstock (corresponding to the green section in Figure 6.3.)

Visible in the figure is the large gradient change over Woodstock - as we can see, a planar estimate of the world is a poor approximation. This altitude change causes a large amount of ground-strike, which induces error into the pose estimates through the erroneous velocity feed. Any system reliant on LIDAR scan-matching will have to accommodate the estimation errors induced by these different testing regions. In the following sections, we explore the root causes of such velocity errors, and methods to account for them.

### 6.2.1 Sources of velocity error

Given that the Wildcat is fitted with a very capable Inertial Navigation System (INS), it provides an excellent baseline with which to compare the velocity estimates from the scan-matching algorithm using the feed from the horizontal LIDAR. The following plots show the error of the scan-matching method measured against the INS, as a function of position:

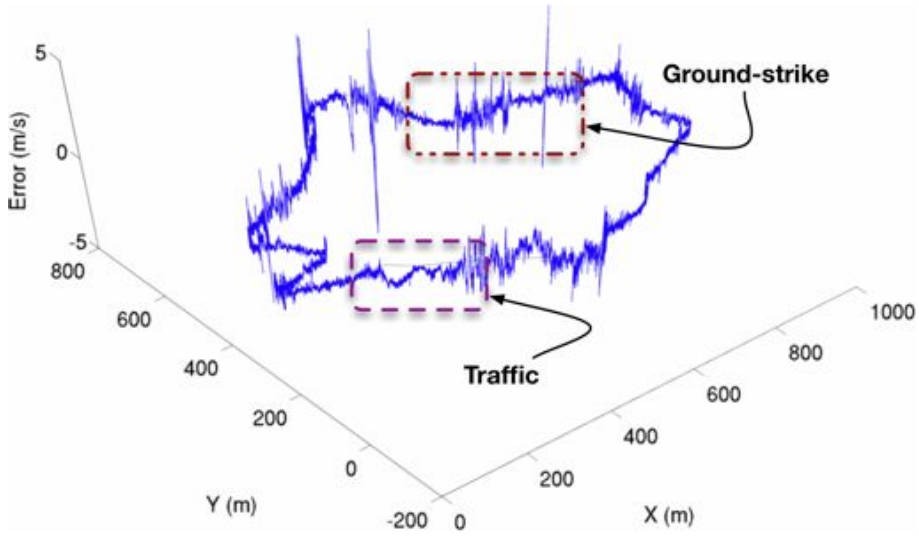



Figure 6.5: **Velocity error: Woodstock.** A plot of the error in linear velocity from the scan-matching algorithm over the course of the Woodstock run, as compared to the INS on the vehicle. In some locations, the error is in excess of $6m/s$. Typical causes of these errors are illustrated in Figure 6.7 through Figure 6.9.

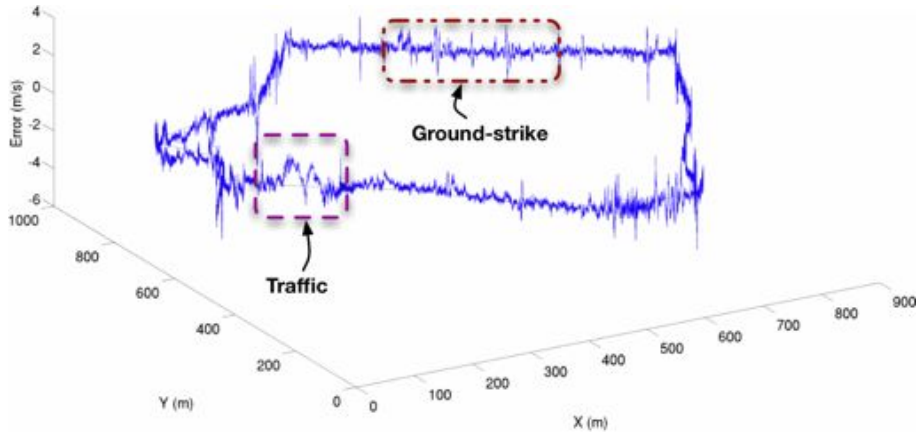
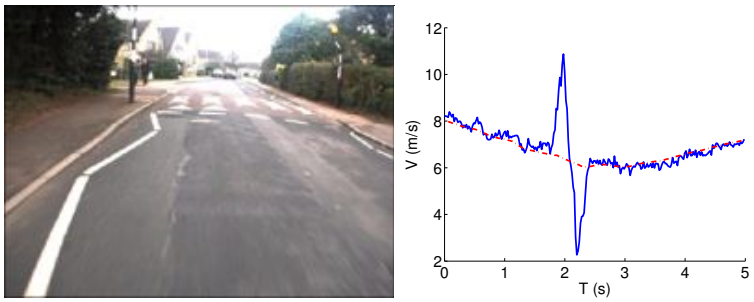



Figure 6.6: **Velocity error: Kidlington.** A similar plot to Figure 6.5 around Kidlington, Oxfordshire. Again we see the effect of ground strike and traffic on the velocity estimates, highlighted in the north-east and west, respectively.

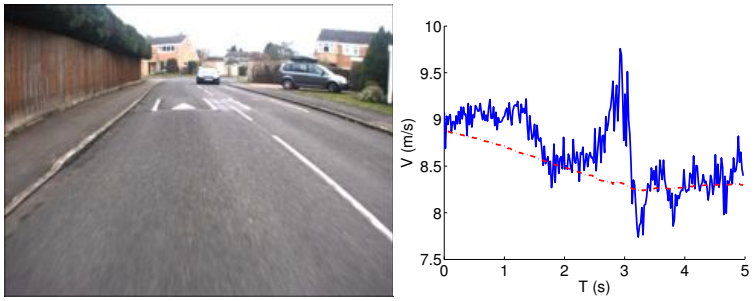
Visible in both Figure 6.5 and Figure 6.6 are a number of regions that cause the scan-matching algorithm to perform poorly. Typical reasons for errors are ground-strike arising from vehicle pitch/roll (Figure 6.7), relative errors from oncoming vehicles (Figure 6.8) and relative errors from cars in front (Figure 6.9). In all these figures, estimated velocity is in (solid) blue, and ground-truth velocity in (dashed)

red:



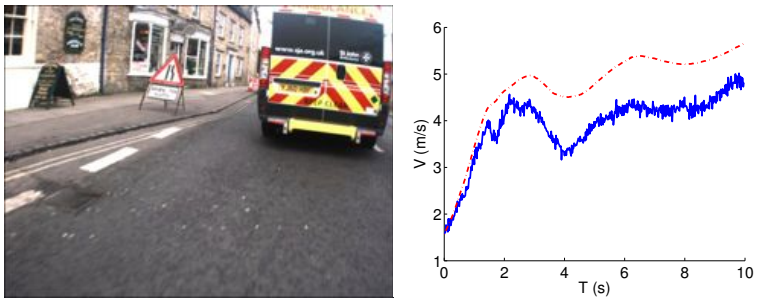

(a) Bumblebee2 image (b) Velocity estimate

Figure 6.7: **Velocity Type 1 errors.** Velocity errors arising from ground-strike after traversing a raised pedestrian crossing.




(a) Bumblebee2 image (b) Velocity estimate

Figure 6.8: **Velocity Type 2 errors.** Velocity errors arising from the relative velocity of oncoming vehicles.




(a) Bumblebee2 image (b) Velocity estimate

Figure 6.9: **Velocity Type 3 errors.** Velocity errors arising from the relative velocity of vehicles ahead.

Of all these errors, Figure 6.9 is particularly troublesome. For example, consider

the experiment depicted in Figure 6.10, which shows the actual velocity of the INS, and a biased version of the same signal:

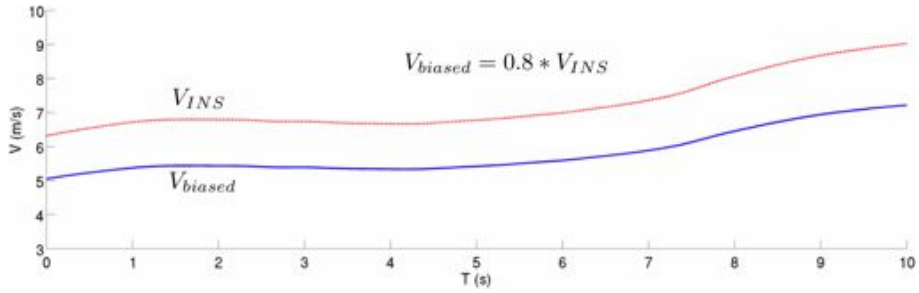
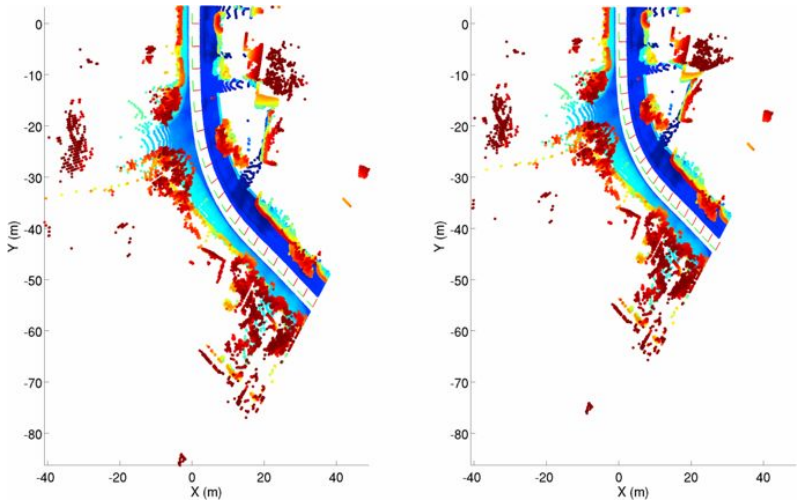



Figure 6.10: **INS velocity and bias.** The biased (blue, solid) vs the actual (red, dashed) velocities used to generate the corresponding point clouds in Figure 6.11. Velocity under-estimates leads to "compressed" point clouds - shown in Figure 6.11(b) - which degrade localisation performance.

These velocities were then used to generate point clouds from the same LIDAR data, as shown in Figure 6.11:




(a) Unbiased cloud (b) Biased cloud

Figure 6.11: **Comparison of biased vs. unbiased point clouds.** (a) Shows an overhead view of the point cloud from the downward-facing LIDAR (coloured by height above the ground, blue being lowest and red highest) using the INS velocity feed. (b) Shows the same LIDAR-data, but generated using the biased velocity estimate (shown in Figure 6.10). This "compression" leads to point clouds that are a poor representation of reality, resulting in localisation error. (In both cases, a centre-swathe of points has been omitted in order to render the poses ).

Visible in Figure 6.11(b) is the "compression" arising from the velocity under-estimate - this is very detrimental to the localisation process, as the statistics of the data have changed significantly with this warping resulting in a biased pose estimate, and ultimately tracking failure. A corresponding over-estimate in the velocity feed will yield the reverse problem - an expansion of the point cloud. While both type 1 and type 2 errors are predominantly short in duration, type 1 errors can exist over significant periods and are the main cause of localisation failure.

However, in all these pathological cases - (Figure 6.7 to Figure 6.9) - there is distinct *context*. Ground-strike tend to occur in regions with sharp gradient change, and errors arising from the relative motion of traffic tends to occur predominantly in the town centre. We leverage this property, and in the following sections develop a context-dependent sensor model that categorically improves the velocity estimates, resulting in robust localisation performance.

## 6.3 Context-dependent sensor models

The previous sections have highlighted the importance of *context* with respect to the errors in inferred velocity from the horizontal LIDAR. This leads us to believe that we require a spatially-varying sensor model that is capable of filtering out areas of the environment that are in motion relative to the vehicle (traffic, people, and ground-strike as it appears in ranging data). In the next section we exploit the use of vehicle context in order to generate a robust model for LIDAR scan filtering.

### 6.3.1 Sensor models as a function of location

To correct the aberrant behaviour shown in Figure 6.7 through Figure 6.9, we need to learn a filter that will allow us to remove points from scans that degrade the performance of the scan-matching algorithm. We seek a way of probabilistically

filtering points in scans that would be good match candidates, *given where we are in the world*. We do not *require* a model for every transient obstacle that we encounter - only a way of determining good vs. bad regions of incoming scans.

We therefore introduce the notion of a position-dependent sensor model. Consider a function $\mathbf{f}$ that maps an input value $\lambda$ to some output space:

$$\mathbf{f}(\lambda) \mapsto \kappa, \quad \lambda \in \mathbb{R}^m, \quad \kappa \in \mathbb{R}^n \tag{6.1}$$

Given that we are traversing a road network, a natural representation of this mapping is a cubic spline, which will map a floating-point value to a global Universal Transverse Mercator (UTM) $(x, y)$ position ($\mathbb{R} \mapsto \mathbb{R}^2$).

A cubic spline is a piecewise, third-order polynomial that interpolates a set of control points. For example, given a one-dimensional input interval over domain $\mathbf{x} \in \mathbb{R}^1$ between two bounds, $a$ and $b$, and its associated output $\mathbf{y}$:

$$\mathbf{x}_{ab} = \{x_0 < x_1 < \cdots < x_n\}$$
$$\mathbf{y}_{ab} = \{y_0, \cdots, y_n\}$$

an interpolating cubic-spline $\mathcal{S}$ [28] is defined to be a piecewise function that satisfies the following conditions:

1. $\mathcal{S}_i(x)$ is a cubic polynomial on the interval $[x_i, x_{i+1}]$ for $x_i \in \mathbf{x}$
2. $\mathcal{S}(x_i) = y_i$ for $x_i \in \mathbf{x}$
3. $\mathcal{S}(x)$, $\mathcal{S}'(x)$, $\mathcal{S}''(x)$ are continuous for $x \in \mathbf{x}$[1]

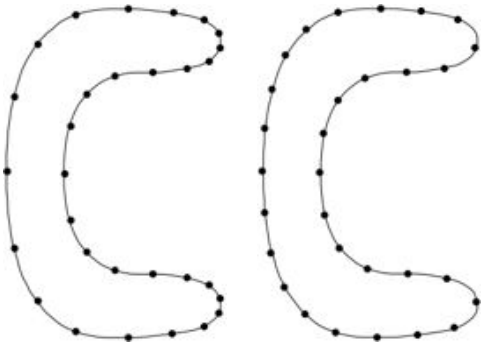
[1] This is known as $C2$ continuity

By setting the first and second derivatives to match at each of the control-points, and with the appropriate boundary conditions (in this case, we are using the "not-a-knot" boundary conditions, where $\mathcal{S}_0'''(x_1) = \mathcal{S}_1'''(x_1)$, and $\mathcal{S}_{n-2}'''(x_{n-1}) = \mathcal{S}_{n-1}'''(x_{n-1})$), it is possible to write the constraints on the system as a system of linear equations and solve for the knot coefficients.

This formulation requires a monotonically increasing $\mathbf{x}$, which is not the case with the trajectory in Figure 6.3. A common approach is then to construct two splines, indexed by a common monotonically increasing parameter, $\lambda$. So, for any trajectory $\mathcal{T} = \{x, y\}_{i=1}^{N}$:

$$\begin{aligned} \mathcal{S}_x &= \{[\lambda_0, \cdots, \lambda_n], [x_0, \cdots, x_n]\} \\ \mathcal{S}_y &= \{[\lambda_0, \cdots, \lambda_n], [y_0, \cdots, y_n]\} \end{aligned} \tag{6.2}$$

This representation leads to a very natural, smooth, parametric interpolating function that takes a single scalar input and maps to a position in $\mathbb{R}^2$ - an ideal *compact* representation of the trajectory. A consequence of this parametric representation is that the $\lambda$ parameter is not uniformly spaced (in terms of arc-length) along the spline, as is shown in Figure 6.12[71]:



(a) Parameter-space (b) Cartesian space

Figure 6.12: **Parametric spline representation.** A comparison of Cartesian position of a spline parameter, $\lambda$. Figure (a) shows equal parameter intervals, Figure (b) shows equal arc-length intervals. Producing equal arc-length intervals requires integrating the arc-distance along the spline. (Image courtesy of [71])

Partitioning the spline into equal arc-length segments requires integrating the function:

$$L = \int_{\lambda'}^{\lambda''} \left\| (\mathcal{S}'_x(\lambda), \mathcal{S}'_y(\lambda)) \right\| d\lambda \tag{6.3}$$

which can then be used to produce equidistant (as a function of arc-length) $\lambda$ values. It should be noted that this representation is only suited to road-networks - we would not employ the same technique for the general localisation problem. However, this spline-based approach is just one realization of this indexing method - instead of relying on a parameter indexing into the global location of the map, we could instead rely on visual cues from an on-board camera to index into a *topological* representation of the workspace. If the intuition is correct that the physical setting influences sensor characteristics, then the approach will be invariant under the indexing method used.

The control points for the spline model are obtained by generating equidistant INS poses in the reference data over the entire $2.6km$ route around Woodstock, as

shown in Figure 6.13:

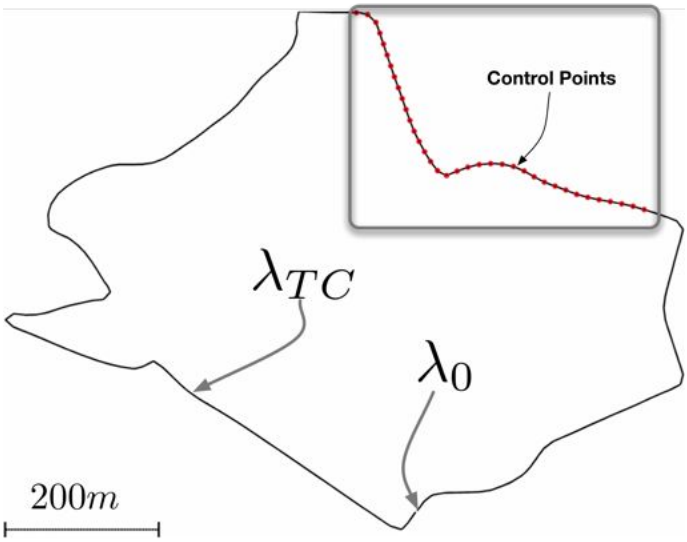



Figure 6.13: **Spline map of Woodstock.** A cubic spline representation around $2.7km$ of Woodstock, with the inset image highlighting the 10 metre-spaced control points. Also highlighted is the start parameter $\lambda_0$, and a spline value that corresponds to the town centre $\lambda_{TC}$.

Given this representation, it is now possible to index into the road network using a single floating-point value, with the location of the vehicle along this manifold path implicitly encoding information about the problematic sections of the LIDAR fan.

As an example of such a case, consider a typical pass through an area with heavy traffic, such as Woodstock town centre. To illustrate the problems arising from such a traversal, we can make use of the various on-board cameras and project points from the horizontal LIDAR into an image. Figure 6.14(b) shows an example LIDAR fan (and the associated image) at the $\lambda_{TC}$ value highlighted in Figure 6.13:

(a) (b)

Figure 6.14: **Woodstock town centre.** (a) An example LIDAR sweep at the $\lambda_{TC}$ position as viewed from an onboard camera, and (b) with the horizontal LIDAR points overlaid. This fan corresponds to the scene shown in Figure 6.9. The points falling on the vehicle in front (highlighted in red) must be filtered out to prevent the velocity error highlighted in Figure 6.9.

Highlighted in red in Figure 6.14(b) are the LIDAR beams falling on a vehicle ahead of the Wildcat. To see exactly why this situation leads to suppressed velocity estimates, we can make use of the accurate pose-estimates from the INS, and project consecutive sweeps from the horizontal LIDAR into a common frame to visualize the point overlay[2]. Figure 6.15 shows two such scans - separated by one second - from the horizontal LIDAR overlaid into a common frame using the INS pose estimates:

[2]In this fashion, we are relieved from having to estimate the pose transformation between scans using point correspondences

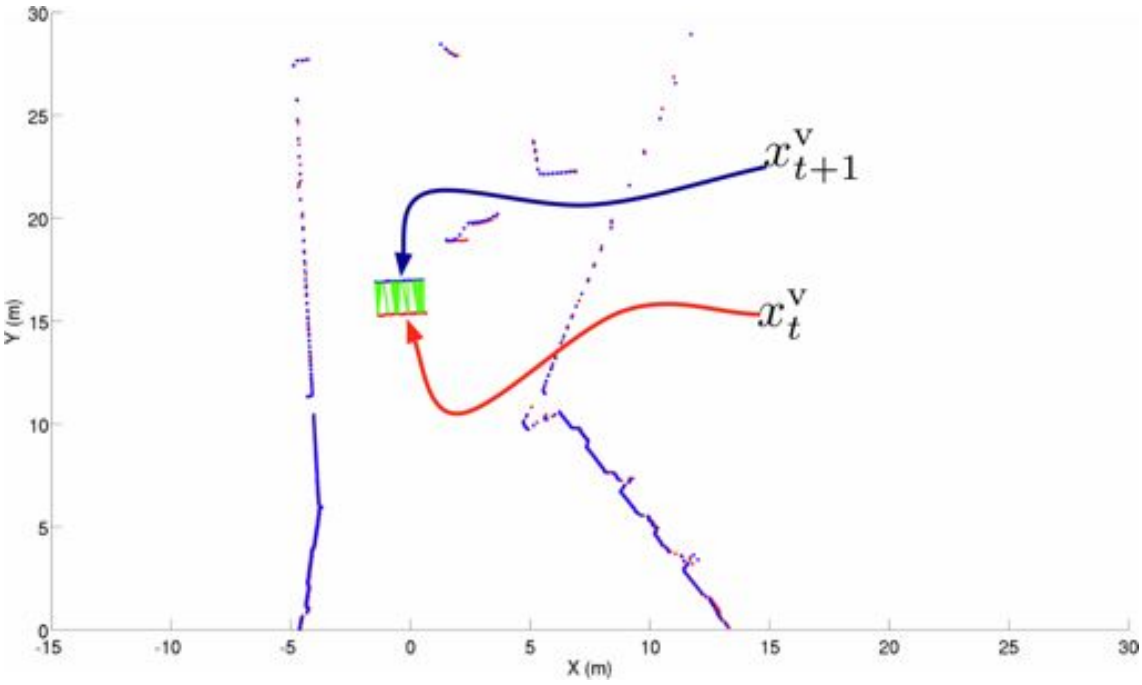



Figure 6.15: **Consecutive LIDAR sweeps in Woodstock town centre.** An overhead view of the scenario depicted in Figure 6.9. Points from one-second separated scans (red and blue, respectively) are overlaid into a global frame using the INS estimates, and correspondences for the van are highlighted in green. As the relative velocity between the van and Wildcat is less than the Wildcat's true velocity, the velocity estimates are correspondingly suppressed (as is visible in Figure 6.9).

As can be seen from the figure, the majority of the scene is static in the period between the two LIDAR scans, and this is apparent from the point overlap. However, the relative velocity of the van in front causes those LIDAR points impacting on it to shift substantially in the interval - this discrepancy then manifests itself in the underestimate of the true velocity, as shown in Figure 6.9.

One way of accounting for this situation is to develop a filter to remove those points from the horizontal LIDAR that are likely to result in inferred velocity error. In a probabilistic fashion, this filter should generate a probability - for every beam in the LIDAR - as to whether the measurement should be incorporated into the scan-matching engine, or filtered out. It should account for the case where *all* points in the LIDAR should be discarded (for example, consider a bustling intersection

with both vehicle and pedestrian traffic - almost all of the incoming data should be filtered).

In the following sections, we learn a probabilistic filter $p(\mathbf{X} \mid \mathbf{r}, \Theta, \Omega)$ where $\mathbf{X} = \{X_1, \cdots, X_N\}$ is a boolean-valued indicator variable corresponding to whether beam $i$ (with range $r_i$ and angle $\theta_i$) is a good candidate to incorporate into the scan-matching engine, and $\Omega$ corresponds to some model-specific parameters.

An important consideration here is the form of this filter - whether it is *stationary*, with constant parameters $\Omega$ that are invariant to any environmental stimuli - or *non-stationary*, with parameters - or form - that is intrinsically tied to some exogenous cues. It is posited that the form of the filter will be directly influenced by the spatial location - we expect very different filtering patterns on the highway, as opposed to a T-junction.

As validation of this concept, the next section shows results from a variety of classifiers (over range and angle data), and shows that taking into account context leads to better classification performance (and hence filtering, and better velocity estimates).

### 6.3.2 Stationary vs. Non-stationary models

The concept of contextual sensor models can be framed in terms of stationarity - we seek to learn a non-stationary sensor model - i.e., one whose intrinsic characteristics vary spatially. This approach is similar to the hierarchical mixtures-of-experts approach [45]; a divide-and-conquer approach to classification [99].

The core HME idea is to partition the input space by means of a *gating-function* into regions that are associated with a domain "expert". Although this method reduces the estimator bias, it tends to increase the variance. Bishop and Svensen [10] present a fully-Bayesian treatment of the hierarchical mixture model, alleviating the need for maximum-likelihood parameter estimation, which could lead to model

over-fitting.

Although the HME and Bayesian-HME are useful tools for determining the number of experts (and appropriate gating networks), we show that using a more simplistic hard-gating (constant arc-distance) function in this context works well.

To contrast the non-stationary and stationary approaches, we will consider a particularly testing section of Woodstock - the town centre. The two models will take a very different approach to classification - the *stationary* approach will try and learn one monolithic classifier for the length of Woodstock town centre. In contrast, the *non-stationary* model will consist of a bank of classifiers, each individually trained on discretized sections of the trajectory. A visual contrast of the different approaches is shown in Figure 6.16:

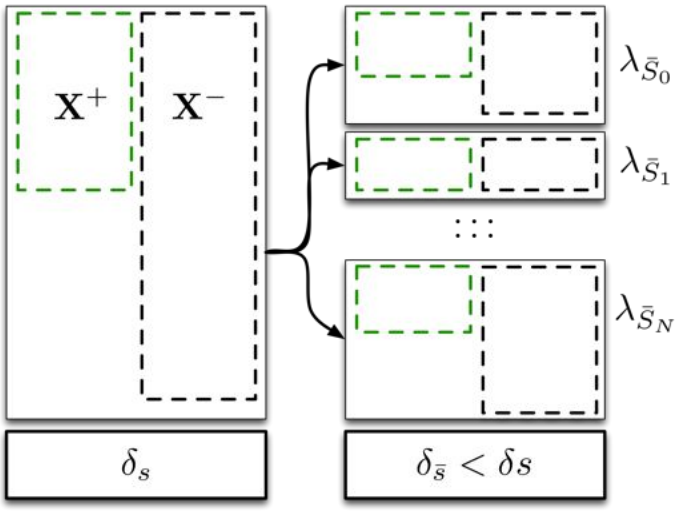



Figure 6.16: **An illustration of the classification approach.** A comparison of the stationary (left), and non-stationary (right) classification approaches. For an arbitrary classification task, we have a set of positive training instances ($\mathbf{X}^+$) and a set of negative instances ($\mathbf{X}^-$). We seek to segment this input data such that the classification error $\delta_{\bar{\mathcal{S}}}$ for the non-stationary approach is *lower* than that of the stationary model, $\delta_{\mathcal{S}}$.

If we consider one of the main sources of error to be vehicle traffic, we require a classification algorithm $\mathcal{A}$ which learns from a set of input data $\mathcal{L} = \{\mathbf{x}, y\}_{i=1}^{N}$ and produces a decision rule, $\hat{f}$. This decision rule maps data from the input domain

$x_i \in \mathcal{X}$ (consisting of range and angle data) to the output domain $y_i \in \mathcal{Y}, [1, -1]$ (corresponding to vehicle/not-vehicle), i.e. $\hat{f} : x \rightarrow y$. At run-time, we then classify all the incoming LIDAR points, and discard those that are likely to be from traffic.

The non-stationary approach will consist of a bank of location-dependent classifiers, with locations chosen uniformly over distance along a spline-representation of the town centre - these locations are shown in Figure 6.17:

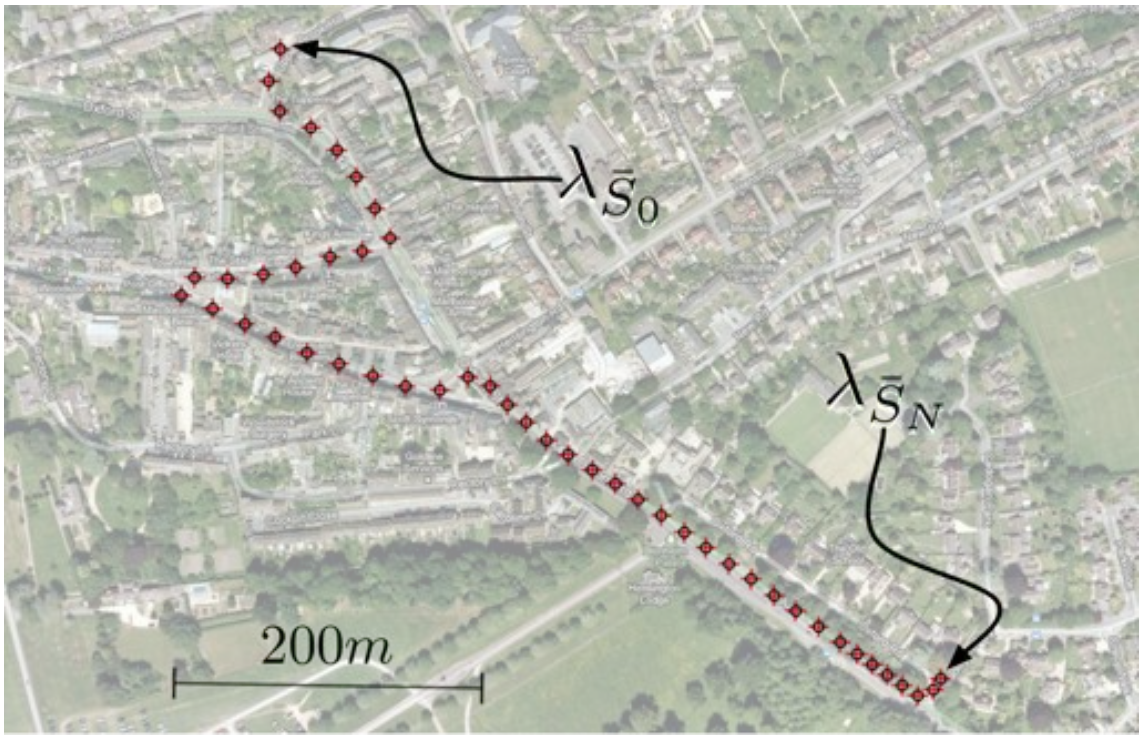



Figure 6.17: **Contextual filters.** Global positions of the [48] contextual filters over Woodstock town centre. Training data for these models was obtained by traversing this route in a north-south direction several times over the course of a 6 month period.

In Figure 6.17, $\lambda_{\hat{S}_0}$ is the first classifier of the non-stationary set and $\lambda_{\hat{S}_N}$ the last of $N$ classifiers (48 in the following experiments). It could be argued that the position of the filters should not be chosen solely on distance, but rather take into account other cues - change in orientation, for example. However, it will be shown that the uniform-distance representation is sufficiently descriptive to allow for long-term localisation in this environment. It should also be noted here that

this is primarily a pedagogical example - moving vehicles are not the only source of scan-matching error, but they provide an illustrative test case.

The following sections compare the results of three standard forms of classification, and show that the performance of the non-stationary representation is unequivocally better. For the following classification comparisons, training data spanning 6 months was obtained by hand-labelling 2731 scans in, and around, the Woodstock town centre. Again, input data $\mathbf{x}$ consists of range and angle data, and target data $y$ is the boolean variable corresponding to whether this point corresponds to a vehicle ($y = 1$), or not ($y = -1$).

### Naive Bayes

Naive Bayes is a simple classification scheme that applies Bayes theorem to classification problems under strict independence assumptions. Given input random variables $\mathbf{x} = \{x_1, \cdots x_N\}$, the posterior distribution according to Bayes rule over the classification output $y$ is:

$$p(y \mid x_1, \cdots, x_N) = \frac{p(y)\, P(x_1, \cdots, x_N \mid y)}{p(x_1, \cdots, x_N)} \tag{6.4}$$

Of course, obtaining an unbiased estimate of $P(\mathbf{x} \mid y)$ becomes intractable with increasing problem size. The Naive Bayes algorithm makes the conditional independence assumption:

$$p(x_i \mid y, x_j) = p(x_i \mid y) \;\; \forall (i, j) \tag{6.5}$$

This assumption of conditional independence is often violated - however, despite this, the performance of Naive Bayes is often comparable to more sophisticated

algorithms. Applying the independence assumption to Equation (6.4) gives:

$$p(y \mid \mathbf{x}) \propto p(y) \prod_{i=1}^{N} p(x_i \mid y) \tag{6.6}$$

We then use the *maximum a posteriori* estimate (MAP) for the most likely class as our learning function:

$$\hat{f} : \mathbf{x} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \; p(y \mid \mathbf{x}) \tag{6.7}$$

Estimating the conditional distributions $p(x_i \mid y)$ is done through kernel density estimation [33], where for a set of samples $N$ from a random variable $x_i$, the estimated density is:

$$\hat{p}(x) = \frac{1}{\mathcal{Z}} \sum_{j=1}^{N} K\left(\frac{x - x_i^j}{h}\right) \tag{6.8}$$

where $K(\cdot)$ is the kernel-function (in this case, a Gaussian) and $\mathcal{Z}$ is an explicit normalization term, ensuring that $\hat{p}(x)$ is a valid probability distribution. The estimation of the bandwidth-parameter, $h$, is the subject of much research - see [96] for an overview. Figure 6.18 highlights the effect of the this bandwidth parameter when estimating a multi-modal distribution:

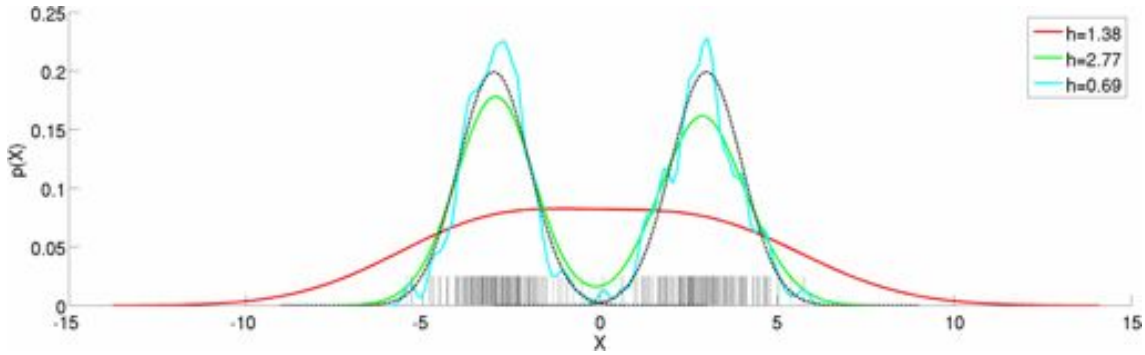



Figure 6.18: **Effect of the bandwidth parameter on density estimation.** Samples from a sum-of-Gaussians (SoG) distribution (black, dotted), with kernel density estimates with various bandwidth parameters $h$ shown. Bandwidth parameters that are too large miss the multiple modes (red), while small parameters begin to over-fit the data.

A solution to the estimation problem is to perform cross-validation on multiple datasets, and is the method employed here. The use of this non-parametric [3] estimator ensures that the model can capture the multiple-modality inherent in the data. Figure 6.19 and Figure 6.20 shows the probability distributions (both in the probability mass function (pmf), and kernel density estimates) over the indicator variables for the *stationary* case:

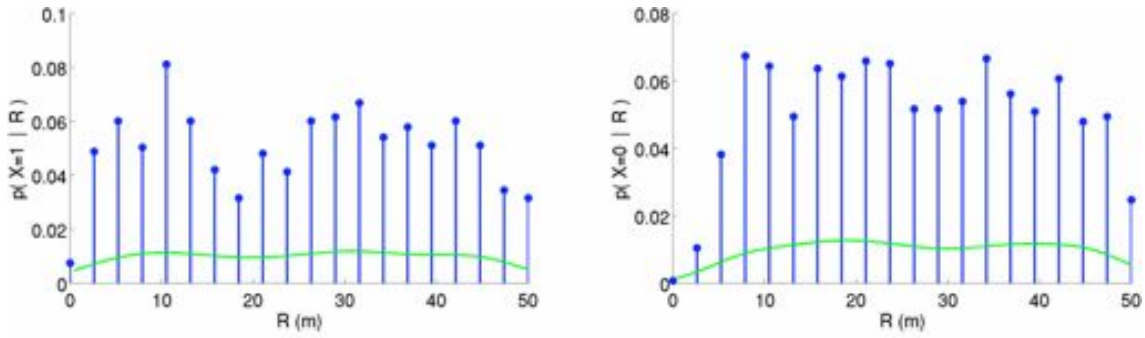



Figure 6.19: **Stationary model conditionals: range.** Conditional distributions for the stationary sensor model over range for both classes. Note how the range measurements are very uniformly distributed for both classes - this makes classification difficult.

[3] Non-parametric in this context does not mean that there are no parameters - rather, that the distribution model does not belong to any parametric family.

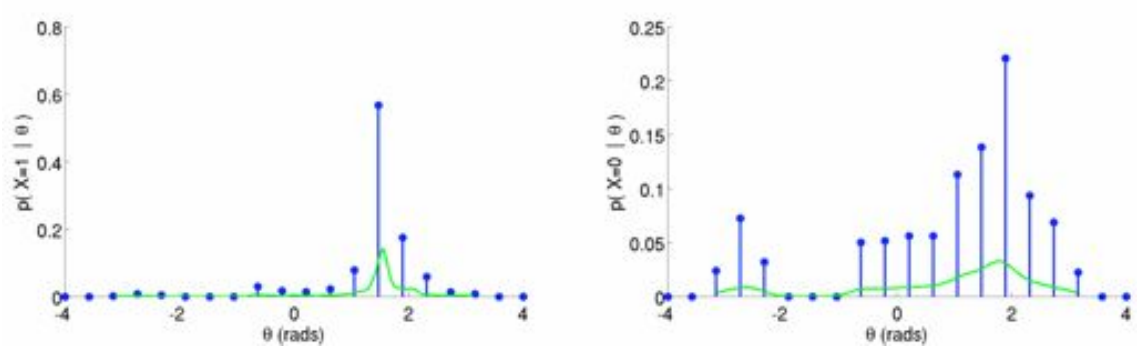



Figure 6.20: **Stationary model conditionals: angle.** A similar graph to Figure 6.19 over angle. For the data collected, vehicles tend to occur at $\theta \in (0, 2)$ rads, which corresponds to the front and right-hand side of the vehicle.

What is interesting to note is that - for the stationary approach - the occurrence of vehicles seems to be uniform over range. This is problematic for any classification scheme, as there is no obvious "decision boundary" - the positive and negative examples seem to fall uniformly over the input domain. This behaviour is contrasted against the conditional distributions from a single model in the non-stationary classifier bank:

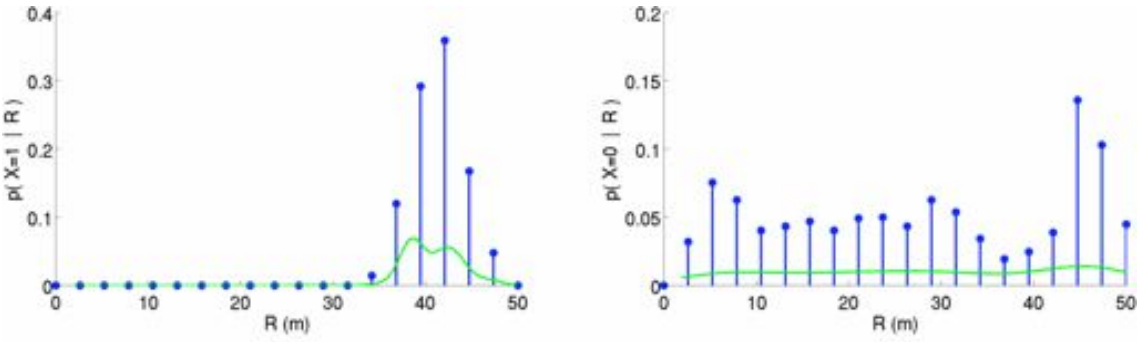



Figure 6.21: **Non-stationary model conditionals: range.** Conditional distributions over range for the non-stationary sensor model at a particular $\lambda$ value. Notice how the form of the distributions (particularly over range) has changed - this gives us an indication that conditioning on place will lead to better classification performance - compare this image to Figure 6.19.

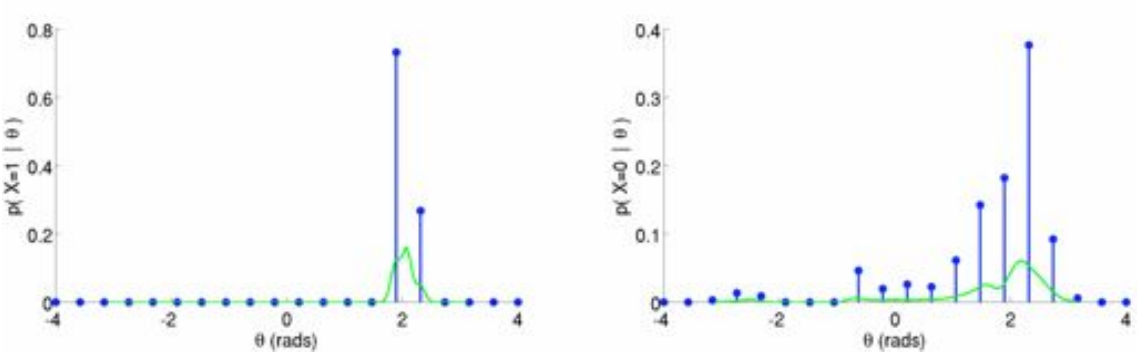
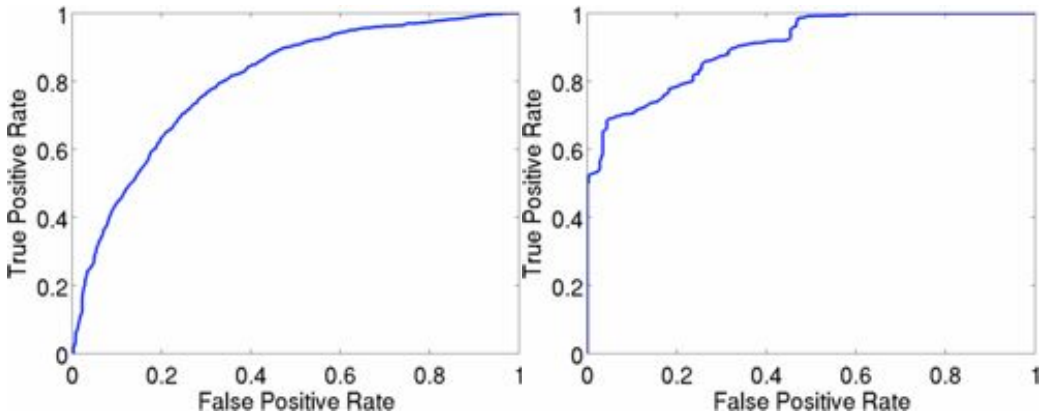



Figure 6.22: **Non-stationary model conditionals: angle.** The conditional distributions over angle for the same model shown in Figure 6.21. The form is very similar to the conditional distribution over angle from the stationary model (Figure 6.20), which implies that while traffic appears uniformly over distance, the same is not true for bearing.

It becomes clear from this data that we cannot - given that Naive-Bayes ignores correlations - expect the same performance as a more "discriminative" learning algorithm. Figure 6.23 shows an example of the Receiver Operating Characteristic (ROC) curve for the stationary and one instance of the non-stationary classifier bank. The ROC curve can be considered to be a *rate* curve, plotting the rate of false-positives against true positives for a classification threshold:




(a) Stationary Model (b) Non-stationary instance

Figure 6.23: **ROC curves.** The Receiver-Operating-Characteristic (ROC) curves for (a) the stationary-model, and (b) a single element of the non-stationary ensemble. A larger area under the ROC curve indicates a better classifier.

In a ROC curve, an ideal classifier would have a performance curve that produced 100% true-positives (TP) and 0% false-positives. As such performance is in practice unattainable, we look for a classifier that is said to "dominate" other classifiers. A classifier $\mathcal{A}_1$ dominates classifier $\mathcal{A}_2$ if:

$$\mathcal{A}_1 : fp > \mathcal{A}_2 : fp \quad \forall fp \in FP \tag{6.9}$$

where $\mathcal{A} : FP \rightarrow TP$ is a function that maps a False-Positive rate to a True Positive rate. This is one metric of comparison - another is to use the area-under-curve (AUC) metric, which does not impose such stringent criteria (these are well reviewed in [61]). In Figure 6.23, the non-stationary instance dominates the stationary model - however, it must be emphasised that the non-stationary model is only valid over the domain for which it is trained.

### Bagged Decision Trees

"Bagging" or *bootstrap aggregating*[16] is an **ensemble** mechanism for improving performance over a classification task. The underlying method is to create a bank of classifiers from training sets that have been "bootstrapped" from the original training data. The resultant classifier output is then averaged over all the models in the ensemble.

"Bootstrapping" [32] is a method for generating training sets from a limited data pool by sampling from the original training data *with* replacement - which will generate training sets in proportion to their representation in the input training data. The aggregated combination of bootstrapped learners tends to reduce the variance of the ensemble - a desirable characteristic.

The learner type is chosen to be decision trees - generalised as Classification

and Regression Trees (CART) in the statistical literature - which are simple rule based binary trees, consisting of intermediate and terminal nodes. Intermediate nodes represent decision boundaries for the input data, while terminal nodes assign a classification prediction. An example is shown in Figure 6.24:

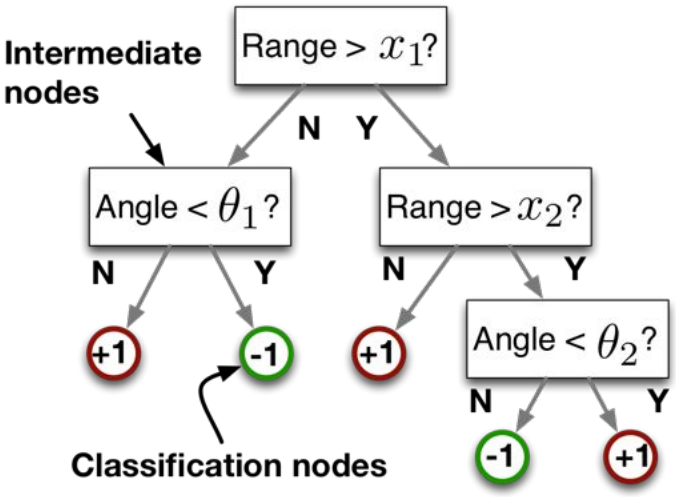
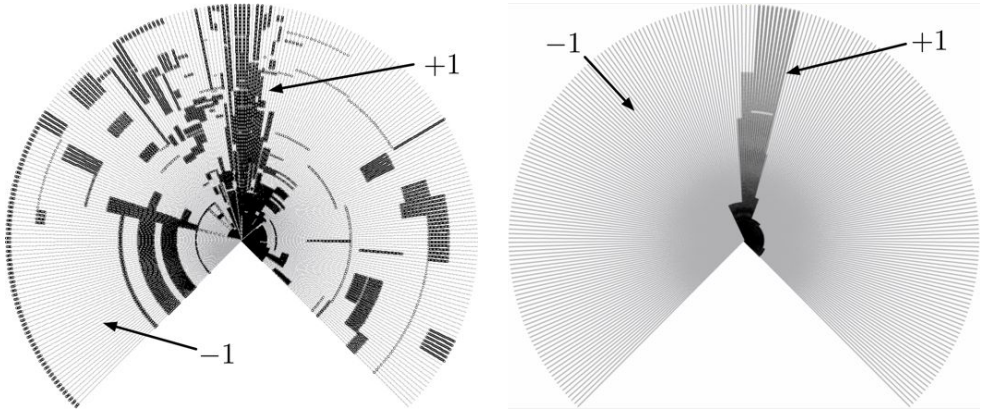



Figure 6.24: **Toy decision tree.** An example decision tree over range and angle data, classifying over two classes. Once trained (i.e. once the intermediate nodes have been built), classification of a new $\{r, \theta\}$ datum is a simple traversal of this tree.

Learning a decision tree corresponds to building the branching rules for the intermediate nodes - one of the most commonly used algorithms is ID3 [73] which uses information gain to determine which attribute - and the corresponding value of the attribute - to assign to a split. An explicit enumeration over the input space for both the stationary and a single element of the non-stationary classification ensemble is shown in Figure 6.25:




(a) Stationary (b) Non-stationary element

Figure 6.25: **CART decision boundary.** (a) The stationary-model decision boundary, and (b) one (of $N$) example non-stationary decision boundaries. The stationary model has to explain the variation across the entire region, whereas the non-stationary element is only responsible for its domain.

Classification trees can exhibit high variance - the algorithm can perform very differently for different training sets, and therefore are a good subject for bagging - an illustration of the bias/variance relationship is shown in Figure 6.26:

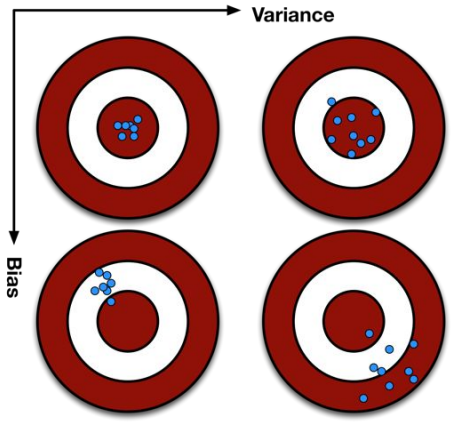



Figure 6.26: **Bias-variance illustration.** A pictorial representation of bias and variance. Classification trees tend to suffer from high variance, as the classification boundary is data-dependent. Bagging and other ensemble training methods reduce this effect.

The bagging algorithm for an arbitrary learner is given in Algorithm 5. Input is the set of labelled training data, the learner class (in this case, a decision tree), and

the desired number of ensemble instances:

**Algorithm 5** Bagging Algorithm

1: **procedure** LEARNENSEMBLE($\{\mathbf{x}, y\}_{i=1}^{N}, \mathcal{A}, B$)
2: $\hat{\mathbf{f}} = []$
3: **for** $b = 1$ to $B$ **do**
4: $\{\mathbf{x}^\star, y^\star\}_{i=1}^{M} \leftarrow$ SampleMWithReplacement($\{\mathbf{x}, y\}$) ▷ $M \leq N$
5: $\hat{\mathbf{f}}_b \leftarrow \mathcal{A} : \{\mathbf{x}^\star, y^\star\}$
6: **end for**
7: $\hat{f} = \frac{1}{B} \sum_{i=1}^{B} \hat{\mathbf{f}}_i$ ▷ Bagged estimator
8: **end procedure**

The bagging algorithm condenses to $\hat{f} : x = \mathbb{E}[\hat{\mathbf{f}} : x]$. Bagging, as an ensemble algorithm, tends to reduce the variance (at a cost of *always* increasing bias [18]) and is most useful for algorithm classes that exhibit high variability as a function of the input data (such as decision trees).

### AdaBoost

In bagging, generating the learning ensemble is left to chance (through the bootstrapping procedure). The alternative is to train the ensemble *sequentially*, at each stage attempting to reduce the training error. This is known as adaptive-boosting [79], and is highlighted in Algorithm 6.

Boosting makes use of an aggregating combination of so-called *weak-learners* - a learner that can be only marginally better than random in predicting the output of a class - into a resulting strong classifier that provides good classification performance.

In Line 2 a uniform prior weighting is placed over the training instances. The algorithm then iterates, for a number of *rounds*, building weak learners over the samples. The hypothesis error is then calculated for every input point, and the samples re-weighted. $\mathbb{I}$ is the indicator function, and $\mathcal{Z}_t$ is a normalization constant. Again we can see that the final classifier output is a combination of the ensemble

**Algorithm 6** AdaptiveBoosting

**procedure** AdaptiveBoosting( $\{\mathbf{x}, y\}_{i=1}^{N}, T$)
  $D_1 = \frac{1}{N}$ ▷ Initialise weighting
  $\hat{\mathbf{f}} = []$
  **for** $t = 1$ to $T$ **do**
    $\hat{\mathbf{f}}_t \leftarrow \mathcal{A} : D_t, \{\mathbf{x}, y\}$ ▷ Train weak learner
    $\mathcal{H}_t \leftarrow \hat{\mathbf{f}}_t : \mathbf{x}$ ▷ Get classifier hypothesis
    $\epsilon_t \rightarrow \sum_{n=1}^{N} D_t(i) \ \mathbb{I}_{\mathcal{H}_t(x_i) \neq y_i}$ ▷ Hypothesis error
    $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
    $D_{t+1}(i) = \frac{1}{\hat{Z}_t} D_t(i) \exp^{-\alpha_t y_i \hat{f}_t(x_i)}$ ▷ Update
  **end for**
  $\hat{f} = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \mathcal{H}_t\right)$ ▷ Final Hypothesis
**end procedure**

hypotheses.

AdaBoost can use any input of weak classifier, provided the classifier can do better than random in predicting the correct output. There are many other boosting variants - Gentle, Logit, Real, and so on - these are well reviewed in [25]. For this experiment, decision trees were again used as the weak learner.

### 6.3.3 Model classification performance

The evaluation method for the stationary/non-stationary comparison proceeds in an accumulative fashion, where the non-stationary models are built up sequentially along the route $\lambda_{\hat{\mathcal{S}}_0} : \lambda_{\hat{\mathcal{S}}_N}$, and compared to the stationary model trained over the *accumulated* data. This is outlined in Algorithm 7:

Algorithm 7 takes an aggregated set of training instances, a learning algorithm type, and returns the associated stationary and non-stationary models. Once we have these two models for the three algorithms mentioned, we can evaluate and compare the classification performance of the different strategies. Figure 6.27 shows the ROC comparison for both the non-stationary and stationary models using the

**Algorithm 7** Learning Strategy Comparison

**procedure** CompareLearningStrategies( $\{\mathcal{L}\}_{j=1}^{M}, \mathcal{A}$)
  **for** $j = 1$ to $M$ **do**
    $\mathcal{M}_{\mathcal{S}} \leftarrow \mathcal{A} : \{\{\mathbf{x}_i, y_i\}_{i=1}^{N}\}_1^j$ ▷ Learn stationary model
    $\mathcal{M}_{\bar{\mathcal{S}}} = []$
    **for** $k = 1$ to $j$ **do**
      $\mathcal{M}_{\bar{\mathcal{S}}_k} \leftarrow \mathcal{A} : \{\{\mathbf{x}_i, y_i\}_{i=1}^{N}\}_k$ ▷ Learn non-stationary model
    **end for**
  **end for**
  return $\mathcal{M}_{\mathcal{S}}, \mathcal{M}_{\bar{\mathcal{S}}}$
**end procedure**

Naive-Bayes classifier:

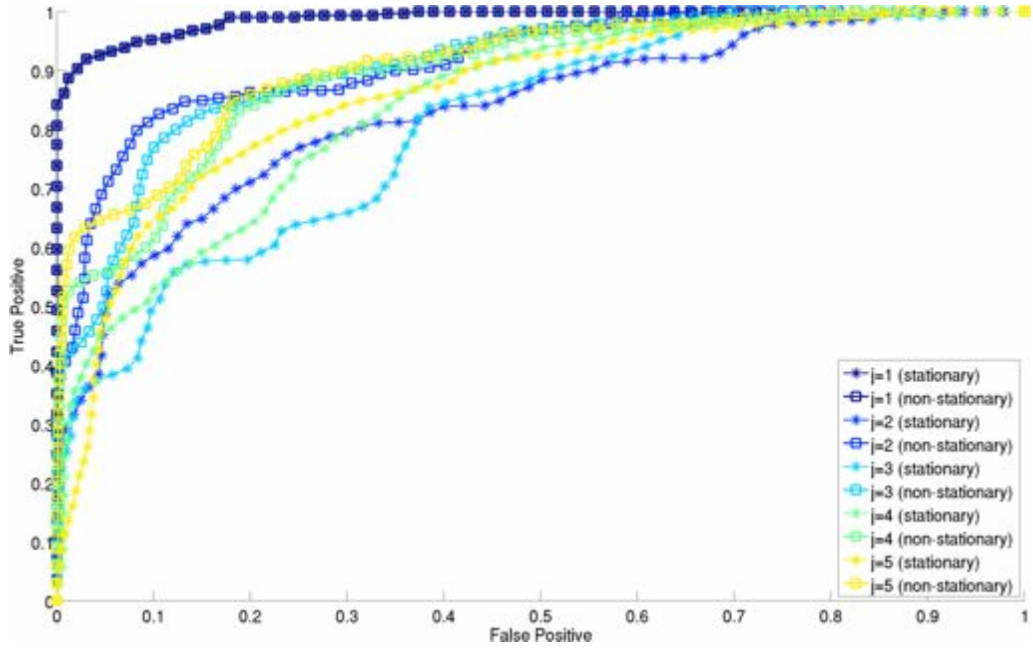



Figure 6.27: **Naive-Bayes stationary vs. non-stationary ROC.** Naive-Bayes ROC curves for different numbers of training instances - $j$ corresponds to the index of one of the 48 model positions shown in Figure 6.17. As we increase the amount of data taken from different places in Figure 6.17, the performance of the stationary model begins to degrade - although this is also true for the non-stationary approach, the degradation is much less rapid.

Figure 6.27 plots the performance of the two classification approaches for increasing context number ($j$ in the above figure) along the route. When $j = 1$, the performance of the two models are identical - this is to be expected, as they both

take on exactly the same form using the same training data. However, as $j$ increases, the performance of the non-stationary approach begins to dominate the stationary approach. This makes sense intuitively, as the non-stationary model has to explain more variation across the route, whereas each stationary model only has to model the data within its domain (as is explicitly illustrated in Figure 6.25).

Table 6.1 shows the Area-Under-Curve (AUC) metric for both the stationary and non-stationary approaches for increasing model number. As can be seen from the table, the non-stationary approach using Naive Bayes *dominates* the stationary approach for all model values[4]:

Table 6.1: AUC Comparison: Naive Bayes

| | $j = 2$ | $j = 2$ | $j = 3$ | $j = 4$ | $j = 5$ |
|---|---|---|---|---|---|
| AUC : Stationary | 0.9863 | 0.9084 | 0.8638 | 0.8560 | 0.8401 |
| AUC : Non-Stationary | 0.9863 | 0.9734 | 0.9771 | 0.9654 | 0.9609 |

Figure 6.28 replicates the same experiment, using the same data, employing bagged decision trees as the classification mechanism:

[4]A perfect classifier has an AUC of 1

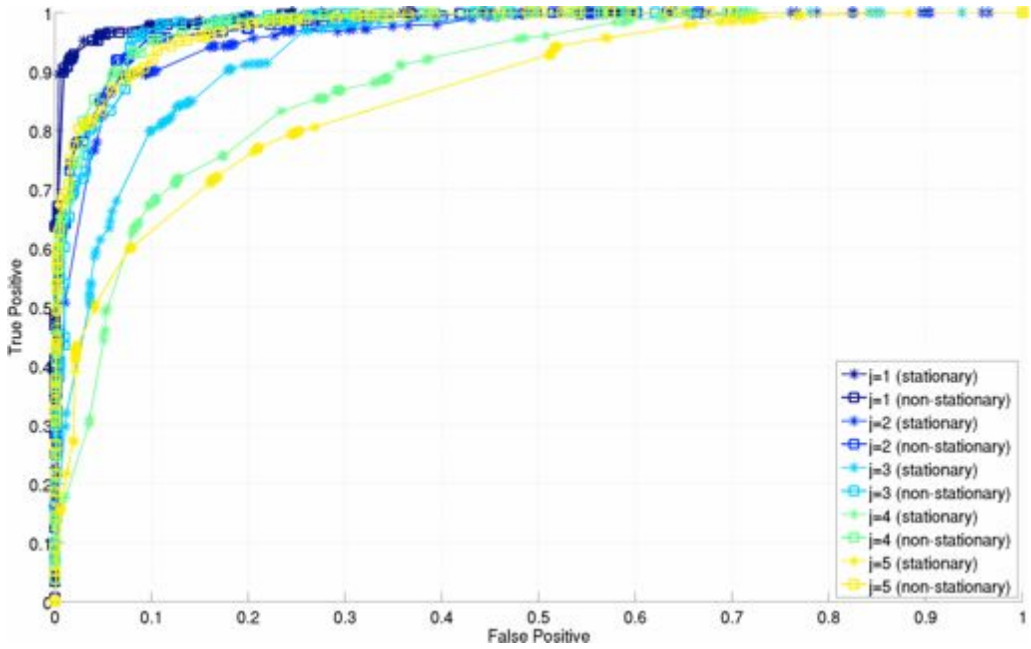



Figure 6.28: **Bagged Decision Trees stationary vs. non-stationary ROC.** Bagged decision tree ROC curves, for the same training instances shown in Figure 6.27. Decision-trees are a more discriminative classifier than Naive-Bayes, and this is reflected in the performance curves - although the non-stationary approach still dominates the stationary approach for all $\mathcal{M}$.

Again the performance of the non-stationary ensemble dominates that of the stationary model, as is made explicit in the AUC comparison in Table 6.2.

Table 6.2: AUC Comparison: Bagged Decision Trees

| | $j=2$ | $j=2$ | $j=3$ | $j=4$ | $j=5$ |
|---|---|---|---|---|---|
| AUC : Stationary | 0.9928 | 0.9613 | 0.9346 | 0.8803 | 0.8640 |
| AUC : Non-Stationary | 0.9915 | 0.9766 | 0.9758 | 0.9789 | 0.9757 |

Given that decision-trees are a *discriminative* classifier - i.e. it does not attempt to model statistics about the observations - both the non-stationary and stationary approaches exhibit better classification performance than the Naive Bayes approach. However, the non-stationary model still outperforms the stationary version, for the same reason as Naive Bayes - each model is only responsible for its domain in which the space the data inhabits is more regular than when taken as a whole.

As final validation, Figure 6.29 shows the performance for AdaBoost over the same training data with the corresponding AUC comparison in Table 6.3:

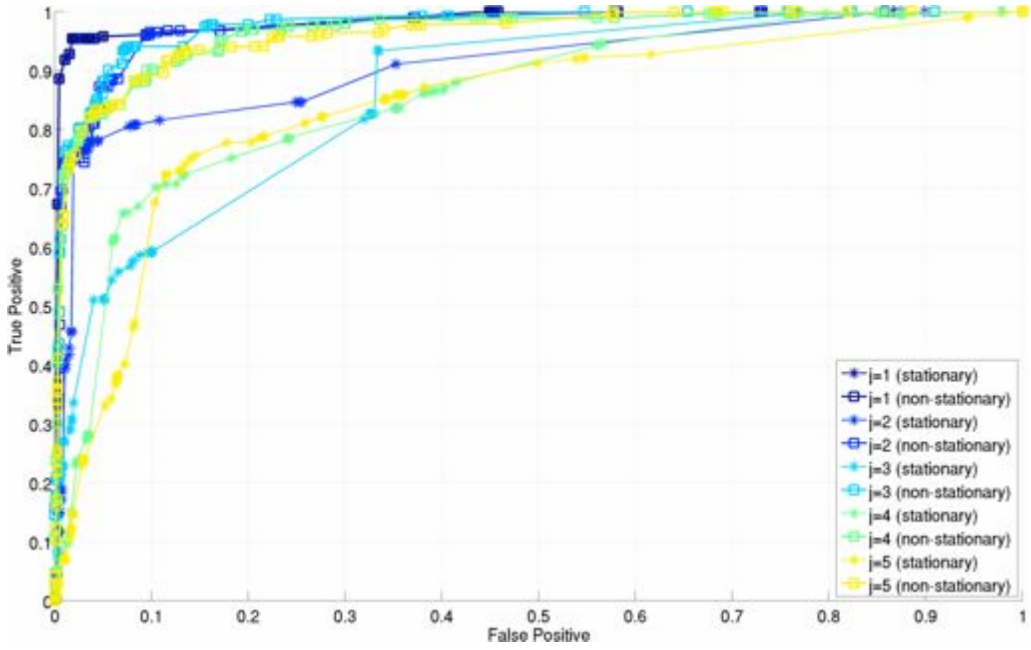



Figure 6.29: **Boosted Decision Trees stationary vs. non-stationary ROC.** Boosted decision trees (using the AdaBoostM1 algorithm) ROC curves, for the same training instances shown in Figure 6.27 and Figure 6.28. The boosted trees again outperform NaiveBayes, and are comparable to the bagged approach, but again show that the non-stationary model is superior to the stationary model.

Table 6.3: AUC Comparison: Boosted Decision Trees

| | $j = 2$ | $j = 2$ | $j = 3$ | $j = 4$ | $j = 5$ |
|---|---|---|---|---|---|
| AUC : Stationary | 0.9968 | 0.9778 | 0.9459 | 0.9019 | 0.8890 |
| AUC : Non-Stationary | 0.9968 | 0.9941 | 0.9891 | 0.9807 | 0.9846 |

Again, the performance of the non-stationary approach dominates that of the stationary model. In all cases, we can see that learning a *contextual*, non-stationary ensemble results in better classification performance for the vehicle classification task. We can say, conclusively based on the results shown, that learning a bank of classifiers - as a function of location, which implicitly captures context - is a method that will give better classification performance for our task (removing vehicles from

the LIDAR fan) than an equivalent stationary approach.

Figure 6.30 is an interesting depiction of the decision boundaries for each of the non-stationary models trained over the region highlighted in Figure 6.17:

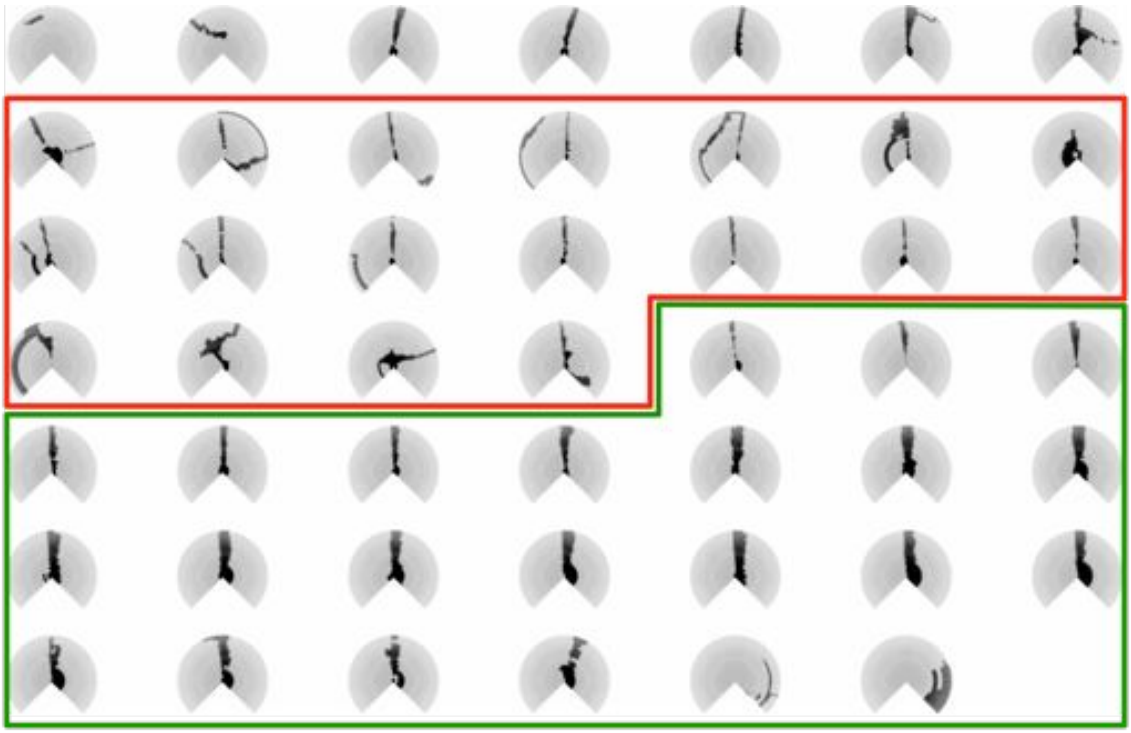


Figure 6.30: **Learned models.** The decision boundaries for the bagged decision-tree models for the locations depicted in Figure 6.17. Noticeable is the contrast between models learned in areas with a large dynamic component (highlighted in red are the models trained in the town centre, showing the variegated decision boundary shapes), and models learned where the traffic flow is more consistent (the green-highlighted models show models mainly on the main road).

The caveat with this approach, is that any individual classifier is only valid within its domain - i.e. we have access to the indicator function:

$$\hat{f}(x) = \mathbb{I}_{x \in \Omega(\hat{f})} \hat{\mathbf{f}}(x) \quad \text{for} \hat{f} \in \hat{\mathbf{f}} \tag{6.10}$$

where $\Omega(\hat{f})$ denotes the domain of the classifier $\hat{f}$. The approach is therefore predicated on the indicator function assigning the input data to the correct model.

However, given that we are focused exclusively on a *tracking* problem and we expect to know - with some confidence - our current position within the map, this is not a limiting factor.

### 6.3.4 Filtered vs. Unfiltered velocity comparisons

Training data for the task was obtained from hand-labelling scans from 3 datasets (spanning 6 months), and then testing on 3 datasets (totalling 10 km) over the next two weeks. Figure 6.31 through Figure 6.34 show representative images from problematic locations ( i.e. sections where, if the velocity is unfiltered, tracking fails) over duration of these test datasets:

(a) 2012-07-12 (b) 2012-07-17 (c) 2012-07-24

Figure 6.31: **Woodstock entry point.** Images from the Woodstock "entry" point ($\lambda_{\mathcal{S}_0}$) over 12 days. Figure (a) is from a Point-Grey Firefly roof-mounted camera, (b) and (c) from a bumper-mounted Point-Grey Bumblebee camera. Visible in all these figures are traffic that leads to errors in the scan-matching velocity estimates - these must be filtered in order to prevent tracking failure.

(a) 2012-07-12 (b) 2012-07-17 (c) 2012-07-24

Figure 6.32: **Woodstock town centre.** An image sequence similar to Figure 6.31, but for the town centre. Again, note the large trucks in (a) and (c) - both sources of large velocity error. This compounded velocity error causes point cloud compression, which subsequently results in tracking failure.

(a) 2012-07-12 (b) 2012-07-17 (c) 2012-07-24

Figure 6.33: **Woodstock T-junction.** Scenes from in and around a busy T-junction in the centre of Woodstock. Traffic joining the main road (a), large vehicles on the main road (b) and vehicles leaving Woodstock (c) all contribute significantly to velocity error.

(a) 2012-07-12 (b) 2012-07-17 (c) 2012-07-24

Figure 6.34: **Woodstock exit.** The final exemplar scene - the busy main road leaving Woodstock. The regular presence of buses (a), garbage-trucks (b) and other vans (c), all *highly* visible from the bumper-mounted LIDAR induce a large velocity error.

Given that there is substantial similarity - with respect to traffic patterns - across all of the datasets at each of the scenes (in Figure 6.31 through Figure 6.34, we expect that a contextual scan-filter would be able to greatly reduce the effect of transient objects in the LIDAR fan. Figure 6.35 through Figure 6.37 show the raw ICS-based scan-matching velocity estimates from the horizontal LIDAR for each of the scenes over all of the datasets. In almost all cases, the effect of traffic on the scan-matching velocity estimates are pronounced:

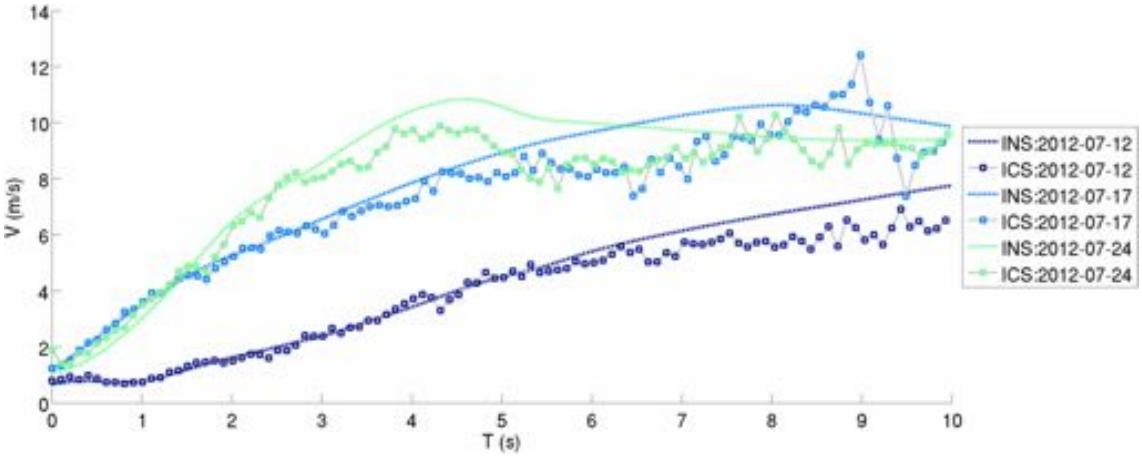



Figure 6.35: **Velocity estimates: INS vs. ICS estimates (Woodstock entry).** A comparison of the velocity estimates (both INS (solid), and ICS (square markers)) for each of the three datasets at the Woodstock entry point. What is apparent is that - due to traffic - the ICS velocities are consistently under-estimated in each of the datasets.

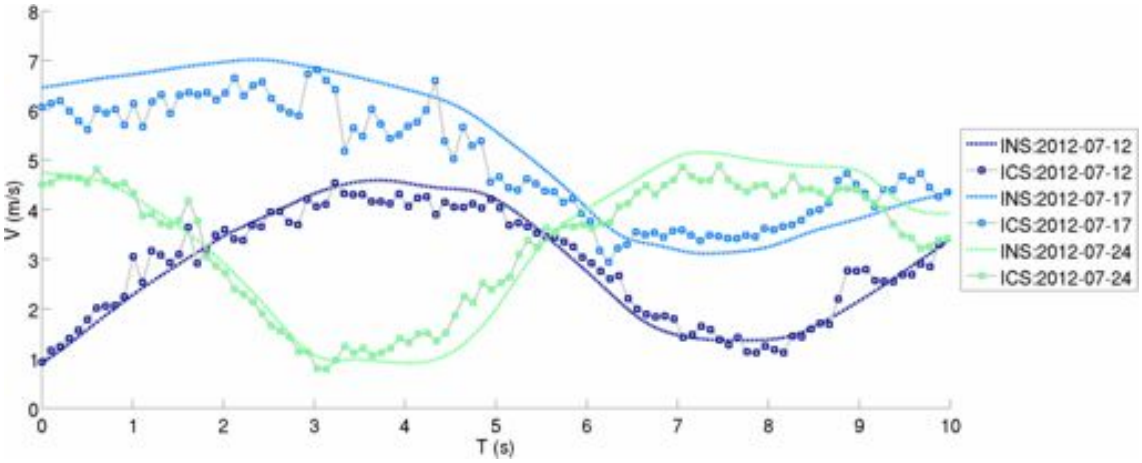



Figure 6.36: **Velocity estimates: INS vs. ICS estimates (Woodstock centre).** Velocity estimates for the same datasets as in Figure 6.35 in the centre of Woodstock, exhibiting similar behaviour. In some cases, on the $12^{th}$ of July for example, the ICS estimates are not unduly biased, due to the lack of traffic at this particular time.

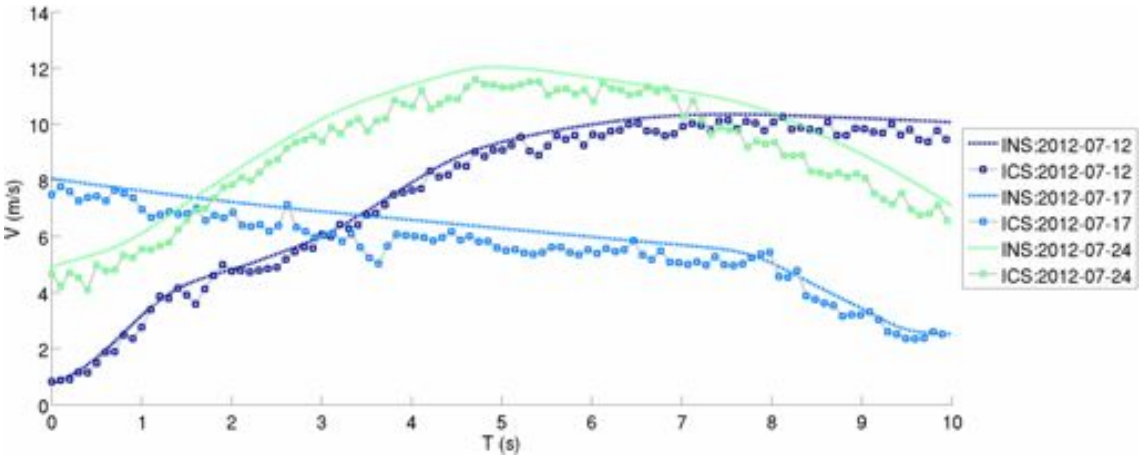



Figure 6.37: **Velocity estimates: INS vs. ICS estimates (Woodstock mainroad).** Velocity estimates for the main road, leaving Woodstock. Again apparent in each of the datasets is the bias inherent in the ICS velocity feed due to traffic on the road. This effect is persistent enough to justify the effort to develop a scan filter that is able to correct this bias.

The persistent bias in the velocity estimates across the datasets causes compression (or expansion) of the point cloud (as illustrated in Figure 6.10). Given that the non-stationary model provides a method of removing - from the LIDAR feed - points that are likely to induce velocity errors, it is possible to compare the performance of

the filtered/unfiltered approaches over the same data, which is shown in Figure 6.38:

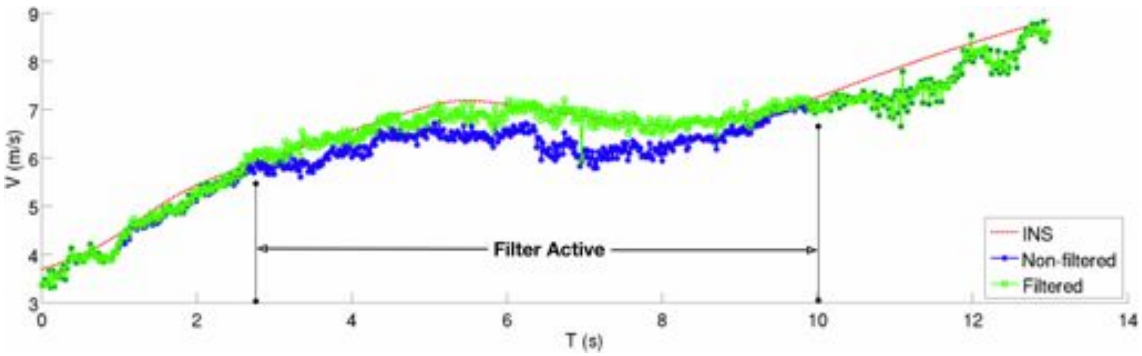



Figure 6.38: **Filtered velocity estimates.** The corresponding *filtered* velocity estimates (shown in green) against the unfiltered estimates (blue) over the scene shown in Figure 6.35, as compared to the INS estimate (red).

For the period that the filter is *active* - i.e. the data falls into the domain of one of the classifiers of the classifier bank, as determined by the indicator function - the velocity estimates have been improved substantially.

Figure 6.39 contrasts the performance of each of the non-stationary classification approaches for the problematic scene highlighted in Figure 6.35:

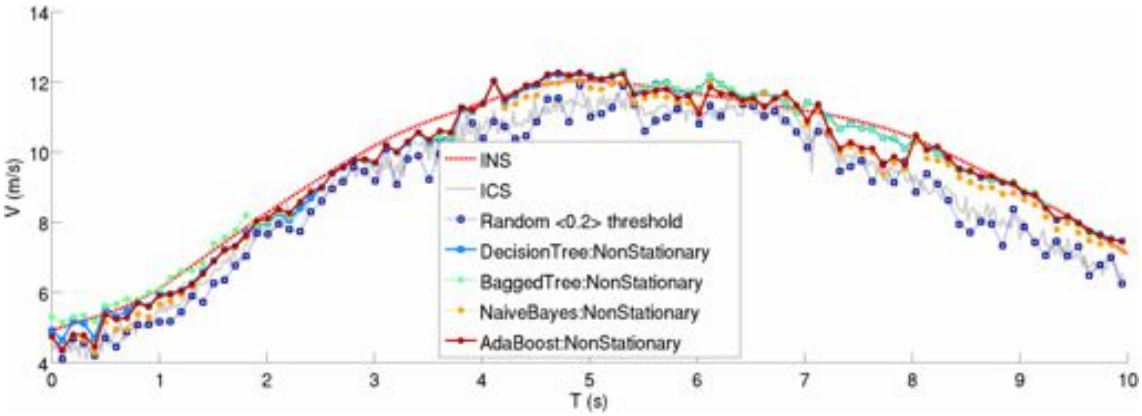



Figure 6.39: **Filtered velocity estimates (Woodstock entry).** A comparison of each of the classification strategies (NaiveBayes, Boosting and Bagging) for the scene shown in Figure 6.31(a). The ICS velocity estimates, due to traffic, underestimate the true velocity. However, filtering out points that are likely to be traffic leads to better velocity estimates for each of the classification schemes.

Figure 6.39 shows a comparison of the root mean square errors (RMSE) for each

of the filtered velocity estimates shown in Figure 6.39. The RMSE is evaluated by comparing the filtered velocity estimates, from each of the classifiers, against that of the INS:

Table 6.4: RMSE Comparison: Woodstock entry

| | ICS | Random (.2) | Tree | BaggedTrees | NaiveBayes | AdaBoost |
|---|---|---|---|---|---|---|
| 2012-07-12 (m/s) | 0.40 | 0.50 | 0.33 | 0.30 | 0.34 | 0.34 |
| 2012-07-17 (m/s) | 0.56 | 0.69 | 0.15 | 0.15 | 0.45 | 0.42 |
| 2012-07-24 (m/s) | 0.81 | 0.99 | 0.32 | 0.26 | 0.45 | 0.37 |

What is clear from Table 6.4 is that - for all of the datasets corresponding to this particular location - the filtered approaches uniformly outperform the raw ICS estimates, as measured by the RMSE. This confirms the hypothesis that by filtering out points in the LIDAR fan that are statistically problematic, we can extract better velocity estimates from the scan-matching algorithm. Figure 6.40 and Figure 6.41 replicate this comparison for each of the remaining scenes:

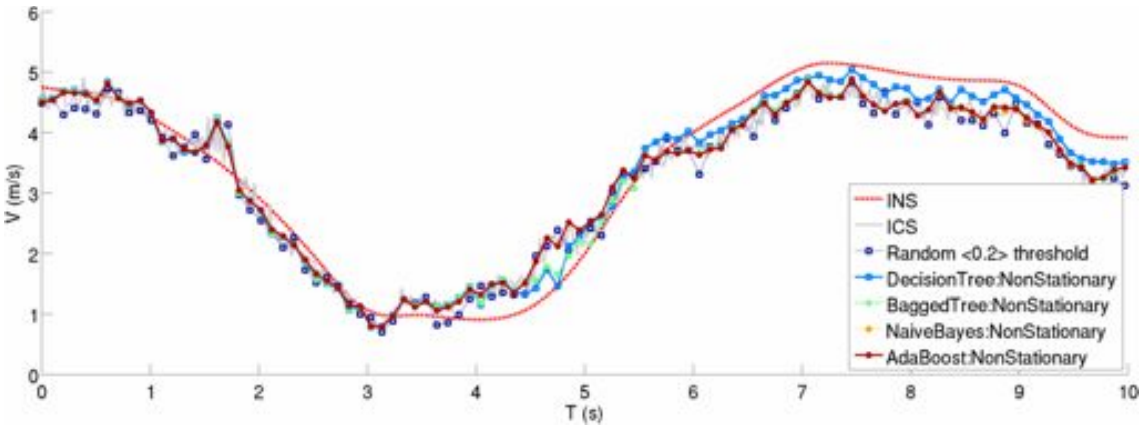



Figure 6.40: **Filtered velocity estimates (Woodstock centre).** Filtered velocity estimates using the various classification techniques over the Woodstock town centre. Again apparent is the improved velocity estimate from the contextually-filtered estimator. RMSE comparisons are shown in Table 6.5.

Table 6.5: RMSE Comparison: Woodstock centre

| | ICS | Random (.2) | Tree | BaggedTrees | NaiveBayes | AdaBoost |
|---|---|---|---|---|---|---|
| 2012-07-12 (m/s) | 0.29 | 0.33 | 0.40 | 0.40 | 0.34 | 0.27 |
| 2012-07-17 (m/s) | 0.60 | 0.71 | 0.39 | 0.40 | 0.63 | 0.70 |
| 2012-07-24 (m/s) | 0.43 | 0.47 | 0.31 | 0.40 | 0.43 | 0.43 |

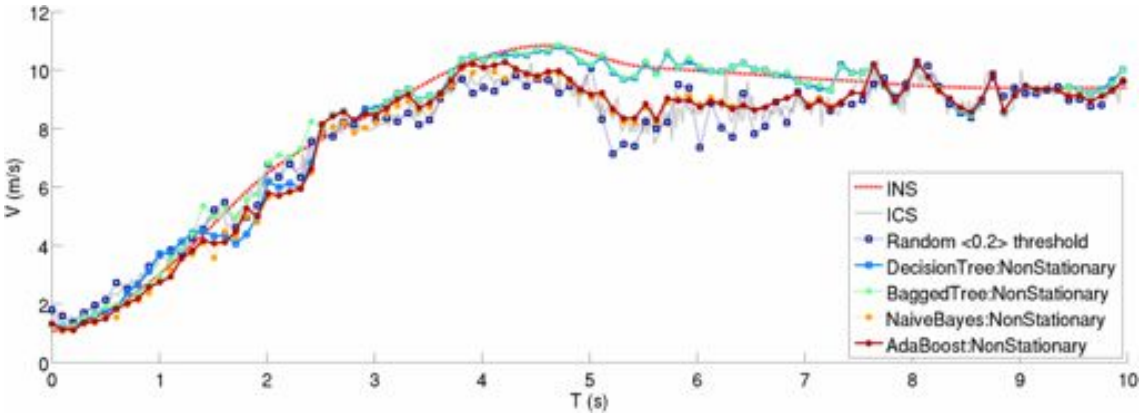



Figure 6.41: **Filtered velocity estimates (Woodstock T-junction).** Filtered velocity estimates using the various classification techniques over the Woodstock T-junction. RMSE comparisons are shown in Table 6.6.

Table 6.6: RMSE Comparison: Woodstock T-junction

| Method | ICS | Random (.2) | Tree | BaggedTrees | NaiveBayes | AdaBoost |
|---|---|---|---|---|---|---|
| 2012-07-12 (m/s) | 0.67 | 0.74 | 0.23 | 0.28 | 0.65 | 0.54 |
| 2012-07-17 (m/s) | 0.93 | 1.08 | 0.62 | 0.56 | 0.90 | 0.87 |
| 2012-07-24 (m/s) | 0.88 | 1.04 | 0.45 | 0.32 | 0.80 | 0.76 |

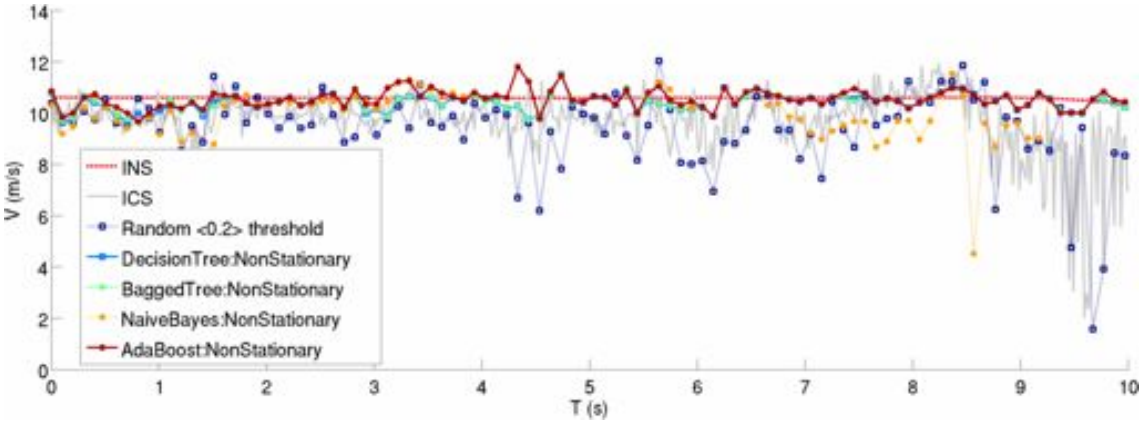



Figure 6.42: **Filtered velocity estimates (Woodstock exit).** Filtered velocity estimates using the various classification techniques over the Woodstock exit. In this example, the ICS velocity error due to traffic is highly visible. RMSE comparisons are shown in Table 6.7.

Table 6.7: RMSE Comparison: Woodstock exit

| | ICS | Random (.2) | Tree | BaggedTrees | NaiveBayes | AdaBoost |
|---|---|---|---|---|---|---|
| 2012-07-12 (m/s) | 1.00 | 1.20 | 0.39 | 0.36 | 1.25 | 0.61 |
| 2012-07-17 (m/s) | 0.86 | 1.20 | 0.48 | 0.51 | 0.90 | 0.75 |
| 2012-07-24 (m/s) | 1.51 | 2.13 | 0.39 | 0.38 | 0.95 | 0.38 |

In all of these cases, Figure 6.39 to Figure 6.40, the velocity estimates from the non-stationary filters (Decision Tree, Bagged Decision Trees, Naive Bayes, Boosted Decision Trees) are better - i.e. the RMSE is *lower* - than that of the raw ICS approach.

An argument that could be levelled against the non-stationary classification approach is that one does not need to be so selective when filtering points - if enough are removed from the LIDAR fan, the problematic points may be filtered out. As a control-experiment, Figure 6.43 shows the ICS velocity filtered by randomly decimating points from the LIDAR fan, for varying degrees of decimation:

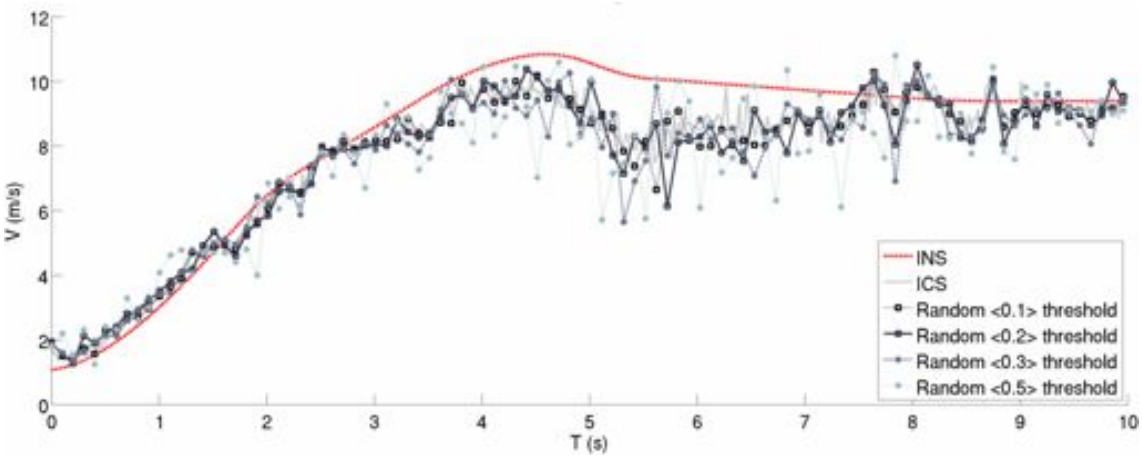



Figure 6.43: **Filter comparisons : Randomized sub-sampling.** Filtered ICS estimates using a random decimation of the incoming points. This naive approach merely increases the signal-to-noise ratio, and does little to compensate for the accrued velocity bias.

Figure 6.43 explicitly shows that this hypothesis is not true - random input decimation only serves to increase the signal-to-noise ratio, and does nothing to compensate for the velocity bias from traffic. Table 6.8 shows that the RMSE increases proportionally with the rejection ratio:

Table 6.8: RMSE Comparison: Random decimation

| Rejection ratio | .1 | .2 | .3 | .5 |
|---|---|---|---|---|
| RMSE (m/s) | 0.96 | 1.07 | 1.16 | 1.48 |

This section has explicitly shown how learning a place-dependant classifier leads to improved velocity estimates for a number of testing locations in a dynamic real-world setting. In the following section, the performance of the localisation engine is contrasted with and without the contextual filter bank, and final trajectory estimates are shown around the Woodstock site.

### 6.3.5 Localisation improvements using contextual filters

The previous section has clearly articulated the need for contextual scan-filtering to improve velocity estimates. We now explicitly demonstrate *why* this velocity filtering leads to improved localisation performance. Figure 6.44 contrasts the effect on *relative* localisation error (as compared to the INS) both with and without the filtered velocity feed:

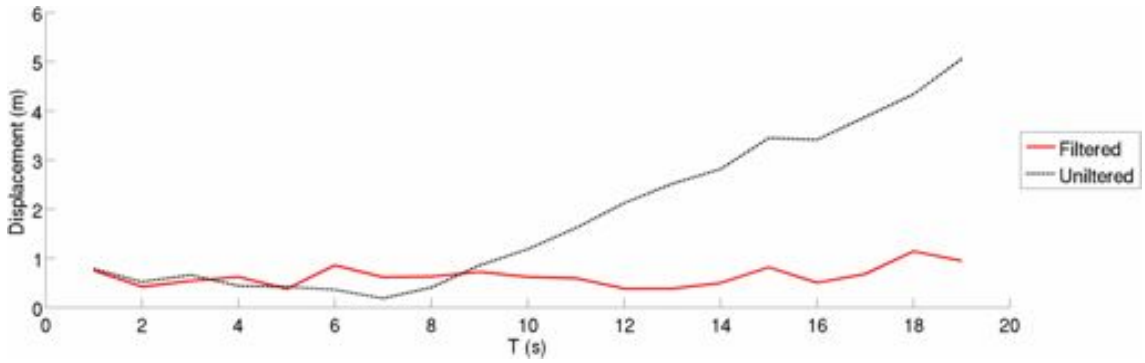



Figure 6.44: **The effect of filtering on localisation.** The difference in sequential pose estimates measured as a function of distance to the current INS pose estimate, using the non-stationary filter (red) and the unfiltered approach (black). Using the unfiltered velocity estimates, the estimate using $L^3$ begins to diverge until failure. However, by filtering the LIDAR points, the velocity estimate exhibits less error, maintaining localisation.

This plot was obtained by measuring the displacement of the current pose estimate (using the $L^3$ approach) to the current INS estimate, with velocities using the bagged decision-tree non-stationary filter (red) against the unfiltered estimate (black). Although the estimator disagrees with the global INS estimate by approximately one metre, this is constant for the filtered approach, but divergent for the non-filtered approach - this is due to the effect of accrued velocity error.

Again we present Figure 6.45, which showed the original failure locations, and provided the motivation for the contextual filter approach:

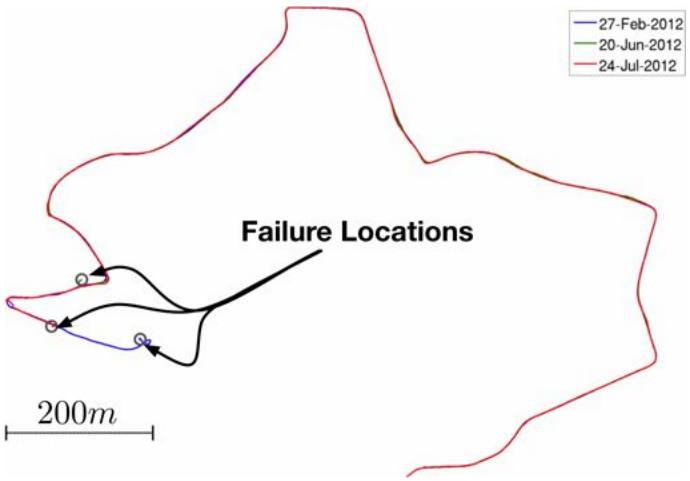



Figure 6.45: **Woodstock failure locations.** Localisation points-of-failure over the Woodstock route over three datasets spanning several months. Note how the failure locations cluster in the town centre - typically due to the presence of traffic.

As we have seen, by filtering the points going in to the scan-matching algorithm, we can correct the velocity estimates such that we do not suffer from a complete system failure, as highlighted in Figure 6.45. Figure 6.46 shows the resulting trajectories after using velocities that have been filtered with the contextual models:

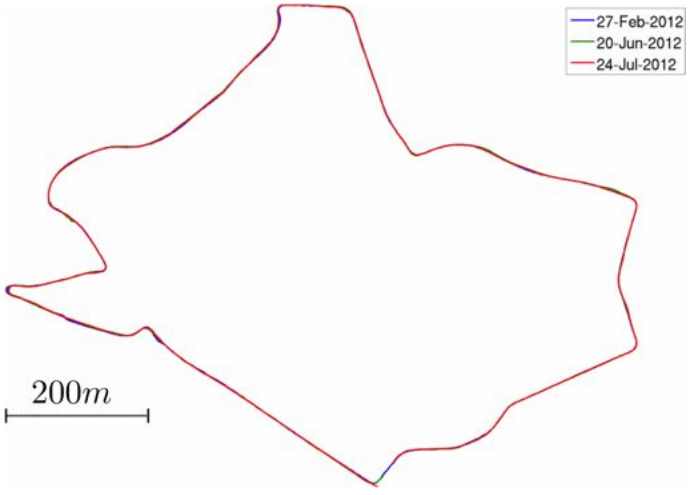



Figure 6.46: **The resulting trajectories using contextual scan-filtering.** After applying the contextual scan-filtering technique (in this case, bagged decision trees), the points of failure have been eliminated and the system stays localised for each of the datasets.

Given that - with the filtered velocity feeds - the system is more robust to failure

induced by traffic, we can again make use of the relative displacement metric for the purposes of comparison. Figure 6.47 compares the performance using this metric of the $L^3$ system (red, using the contextually-filtered LO velocity) against the INS (blue) for one loop around Woodstock:

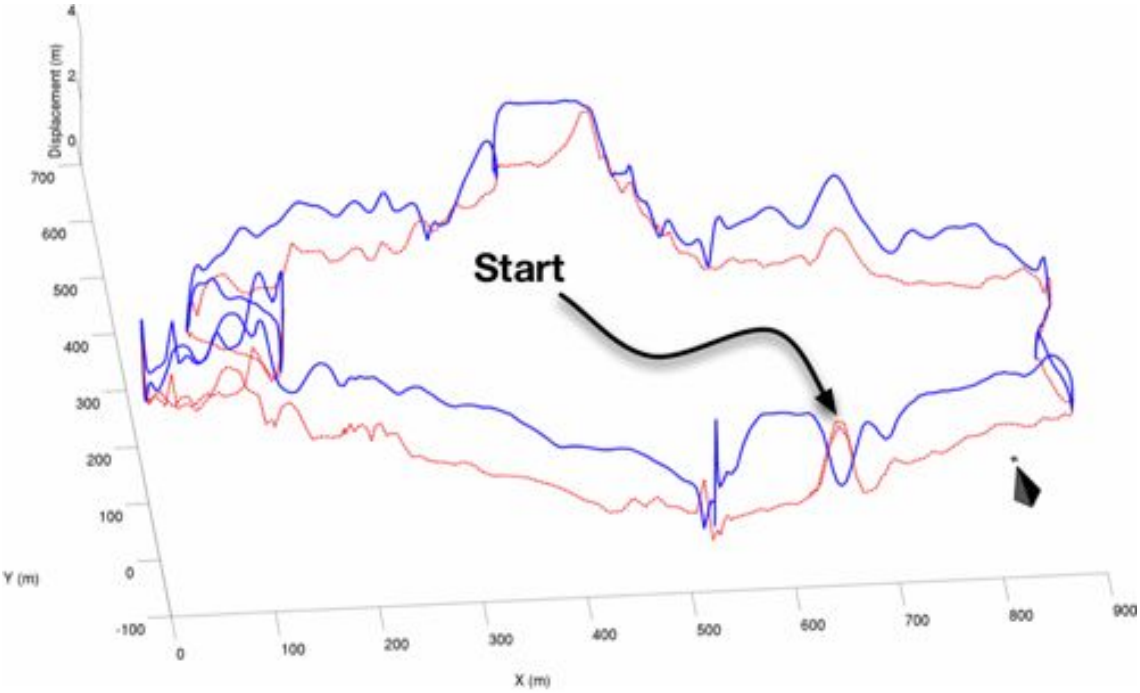



Figure 6.47: **Average relative displacement.** The relative displacement comparison of estimated trajectories from the INS (blue, solid) and the $L^3$ system (red, dashed) against a reference trajectory over 2.7km. As can be seen, the $L^3$ approach exhibits a lower displacement as compared to the INS over the course.

This is similar to the images shown in Chapter 5. To reiterate, in Figure 6.47 the displacement of the INS to the reference trajectory (blue) is shown against that of the $L^3$ system (red). On average we expect this displacement to be small, as the vehicle is constrained by the road-network, and this is shown to be true in the figure.

Figure 6.48 again shows the averaged relative displacement of the trajectories (INS, and $L^3$ using scan-matching), for Woodstock as compared to Kidlington:

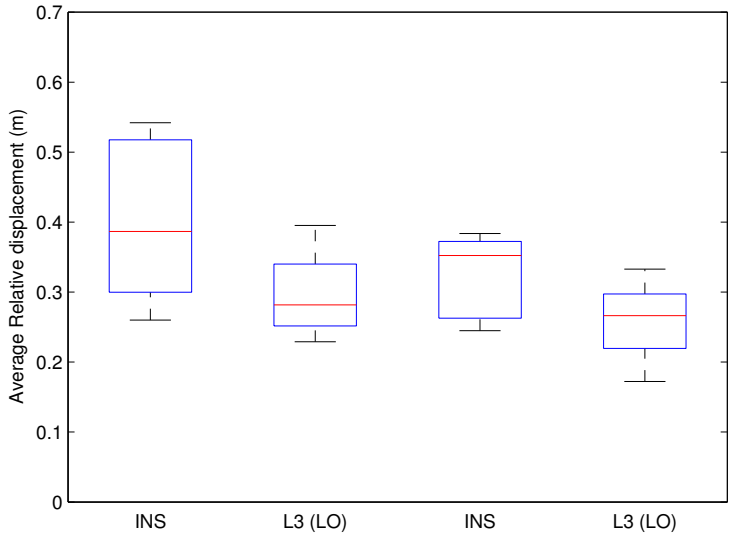



Figure 6.48: **Woodstock/Kidlington INS/$L^3$ displacement comparison.** Averaged relative displacement for the data collected over Woodstock (INS and $L^3$, left), and Kidlington (INS and $L^3$). The lower average relative displacement of $L^3$ is visible in both of these locations.

Again we can see that the performance, in terms of repeat localisation, is superior to that of the INS. Table 6.9 presents a performance summary of the entire system:

| **TTM** (days) | **Distance** (km) | **Cumulative** $L^3$ (INS) (km) | $L^3$ (LO) (km) | **Location** |
|---|---|---|---|---|
| 0-31 | 26.00 | 26.00 | - | Begbroke |
| 0 | 10.56 | 36.56 | - | Woodstock |
| 49 | 10.47 | 47.03 | - | Woodstock |
| 114 | 5.22 | 52.25 | **5.22** | Woodstock |
| 119 | 2.23 | 54.48 | **7.45** | Woodstock |
| 136 | 10.05 | 64.53 | 17.50 | Woodstock |
| 141 | 12.93 | 77.46 | 30.43 | Woodstock |
| 148 | 4.91 | 82.37 | 35.34 | Woodstock |
| 178 | 5.51 | 87.88 | **40.85** | Woodstock |
| 0 | 7.86 | 95.74 | - | Kidlington |
| 23 | 13.19 | 108.93 | **54.04** | Kidlington |

Table 6.9: Localisation summary

Table 6.9 summarises the localisation results, using the contextual-filter approach. This summary shows that - by learning a filter model predicated on our position in the world - we can stay localised in areas that previously caused failure (text in bold). The dashed datasets in Table 6.9 were omitted, as they were used to train the filters.

However, the presence of vehicles on the road is not the only source of errors for the scan-matching algorithm. Ground-strike in particular contributes greatly to the signal-to-noise ratio of the estimated velocity feed, and cannot be filtered out using the supervised training approach as outlined. Figure 6.49 replicates the results shown in Figure 6.41 with the portion corresponding to ground-strike emphasised:

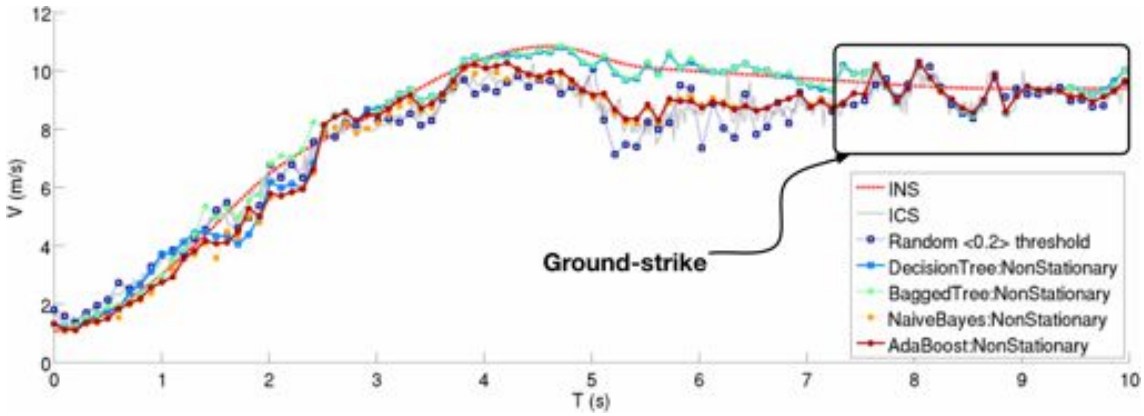
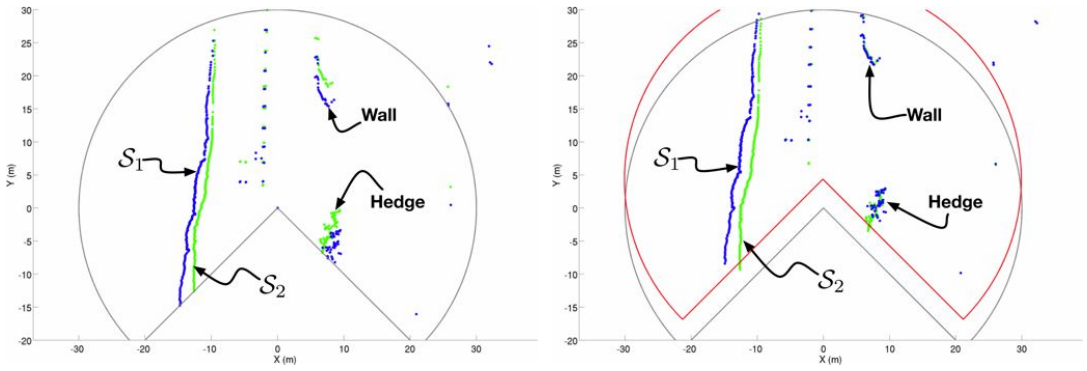



Figure 6.49: **Velocity errors induced from ground-strike.** The velocity error arising from ground-strike in the LIDAR fan. Although the decision tree and bagged decision tree models have reduced the signal-to-noise ratio by filtering out the traffic, both incorporate points corresponding to ground-strike, leading to velocity error.

Although we could envision leveraging the geometry of points in the LIDAR fan to build a vehicle classifier, this becomes much more difficult for ground-strike. Therefore, we seek a method of learning a contextual filter that can accommodate the errors accrued from both traffic and ground-strike. In addition, we require a model that can learn such a filter in an **unsupervised** fashion - it is too burdensome to require hand-labelled scans for each new mission context. The approach for achieving this is outlined in the following section.

## 6.4 Learning unsupervised filters

Although vehicles and ground-strike can appear very differently in the LIDAR fan, both share a common trait - both result in substantial point discrepancy in sequential LIDAR sweeps. As we saw in Figure 6.15, by using the INS to project sequential LIDAR sweeps into a common frame, we can immediately identify regions where there is little overlap - this corresponds to points in the LIDAR fan that will lead to poor velocity estimates. Figure 6.50 shows two scans, separated by one second, rendered into the same frame, i.e. $\mathcal{T}_{\mathcal{S}_1} = \mathcal{T}_{\mathcal{S}_1} = \mathcal{T}_0$:




(a) Relative (b) Global

Figure 6.50: **LIDAR point-overlay comparison.** (a) A comparison of two 1-second separated LIDAR fans from the horizontally-mounted LIDAR, showing the point-comparison when projected into the same frame. Visible on the right-hand side are a wall and a roadside hedge. (b) The same fans, rendered into a global frame using the INS. The wall and hedge are now well aligned - however, the points on the left are not aligned, indicative of ground-strike.

What is apparent from Figure 6.50(a) is that in both fans, the LIDAR points on the left-side of the fan seem to be good candidates for matching, i.e. they represent some distinct feature in the world. However, if instead the LIDAR fans are projected into a global frame (Figure 6.50(b)) the discrepancy becomes noticeable, which intimates that the line-like feature is merely an artefact caused by ground-strike. This is problematic when performing scan-matching through any of the ICP variants, as the mass of points on the left will dominate, giving rise to erroneous

estimates.

Given the mercurial nature of the ground-strike process, a new LIDAR filter model is proposed - one that will take into account the context giving rise to ground-strike, as well as relaxing the need for copious amounts of hand-labelled training data.

### 6.4.1 Learning transiency from data

We observe - during training - noisy estimates of the transiency of scan cells by overlaying points from consecutive scans using DGPS-corrected INS data. We then generate point correspondences across scans using a nearest-neighbour search and culling points closer than a certain threshold - this ensures the remaining points have moved substantially between consecutive scans. The raw LIDAR points from Woodstock centre are shown in Figure 6.51:

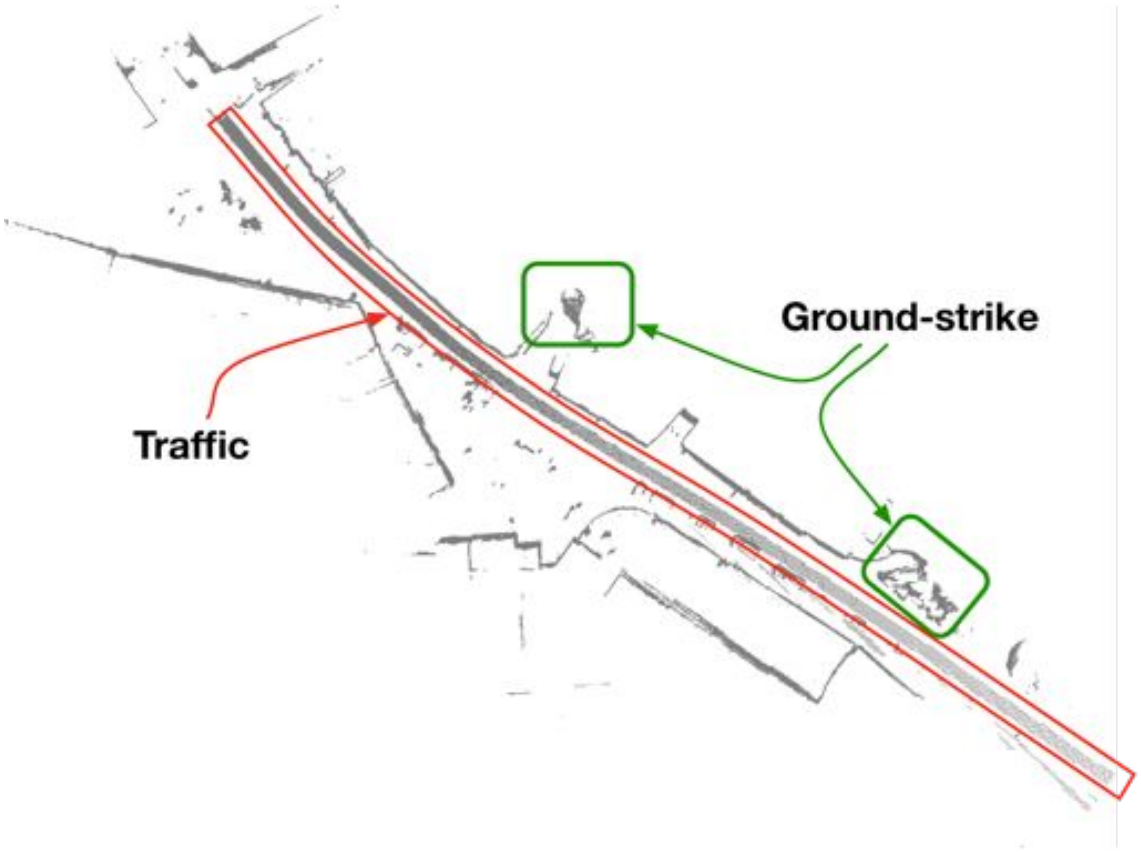



Figure 6.51: **Woodstock LIDAR transiency illustration: Raw LIDAR points.** The raw LIDAR points collected from the horizontally-mounted LIDAR, projected into a global frame using the INS. Very apparent in this figure are the "tracks" left by vehicles in front of the Wildcat; also visible are ground-strike points arising from the pitching and rolling motion of the vehicle on the road.

Highly visible in Figure 6.51 is the band of points that correspond to traffic. Also visible are points arising from ground-strike (highlighted in green). Ground-strike tends to appear as a moving wavefront in the LIDAR scan, and is to be removed in order to prevent the type of errors shown in Figure 6.49. In both cases (traffic and ground-strike), there is substantial discrepancy in point-correspondence when overlaid into a common frame (using the INS, as shown in Figure 6.50).

The following algorithm presents a simple way of determining these discrepancies. It takes, as input, two LIDAR scans, their associated pose (given by the INS), a set of parameters $\theta$, and returns points that exhibit a high degree of transience:

**Algorithm 8** Transiency estimation

**procedure** EstimateTransientPoints($\mathcal{S}_1, \mathcal{S}_2, \mathcal{T}_1, \mathcal{T}_2, \theta$)
  $\mathbf{X}_1 \leftarrow \text{Proj}(\mathcal{S}_1, \mathcal{T}_1)$ ▷ Project scans, given pose
  $\mathbf{X}_2 \leftarrow \text{Proj}(\mathcal{S}_2, \mathcal{T}_2)$
  $\mathcal{F} \leftarrow []$
  **for all** $x \in \mathbf{X}_2$ **do** ▷ Inter-scan cluster
    $\{x', d\} \leftarrow$FindKNearestNeighbours( $x, \mathbf{X}_1, k = 1$)
    **if** $\theta_{d_L} < d < \theta_{d_U}$ **then**
      append($\mathcal{F}, x'$)
    **end if**
  **end for**
  $\mathcal{O} \leftarrow []$
  **for all** $x \in \mathcal{F}$ **do** ▷ Secondary cluster
    $\{\mathbf{d}\} \leftarrow$FindKNearestNeighbours( $x, \mathcal{F}, k = \theta_{nn}$)
    **if** $\mathbb{E}[\mathbf{d}] > \theta_m$ **then**
      append($\mathcal{O}, x'$)
    **end if**
  **end for**
  return $\mathcal{O}$
**end procedure**

where the function FindKNearestNeighbours($\cdot$) returns K points and Euclidean distances for an input datum against an array of data. The initial loop in Algorithm 8 is an inter-scan clustering, that calculates the distance from each point in the first scan to each point in the second scan, with those points with the nearest neighbour distance falling into the range specified by $\theta_{d_L}$ and $\theta_{d_U}$ being retained. This approach produces a broad estimate of discrepancy between point clusters in consecutive scans, but always incorporates outlier points.

To account for this, the second loop averages the distance of each point to $\theta_{nn}$ of its neighbours, which are retained if they fall above a distance specified by $\theta_m$. If a point has less than $\theta_{nn}$ neighbours, then the point is discarded - this ensures that outlier points are not always incorporated.

Applying this filtering approach to the points in Figure 6.51 results in the following filtered transient points:

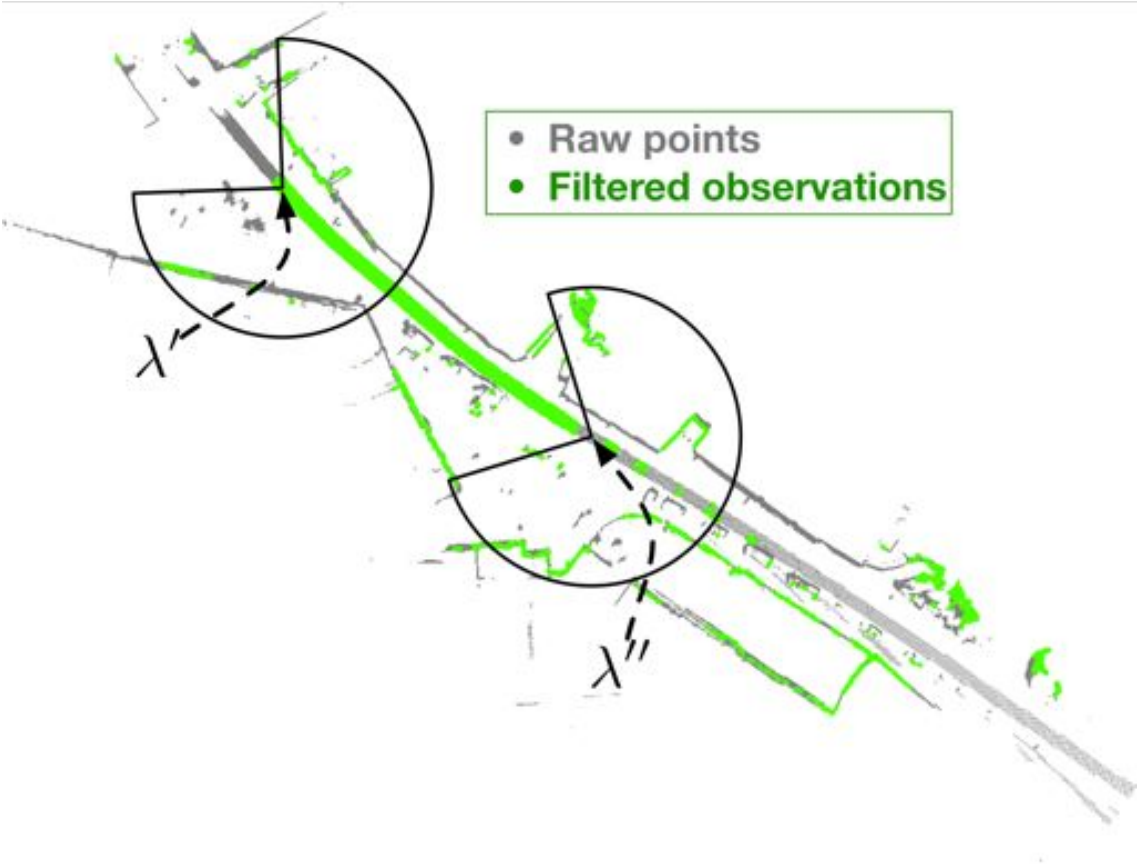



Figure 6.52: **Woodstock LIDAR transiency illustration: Transient observations.** The filtered LIDAR points from Figure 6.51, filtered as per Algorithm 8. By fusing INS and LIDAR data during the initial run and applying simple clustering methods, we can isolate areas in the environment that exhibit a high degree of transience. In addition to vehicles, ground strike (top-right) is also detected - this is to be expected, as ground-strike can be characterized as a fast-moving wavefront. Some static structures are incorrectly observed as transients - however, the number of these observations is small. Also shown are two example LIDAR fans rendered at two successive $\lambda$ positions.

At this juncture, we move away from relying on a point-based classification approach toward a discretized model of scan-transiency that enables us to incorporate prior knowledge as to how point-discrepancy appears in the LIDAR fan. The LIDAR domain is now partitioned into discrete segments, as is shown in Figure 6.53:

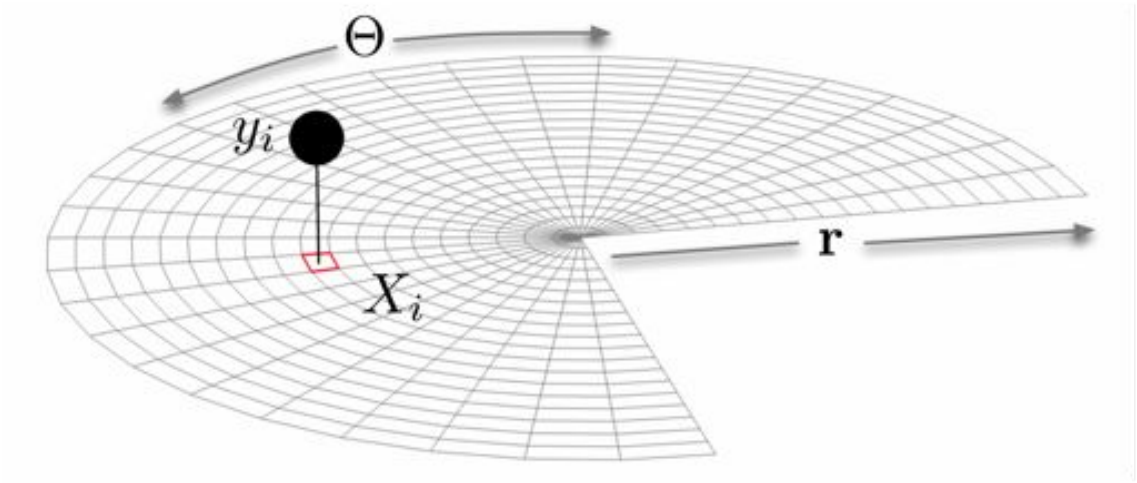



Figure 6.53: **Discretized LIDAR model.** A visualisation of the model used to estimate the **transience** of areas in the LIDAR scan plane (given a location in the world, indexed by $\lambda$). The unobserved latent states, $\mathbf{X}$, constitute the underlying transience of a certain location in the beam plane of the LIDAR. The observed value $\mathbf{y}$ are noisy estimates from training data, estimated by observing point misalignment in consecutive laser scans.

The input domain (range and angle) is now partitioned into a grid of random variables, $\mathbf{X}$, where $X_i$ is a binary variable denoting the **transiency** of laser data observed in a discrete $\{\theta, r\}$ cell in the scan plane. This measure captures how reliable sensor data from a certain cell will be and allows us to determine, probabilistically, how much we can trust LIDAR data from a particular point in the world.

The advantage of this representation is that we can fold in a prior intuition as to the relationship of neighbouring cells in the LIDAR fan - for instance, if a certain cell $X_i$ is marked as "transient", the likelihood of its neighbours being transient is higher. This probabilistic model, formulated as a graph, enables us to capture these correlations. The approach is explored fully in the following section, beginning with an overview of Markov Random Fields.

### Representing the filter as a Markov Random Field

We can consider the LIDAR segmentation in Figure 6.53 to be a graph $\mathcal{G}$ characterized by vertices $\mathcal{V}$, with latent variables $\mathbf{X}_{v\in\mathcal{V}}$ connected by undirected edges $\mathcal{E}$, each taking on a label in the set $\mathcal{L} = \{0, 1\}$. If we assume a strict positivity constraint, such that:

$$P(X_i) > 0 \;\; \forall \;\; X_i \in \mathbf{X} \qquad \text{(Positivity)} \tag{6.11}$$

and a neighbourhood constraint:

$$P(X_i \mid \mathbf{X}_{\setminus\{\mathbf{i}\}}) = P(X_i \mid X_{\mathcal{N}_i}) \qquad \text{(Markovianity)} \tag{6.12}$$

where $X_{\setminus i}$ denotes the set-difference operator, and $X_{\mathcal{N}_i}$, are the neighbouring sites of $X_i$ then the graph is said to represent a Markov Random Field. Then, according to the Hammersley-Clifford theorem [6], the MRF is equivalent to the Gibbs distribution over $\mathcal{G}$:

$$P(\mathbf{X}) = \frac{1}{\mathcal{Z}} \prod_{c\in\mathcal{C}(\mathcal{G})} \exp\left\{\theta_c\{X_c\}\right\} \tag{6.13}$$

where $\mathcal{Z}$ is the *partition function* ensuring that $P(\mathbf{X})$ is a valid probability distribution, and $\theta_c\{X_c\}$ are the potentials associated with the maximal *cliques* $c \in C(\mathcal{G})$. Maximal cliques for $1-4$ variables are shown in Figure 6.54:

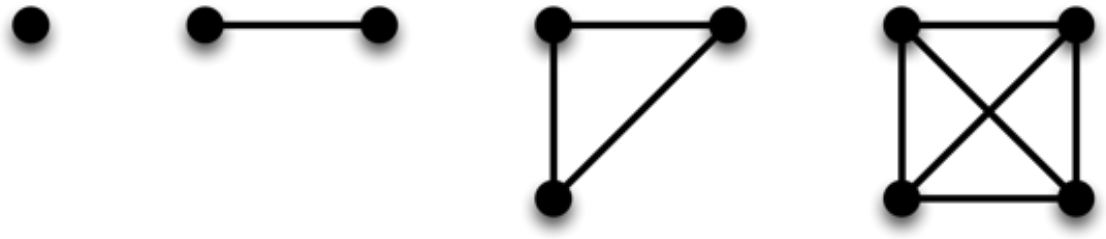


Figure 6.54: **Fully connected sub-graphs, or "cliques".** Fully connected cliques for 1,2,3, and 4 variables. The pairwise cliques in the MRF model correspond to a fully-connected graph of size 2.

Cliques are fully-connected subgraphs, which in the case of the MRF are pairwise - higher order cliques can be specified depending on the task. Equation (6.13) can be re-written as:

$$P(\mathbf{X}) = \frac{1}{\mathcal{Z}}\exp\{E(\mathbf{X})\} \qquad \text{where} \tag{6.14}$$

$$E(\mathbf{X}) = \sum_{c\in\mathcal{C}} \theta_c\{X_c\} \tag{6.15}$$

where $E(\mathbf{X})$ is known as the energy function, which decomposes into *unary* and *pairwise* terms:

$$E(\mathbf{X}) = \sum_{i\in\mathcal{V}} \Phi(X_i) - \sum_{(i,j)\in\mathcal{E}} \Psi(X_i, X_j) \tag{6.16}$$

where $\Phi(\cdot)$ is the *node* potential and $\Psi(\cdot)$ is the *pairwise* potential. Given that we are seeking a binary segmentation, a natural prior over the pairwise term is the Ising model:

$$\Psi(X_i, X_j) = \gamma \mid X_i - X_j \mid \tag{6.17}$$

The Ising potential increases the contributed energy when spatially-adjacent variables have differing values, enforcing spatial consistency[5] . The MRF is now used as the prior over X, which is then associated with a set of observations, $\mathbf{y}$. Given that this is a *segmentation* problem, i.e. $\mathbf{X} \in \{0, 1\}$, we would like to estimate the most probable labellings for all the latent variables in the graph given the observed data. In a Bayesian setting, this posterior distribution over latent variables is:

$$P(\mathbf{X} \mid \mathbf{y}) \propto P(\mathbf{y} \mid \mathbf{X})P(\mathbf{X}) \tag{6.18}$$

The assumption made for tractability here is over the likelihood term:

$$P(\mathbf{y} \mid \mathbf{X}) = \prod_{i \in \mathcal{V}} P(\mathbf{y}_i \mid x_i) \tag{6.19}$$

Note that only the prior has been specified to be a Markov Random Field - however, given the conditional independence assumption, the resulting posterior is also a MRF. Another important aspect here is that the label interactions here are specified a-priori - they are not a function of the data.

The most probable setting for the latent configuration is the *maximum a-posteriori* estimate, given by:

[5]The Ising prior is said to be homogeneous and isotropic - the interaction potential is stationary across the field, and it is invariant to rotation.

$$\text{MAP} = \underset{x \in \mathbf{X}}{\text{argmax}} \; P(\mathbf{X} \mid \mathbf{y}) \tag{6.20}$$

which corresponds to a minimisation of the energy function in Section 6.4.1. For the pseudo-boolean[6] function that corresponds to the Ising prior, the MAP estimate can be computed in polynomial time using graph cuts [15], given that the formulation obeys submodularity constraints [47].

The key advantage here is that, due to the explicit maximization, evaluation of the partition function $\mathcal{Z}$ is not required - this is not true for computing the marginals, for which we must resort to approximate methods such as sum-product Belief Propagation (BP). A good overview of BP, and inference in graphs, is given in [48]. However, for our scan-segmentation task, the MAP estimate is sufficient for a "hard-masking" approach.

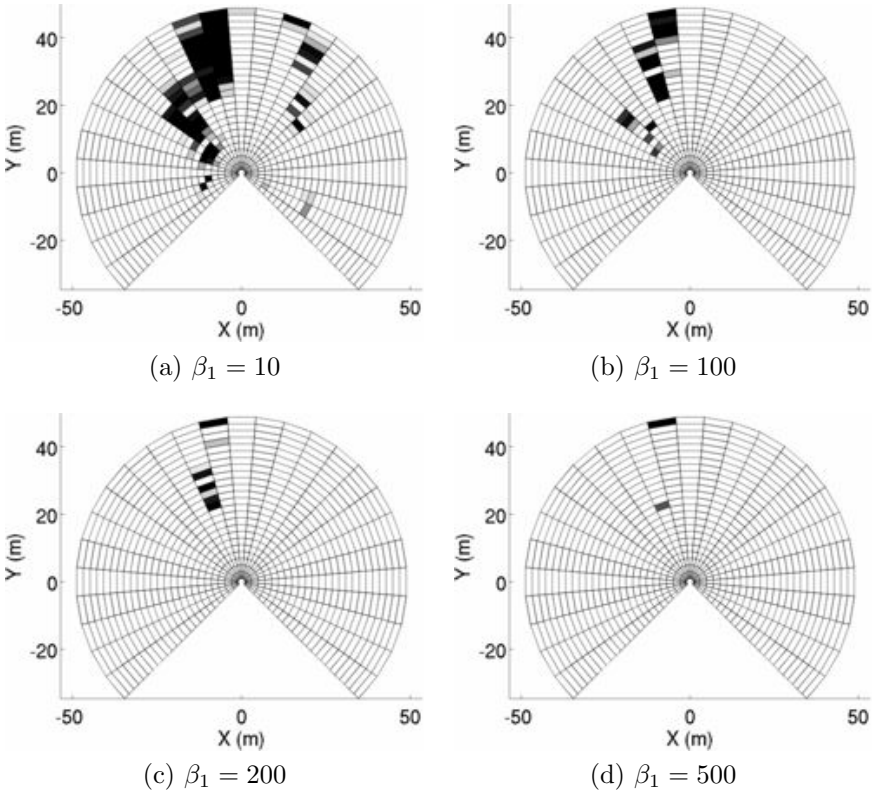
The Conditional Random Field [52] is a form of MRF that also models the posterior distribution $P(\mathbf{X} \mid \mathbf{y})$ - however, the factorization into prior and likelihood is not made explicit, as mentioned in [11]. Instead of modelling the data in a generative fashion, we utilize a discriminative classifier as the unary potential, as formulated by [50]. For the segmentation task, we make use of logistic regression:

$$P(X_i \mid y_i) = \frac{1}{\exp^{-(\beta_0 + \beta_1 y_i)} + 1} \tag{6.21}$$

where the observations $\mathbf{y}$ are histogram counts of per-cell transience given some $\lambda$ index (as illustrated by the two example fans in Figure 6.52). Given the projection of the global points into the LIDAR fan, we apply a histogramming operation to produce counts of per-cell transience. Figure 6.55 shows the effect on the cell

[6] A function whose domain is boolean, but the range is real-valued

transience probability for various $\beta$ parameters as given by logistic regression:




(a) $\beta_1 = 10$ (b) $\beta_1 = 100$

(c) $\beta_1 = 200$ (d) $\beta_1 = 500$

Figure 6.55: **Cell transiency probabilities for varying $\beta$.** (a) $\beta_1 = 10$, (b) $\beta_1 = 100$, (c) $\beta_1 = 200$, (d) $\beta_1 = 500$. Darker cells indicate higher probability.

Varying the $\beta_1$ parameter adjusts the estimated probability of a cell being marked "transient" as a function of the number of points in the cell. A value of $\beta_1 = 100$ works well for this problem. Given this regression model, the posterior distribution becomes:

$$P(\mathbf{X} \mid \mathbf{y}) = \frac{1}{\mathcal{Z}} \exp - \left( \eta \sum_{i \in \mathcal{V}} -\log(p(X_i \mid y_i)) + \gamma \sum_{i,j \in \mathcal{E}} \mid X_i - X_j \mid \right) \tag{6.22}$$

This representation, where we are no longer modelling the data in a generative fashion is known as a Conditional Random Field - the model is *conditioned* on the observed data. More specifically, the use of discriminative classifiers in the unary or

pairwise terms is known as a Discriminative Random Field (DRF) [50].

This model has a number of parameters - $\eta$ and $\gamma$ which trade off the relative contribution of the unary and pairwise potentials, with $\beta_0$ and $\beta_1$ modulating the behaviour of the logistic function. In general, these parameters would be optimised with respect to some training data - however, since we have no explicit labellings for transiency given histogram value, we rely instead on hand-tuned parameters. However, even with sub-optimal model parameters, we show that the resulting velocity estimates (and therefore, localisation estimates) are comparable to supervised classifiers.

### Estimating the most probable filter state

Given a set of parameters for the model, we want to estimate the MAP labelling for the latent states - this is termed *decoding*. Sophisticated methods based on graph cuts allow for polynomial-time decoding (as a function of the number of latent states) for pseudo-boolean graphs. For ease of implementation, we utilise a decoding scheme from the family of move-making algorithms, known as Iterated Conditional Modes (ICM) [9].

ICM starts with an initial configuration, and proposes moves in the configuration space of $\mathbf{X}$. Moves that decrease the energy are accepted with probability 1, and the algorithm iterates until a convergence tolerance is reached. Given that any latent state in the graph is conditionally independent of the rest of the graph given its neighbourhood[7], ICM can be formulated as the following minimisation:

$$X_i = \underset{X_i \in \mathcal{V}}{\operatorname{argmin}} \; E(\{X_j \mid j \neq i\}, X_i) \tag{6.23}$$

[7]This is also known as the Markov blanket

ICM works in a coordinate-descent fashion, proposing changes in the *move space* of the random field, halting when no other change leads to a decreased energy. For the data gathered over the location marked $\lambda''$ in Figure 6.52 the most probable labelling of $\mathbf{X}$ given the observed data $\mathbf{y}$ and employing ICM is shown in Figure 6.56.

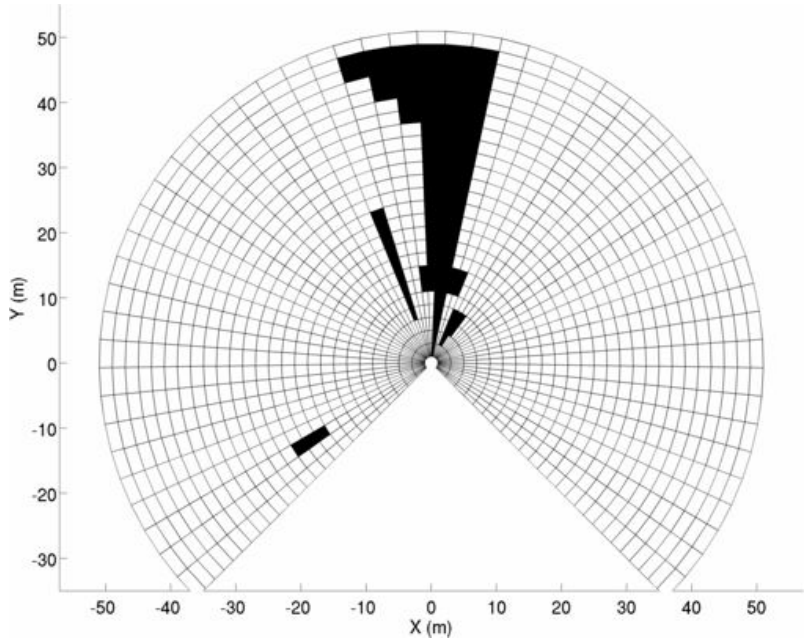
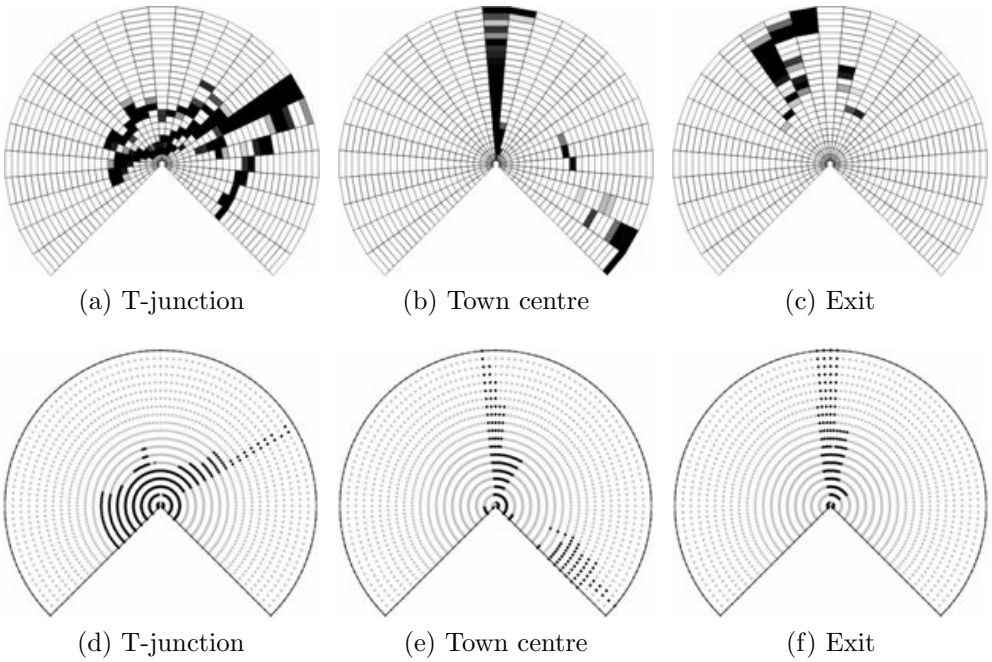



Figure 6.56: **Iterated Conditional Modes (ICM).** The resulting locally maximal $\mathbf{X}$ given by applying ICM to the observed data for the location $\lambda''$ shown in Figure 6.52 over the model depicted in Figure 6.53. Note how the road has been learned to be an unreliable place for scan-matching, given the presence of vehicles. Also, the entryway between two buildings has also been classified as transient - this is due to repeated induced roll experienced by the vehicle at that particular point.

Note crucially that we have not explicitly encoded a vehicle model, but we have learned that roads - due to cars - are poor places to utilize scan match data. We have **also** simultaneously learned that ground strike is also undesirable - Figure 6.56 classifies a region between two buildings as a source of transient LIDAR data. This is due to the repeatedly observed ground-strike arising from vehicle roll in that particular area.

Figure 6.57 shows a side-by-side comparison of the supervised/unsupervised filters for equivalent locations in Woodstock:



(a) T-junction (b) Town centre (c) Exit

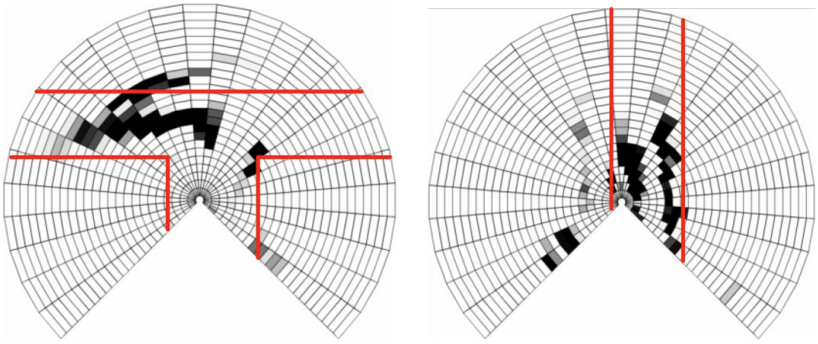
(d) T-junction (e) Town centre (f) Exit

Figure 6.57: **Comparison of supervised/unsupervised filters.** Top row is the transiency-based estimation over the LIDAR model, bottom-row is the fully-enumerated decision boundary for the equivalent non-stationary supervised classifier at the same location (Decision Trees, in this example). (a)(d) are the Woodstock T-junction, (b)(e) the town centre, and (c)(f) the Woodstock exit.

One aspect to note is that the forms of the filters are fundamentally different - in the unsupervised case, we explicitly learn a filter over a discretized domain, whilst in the supervised case the form of the filter shown is plotted by explicitly enumerating the decision boundary for the entire input space.

We now can use this learned distribution at run-time to filter out LIDAR points that have a high probability of originating from a transient object - be it a vehicle, or ground-strike from the rolling/pitching motion of the vehicle. The key advantage of this unsupervised method is that we can apply the same approach to new scenes, without having to rely on hand-labelled training data.

Figure 6.58 shows two example filters learned at different locations around the Kidlington test route, with the road boundaries highlighted:



(a) X (b) Z

Figure 6.58: **Kidlington: Unsupervised filters.** Unsupervised filters learned for a (a) T-Junction, and (b) straight road section.

It is apparent that this method generalizes to new environments well, and relieves the burden of having to obtain hand-labelled training data. We have focused exclusively in the previous sections on varying the form of the filter - the following brief section enumerates an interesting result of varying the intrinsic parameters of the scan-matching engine, concluding this chapter.

## 6.5 Contextual model parameters

An equally valid approach to the problem is to vary some *intrinsic* properties of the matching algorithm - for instance, the outlier threshold in the conventional ICP formulation. Figure 6.59 shows a result of this experiment:

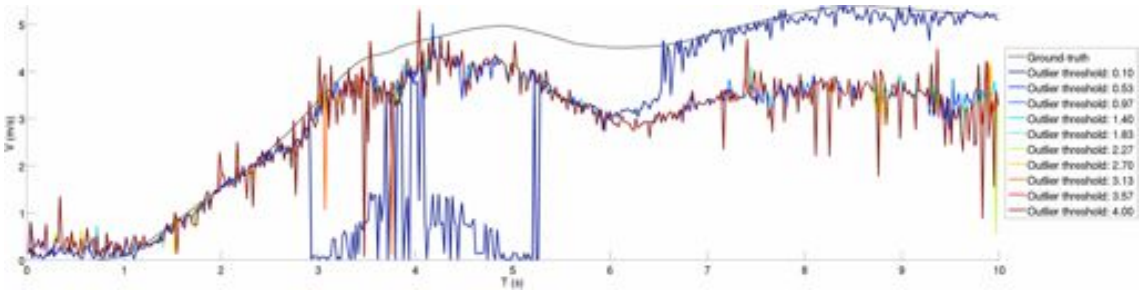



Figure 6.59: **Comparison of ICP parameters in Woodstock centre.** Velocity estimates arising from different outlier thresholds in an ICP scan-matching engine.

An interesting aspect of Figure 6.59 is the behaviour of the matching algorithm when the outlier threshold is set to a value of 0.1. For $t \in [0, 6.5]$, the behaviour of ICP is as poor as (if not worse) than other thresholds. However, from $t = 6.5$ onwards, the system recovers and begins to estimate the vehicle velocity well, despite the relative velocity of the vehicle in front.

This intimates that the outlier threshold is successfully filtering out the beams impacting on the vehicle, leading to better velocity estimates - however, this outlier distance is dependent on the relative vehicle velocity (as compared to the Wildcat). Given that we can augment the sensor suite, an interesting avenue of research would be the servoing of this parameter value given some extrinsic cues about vehicles on the road ahead.

## 6.6 Summary

This section has outlined methods and techniques for generating robust velocity estimates over extensive real-world data from a horizontal LIDAR. The key focus has been the development of a context-specific sensor model, that - when learned in either a supervised or unsupervised fashion - serves to significantly reduce the signal-to-noise ratio on the velocity signal, leading to robust long-term large-scale pose estimates from the $L^3$ system.

# Chapter 7

# Conclusions

This section serves to summarise the previous chapters, providing a retrospective view of the material covered, and providing an outline for future work.

## 7.1 Summary of contributions

This work has focused on the accurate, long-term localisation of a road vehicle with an inexpensive sensor suite, centred around the use of 2D LIDARs. We observed at the beginning of the thesis that accurate knowledge of the vehicle position within the world is an absolute requirement for any of the higher-level tasks we would like an autonomous car to perform.

Chapter 2 presented an overview of the current approaches for large-scale localisation for road vehicles, with a high-level overview of the different sensing modalities. This chapter highlighted the common thread that localisation within a given map - whether it is obtained from a heavily instrumented survey vehicle or through other means - is crucial for any extended higher-level tasks. This is implicit - how can we know where to go if we don't know where we are? We also articulated the need for an active sensor, LIDAR, for its robust performance in a range of conditions and

environments, and why it continues to be a vital component for field robots.

Reliable, efficient data collection is a core competency in robotics, and Chapter 3 developed the means for both the *synthesis* and acquisition of large scale datasets. We presented an integrated simulator that was capable of generating realistic city-sized datasets in real-time, and showed how the proliferation of digital CAD data has allowed for increased realism when building virtual representations of cities.

A stand-alone dual-purpose localisation/surveying unit - the NABU - was designed and fabricated, allowing for seamless data collection in a platform-independent way. The design of the NABU incorporated a variety of sensing modalities - from LIDAR to vision - and was built to allow for the rapid re-configuration of the sensor payload. The usefulness of this unit has been validated with many kilometres travelled (and gigabytes logged) by members of the Mobile Robotics Group.

Given the ability to acquire large-scale datasets - and seamlessly synthesise arbitrary ones of high realism if required - we moved on to the core application: localisation within a prior map using inexpensive sensor suites (like the NABU) over the long-term. This focus formed the core of Chapter 4. The motivation behind this work - in particular the issues arising with conventional localisation techniques due to scene change - were highlighted, and this provided the impetus for the development of the retrospective swathe-based matching algorithm.

Framing the localisation process as an entropy-based optimisation problem served to provide a robust way of localising the vehicle over extended durations (both temporal and spatial). The techniques were designed and validated around Begbroke Science Park, and subsequently extended into the nearby towns of Woodstock and Kidlington in Chapter 5. The performance of the algorithm was validated extensively over a wide variety of weather, conditions, and experiencing all the scene change common to a typical bustling urban centre.

Of course, operating in these complex, highly dynamic environments provided a

number challenges. Of vital importance to the estimation algorithm in Chapter 4 was the accuracy with which the swathe could be built - this relied on unbiased sources of velocity estimates, which proved to be difficult to obtain in a fluid, dynamic setting like the centre of a town.

The core insight to correcting this bias was that the *context* of the sensor data was highly influential, and this was the focus of Chapter 6. By building a context-specific model of dynamic parts of the scene, it became possible to filter out regions that were known to induce velocity errors. This was validated extensively by using hand-labelled training data, which - given the scales operated over in this thesis - becomes rapidly untenable. The solution to this was to recover these sensor models by *learning* about the problematic areas from repeated passes through the workspace, and this allowed for the unsupervised synthesis of these non-stationary models.

## 7.2 Future work

Some of the limitations and failure cases of the system have already been articulated. Improvements and refinements can be broken down into three categories:

### Improved ego-motion estimates

Having multiple sources of velocity data is of course highly beneficial. Utilizing multiple modes is particularly useful, as often the failure cases for each sensor will not overlap - see Appendix A. This is particularly pertinent if we consider the major failure modes for the two odometric measures compared in this thesis - VO and LO - whose prime failure conditions are darkness and falling rain, respectively.

Of course, for conditions falling in the spectrum between these two extremes, it may not be possible to know how to weight the importance of each of these sensors. Developing another contextual - and *temporal* - weighting parameter would allow

us to develop *confidence* regions for each sensor, leading to better overall velocity estimates.

### Cross-sensor integration

Although we have access to multiple sensor modalities on the NABU, information is not shared about the scene as perceived by each of the individual sensors. For example, if we *know* - with a high degree of confidence - that certain LIDAR points in the fan should be filtered, these could be referenced - using the known extrinsic calibration between the sensors - to pixels in imagery from the on-board cameras. These would provide a strong *prior* that would allow us to filter features in the *camera* image (using some form of segmentation for example) that would lead to better VO estimates (for example).

### Learning semantic regions

In the current approach, we learned a sensor model that was predicated only on the distance around a canonical spline map of the environment. While this approach proved to be satisfactory for all the data collected in this thesis, it does not answer the interesting question of how these models relate to each other - for example, we expect models on the highway to be self-similar to other highway models.

By identifying commonality between these models, we can identify regions with semantic cohesion which will be necessary - when considering country-scale navigation - for the data-compression opportunities it will provide.

## 7.3 Concluding remark

In this thesis we have developed the means to bring reliable, inexpensive, robust long-term localisation capabilities to vehicles of any configuration, and it is hoped

that the framework in its entirety will be beneficial to researchers going forward.

# Appendix A

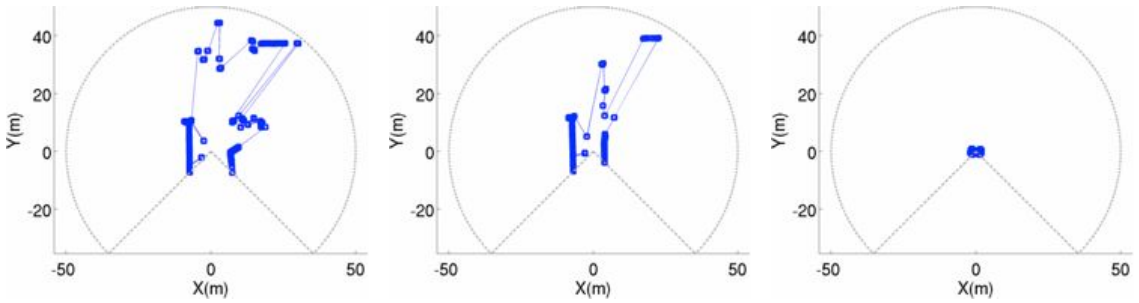
# Sensor failure modes

This section briefly illustrates the advantage of dual odometry-estimation modalities. Figure A.1 illustrates some of the weather encountered at Begbroke Science Park across the seasons:

(a) Sunshine (2011-12-05) (b) Snow (2012-02-06) (c) Snow and mist (2012-02-06)

Figure A.1: **Sensor characteristics across seasons.** A sequence of images showing the varying faces of Begbroke Science Park. Shown in (a) is a dry, cloudless day - ideal operating conditions for any sensor. Figure (b) is of a day with intermittent snow storms, and (c) is during a mist/snow shower.

As can be seen from the image sequence, there is still noticeable structure, even during the snow/light rain shower in Figure A.1(c). Now compare the performance of the horizontal LIDAR over the same data:




(a) Sunshine (2011-12-05) (b) Snow (2012-02-06) (c) Snow and mist (2012-02-06)

Figure A.2: **LIDAR failure modes.** A contrast of the LIDAR performance across the same conditions. Figure (a) shows the LIDAR working as expected. In (b) we begin to notice the LIDAR signal degradation, while finally in (c) the LIDAR has degraded completely.

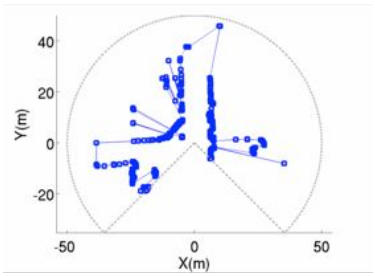
As can be seen from the LIDAR plots in Figure A.2, during the mist/show shower, there are no valid range-returns - this is problematic if we are using LIDAR to navigate during such an event. It should be stressed here though that Figure A.2 was an outlier, and only one of two instances of a rainy dataset collected over the duration of this thesis.

We now examine the reverse of this situation - consider Figure A.3, showing images from the same location (on successive loops) around the Woodstock test site:

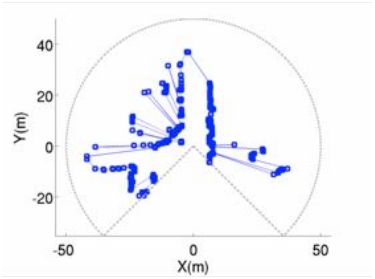
(a) 2012-04-16: Loop 1 (b) 2012-04-16: Loop 2 (c) 2012-04-16: Loop 3

Figure A.3: **Camera failure modes.** The vehicle lighting and evening conditions combine to make this a very testing image sequence to perform VO on.

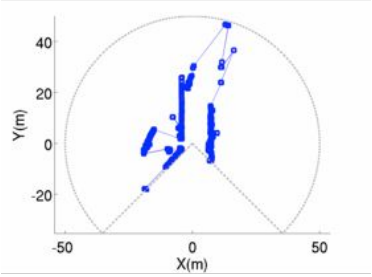
Contrast this with the performance of the LIDAR data, taken at the same locations:




(a) 2012-04-16: Loop 1




(b) 2012-04-16: Loop 2




(c) 2012-04-16: Loop 3

Figure A.4: **LIDAR evening data.** LIDAR data corresponding to the images in Figure A.3.

The obvious point here is that different sensor modalities are advantageous for any autonomous system (or payload).

# Bibliography


[1] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, May 1987.

[2] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, D. Anderson, et al. Odin: Team victortango's entry in the darpa urban challenge. *Journal of Field Robotics*, 25(8):467–492, 2008.

[3] H. Badino, D. Huber, and T. Kanade. Real-time topometric localization. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1635–1642. IEEE, 2012.

[4] D. Badouel. An efficient ray-polygon intersection. In *Graphics gems*, pages 390–393. Academic Press Professional, Inc., 1990.

[5] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *Visualization and Computer Graphics, IEEE Transactions on*, 5(4):349–359, 1999.

[6] J. Besag. Statistical analysis of non-lattice data. *The statistician*, pages 179–195, 1975.

[7] P. Besl and N. McKay. A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence*, 14(2):239–256, 1992.

[8] P. Biber and W. Straßer. The normal distributions transform: A new approach to laser scan matching. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2743–2748. Ieee, 2003.

[9] C. Bishop. *Pattern recognition and machine learning*, volume 4. Springer, New York, 2006.

[10] C. Bishop and M. Svenskn. Bayesian hierarchical mixtures of experts. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 57–64. Morgan Kaufmann Publishers Inc., 2002.

[11] A. Blake, P. Kohli, and C. Rother. *Markov Random Fields for Vision and Image Processing*. Mit Pr, 2011.

[12] 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*, volume 2, pages 1899–1906. IEEE, 2003.

[13] M. Bosse and R. Zlot. Map matching and data association for large-scale two-dimensional laser scan-based slam. *The International Journal of Robotics Research*, 27(6):667–691, 2008.

[14] M. Bosse and R. Zlot. Continuous 3d scan-matching with a spinning 2d laser. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4312–4319. IEEE, 2009.

[15] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, 2001.

[16] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[17] R. Brent. *Algorithms for minimization without derivatives*. Dover Publications, 2002.

[18] A. Buja and W. Stuetzle. Bagging does not always decrease mean squared error. *Preprint*, 2000.

[19] P. Carle, P. Furgale, and T. Barfoot. Long-range rover localization by matching lidar scans to orbital elevation maps. *Journal of Field Robotics*, 27(3):344–370, 2010.

[20] Y. Cheng, M. Maimone, and L. Matthies. Visual odometry on the Mars exploration rovers. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 1, pages 903–910. IEEE, 2005.

[21] W. Churchill and P. Newman. Practice Makes Perfect? Managing and Leveraging Visual Experiences for Lifelong Navigation. In *Proc. IEEE International Conference on Robotics and Automation (ICRA2012)*, Minnesota, USA, May 2012.

[22] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, and G. Marchal. Automated multi-modality image registration based on information theory. In *Information processing in medical imaging*, volume 3, pages 264–274, 1995.

[23] O. C. Council. Oxfordshire city council transport policies and plans. `http://www.oxfordshire.gov.uk/cms/content/transport-new-developments`. Accessed: 13/08/2012.

[24] I. Csiszár and P. Shields. *Information theory and statistics: A tutorial*. Now Publishers Inc, 2004.

[25] M. Culp, K. Johnson, and G. Michailidis. ada: An r package for stochastic boosting. *Journal of Statistical Software*, 17(2):9, 2006.

[26] F. Daum. Nonlinear filters: beyond the Kalman filter. *Aerospace and Electronic Systems Magazine, IEEE*, 20(8):57–69, 2005.

[27] S. Davey. *Extensions to the probabilistic multi-hypothesis tracker for improved data association*. PhD thesis, The University of Adelaide, 2003.

[28] C. De Boor. *A practical guide to splines*, volume 27. Springer Verlag, 2001.

[29] H. Dinh and S. Kropac. Multi-resolution spin-images. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 863–870. IEEE, 2006.

[30] A. Diosi and L. Kleeman. Fast laser scan matching using polar coordinates. *The International Journal of Robotics Research*, 26(10):1125–1153, 2007.

[31] A. Doucet, N. De Freitas, and N. Gordon. *Sequential Monte Carlo methods in practice*. Springer Verlag, 2001.

[32] B. Efron. Bootstrap methods: another look at the jackknife. *The annals of Statistics*, 7(1):1–26, 1979.

[33] Ö. Eğecioğlu and A. Srinivasan. A fast non-parametric density estimation algorithm. *Communications in numerical methods in engineering*, 13(10):755–763, 1997.

[34] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[35] C. Ericson. *Real-time collision detection*, volume 1. Morgan Kaufmann, 2005.

[36] R. Fletcher and M. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168, 1963.

[37] P. Furgale and T. Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.

[38] N. Gelfand, N. Mitra, L. Guibas, and H. Pottmann. Robust global registration. In *Proceedings of the third Eurographics symposium on Geometry processing*, pages 197–es. Eurographics Association, 2005.

[39] A. Harrison and P. Newman. Ticsync: Knowing when things happened. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 356–363. IEEE, 2011.

[40] J. Havel and A. Herout. Yet faster ray-triangle intersection (using sse4). *Visualization and Computer Graphics, IEEE Transactions on*, 16(3):434–438, 2010.

[41] M. Hebel and U. Stilla. Als-aided navigation of helicopters or uavs over urban terrain. *International Society for Photogrammetry and Remote Sensing*, 2010.

[42] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *JOSA A*, 4(4):629–642, 1987.

[43] Intel. SSE4 Programming Reference. `http://software.intel.com/`, April 2007.

[44] A. Johnson. *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1997.

[45] M. Jordan and R. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[46] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Int. symp. aerospace/defense sensing, simul. and controls*, volume 3, page 26. Orlando, FL, 1997.

[47] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2):147–159, 2004.

[48] F. Kschischang, B. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, 2001.

[49] S. Kullback and R. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[50] S. Kumar and M. Hebert. Discriminative random fields. *International Journal of Computer Vision*, 68(2):179–201, 2006.

[51] R. Kümmerle, P. Pfaff, R. Triebel, and W. Burgard. Active monte carlo localization in outdoor terrains using multi-level surface maps. *Autonome Mobile Systeme 2007*, pages 29–35, 2007.

[52] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann Publishers Inc., 2001.

[53] S. LaValle. *Planning algorithms*. Cambridge Univ Pr, 2006.

[54] S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 1225–1232. IEEE, 2000.

[55] J. Levinson, M. Montemerlo, and S. Thrun. Map-based precision vehicle localization in urban environments. In *Proceedings of the Robotics: Science and Systems Conference*. Citeseer, 2007.

[56] J. Levinson and S. Thrun. Robust vehicle localization in urban environments using probabilistic maps. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4372–4378. IEEE, 2010.

[57] Y. Li and E. Olson. A general purpose feature extractor for light detection and ranging data. *Sensors*, 10(11):10356–10375, 2010.

[58] K. Lingemann, H. Surmann, A. Nuchter, and J. Hertzberg. Indoor and outdoor localization for fast mobile robots. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2185–2190. IEEE, 2004.

[59] M. Löfstedt and T. Akenine-Möller. An evaluation framework for ray-triangle intersection algorithms. *Journal of Graphics, GPU, & Game Tools*, 10(2):13–26, 2005.

[60] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of intelligent & robotic systems*, 18(3):249–275, 1997.

[61] M. Majnik and Z. Bosnic. Roc analysis of classifiers in machine learning: A survey. *University of Ljubljana, Ljubljana, Slovenia*, 2011.

[62] M. Milford and G. Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1643–1649. IEEE, 2012.

[63] T. Minka. Bayesian inference, entropy, and the multinomial distribution. *Scientific Commons*, 2000.

[64] T. Möller and B. Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of graphics tools*, 2(1):21–28, 1997.

[65] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597, 2008.

[66] F. Moosmann and C. Stiller. Velodyne slam. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 393 –398, june 2011.

[67] Y. Morton, Q. Zhou, and F. van Graas. Assessment of second order ionosphere error in gps range observables using arecibo incoherent scatter radar measurements. *Radio Sci*, 44, 2009.

[68] A. Napier, G. Sibley, and P. Newman. Real-time bounded-error pose estimation for road vehicles using vision. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 1141–1146. IEEE, 2010.

[69] U. Nehmzow. *Mobile robotics: a practical introduction*. Springer Verlag, 2003.

[70] J. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[71] J. Peterson. Arc length parameterization of spline curves. *Computer-Aided Design*, 14(degree 1):1–11, 2006.

[72] J. Pluim, J. Maintz, and M. Viergever. Mutual-information-based registration of medical images: a survey. *Medical Imaging, IEEE Transactions on*, 22(8):986–1004, 2003.

[73] J. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[74] F. Ramos, D. Fox, and H. Durrant-Whyte. Crf-matching: Conditional random fields for feature-based scan matching. In *Proc. of Robotics: Science and Systems*. Citeseer, 2007.

[75] J. Rehder, K. Gupta, S. Nuske, and S. Singh. Global pose estimation with limited gps and long range visual odometry. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 627–633. IEEE, 2012.

[76] D. Reid. An algorithm for tracking multiple targets. *Automatic Control, IEEE Transactions on*, 24(6):843–854, 1979.

[77] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001.

[78] R. Rusu, N. Blodow, Z. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3384–3391. IEEE, 2008.

[79] R. Schapire. A brief introduction to boosting. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1401–1406, 1999.

[80] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

[81] J. Shlens. Notes on kullback-leibler divergence and likelihood theory. *System Neurobiology Laboratory, Salk Institute for Biological Studies, California*, 2007.

[82] G. Sibley. Relative bundle adjustment. *Department of Engineering Science, Oxford University, Tech. Rep*, 2307(09), 2009.

[83] M. Smucker and J. Allan. An investigation of dirichlet prior smoothing's performance advantage. Technical report, Technical Report IR-391, The University of Massachusetts, The Center for Intelligent Information Retrieval, 2005.

[84] H. W. Sorenson. Least-squares estimation: from Gauss to Kalman. *IEEE Spectrum*, 7:63–68, July 1970.

[85] A. Stewart and P. Newman. Laps - localisation using appearance of prior structure: 6-dof monocular camera localisation using prior pointclouds. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Minnesota, USA, May 2012.

[86] E. Takeuchi and T. Tsubouchi. A 3-d scan matching using improved 3-d normal distributions transform for mobile robotic mapping. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3068–3073. IEEE, 2006.

[87] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.0 edition, 2012. http://www.cgal.org/Manual/4.0/doc_html/cgal_manual/packages.html.

[88] S. Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[89] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, et al. Minerva: A second-generation museum tour-guide robot. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 3. IEEE, 1999.

[90] S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic robotics*, volume 1. MIT press Cambridge, MA, 2005.

[91] S. Thrun, M. Diel, and D. Hähnel. Scan alignment and 3-d surface modeling with a helicopter platform. In *Field and Service Robotics*, pages 287–297. Springer, 2006.

[92] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial intelligence*, 128(1):99–141, 2001.

[93] G. Tipaldi and K. Arras. Flirt-interest regions for 2d range data. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3616–3622. IEEE, 2010.

[94] M. Tomono. A scan matching method using euclidean invariant signature for global localization and map building. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 866–871. IEEE, 2004.

[95] C. Toth, D. Grejner-Brzezinska, and Y. Lee. Terrain-based navigation: Trajectory recovery from lidar data. In *Position, Location and Navigation Symposium, 2008 IEEE/ION*, pages 760–765. IEEE, 2008.

[96] B. Turlach. Bandwidth selection in kernel density estimation: A review. *CORE and Institut de Statistique*, 19(4):1–33, 1993.

[97] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[98] P. Viola and W. Wells III. Alignment by maximization of mutual information. In *iccv*, page 16. Published by the IEEE Computer Society, 1995.

[99] S. Waterhouse and S. Waterhouse. Divide and conquer: Pattern recognition using mixtures of experts. *Doctorate's Thesis, University of Cambridge, Cambridge, England*, 1997.

[100] C. Yang and G. Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[101] R. Yang and J. Berger. *A catalog of noninformative priors*. Institute of Statistics and Decision Sciences, Duke University, 1996.

[102] L. Zöliei, J. Fisher, and W. Wells. An introduction to statistical methods of medical image registration. *Handbook of Mathematical Models in Computer Vision*, pages 531–542, 2006.