Representing 3D Shape in Sparse Range Images for Urban Object Classification

Alastair James Quadros

A thesis submitted in fulfillment
of the requirements of the degree of
Doctor of Philosophy

THE UNIVERSITY OF
SYDNEY

Australian Centre for Field Robotics
School of Aerospace, Mechanical and Mechatronic Engineering
The University of Sydney

May 2014

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgment has been made in the text.

Alastair James Quadros

7 May 2014

Abstract

Alastair James Quadros
The University of Sydney

Doctor of Philosophy
May 2014

Representing 3D Shape in Sparse Range Images for Urban Object Classification

This thesis develops techniques for interpreting 3D range images acquired in outdoor environments at a low resolution. It focuses on the task of robustly capturing the shapes that comprise objects, in order to classify them.

With the recent development of off-the-shelf 3D sensors such as the Velodyne, it is now possible to capture range images at video frame rates, allowing mobile robots to observe dynamic scenes in 3D. Classifying objects in these scenes is a core component of autonomous perception, as it enables reasoning at the semantic level, allowing a robot to be more proficient in interacting with the world. To perform this task, features are extracted from the data, which allows different regions to be compared and matched. However, range images acquired at this speed are of low resolution, resulting in sparse 3D point clouds, with sometimes only a few points per object. Changes in sensor viewpoint greatly affect the distribution of 3D points on a surface, and occlusion is prevalent. In this context, existing methods for feature extraction do not perform well.

This thesis contributes algorithms for the robust abstraction from 3D points to object classes. Efficient region-of-interest and surface normal extraction are evaluated, resulting in a keypoint algorithm that provides stable orientations. These build towards a novel feature that is designed to consistently capture local shape, regardless of sensor viewpoint. It does this by explicitly reasoning about the difference between known empty space, and space that has not been measured due to occlusion or sparse sensing. The feature involves a set of parallel 3D lines 'probing' the environment for a surface, and so the feature is called the 'line image.' When lines intercept the interpolated surface, the depth is recorded, producing a local 2½-D representation. Otherwise, if they go behind a surface, or encounter a large gap in the data, unknown space is registered. Visibly empty space, often found along the informative silhouette of an object, is also captured. This allows parts of objects to be consistently represented, capturing sufficient shape information for distinguishing class.

A dataset of objects scanned with a Velodyne was collected and hand labelled, in order to compare this feature with several others on the task of classification. The dataset comprised of 588 urban objects from 14 classes of vehicles, signs, buildings, people and trees. First, a simple k-nearest neighbours approach was used, where the line image showed improvements. Second, more complex classifiers were applied, requiring the features to be clustered into the equivalent of visual words. This presented challenges, as some dimensions are effectively missing due to the incorporated unknown regions. The clusters were used in topic modelling, allowing specific sub-parts of objects to be learnt across multiple scales, improving accuracy by 10%.

This work is applicable to any range image data. In general, it demonstrates the advantages in using the inherent density and occupancy information in a range image during 3D point cloud processing. The line image encodes non-visible regions as missing dimensions. This leaves open more probabilistic approaches to feature matching, where limited visibility results in a distribution over potentially matching object parts.

Acknowledgements

First I would like to thank my supervisors. James Underwood, for his dedicated attention, and his efforts in organising and creating our research group. Thanks to Bertrand Douillard for his friendly support, guidance, and being there to bounce ideas off.

For code used in this work, thanks to Peter Morton, Vsevolod Vlaskine, Cedric Wohlleber, Mark De Deuge and Lionel Ott. To the rest of the perception team, thanks for helping collect datasets. Also Mark Calleija and the tech staff who developed the robotic platforms used here.

The ACFR has been a nexus of amazing people and inspiring work. Thanks to all those who gave me advice, including Hugh Durrant-Whyte, Thierry Peynot, Graham Brooker and Tim Bailey. Also, the past four years would not have been the same without all my fellow students, including Lachlan, John, Asher, Dan, Ariel, Don, Brian and many more. I have cherished our lunchtime discussions.

To my family, thanks for your love and support.

I am also indebted to the free and open-source software movement, which has provided me with so many powerful tools, and a constant source of learning.

“it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility.

...

“Encoded in the large, highly evolved sensory and motor portions of the human brain is a billion years of experience about the nature of the world and how to survive in it. The deliberate process we call reasoning is, I believe, the thinnest veneer of human thought, effective only because it is supported by this much older and much more powerful, though usually unconscious, sensorimotor knowledge. We are all prodigious olympians in perceptual and motor areas, so good that we make the difficult look easy. Abstract thought, though, is a new trick, perhaps less than 100 thousand years old. We have not yet mastered it. It is not all that intrinsically difficult; it just seems so when we do it.”

- Hans Moravec, 1988

Contents

List of Figures

List of Tables

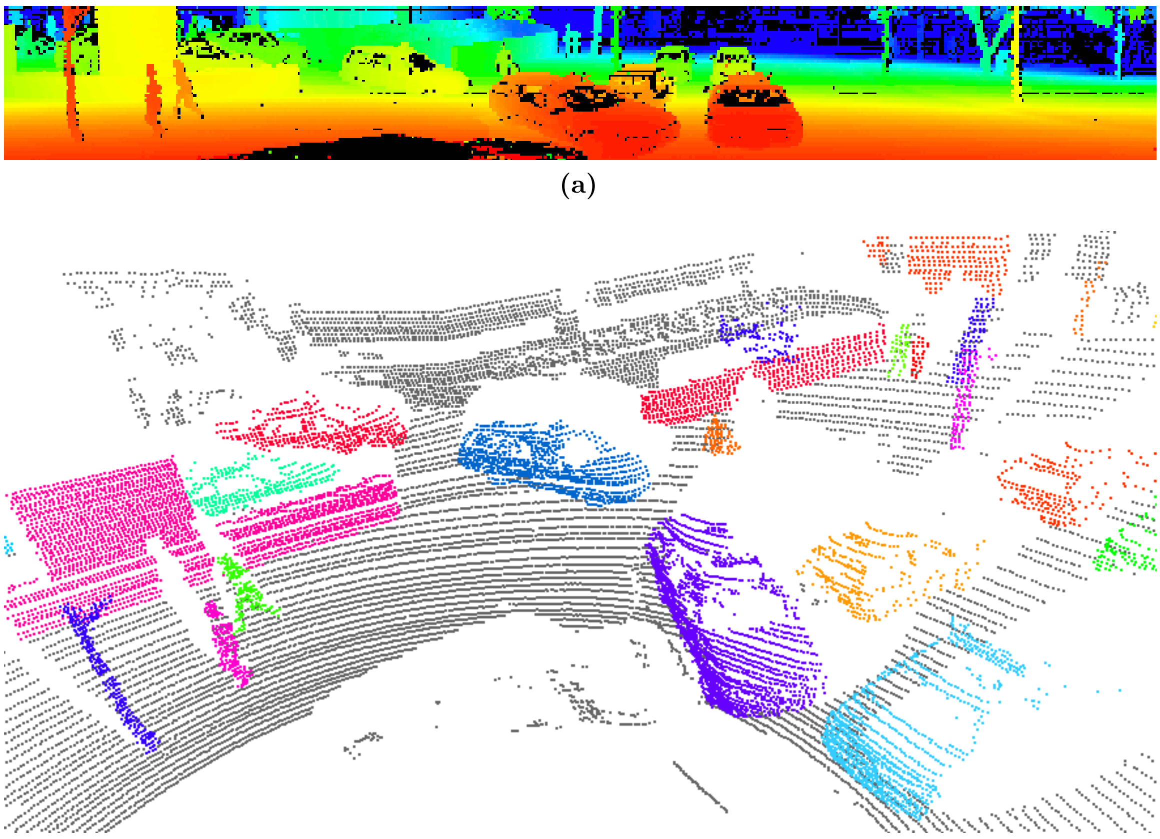
Chapter 1

Introduction

This thesis addresses the task of classifying objects in outdoor scenes using a form of 3D sensory data called the range image. Classification is the task of assigning a label to an object, allowing it to be perceived semantically, which is a core challenge in autonomous perception. 3D sensing provides an alternative approach from traditional imagery in computer vision, allowing the 3D shapes that comprise objects to be used for classification.

In a range image, pixels record depth rather than colour, allowing the 3D location of each pixel to be calculated in a 3D point cloud. These points trace out the shapes of objects, as shown in Figure 1.1. However, there are limitations in sensing for unmanned ground vehicles (UGVs) in dynamic outdoor scenes such as urban environments. Consider Figure 1.1. The image resolution is relatively low, which means 3D points at further ranges are far apart, and the range readings contain noise. Objects are only partly visible, and sometimes in the shadow of another, foreground object.

A focus in this thesis is the *representation* of shape in such a range image. In essence, groups of 3D points are not inherently comparable, they must be transformed in some way to facilitate the comparison of shape. This representation of shape is then fed to a classification algorithm, which uses this information to infer class.



(a)

(b)

Figure 1.1 – (a) A range image from a Velodyne, coloured by depth. (b) The associated 3D points, coloured to show some objects.

The objective of this thesis is to interpret the above data in a manner that is: (1) robust to low density, noise and occlusion; (2) capable of identifying many object categories; and (3) able to do so when trained from a limited dataset. To achieve these, the following tasks are addressed.

- Analysing basic geometric information from noisy, unevenly-sampled data.
- Creating a sufficiently distinguishable, viewpoint invariant shape representation for sensed object parts.
- Classifying objects from their constituent parts.

1.1 Motivation

The ability to identify a wide range of objects is important for many areas of robotics. Seemingly simple everyday tasks require responding to objects in the environment which are either related to the task at hand, or are critical for safety and efficiency. Interpreting urban environments in particular provides a challenge for UGVs such as self-driving cars. The 2007 DARPA Urban Challenge demonstrated the need for vehicle classification in order to plan and navigate traffic [85].

While specific objects can be detected by engineering sensors and algorithms for a narrow application, *general* object recognition significantly broadens the scope of a robot. A limitation in bringing robotics into everyday life is the inability to reliably interpret a relatively unstructured world, compared to factory floors where robotics has been very successful. Beyond simply allowing a robot to do more than a single task, more general object recognition can provide increased interpretive and predictive power.

Beyond robotics, the symbol grounding problem [37] considers how words and concepts (symbols) can gain meaning, which has significance for artificial intelligence. Object classification permits the real world to be transformed into symbols, where objects are defined by learned sensory properties rather than limited dictionary definitions. The ability to reason about objects brings AI closer to more general capabilities.

1.2 Background

Object classification begins with the processing of some form of sensor data. Colour vision has been the mainstay of computer perception for obvious reasons. The sensors are relatively inexpensive, they produce the sort of visual data humans are familiar with, and the data itself contains a wealth of information in colour, texture and shape.

Point feature descriptors are commonly employed in vision to find matching image patches. Features in general convert a portion of sensory data into a vector that *represents* some aspect of the data, making it more amenable to comparison. As image data varies greatly due to changes in viewpoint and lighting, image features such as the Scale Invariant Feature Transform (SIFT) [52] seek to limit this variation, allowing a given image patch to be recognised under varying conditions. This means that a given object would be more likely to have the same features when observed under different conditions, making it more suitable for classification. However, this *invariance* is limited by the 2D nature of images, where changes in viewpoint eventually change the image structure significantly. The three dimensional world is transformed into a two dimensional space of pixels, from which the world must be interpreted.

With 3D data, the three dimensional world is sensed, allowing surface geometry (i.e. 3D shape) to be used as the basis of recognition. This is an object-centric property, independent of viewing angle and lighting. While lacking colour and texture, the 3D shape of an object is often very informative by itself. However, geometry is not sensed, it must be inferred from 3D points; this process is a primary focus of this thesis. Several 3D features exist which represent the geometry, elaborated in Section 2.3. In this way, point features can facilitate matching parts of objects, as shown in Figure 1.2.

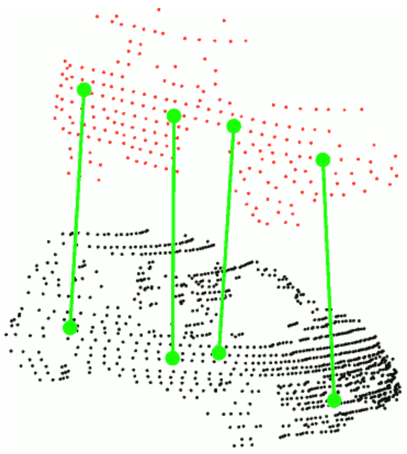


Figure 1.2 – Point feature matching between two point clouds of vehicles.

The focus in this work is on a relatively low quality form of point cloud, where existing features from the literature are not well suited. In the context of UGVs, vehicle motion and dynamic objects in the environment necessitate that range images be taken at a video-like frame rate. Sensors such as the Velodyne provide range images at up to 20 Hz, but are limited to only 64 'pixels' in height (Figure 1.1). In general, higher resolutions and point densities are difficult to achieve due to cost, image acquisition time and outdoor distances (Section 2.1.2).

The single-viewpoint nature of sensing in a range image leads to sparse 3D points and occlusion, which change depending on the position of the sensor. For example, consider the regions in Figure 1.1 that are in shadow, and distant surfaces that are facing away from the sensor. A good shape representation must allow similarly shaped regions to be identified as similar, even across different sensing conditions, and in the presence of partial geometric data, which is often the case.

This *viewpoint invariance* is important in the context of classification. First, consider the task of recognition, which is defined as identifying a single, specific object that has been previously observed. As the object may appear at any relative pose, an effective representation would allow the object to be identified regardless of pose. Classification goes further than recognition, as it aims to identify a particular object instance that has never been seen before, given a set of previously observed training objects in various categories. The properties of that object are used to place it in the correct category. Extracting these properties from the raw sensory data in a robust manner is of prime importance to robust classification. From [8]:

> One advantage [of invariant features] is that [they] can correctly extrapolate well beyond the range of transformations included in the training set. However, it can be difficult to find hand-crafted features with the required invariances that do not also discard information that can be useful for discrimination.

In summary, 3D sensing permits the 3D shape of objects to be used for classification. In order to compare parts of objects using a sensed 3D point cloud, informative, viewpoint-invariant features are needed. These must capture discriminative shape information, while being robust to the non-ideal nature of range images sensed in dynamic urban environments.

1.3 Contributions

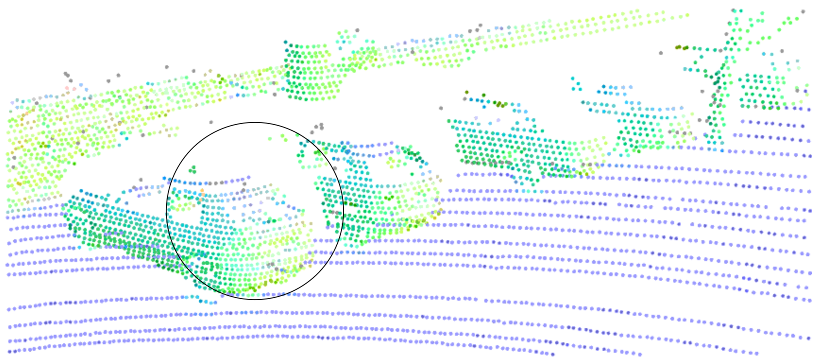
The primary contribution of this thesis is a novel feature that aims to achieve the above. Detailed in Chapter 4 and shown here in Figure 1.3, it consists of a set of parallel 3D lines probing the environment for a surface. As such, the feature is called the *line image*. When lines intercept the interpolated surface, the depth is recorded, producing a local 2½-D representation. Otherwise, if they go behind an occluding surface, or encounter a large gap in the data, *unknown* space is registered. In addition, *visibly empty* space is registered as a different item, as it provides useful information about the silhouette of an object. The unknown space changes with viewpoint, but the surface and visibly empty space remains constant and provides distinguishing information.

The line image relies on some lower level building blocks, and so these are presented first in Chapter 3. These address assumptions of data density and quality, with the following contributions.

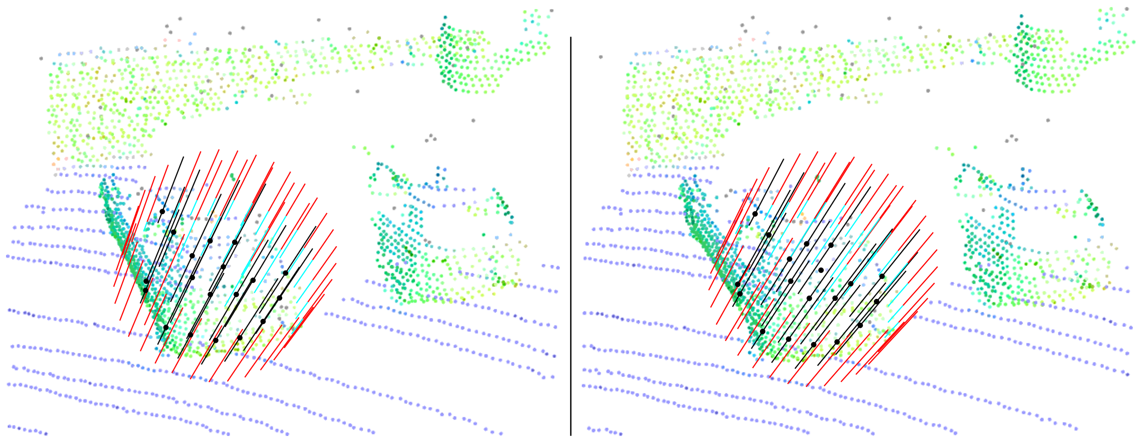
- The use of the range image structure to facilitate the selection of points within a 3D region.
- The use of a graph structure to reason about sufficient sampling and neighbourhood selection, permitting surface normal computation.
- A keypoint algorithm to identify positions and orientations on objects for feature computation, where orientation stability is used as a criteria.

The line image is then detailed in Chapter 4, comprising the following contributions.

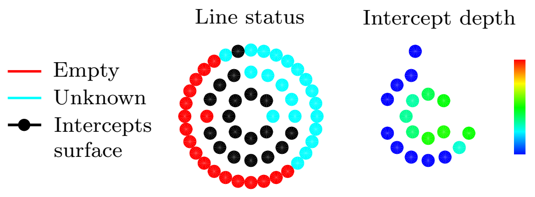
- Efficient occupancy detection along a 3D line using joint reasoning in the range image and 3D point cloud.
- An approximate, fast surface interpolation method using PCA.
- The line image formulation using the above, with a distance metric to allow regions to be compared consistently.
- A dataset is contributed, consisting of 588 urban objects across 14 classes, scanned with a Velodyne.



(a)



(b)



(c)

Figure 1.3 – (a) The rear of a parked car, where a line image is to be computed. (b) A line image computed on the region, with lines and black surface intercepts. This is a cross-eyed autostereogram; to see in 3D, cross your eyes until you see three images, then focus on the centre one. (c) Each line is summarised by the bottom 2D figure, showing which were completely in empty space (red), which found a surface (black, with associated depths), and which encountered unknown space (cyan) behind the car.

- A feature comparison study with spin images, NARF and FPFH, applied to object classification with a k-nearest neighbours classifier.

Chapter 5 then focuses on processing line images for use in other classification algorithms, contributing the following.

- An approach to clustering partially-missing line image data, using an asymmetric distance metric and affinity propagation.
- The application of multi-class supervised latent Dirichlet allocation to 3D object classification, using multiple scales of analysis.

1.4 Thesis Overview

Chapter 2 presents the background of this thesis. The concepts of range imaging and the properties of the data are described, followed by segmentation techniques. Existing global and local 3D features are outlined, motivating the use of local features. Methods for classifying objects from a collection of local features are described.

Chapter 3 starts from the initial processing of computing simple properties on the raw data, forming basic building blocks that are often required prior to computing detailed local features. Region selection and surface normal computation are explored, and efficient and robust methods are contributed. Keypoints are then developed, which define local positions and orientations for computing complex features.

Chapter 4 examines local features, briefly describing several existing techniques. The line image feature is then described in detail, along with a distance measure. An object dataset is presented and then classified using a k-nearest neighbours (k-NN) classifier. The line image is compared with existing features in classification, and shows improved performance at smaller, local scales of analysis.

Chapter 5 develops further processing for the line image in order to use more effective classifiers. First, clustering is examined, with affinity propagation applied to the dataset to produce a set of line image exemplars. Second, in order to classify objects from multiple scales of constituent local features, supervised topic modelling is applied to the clustering results. This approach shows a 10% improvement over k-NN.

Chapter 6 presents conclusions and potential future work.

Chapter 2

Background

Many approaches exist for classifying objects from range images, with relevant techniques from other 3D processing fields, as well as standard colour vision. The focus of this thesis, representing shape, is one step in a pipeline of processing steps, from the raw range image to a set of segmented, classified objects. This chapter provides context for the pipeline that is developed in this thesis. It justifies the approaches taken by evaluating techniques from the literature in the context of sparse range imaging.

The following items will be discussed in order of processing, summarised in Figure 2.1.

- Range Sensing. The nature of sparse range imaging will be described, and why it presents a challenge for 3D perception.
- Segmentation. This stage separates each object from its surrounds. This thesis does not focus on segmentation, as existing techniques can perform it prior to complex shape representations. This approach will be justified, and it's effects on classification outlined.
- Forming efficient, informative representations of objects. The raw data exists as a set of 3D points, which must be compared with prior models. This is facilitated by transforming the object into feature vectors, which represent the 3D shape, and allow it to be efficiently compared.

- Matching and learning from these representations, ultimately allowing classification.

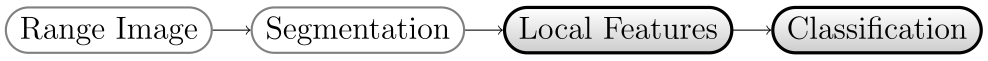


Figure 2.1 – A rough outline of the processing pipeline.

This thesis focuses on developing shape representations for object classification, which is performed within the shaded blocks, in the context of the entire classification pipeline.

2.1 Range Sensing

A number of range sensors are commonly used for machine perception; this section will provide a brief overview of their operating principles and the type of data they produce. This thesis focuses on a form of data called the range image, which will be described here. In particular, low resolution or sparse range images are a focus, and are relevant due to the limitations of denser sensing in robotics.

LiDAR sensors are suited for outdoor use at reasonable ranges (eg. 1 - 100 m) for field robotics in urban environments. They are a time-of-flight (TOF) sensor, and operate by firing a short, near infrared laser pulse, and measuring the time it takes for the light to return. The range and angular orientation of the laser beam allows the calculation of the 3-dimensional location of a reflecting surface with respect to the sensor. By taking multiple range readings at varying orientations, a 3D point cloud can be constructed.

Depending on the sensor, LiDARs can sense depth at centimetre-resolutions, and fire at many thousands of times a second. However, in comparison to cameras, LiDAR sensors can be significantly more expensive. They are also an active sensor, using more power and adding energy to the environment.

A common arrangement is a single laser directed at a spinning mirror, which measures range at close intervals to produce a ‘scan line’, or 2-dimensional slice of the world.

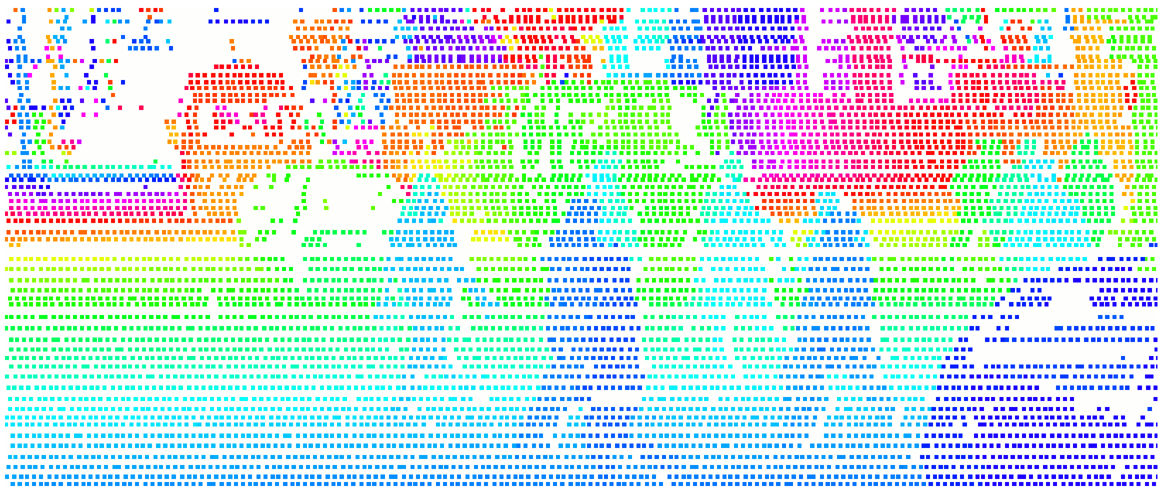
Multiple slices can build up a 3D view of the world, but this process takes time, which limits it's use in dynamic scenes. However, within the past decade, a number of off-the-shelf sensors have become available which provide a full 3D scan of the environment, at a rate which allows for real-time perception in robotics. One form of this 3D scan is the range image, which provides additional structure to the 3D point cloud, and is a focus of this thesis.

2.1.1 Range Images

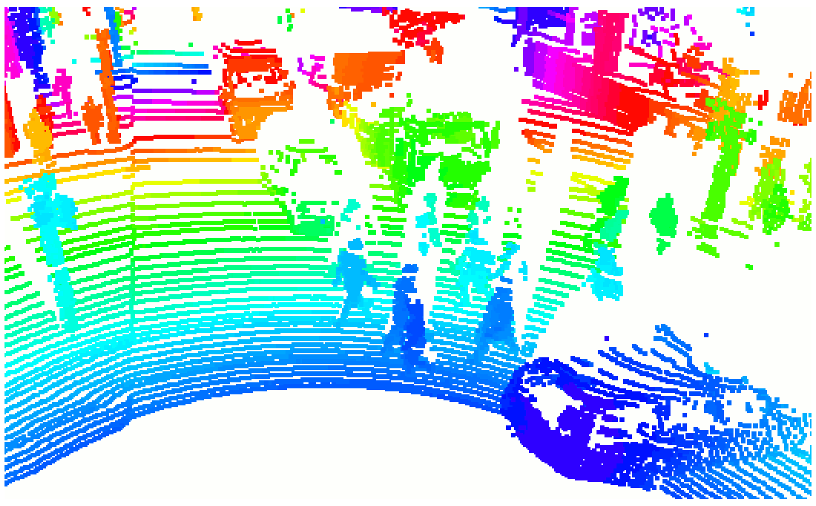
When a set of range measurements are made from a single location in a 2D pattern, the data can be organised as a range image (Figure 2.2a). The image coordinates can be spherical (azimuth and elevation), often encountered in LiDAR, or other mappings such as perspective for stereo vision. For spinning LiDAR sensors, unlike a camera, each range measurement does not necessarily take place within a precisely regular grid of pixels. Instead, to keep generality, a range image will be defined as a set of *continuous* (rather than discrete) 2D points, with a range value for each. This is known as a 2½-D representation. This thesis primarily uses LiDAR, and so 2D space and 2D points, in the context of range images, will refer to the continuous elevation & azimuth polar space.

The range image representation is an alternative to the 3D point cloud, and has some additional implicit information. One is the notion of neighbouring data points and scanning density. Two neighbouring points in the 2D range image space, with no further points between them, define the density of data in that region. Often the polar density of this 2D image data is quite regular, for example with LiDAR sensors spinning at a constant rate, producing a depth reading at a constant frequency. This is in contrast to the 3D data, where the density is a function of the scene. Even on a scanned surface, the 3D Cartesian density varies by the relative angle to the surface.

Another benefit of the range image is the implicit occupancy information. A sensed 3D point not only indicates the location of a surface, but also that there is empty space along the laser ray, between the sensor and surface. The concept of occupancy is about



(a)



(b)

Figure 2.2 – (a) The 2D points from a Velodyne (azimuth & elevation) coloured by depth. They do not form a rectilinear grid of discrete pixels. (b) The associated 3D points.

determining whether each region in 3D space is empty, occupied, or simply unknown due to no laser rays passing through the region. An occupancy grid map [81] is a way of storing this information for arbitrary rays, where ray tracing identifies which grid cells are empty. In a range image, each 2D point is essentially looking down the ray. The range image explicitly denotes a surface, where the 3D Cartesian space in front of it is empty, and the space behind it is unknown. This is contrasted by an unstructured, Cartesian 3D point cloud, where the difference between empty and unknown space is not explicitly represented and therefore indistinguishable. Occupancy forms an important part of shape representation, which is introduced in Section 2.3, and fully explored in Chapter 4.

2.1.2 Sensors

A number of sensors produce range images, including various LiDAR sensors and stereo vision sensors, of which a brief overview will be given here. It is also worth noting which common 3D sensing arrangements do not produce range images, and why these present their own problems. While 3D sensors have steadily increased in capabilities, those that produce sparse range images, as opposed to dense, accurate images, are likely to be an attractive sensor for field robotics for some time due to typical trade-offs in acquisition speed, point density and cost. Costs denoted here are for rough comparison only, to highlight significant differences in affordability.

A recurring notion in sensing is that of a 'scan' of the scene, which is defined here as the set of range measurements that define a single 3D snapshot of the scene, typically to be repeated. In this sense, a scan is like an image or frame in a video.

2.1.2.1 The Velodyne Sensor

The Velodyne sensor and the data it produces will be the primary focus of this thesis, and so will be described here in detail. The type of data it produces provides a relevant challenge for object classification in field robotics that extends beyond this one sensor.

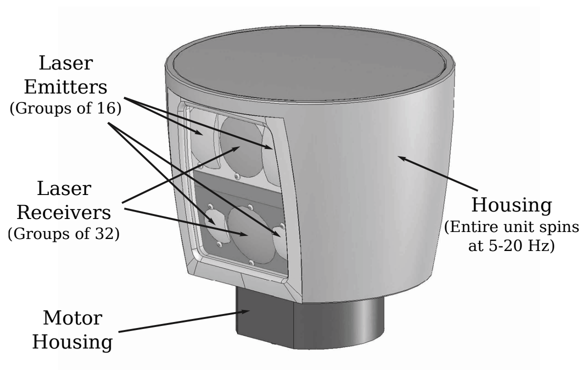


Figure 2.3 – The Velodyne HDL-64E, from [86].

The sensor contains 64 fixed lasers arranged in a block (Figure 2.3), each with a fixed elevation angle ranging from 2° above horizontal, to 24.33° below horizontal.

The entire block rotates at a fixed speed (5-20Hz), providing a 360° field of view, and producing 1.3 million points per second in total.

The sensor originated from the 2005 DARPA Grand Challenge, with it subsequently being used in the 2007 DARPA Urban Challenge by five out of the six finishing teams. The urban challenge involved driving autonomously through 97 km of urban roads with other moving vehicles. Teams used Velodyne data for ground mapping, obstacle detection and tracking [56, 57, 85, 49, 12]. The sensor retails for approximately $80 000 USD[1], and is 13.15 kg in weight. A lower-cost, lighter, 32-laser version was released in 2010.

The data from a single 360° rotation is denoted here as a *scan*, and is a convenient way to delimit data which is, in reality, a constant stream. At a rotation speed of 20 Hz, about 66 667 range measurements are made each scan, of which about 50 000 result in a valid 3D point in a typical urban scene. The operating range is 1-50 m for low-reflectivity surfaces, up to 120 m for highly reflective surfaces, with a 2 cm accuracy [86].

However, the 64 lasers must be calibrated to precisely determine their position and orientation in the sensor block. Numerous methods have been used to improve upon factory calibration [50, 36, 53]. Imperfect calibration can result in alternating laser scan lines being erroneously offset in range and azimuth, effectively reducing the sensor accuracy.

While each scan line produced from one of the 64 individual lasers has a fixed elevation, the spacing between them is not equal (see Figure 2.2). In the S2 model, spacing is closer on the higher elevation lasers, as well as minor variations determined from calibration. In azimuth, each laser fires at approximately a fixed rate, but firings do not occur at regular, repeatable azimuth locations, and the spacing has slight variations. This means that a Velodyne scan cannot be converted to a regular grid of pixels without significant data loss, errors due to interpolation, or gaps in the image. This motivates treating range images in the general sense, where continuous 2D points are considered instead of discrete 2D pixels.

[1]Informal quote from Velodyne in 2012

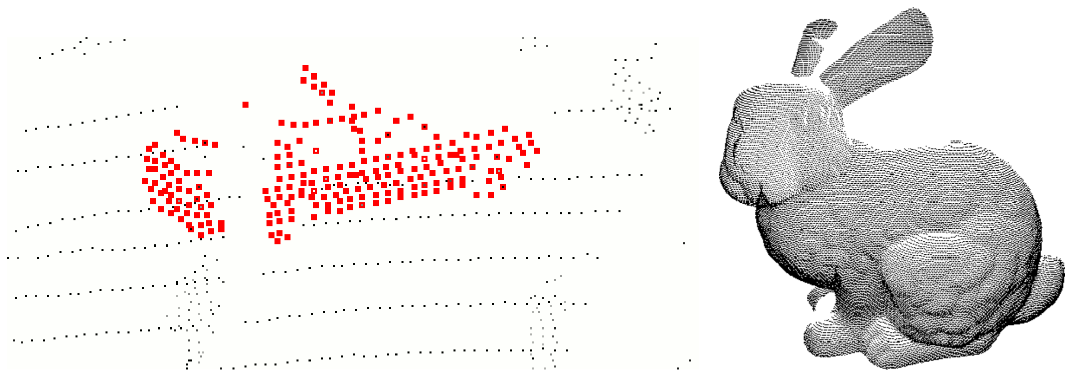


Figure 2.4 – (a) A typical car (red) in a Velodyne scan, 219 points. (b) Stanford bunny, 40 000 points.

A Velodyne scan will be described in this thesis as a *sparse* range image, due to the relatively low data density produced in an outdoor environment. A typical car (Figure 2.4a) 21 m away has only 219 points, and a person 13 m away only 90 points. This is in contrast to other 3D sensing applications such as scans of manufactured parts, sculpture or architecture. Many 3D shape analysis tools begin with benchmark objects such as the Stanford bunny (Figure 2.4b) [83], which contains over 40 000 points. More details about the point clouds of objects will be given in Section 2.1.3.

This thesis will focus exclusively on processing data from the Velodyne HDL-64E. However, in order to demonstrate how the sparsity of range images is a general limitation in outdoor dynamic sensing, a few other common sensors will be described. These provide different trade-offs in point density, scanning time, range and cost; they highlight the limitations of denser 3D sensing.

2.1.2.2 Tilting 2D LiDAR

The least expensive LiDAR sensors consist of a single laser directed at a spinning mirror, producing a 2-dimensional slice of the world. To acquire a full, 3D range image, the 2D laser can be attached to a servo motor to rotate it along the perpendicular axis. The recent PR2 robot platform [34] utilises such a setup, with a horizontal planar 2D LiDAR on tilting servo. It uses a Hokuyo UTM-30LX sensor, which is about \$5600

USD, and weights 210 g. It has a 30 m sensing range and makes 43 200 measurements per second in a 270° horizontal field of view. The tilting period and angular range can vary, with faster scanning reducing data density. This is ideal for the PR2's domain of relatively close-range, indoor environments, and tasks such as manipulation that can wait for several seconds for a dense scan. However, dynamic scenes require faster scan rates, especially in fast-moving outdoor environments. This results in a lower data density. The measurement rate is 30 times lower than a Velodyne, resulting in an equivalent drop in data density for the same scan rate and field of view.

2.1.2.3 Stereo Vision / Structured Light

Stereo vision sensors consist of two cameras at a known offset, and produce a range image by computing the disparity between visual features. The operating range is limited by the disparity resolution, and areas without sufficient texture cannot be processed.

More recently, the Microsoft Kinect [30] has enabled widespread, low-cost range imaging, priced at under $200 USD. It is a structured light stereo camera, where a near-infrared laser projector is used to project a distinct pattern onto the scene. An infrared camera detects the projected pattern, distorted by the geometry of the scene. The disparity against the known, undistorted pattern is computed internally, which provides a measure of depth. This addresses the issue of non-textured surfaces, and simplifies the depth computation. In addition, a co-located colour camera allows colour and texture to be applied to the range image, producing what is known as RGB-D data. The range image resolution is 640×480 pixels sensed at 30 Hz, or 9.2 million points per second.

However, this sensor cannot be used outdoors in the daytime, as sunlight interferes with the infrared image. The range is also limited to approximately 5 m, with the resolution of each depth measurement reducing significantly with range [45]. Its use has primarily been in indoor robotics, where these limitations are less problematic. It is worth noting that the techniques in this thesis could be applied to this low-cost, popular sensor in indoor settings.

2.1.2.4 Dense, Slow 3D LiDAR

Long range, high accuracy and high scanning density are achieved with static, slowly scanning LiDAR sensors such as the RIEGL LMS-Z620. This sensor has a vertically spinning 2D scan line attached to a slowly rotating body, with an optional camera mounted for colouring each point. The laser produces up to 11 000 measurements per second, forming a dense 360° scan of approximately 1.7 million points in 2 minutes, with ranges up to 2000 m. One such scan is shown in Figure 2.5. These sensors are also very expensive, at approximately \$250 000 USD, and find their use primarily in surveying. While the data density is ideal, the cost and long scanning time exclude its use in real-time dynamic robotic perception.

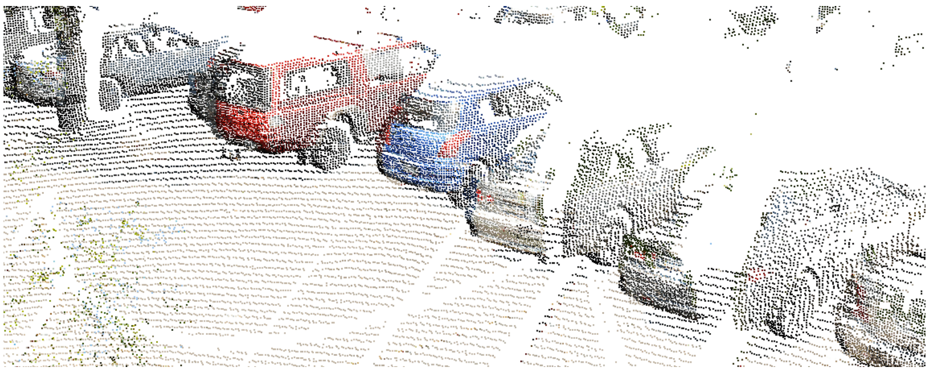


Figure 2.5 – A dense, coloured RIEGL scan of a car park.

2.1.2.5 Accumulated LiDAR with Localisation

An example of a common 3D sensor setup that does not produce range images is where an inexpensive 2D laser scanner is rigidly mounted to a mobile robot platform. With sufficiently accurate localisation data, for example with an inertial navigation system (INS) and global positioning system (GPS), the separate 2D laser scan lines can be accumulated into a single, global reference frame. One such arrangement is with a vertical scan line, swept over the scene as the robot moves, or a horizontal line (known as a push-broom arrangement). This approach is common in map-building, and can produce dense 3D point clouds with relatively inexpensive hardware.

However, this form of data has several distinct differences to range-imaging as earlier described. To begin with, data that is very similar to a range image could be produced if the robot drives in a straight path. Rather than tilting the 2D LiDAR to produce an image of elevation and azimuth, such an image would have coordinates in elevation and the forward axis of motion (for a push-broom setup). However, more typical driving will introduce angular and linear motion, leading to overlapping rays, and 3D points that can no longer be mapped to a 2D image space. In addition, any given 3D region of the scene will have a variable density between scan lines, and scan lines may overlap in irregular ways. Noise in localisation and sensing warrant more complex filtering or surface models to fuse the data. Moving objects must be removed from the map, and any analysis of these must be treated completely differently. The concept of occupancy is lost, unless the full ray of each point is stored, for example in a 3D occupancy grid. As such, shape analysis on this data cannot be treated as the same class of problem, and many algorithms in this thesis cannot be directly applied to this form of data.

On a related note, accumulating several sparse 3D range images suffers from problems such as these. While the data density and coverage may be increased, the very nature of the data is changed, and presents problems of data fusion. Processing becomes centred on an accumulated map, where tasks like localisation and tracking become prerequisites for classification.

This thesis is about interpreting single scans, or range images acquired at relatively fast speeds such as to allow real-time robotic perception in dynamic, 3D environments. Ideally, the algorithms would be robust to any issues caused by low data density, such that object classification can occur immediately instead of waiting for the ideal data to be collected and merged. While Velodyne data may be too sparse to distinguish some objects, many common urban categories such as pedestrians, signs, and vehicles should be distinct enough. A human can distinguish these categories by visually inspecting the point cloud, so it is possible.

Furthermore, operating on a per-scan basis allows several tasks to be done in a pipeline of operations (such as segmentation, tracking and classification). This greatly simplifies the software implementations in a robotic system, where the coupling between components is a simple per-scan input/output.

Data fusion from multiple scans can potentially occur later on in the pipeline, and could benefit from the single-scan information. In [80], the position of each object segment was tracked, and the pool of per-scan object descriptors were used in classification. Tracking could use the single-scan shape representations for making correct associations. These representations could be improved in a filter, once properly aligned with their corresponding observations in subsequent scans. Object classification can proceed immediately, and if the data is found to be insufficiently distinguishing, the robot can then proceed to gather more information as required.

2.1.2.6 Future sensing capabilities

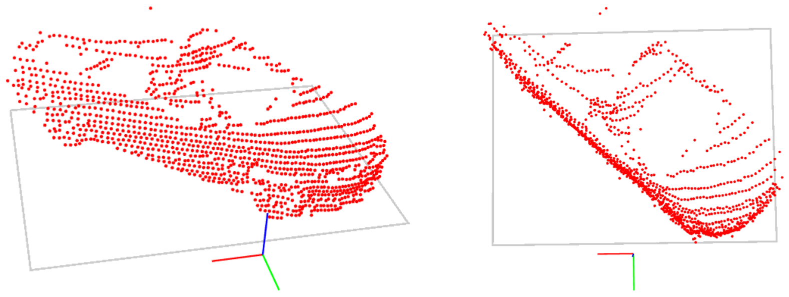
The capabilities and affordability of LiDAR and other 3D sensors have steadily improved, and so a valid question is whether it is worth addressing the issues of sparse, noisy range images such as those from the Velodyne. In particular, as the automotive industry becomes increasingly interested in LiDAR for assisted and autonomous driving, sensor costs may drop with economies of scale. However, there will likely always be trade-offs in affordability, scan density, accuracy and range. The Velodyne is already a very expensive sensor in comparison to visual cameras, and can easily be the most expensive component of a robotic system. More affordable sensors are likely to have even less resolution. Stereo camera-based sensors like the Kinect are significantly cheaper, but the physics of their sensing make high resolution, outdoor, long range measurements difficult. The result is that sparse range images are an attractive sensing modality for dynamic sensing in field robotics.

2.1.3 Sensed Objects

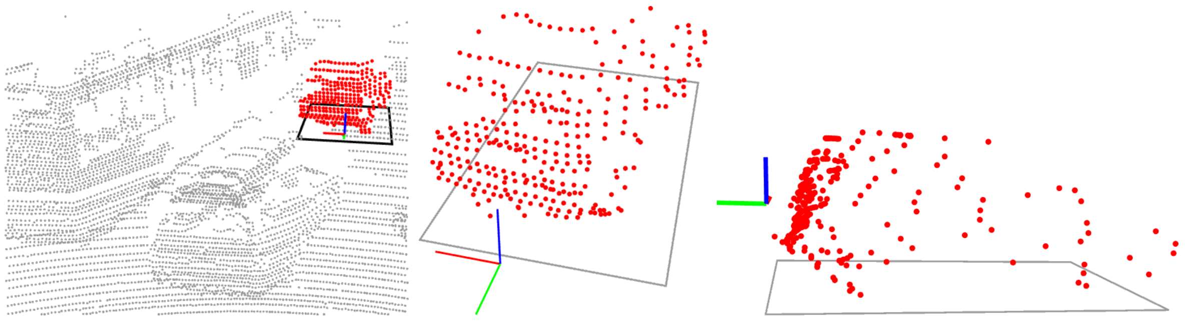
Considering the choice to focus on sparse 3D LiDAR sensing, and the physical constraints of this modality, this section will describe the properties of the point clouds of objects scanned with a Velodyne. The primary properties are the levels of occlusion, density and noise, and they must be considered when designing algorithms for representing and matching objects. Some typical objects demonstrating these are shown in Figures 2.6 to 2.8.

All objects self-occlude, meaning the side further from the sensor is never visible in a single range image, because it is in the shadow of the near side. More distant objects are additionally occluded by shadows cast from nearer ones.

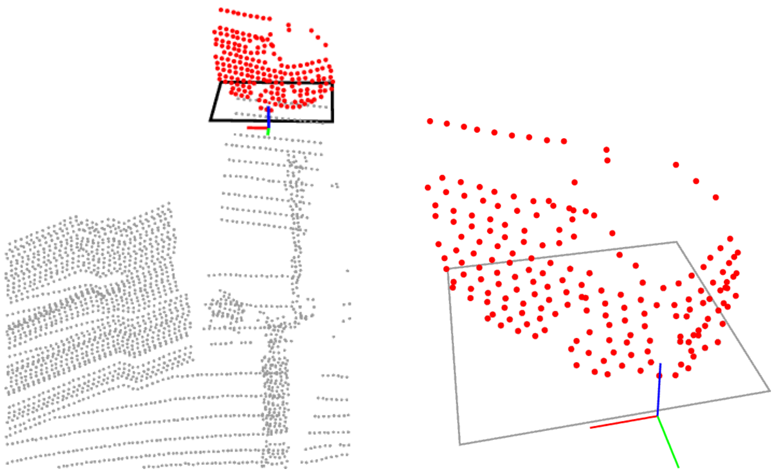
The density of object point clouds vary significantly, primarily due to the distance to the object. Within an object, the density varies as a function of the relative surface angle to the sensor. This can produce very sparse points along buildings and car sides at a steep angle. Noise also exists, in the range of each laser, and between lasers due to slight calibration errors (see Figure 3.4).



(a) Well-sampled car, exhibits self occlusion.

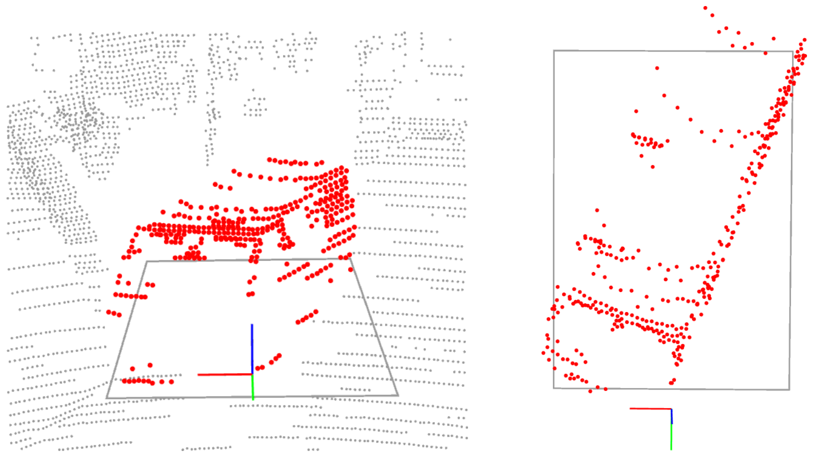


(b) 4-wheel drive vehicle, with dense rear and sparse side, due to orientation.

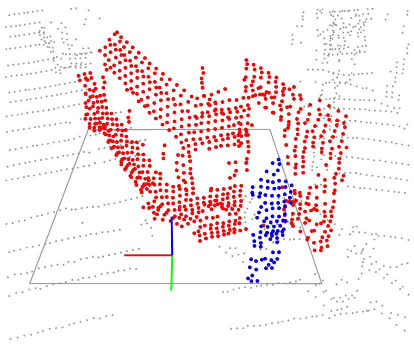


(c) Van occluded by a foreground object.

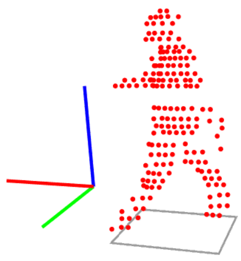
Figure 2.6 – Point clouds of objects come in varying quality, shown in the above and the following few pages. Each sub-figure is a single object shown from multiple views, with the originating full scene also shown in grey. The bounding box is shown at ground level, oriented towards the sensor. The green axis also points to the sensor, with the blue axis facing vertical. (a) The best object scans (such as this) are still self-occluding. (b) Surfaces at an increasing angle to the sensor become sparse. (c) Objects can be occluded by others.



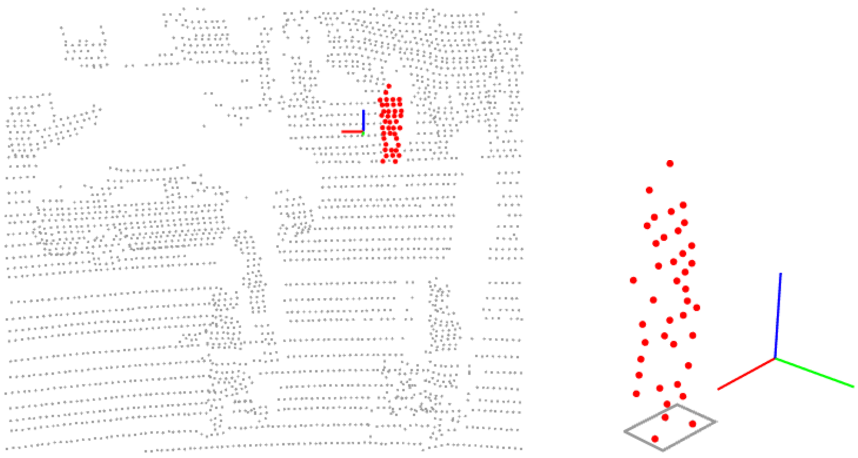
(a) Van irregularly occluded by robot antenna.



(b) Van with a variation in shape (open door) and clutter (person).

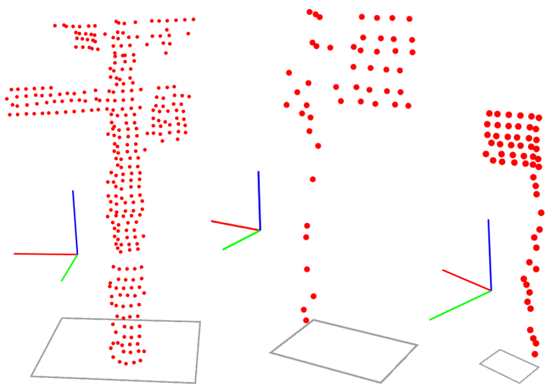


(c) Person, well-sampled.

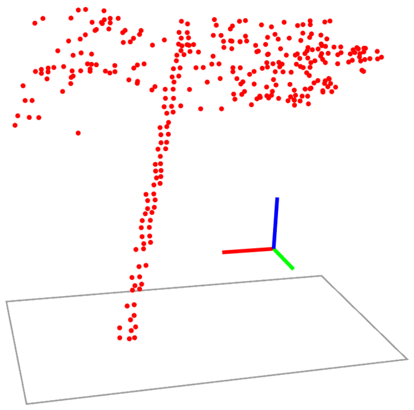


(d) Person, distant and low resolution.

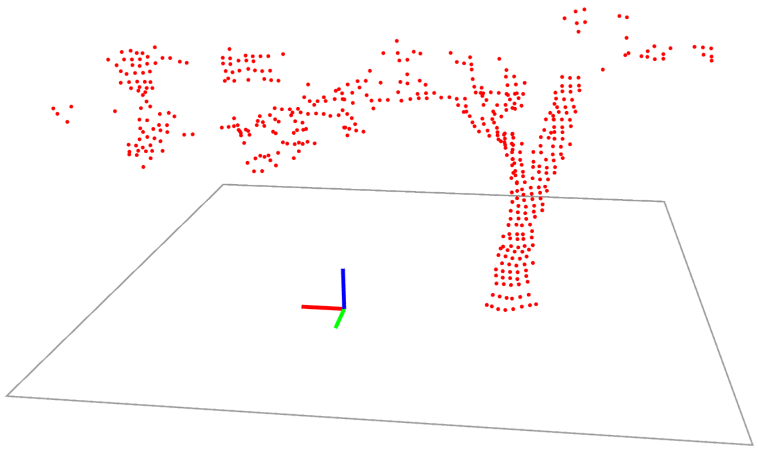
Figure 2.7 – The range image can also be missing data, as in (a). In this case, an antenna on the robot is positioned closer than the minimum operating range of 0.9 m, and so no object is observed 'in front' of the van; it is still recognisable to a human. This also occurs due to non-returns, where the laser beam does not reflect back to the sensor. (b) Objects can also vary in shape, or be cluttered with others, such as this van with the back door open, and a person (blue). The van and person are segmented into the same object by the mesh technique [23] due to their proximity. (c) A person can be clearly discerned at close range, but a person across the intersection in (d) forms only a vague shape of the correct size.



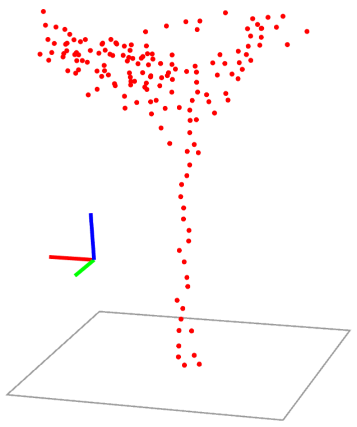
(a) Left to right: two traffic lights (near & far), and a near traffic sign.



(b) Tree.



(c) Large tree.



(d) Distant tree.

Figure 2.8 – (a) Distant traffic lights and nearby traffic signs have only a thin line of points on the pole. (b) (c) (d) Trees show a great deal of variation in shape, with the canopy only partly visible as a scatter of points.

Figures 2.6 to 2.8 illustrate the levels of occlusion, density and noise present in the data. These are a product of range based sensing in real environments. Algorithms that analyse the 3D shape of objects using this data must deal with these complexities.

2.2 Segmentation

One part of classifying an object is in separating, or segmenting it from the scene. This thesis does not focus on segmentation, instead relying on existing techniques which will be briefly described here. However, the assumption of segmentation influences how other processing aspects are done. Failures in segmentation occur as either over-segmentation, where single objects are split up, or under-segmentation, where separate objects or regions are joined into one segment. Segmentation in the context of object classification can be approached in one of two ways:

(1) Segmentation first. Simple properties of the data can be used to segment each object, allowing the whole object to then be analysed and classified.

(2) Segmentation last, or at the same time as classification. Small regions of the data can be analysed and potentially even classified as belonging to object classes, followed by segmentation as a final step of grouping parts together.

Another way to consider this is how much processing is required in the segmentation step, and to what degree it overlaps with object representation and classification. When segmentation occurs as a first step, errors can carry over to the next step of object representation, discussed in the next section. However, this approach subsequently allows the whole object to be analysed and a single representation formed, or for the analysis of smaller regions within the object to be pooled together to perform classification.

Segmentation is an extensive topic in computer vision, with the operation being similar to the 3D case. It is often posed as a graph clustering problem, where vertices of the graph represent pixels within the image, and edges denote some distance [78]. This can be be based on pixel properties such as intensity and colour, or slightly larger regions of textures or edges. A common approach is to then use Markov random fields (MRFs) to solve graph cuts to segment the image [15].

However, these simple per-pixel properties are often insufficient for semantically meaningful classification, as objects and regions contain a variety of edges and colours. More successful approaches use more information, such as complex local features (eg. SIFT [52]), or regions grouped into superpixels [64]. In [33], superpixels form nodes in a graph, and are classified with a conditional random field (CRF) [47], performing segmentation and classification at the same time.

In 3D data, segmentation can be based on simple 'per-point' properties such as physical separation, surface connectivity or normals; these are often clearer than edges and colours in visual images for separating whole objects. Computing these simple properties is the subject of Chapter 3. Several segmentation techniques have been successful on Velodyne scans using these simple 'segmentation first' point-based properties.

Surface concavity can form the basis of separating regions [58], where highly concave angles between surface normals separate surfaces. For example, when the surface of an object meets the ground, it often forms a concave angle. This algorithm relies on good surface normals, and can over-segment non-smooth surfaces, such as typically caused by vegetation for example. It can also under-segment when points of contact between two objects are not sufficiently concave.

Another approach is to first isolate the ground, followed by clustering connected regions [23, 56, 51]. This assumes all objects are physically separated, except for the ground, which is explicitly modelled and removed first. A number of different ground detection techniques can be used [55], with a mesh based approach from [23] found most suitable for Velodyne data (this mesh, or graph, is also used in Section 3.1.2). This is shown in Figure 2.9. Real time performance has been achieved due to the relatively simple operations [23]. While many objects can be clearly segmented with these techniques, results can degrade with clutter, as objects too close to each other have connected voxels, resulting in under-segmentation. Very sparse or missing data can lead to over-segmentation, where voxels are not sufficiently close, or are broken up by occlusion.

Similar to techniques applied in vision, a Markov network can be used with local 3D statistical features [60] to perform segmentation and classification at the same time, producing simple categories. While 'segmentation last' approaches such as these consider more information to solve both problems jointly, there are a number of advantages to using the 'segmentation first' approach. First, by keeping segmentation

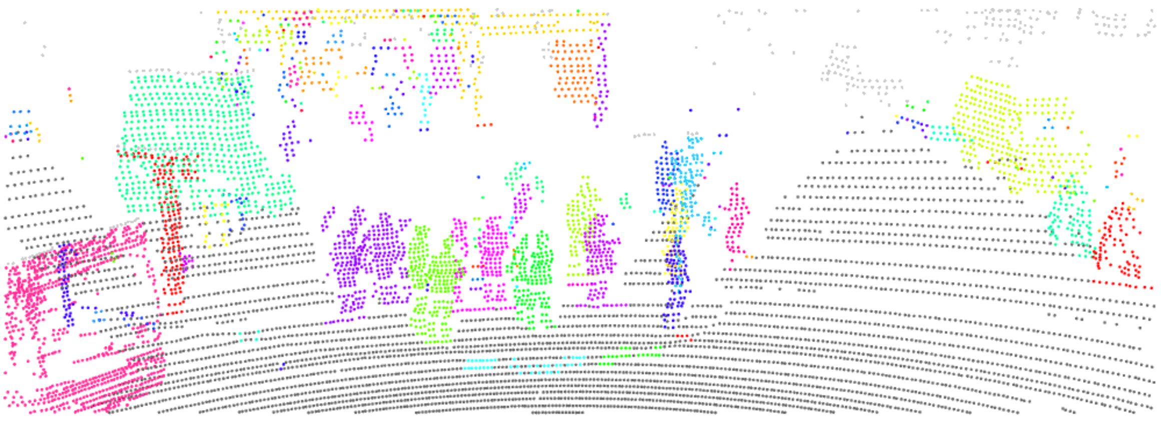


Figure 2.9 – Mesh-segmented scan [23] of some pedestrians (centre), cars (right), bus (left), buildings (upper-left).

functionally separate, it can be used to reduce the large amount of data prior to more intensive processing. This allows complex features to be computed solely on objects rather than the whole scan, reducing computation time. It also permits other perceptual tasks to run independently, such as tracking. In the DARPA Urban Challenge, Team Cornell used segmented objects as the starting point for more advanced perception such as tracking [56]. More recently, Stanford's vehicle takes this approach with object classification [51].

Another benefit to this approach is that, when classifying an object, the assumption that a set of parts belongs to an object allows techniques that are not otherwise possible. This is further elaborated in Section 2.4. MRFs are limited to modelling relationships between directly neighbouring components, and so cannot do inference between all parts of an object without requiring significant computational resources, or unless there are few parts[2].

This thesis largely presumes segmented objects as the input to object representation and classification. As such, the possibility of incorrect segmentation must be considered during these tasks.

[2] Reducing an arbitrary object down to a few parts in a repeatable manner is a subject of this thesis, and so Markov network approaches may be an area of future work.

2.3 Object Representations

This section will outline existing techniques that take a 3D point cloud of an object, and create from it a representation which (1) summarises relevant information and (2) facilitates the comparison of objects. Of particular focus is how robust these algorithms are to sparse range imaging in the field, where varying, non-ideal data is present, as shown previously in Section 2.1.3. There are many techniques for representing a 3D object [17, 79]. Many originate from 3D modelling, where the task is to match complete, dense, meshed 3D vertices. However, these present problems when applied to noisy, incomplete range image data. To represent objects, three broad approaches exist: global features, featureless alignment and local features. This last approach is a focus of this thesis due to its robustness and representative power, which will be illustrated in this section.

Feature vectors are a common method of comparing two point clouds, which transform the raw data to provide salient information for matching. Feature vectors are also known as descriptors, or signatures, and will be referred to here simply as *features*. The term *global* refers to computing the feature over the whole point cloud of the object, as opposed to a *local* feature computed on a small region of the object. Often, features can be applied globally or locally. Global features allow an entire object to be matched directly, but have limitations in generalising to variations in sensing and variations within a class. Local features allow objects to be compared piece-wise, which presents numerous benefits and challenges.

2.3.1 Aims

First, to define the aims of object representations such as features, consider the problem of determining if two separate 3D point clouds are similar, which is a task that often underlies object classification. In the simplest case, the two point clouds to be compared are cleanly segmented objects, and are scans of the exact same object taken at different positions. This is encountered in the task of object *recognition*, where a previously observed object must be found in the scene. A point cloud can vary by: (1) The pose (position and orientation) of the object relative to the sensor; (2) The point cloud density, particularly if an object is further away; (3) Occlusion, possibly by other objects, and always by itself. Methods of matching that are largely *invariant* to these will likely be more successful.

Beyond recognition, object classification has additional requirements of the matching process. Consider supervised classification, wherein a set of labelled *training* objects are available, and a new, unknown *test* object is to be classified. One simple method is to match the test object to each of the training objects. If matches can be made across largely varying conditions, less training data is required. In addition, the test object may have never been observed before, and may vary significantly from all training objects in the correct class. This intra-class variation motivates representations which allow generalisation.

In summary, a good object representation can match objects that are sensed from different positions, and general enough to match varying objects within a class. In light of these aims, object representations in the literature will be introduced, motivating the use of local features. Note that 3D descriptors are an active field of research, for example, with many newly developed algorithms being made available in the recent Point Cloud Library (PCL) [68]. More details on a few local descriptors is also given in Chapter 4, where they are used in comparative experiments.

2.3.2 Global Methods

Global features transform the entire point cloud of an object, allowing the whole object to be compared at once. This can simply be the extents of the object, or a more detailed feature vector summarising constituent shapes. Global properties of an object are often distinct even in low density point clouds, where smaller shapes are not salient or cannot be resolved. However, they are sensitive to prior segmentation errors, as over- or under-segementation has a significant impact on the global properties of an object.

A number of approaches collapse an object down into a histogram. Shape Distributions [61] sample the probability of a function of two or three random points on an object; for instance, the probability function of the distance between two random points. This is invariant to translation and rotation, and does not require anything beyond the 3D object points. Other approaches require surface normals to be computed from the point cloud, binning these in a histogram. For example, the Extended Gaussian Image [42], or orientation histogram, is defined on the surface of a sphere, partitioned into bins. Each 3D point (with a surface normal) is placed on the location of the sphere which has the same surface normal. In addition to the surface normal, the local curvature can also be used to define a histogram [38, 7].

A more recent global feature is the Viewpoint Feature Histogram [70], developed for object and pose recognition using noisy, sparse stereo data. It is part of a class of 'Point Feature Histogram' descriptors [71], which capture various point-to-point angular differences between surface normals. As the target application required the object pose for grasping, the Viewpoint Feature Histogram was designed *not* to be rotationally invariant. Instead, all training objects (various cups and bowls) were scanned at many orientations on a turn-table, with the matched object simultaneously determining the object pose. While this approach worked well for the task of recognising previously observed objects, the need to collect significant amounts of training data makes scaling difficult.

The above approaches result in a histogram, where different regions of the object are accumulated into a single bin. When a given region is not observed due to occlusion, some bin values will change, negatively affecting comparison. In addition, the spatial structure of the object is not intuitively preserved, with many arrangements potentially mapping to one histogram.

In contrast, another method of capturing global shape is by defining a set of spatial bins which encompass parts of the object, and performing operations over these bins. These include 3D Shape Histograms [1], where a sphere is aligned with the object centre, and split into bins along radial and angular sections. The object's 3D points are placed in these bins, forming a spatial histogram.

Many global features such as this require a central point and axes to align the object reference frame, called normalisation. For example, for the above 3D Shape Histogram to be repeatable, the correct centre point and orientation of the bins must be chosen across all instances of matching objects. These can be the mean point of the object's point cloud, as well as axes aligned to the vector of greatest variance, found from principal component analysis (PCA). However, these are not reliable under partial occlusion, or when an object's overall geometry does not provide a clear longitudinal axis. The mean point is also skewed towards regions of higher point density, such as surfaces facing the sensor. While the rotational alignment can be removed for some features by using Spherical Harmonics [44], a centering point is still needed, and occlusion is still an issue.

A different approach is to align the point clouds, directly determining 3D point-to-point or surface similarity [22, 24]. Various forms of the Iterative Closest Point (ICP) algorithm exist [67], which aligns two point clouds given a rough initial alignment. Several initial orientations can be tried, or local feature matches can provide an initial alignment using sample consensus [71]. This approach matches large scale structure without an exact predefined centre and orientation. However, correct alignment is not guaranteed, and is more difficult with occlusion.

In addition to matching objects in the face of occlusion and view-point changes, matches must be made across differently shaped objects in the same class. The notion of similarity captured by global spatial binning or alignment has limitations, as portions of the objects may be similar, but could never rigidly align. As a trivial example, compare a limousine to a car. While the front and rear portion may match well, the distance between them is rigidly fixed in the representation, preventing a match. Other classes such as building facades and trees can exhibit even more variability.

2.3.3 Local Features

The above issues motivate *local* shape representations, which allow objects to be compared piece-wise. This is a common approach in vision, where SIFT [52] is prevalent. By matching small regions within an object, issues of occlusion, bad segmentation and intra-class variation can be tackled. This comes at the cost of more complexity, as several new issues must be addressed.

(1) Specific regions must be selected for computing features and performing matching. An object can either be segmented into a set of exclusive patches, such as curves [20, 40]; or a set of overlapping spherical regions can be defined by *keypoints*. The latter approach is taken in this thesis, and addressed in Section 3.3.

(2) Given a set of local matches, a single class label must be determined for the whole object, addressed in Chapter 5.

In addition, the original requirements of matching across variations in pose, density and occlusion also apply to local regions, with the concession that not all regions have to match. Local 3D features are the subject of Chapter 4, where a more detailed overview is given, and a novel local feature, called the *line image*, is developed that addresses these requirements. A brief overview of existing local features will be given here to provide context.

The design of local features is similar to global features, which in many cases can simply be applied to a local region of points. The centre of the region, and the corresponding surface normal can provide a set of reference axes for alignment, e.g. for spatial bins. Some examples are: a histogram of relative surface normals [71], statistical descriptors such as eigenvalues from PCA [16], a 3D grid with moments computed on each cell [14].

The spin image [43] is a widely-used feature that is computed at a central point, with a precomputed surface normal. Each sensed point in the region is transformed into coordinates based on the surface normal (see Figure 2.10). This can be visualised as a 2D image, or grid, spinning about the surface normal, defining hollow cylindrical 3D bins. Sensed points are binned to form a 2D histogram. The resulting spin image allows different regions of points to be compared and matched, using 2D image processing techniques.

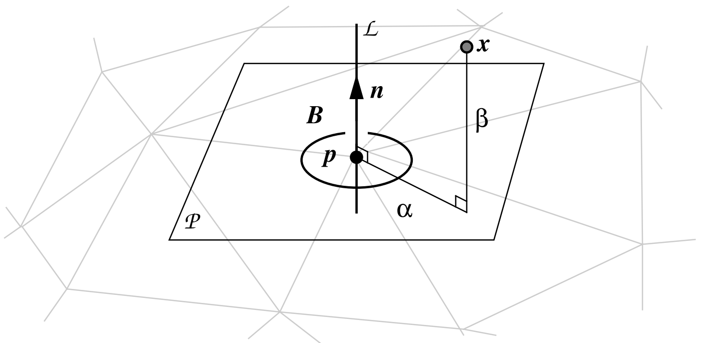


Figure 2.10 – From [43], the spin image computed at a vertex p of a meshed surface. For each sensed point x, the spin coordinates (α, β) are calculated, which then form a histogram. α is the distance of x from the normal line L, and β is the distance from the plane P.

The issues with local features such as these are the topic of Chapter 4, which describes their limitations when applied to sparse range image data, and a novel feature is developed to address these. By matching local regions, an object can be matched even if it is partly occluded. As objects in range images self-occlude in ways dependent on their relative pose to the sensor, the ability to make partial matches improves the viewpoint invariance of the overall classification system. In addition, large variations within each class can be tolerated, as only some, salient parts of the objects must match.

2.4 Matching and Learning

Given an object represented by a set of local features, this section describes how classification is performed, from local regions up to whole objects. This provides context for local features, which must be compared and manipulated to achieve the final goal of object classification. One desired aspect is the ability to generalise, from limited training data to classes with variability in sensing conditions and underlying shape. Ideally, the approach is also applicable to many more categories of objects.

There are two aspects to classifying an object given its parts: (1) performing inference on the parts (local feature vectors), followed by (2) classifying the object from these parts. The former step can involve classifying the object part, giving it an object class label (e.g. denoting it as belonging to a car, or a pedestrian). Then, the second step can involve pooling all part labels to determine the whole object's class label. Alternately, the inference on parts can involve clustering the training data, turning a massive set of feature vectors into a finite number of primary shapes. Statistics can be collected on this set of shapes, allowing more effective whole-object classification methods, such as topic models.

2.4.1 Local Feature Classification

The typical task of supervised classification is to determine the label of a new, unknown feature vector given a labelled training set. In this instance, the feature vector corresponds to an object part, and is to be classified as belonging to a certain object class. A feature vector can be considered as a point in a *feature space*, and comparing two features is posed as computing their distance in feature space. Many classification algorithms exist [8], and they generally seek to separate the feature space into class-specific regions.

The simplest method to classify an object is to first classify its parts independently, followed by a pooling operation. A simple, yet often effective classification method is k-nearest neighbours (k-NN), which finds the closest k training points (feature vectors) to the test point. The k nearest training points 'vote' on the class label to apply. When classifying an entire object from a set of feature vectors, these votes can be pooled; the class with the most votes is the classifier output. This applies to any discriminative classifier.

Similarly, classifiers such as Naive Bayes provide a class posterior probability for each part, and the posteriors can be multiplied together to arrive at a single object label. One approach that bridges Naive Bayes with k-NN is the Naive Bayes Nearest Neighbours (NBNN) algorithm [13], which was applied to classifying each image from a set of features. It approximates the optimal classifier under a Naive Bayes assumption in terms of summing nearest neighbour distances.

However, Naive Bayes assumes (Equation (2.1)) that object parts S_i are conditionally independent given the class C, which is likely not the case. This is because several parts of an object which commonly occur together (i.e. are correlated) result in salient larger structures. Naive Bayes collapses together all parts from all training objects for a given class, simplifying inference but losing information. Different classes of objects can share many of the same small, individual parts, meaning there is significant overlap between object classes in the feature space.

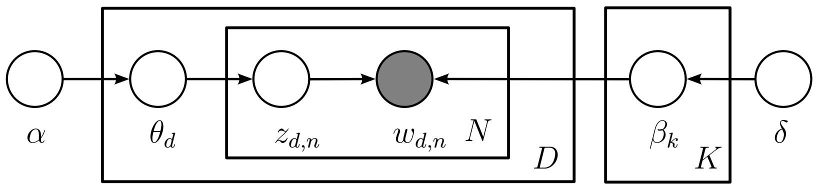
$$p(S_1, S_{\dots}, S_n|C) = \prod_{i=1}^{n} p(S_i|C) \tag{2.1}$$

For example, consider performing Naive Bayes on two point clouds of objects, such as a car and a truck, which have been cut up into many overlapping spheres of 1 m radius. Looking at each part by itself, its probability of being a car or a truck is evaluated, with no knowledge of the other parts. Finally, the probabilities from all the constituent parts are combined for each object. Unless there were, on average, sufficiently distinguishing isolated parts, the task will not succeed. This motivates methods that consider several parts at once by modelling inter-part statistical dependencies.

2.4.2 Topic Modelling

Topic models are a recent approach which are relevant to this goal [9]. Originally used for analysing collections of text documents, topic models have been applied successfully to other fields such as classifying visual images from constituent local features [28]. It is the subject of Section 5.2.

It functions on a 'bag of words' representation [21], which is a way of reducing a large set of high-dimensional feature vectors into a more manageable form. In particular, it permits techniques from document classification such as topic modelling to be applied to visual or 3D data. Features are first clustered or quantised, which is the subject of



$w_{d,n}$: the n^{th} word from document d.
$z_{d,n}$: the topic which generated the corresponding word.
θ_d : topic proportions for document d.
$\beta_{1:K}$: the K topics (distributions over words).
α δ : hyperparameters of the model.

Figure 2.11 – Graphical model of LDA, showing the conditional dependencies of each variable. Plates denote replicated variables, empty circles are unobserved variables, and shaded are observed.

Section 5.1. Each cluster forms a *visual word*, with the entire set of clusters forming a *vocabulary*. When a feature vector is associated with a cluster, it is summarised by the word (an index number of the vocabulary). Thus, a set of feature vectors comprising an object is reduced into a set of words. In this thesis, a feature vector represents the shape of a part of an object, and so words form repeatably detectable parts. Techniques such as topic models are then able to infer the statistical relationships between parts (words).

The simplest form of topic model is Latent Dirichlet Allocation (LDA) [10], a generative probabilistic model, shown in graphical form in Figure 2.11. In the context of document modelling, the input data consists of a set of documents, with each document containing a set of words from a fixed vocabulary. In the generative model, there are a predefined number of topics, with the following structure:

(1) Each topic β_k is a distribution over the vocabulary of words.

(2) Each word w in a document d is generated from a topic ($z_{d,n}$ is the assignment of word n in document d to a topic).

(3) Each document has a distribution over topics θ_d.

This is known as a mixed membership model, as each document has a mixture of topics.

Distributions are encouraged to be sparse, so that documents are effectively assigned a few topics, and topics a few words. Topics loosely capture a set of co-occurring words. In the context of document analysis, this groups words from similar topics, for example a topic pertaining to genetics would contain the words ‘sequence’, ‘molecular’ and ‘genome’ [9]. These words were likely to co-occur in a document about genetics.

Translating to the problem of 3D object classification, each object is a document, and each local feature (associated to a cluster) is a word, or predefined shape. Topics then capture sets of loosely co-occurring shapes. Rather than requiring every object in a class to have one distribution of shapes (i.e. topic), topic modelling provides an extra layer; the number of topics can exceed the number of classes. As such, the abstraction goes from isolated shapes, to sets of co-occurring shapes, to classes. For instance, vehicles may have ‘wheels, bottom edges and corners along the ground’ as one topic, with another set of shapes further distinguishing trucks, sedans etc. Indeed, hierarchical forms of topic models exist [74]. There are many variants of topic models due to the ease of modifying the graphical model.

In order to classify objects from topics, one approach is presented in supervised topic models [11], where training labels are incorporated into the graphical model, allowing the class labels of test objects to be inferred. Alternately, topics can be considered as a form of dimensionality reduction, with classes inferred from standard classifiers such as k-NN, support vector machines (SVM) etc. applied to the topic vector.

In summary, local features can be classified independently and pooled to classify an object. Alternately, features can be clustered to find the equivalent of words in a common vocabulary, and then used in frameworks such as topic models to learn co-occurring sets of shapes. This provides a more complex but possibly more powerful approach.

2.5 Conclusion

Dynamic range imaging in the field from a mobile robot is an informative sensing modality. However, the nature of the environment and sensing technology results in relatively sparse 3D data, significant amounts of occlusion and variation in point density, all of which change with the relative pose of the sensor in the scene.

Objects can be segmented out from their surrounds using existing techniques based on simple properties like spatial connectivity, allowing whole objects to be analysed. This provides an efficient modular mechanism for classification, in contrast to techniques that first compute features everywhere and perform intensive tasks like CRFs for segmentation and classification together. The former approach is used in this thesis for speed and simplicity, although the second is a viable alternative once an effective local representation is developed.

Representing an object in terms of a feature reduces it into an easily comparable numerical form. Global, object-wide features require some form of positional and rotational normalisation, or may be invariant to these changes by design. However, the extensive amount of occlusion, density variation and class variability motivate the use of local features.

Once an object is represented by a set of local features, it can be classified by matching and learning from these parts. In one approach, the parts can be independently classified and the results pooled together. Another method is to cluster the local features, forming repeatable shapes, or words. This allows co-occurring shapes to be found with topic modelling, providing an intermediate level to represent classes.

The remaining chapters of this thesis are organised as follows: Chapter 3 examines basic building blocks for shape analysis, required for Chapter 4, which examines local features and provides the full formulation of the line image feature. An object dataset is introduced, and classified with k-NN. Chapter 5 then looks at processing required for using topic models to classify the 3D object dataset. These are arranged into a pipeline of increasing abstraction in Figure 2.12.

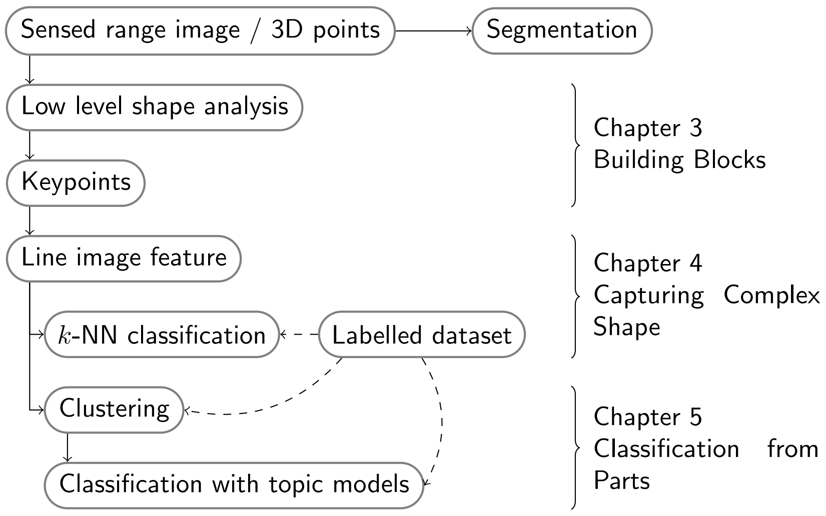


Figure 2.12 – An outline of this thesis as a pipeline from sensory data to classification, linked to each relevant section.

Chapter 3

Building Blocks for 3D Shape Analysis

The overall objective of this thesis is to compare the various shapes within objects, finding similarities and using these to perform classification. The sensed data, however, begins with depth measurements, and so these must be transformed into a measure of shape. This process, generally known as feature extraction, will be considered in the following steps.

(1) Surface region definition: The depth measurements, often irregularly spaced and sparse, will be broken down into representative regions of points, with surface properties such as normals computed on each region.

(2) Keypoint definition: These properties will be used to align 3D keypoints, which gives each region of points a fixed axis of alignment.

(3) Feature computation: With this information, more complex shape descriptors can be computed robustly on these local regions (e.g. with respect to varying point density).

This chapter addresses the first two items. Efficient region selection techniques are explored, followed by applying these to the task of surface normal computation in the context of noisy, sparse LiDAR measurements. Keypoints are then considered, with a contribution in using surface orientation stability rather than location stability. The methods in this chapter form some of the basic building blocks of more complex 3D processing tools presented in following chapters. These properties, and how robustly they can be computed, affect how the more complex shape analysis is done.

3.1 Region Selection

When analysing the shape at a point on a surface, the support region must be defined, determining the scale of analysis. This is typically a neighbourhood of points within a spherical region centred on a target point of interest P_c, with radius r. Using this set of points, properties of the surface are then inferred, such as surface normals; thus the neighbourhood of points must sufficiently represent the surface shape.

The following competing aspects are involved in region selection algorithms.

(1) Desired geometric information (small scale surface shape, or broad, large-scale shape). For example, a highly curved region can be considered flat when 'zoomed in' to a smaller scale.

(2) Available geometric information, due to the sampling rate. There may be insufficient sampling to analyse shape at smaller scales, similar to how the Nyquist frequency limits high frequency analysis in sampled data.

(3) Computation time considerations for selecting the supporting points, and what sampling requirements a particular algorithm has (e.g. only a few, approximate neighbours may be needed). For real-time applications, the selection of the sensed points can sometimes be the bottleneck in a processing pipeline.

The task of region selection is also known as the fixed radius near neighbours problem; an early survey is given in [4]. It is similar to the k-nearest neighbour problem, where a set of k points closest to a query point are desired. In this case, k may vary, with the distance to the query point being the gating mechanism.

This section presents several methods for selecting neighbourhoods of 3D points from the literature, and contributes a novel fast method which exploits the range image structure. The time complexity of each algorithm is explored, and computation times are measured on Velodyne data. These techniques are then used in the next section to compute surface normals.

3.1.1 *k*-d Tree

A k-d tree [6] is a standard method of finding nearest neighbours in a k-dimensional point cloud, and does not require any specific point structure. In this case, $k = 3$. Points are organised in a binary search tree, where each branch partitions space in one dimension, shown in Figure 3.1. The branching is terminated when each *leaf node* contains fewer than a given number of points. In 3D, the k-d tree forms a hierarchy of rectangular cuboids.

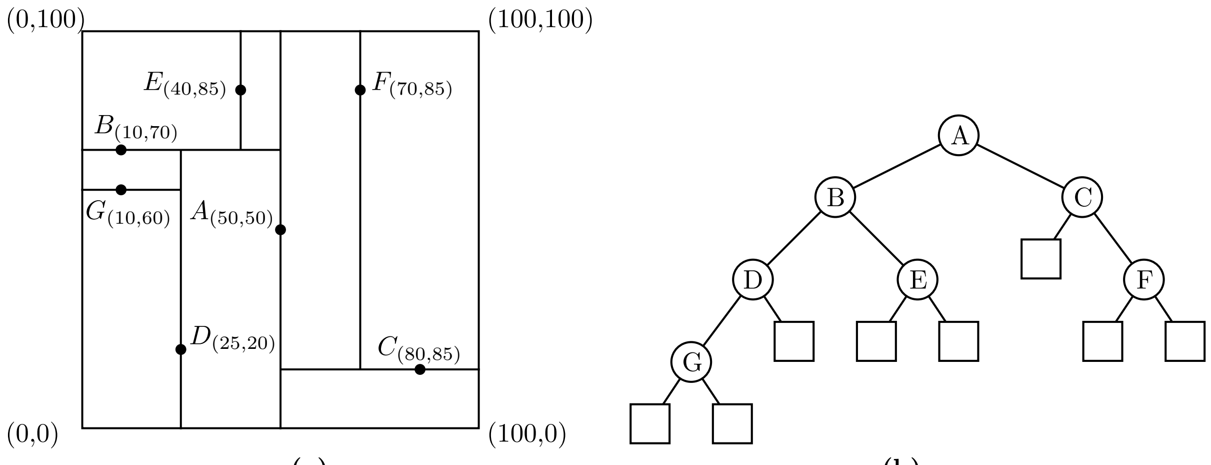


Figure 3.1 – Adapted from [6], a k-d tree visualised in 2D (a), with each splitting node partitioning the space into rectangles of smaller size. Note that in 3D, these are rectangular cuboids. (b) The same k-d tree in graphical form. The dimension which is split changes at each depth, cycling through all dimensions. Remaining data points (not shown) are stored in the bottom 'leaf nodes' (square).

Region selection is performed by finding which rectangular cuboids are partially or fully within the sphere of selection. This is implemented by traversing down the tree, inspecting each node, which defines a cuboid. If the distance of the target centre to at least one of the 8 vertices of the cuboid is within the radius r, deeper nodes of the tree are checked. Eventually, a leaf node is reached, wherein the distance of each contained data point is checked. Various shortcuts can be taken, such as not traversing any deeper when a cuboid is fully within the sphere, allowing all containing points to be added to the region without further distance computations.

The implementation in [59], known as the Fast Library for Approximate Nearest Neighbors (FLANN) was used[1], providing a baseline for comparing other selection methods. While approximations such as in [59] can provide speed improvements at the cost of accuracy, they are focused on high-dimensional data, and have not been applied to the task of 3D region selection. The effect they have on sampling uniformity and subsequent geometric analysis is unknown, and will not be considered here.

It is worth noting that k-d trees are used extensively in many fields, and a large amount of implementation effort is ongoing in the community. A very recent comparison of k-d tree software libraries on unstructured 3D LiDAR data is given in [26]. While optimisations, parallelization and GPU implementations could be applied to most types of neighbourhood selection algorithms, the availability of open source k-d tree libraries gives them practical benefits.

3.1.2 Bearing graph

Range images provide additional structure that can be used to facilitate region selection. A single scan of the Velodyne can be treated as a set of 2D points in elevation and azimuth (Section 2.1.2.1). A graph structure as in [58] provides neighbourhood connectivity between these 2D points, shown in Figure 3.2. This structure will be referred to as the *bearing graph*. Each point is connected to the preceding and following points in the scan line, as well as to points above and below (in elevation). Links are

[1]The index type KDTreeSingleIndexParams was used, with the default max leaf size of 10, as recommended for 3D point clouds in the FLANN manual.

then removed based on heuristics to separate foreground and background points. It is efficient, as data is already ordered by azimuth and elevation. It can also incorporate segmentation information (by removing links between different segments) to prevent a selection over more than one object.

3.1.2.1 Selection

To select all points within a given radius of a central point, an algorithm similar to a breadth-first search (BFS) [75] can be used. Points (vertices in the graph) are added to the selection as they are 'marked' as visited, and within the specified region.

Alternately, just four neighbours of a point can be used as a representative sample. This will be referred to as the *sparse* graph selection, and is particularly relevant for surface normal computation, as in [58] and discussed further in Section 3.2.1.1. However, as the majority of the noise is in the range, close, directly connected neighbours can be too close for accurate surface analysis. This is shown in (Figure 3.4). Surface normals are smoothed in [58] by averaging them, but this is only effective for unbiased errors, which is not the case here. Instead, this work selects four points further away, utilising a graph traversal algorithm shown in Algorithm 1. Each direction (up/down/left/right) is traversed up to a given distance to acquire each neighbour.

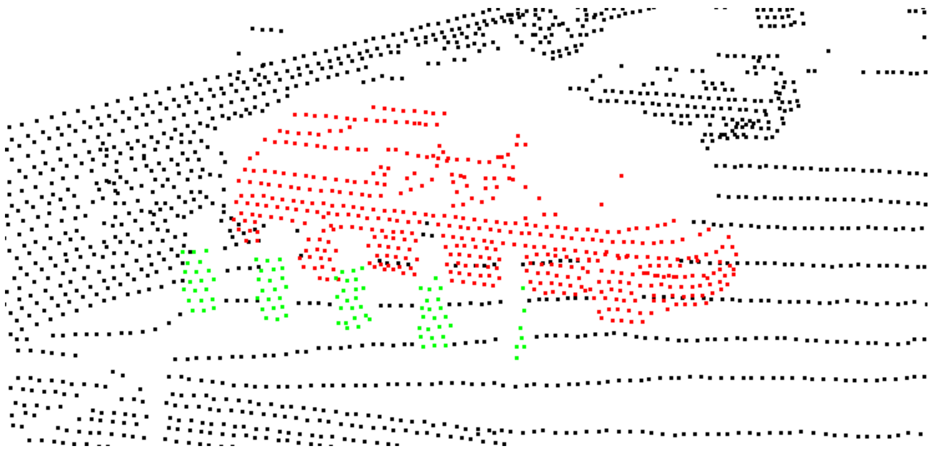
An issue with performing graph traversal on this data is the numerous missing edges. Due to the simple construction of the graph (linking neighbours, then removing them if they are relatively long), an optimal mesh is not achieved, unlike more intensive algorithms such as Delaunay triangulation. In addition, the angular distances between neighbouring points are not precisely equal, and there is noise in the range. In ideal data, two adjacent edges (i.e. connecting three neighbouring points) along a flat surface would be of similar length. With the above issues, one of the adjacent edges can be significantly larger. As a result, the graph construction algorithm can sometimes remove an edge that should have been kept.

The resulting missing edges in the graph can limit the availability of neighbours in some directions, or the distance that can be traversed in a given direction. To account for this, the algorithm can side-step missing links when traversing in a given direction. While this may result in a neighbour situated diagonally from the centre point, this provides a more complete neighbourhood. The effect this larger-scale neighbourhood has on surface normal computation is further discussed in Section 3.2.1.1.

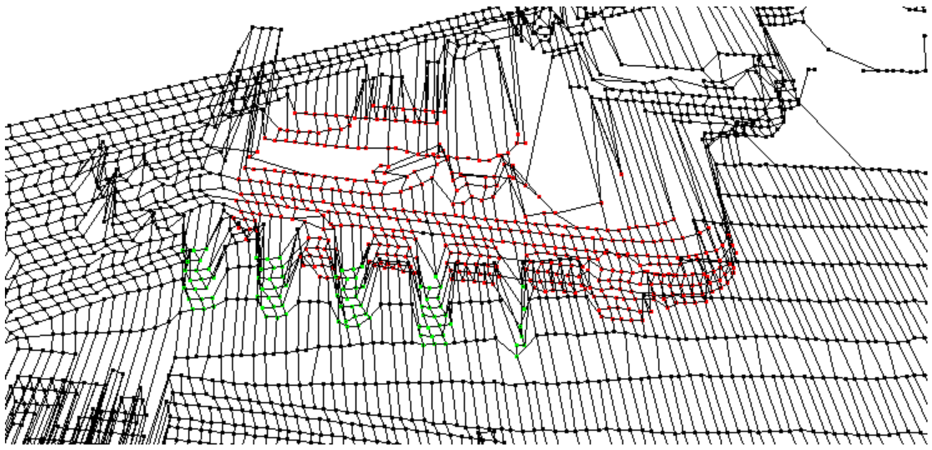
3.1.2.2 Minimum valid scale

In addition to region selection, the graph can help define the minimum scale of shape analysis. As a link between two points indicates that they belong to the same surface, the distance between directly connected points defines the smallest spherical radius at which the region can be analysed. For instance, if point p's nearest connected neighbour in elevation (from the scan line above or below), p_e, is 0.2 m away, the selection of all points inside a spherical region of 0.1 m radius about p will not include p_e. Lacking support points in elevation, the distribution of points will not be representative of the surface. This is a common issue in sparse range image data, as shown in Figure 3.3, where the ground becomes a set of well-separated scan lines. In this example, the minimum scale of analysis is 0.2 m (assuming any neighbours in azimuth are within this range). Note that, in the *absence* of a graph connection, the geometry of the region may be accurately represented at the smaller scale; consider a horizontal wire, with no graph edges in elevation. This property of scale selection from the graph will be utilised in Section 3.2.1.2.

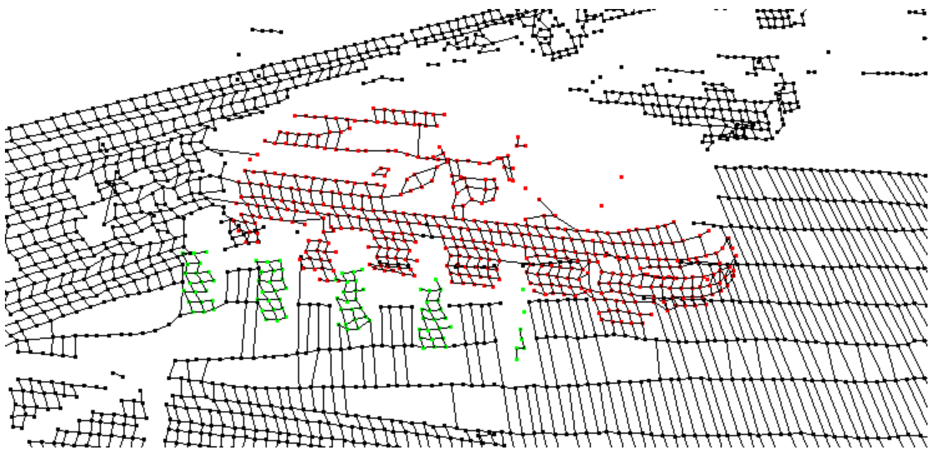
This notion of a *representative* neighbourhood of points brings up an insight into 3D shape analysis on points. Shape is determined not only from the existence of points, but also their absence in the empty space surrounding a surface. Unfortunately, the absence of points may be due to the lack of samples rather than a lack of a surface. The structure of the range image, and subsequent bearing graph, allow some reasoning about sufficient surface sampling. This leads to the concept of occupancy maps (introduced earlier in Section 2.1.1), where unknown space is distinguished from known-empty and occupied space. This concept is explored further in Chapter Chapter 4, where it becomes key in capturing complex 3D shape in sparse range image data.



(a)



(b)



(c)

Figure 3.2 – Constructing a bearing graph [58]. (a) A scan of a car behind some posts. (b) The full graph, connecting neighbours in elevation and azimuth. (c) The final graph after 'cleaning' by removing large edges. In this case, edges in elevation that were more than 3 times the length of adjacent edges (in elevation) were removed. The same was performed for azimuth. Finally, any edge more than 5 m long was removed.

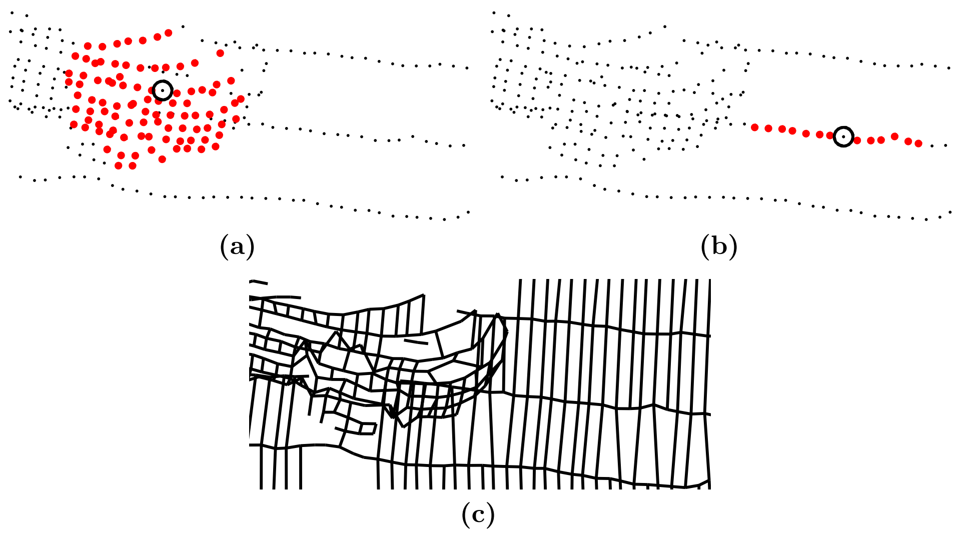


Figure 3.3 – Determining the minimum valid scale for region selection on a Velodyne scan. (a) A well sampled spherical selection on the front of a car. (b) A spherical selection on the ground, too sparse to represent the planar nature of the region. The large angle of the surface relative to the sensor results in points spread out in this manner, leaving the other rows outside of the spherical threshold. When analysing shape, such results must be filtered out. The mesh shown in (c) (built as in [58]) connects sparse points along a surface. Consider selecting a spherical region of fixed radius r centred at point p. In order to check if the region will contain a sufficient sample, the adjacent points to p in the graph can be inspected. If p has neighbours in elevation, and they are all beyond the radius of the sphere, the region is too small for surface analysis. The same applies for azimuth.

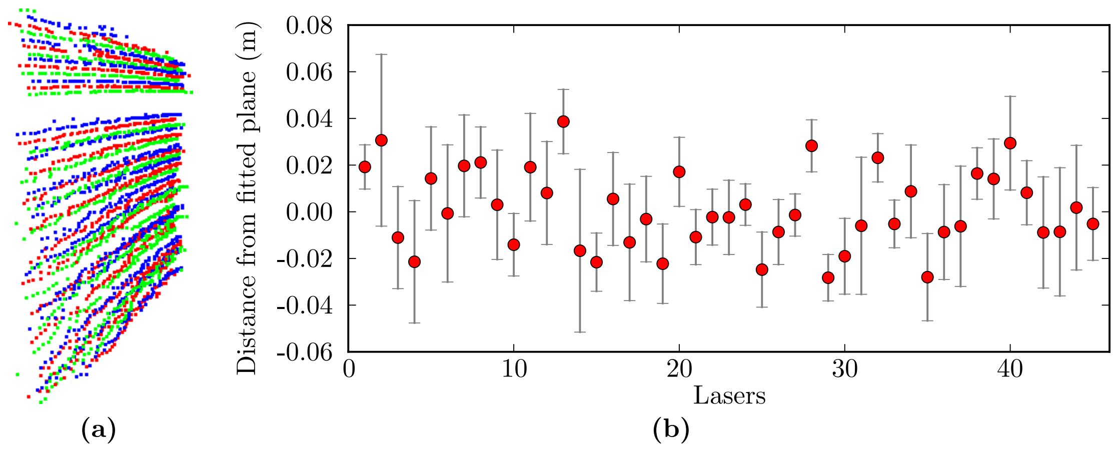


Figure 3.4 – (a) A planar surface, with each laser coloured separately. (b) The mean distance from the fitted plane for each laser, showing strong biases. While each laser may have 2 cm accuracy, the errors in calibration between lasers introduce significant errors in depth. As a result, using two directly neighbouring points between lasers (i.e. in elevation) for surface normal computation will not be representative of the surface. This can be mitigated by using neighbours further away; this is detailed in Section 3.2.1.1 and Figure 3.10.

```
Input   : point index $start$, direction $d \in$ {left, right, up, down}
Output  : neighbour point index
prev ← start;
while True do
    current ← Graph(prev, d);
    if current ∈ ∅ then
        // Check perpendicular neighbours for a way through.
        for $d_p \perp d$ do
            perpNeigh ← Graph(prev, $d_p$);
            current ← Graph(perpNeigh, d);
            if current ∈ ∅ then
                continue;
            else
                break;
    if ¬ WithinRegion(current, start) then
        if prev == start then
            Return current ;
        else
            Return prev ;
    prev ← current;
```

Algorithm 1: Traversing the graph in a given direction to find a point just within a given region. The routine goes around breaks where possible. If the only neighbour is outside of the region, it returns it anyway, such that the region can still be analysed when the scan density is low.

3.1.3 Range image grid

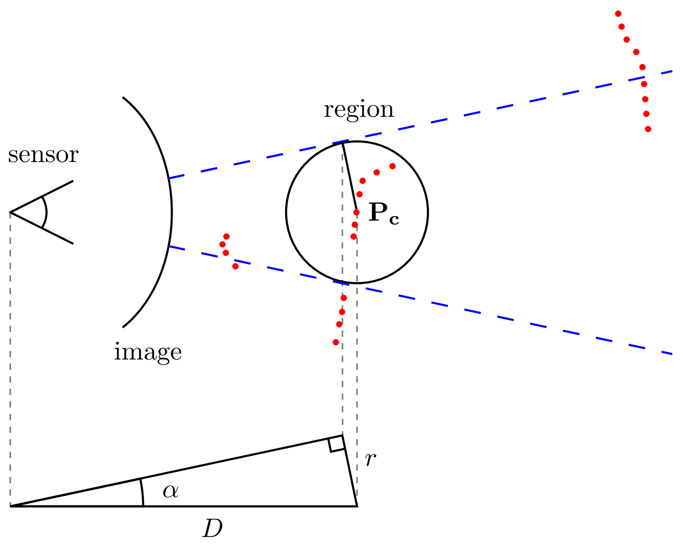


Figure 3.5: Selecting a 3D spherical region in the range image. Using the depth D of the centre point $\mathbf{P_c}$ and the desired region radius r, the angular width α can be determined: $\alpha = arcsin(\frac{r}{D})$. This defines a patch (θ, ϕ) in the range image about the central point's angular coordinates (θ_c, ϕ_c):

$$\theta \in [\theta_c - \alpha, \theta_c + \alpha]$$
$$\phi \in [\phi_c - \alpha, \phi_c + \alpha]$$

This patch is illustrated by the blue dashed lines. All sensed points (red) between these lines are then filtered by their distance to the centre point to obtain the final selection.

The range image organises range data by azimuth and elevation, with each 'pixel' having a depth value. While some sensors genuinely acquire one depth measurement per pixel, laser scanners such as the Velodyne sample depth along the azimuth coordinate irregularly. Gridding these points by azimuth invariably results in some cells getting two points, or none at all. In addition, the elevation spacing between individual lasers is constant but not regular.

Nevertheless, the range image provides a simple, efficient data structure upon which to do region selection and other inference. Rather than 'pixelising' the data by selecting a single depth for each pixel, this work instead stores 3D points in their corresponding 2D cell. When selecting a spherical region about a given centre point, the range image can be used to efficiently select the contained points, detailed in Figure 3.5. Simple trigonometry defines a patch in the range image that contains the sphere. For each 2D cell within this patch, the algorithm checks whether the contained 3D points are within the sphere in Cartesian space.

This method can be considered a type of cell technique [4], which works well on relatively uniform data in low dimensions. In this case, the uniformity of the 2D bearing space (elevation and azimuth) is exploited. In addition, the linear data structure of the grid means that iterating over each row of 3D points is fast in practice. This is because of the concept of locality of reference, where accessing consecutive memory locations is faster than accessing memory in disparate locations.

3.1.4 Computation time

These methods for selecting a region of points will now be compared. To summarise the previous sections, the methods under comparison are:

(1) The k-d tree.
(2) The bearing graph.
(3) The *sparse* bearing graph selection, in which only four neighbouring points are selected from the bearing graph, at a specified distance away (useful for computing surface normals).
(4) The range image grid.

For each of these, the experimental computation time and algorithmic time complexity will be examined. In considering time complexity, the focus is on increasing the number of points within the region to be selected. This can either be from denser sensing, or a larger region.

Computation times[2] were measured for each region selection method. Table 3.1 shows initialisation and selection times for a radius of 0.3 m, a typical scale used in this work for surface normal computation on Velodyne data. Figure 3.6 shows how the selection time for each algorithm scales with the radius of the region.

To generalise these results, the algorithmic time complexity of each stage of a selection algorithm will now be considered. Region selection algorithms can be broken down into the following stages:

[2]Algorithms were run on a Core2Duo E8400 3 GHz machine; they were implemented in C++.

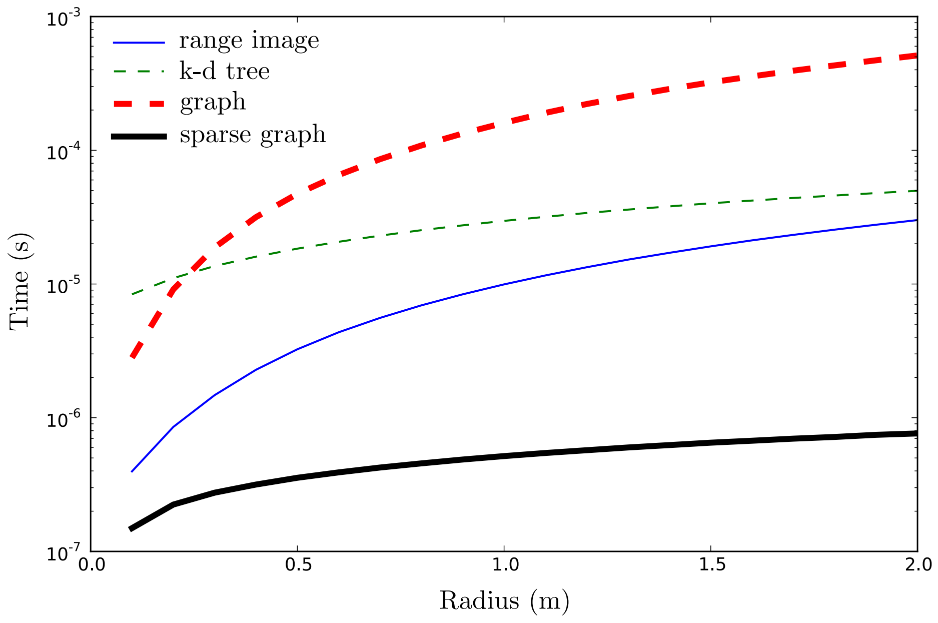


Figure 3.6 – Computation times for each region selection method for a range of scales. These times are an average over 20 scans of different urban environments, where in each scan, a region about every point was selected.

Selection Method	Initialisation time (**ms**)	Selection time for 0.3 m radius (**µs**)
k-d tree	14.7	12.8
Graph	4.88	23.1
Sparse graph	4.88	0.284
Image grid	0.567	1.65

Table 3.1 – Computation times for region selection methods for a radius of 0.3 m. Note that initialisation operates on the whole range image, while selection is the average time for a single region (note the different units).

(1) Data structure initialisation, where all data points are organised in some fashion. This only needs to be done once (per scan).

(2) Retrieval of a subset of points from the data structure, which may include points outside the region to be selected. For example, the k-d tree retrieves rectangular cuboids, some of which contain points outside of the target spherical region.

(3) Compute the distance from the region centre to each point in the above set, to identify which ones are inside the region. This may occur during the set retrieval, such as in the bearing graph.

There tends to be trade-offs between these stages. For example, the k-d tree has a large initialisation time due its detailed data structure. This structure then allows accurate subsets of points to be retrieved, meaning fewer subsequent L2 norm computations need to be done on points outside the desired region. A summary of the time complexities is given in Table 3.2, with the details outlined as follows.

3.1.4.1 Stage 1: Initialisation

k-d tree With n as the number of data points, the time complexity of initialising a k-d tree is $O(n \log n)$ [5].

Graph As the data is already sorted by azimuth and elevation, graph construction is an $O(n)$ operation.

Sparse graph The same graph is constructed for the sparse graph.

Image grid The range image grid is also $O(n)$, with n binning operations.

3.1.4.2 Stage 2: Subset / cell retrieval

k-d tree For k-d trees, the complexity of a region selection operation in [5] is denoted as $O(\log n)$. The primary computations can be grouped into computing distances of cuboid vertices (stage 2, considered here), and computing distances of data points (stage 3). With more points, the k-d tree will grow more layers of nodes (under a max-leaf size constraint), resulting in more vertices to check, but scaling sublinearly as $O(\log n)$.

Graph For the graph selection method, retrieval involves the breadth-first-search (BFS) algorithm, which scales with $O(|V| + |E|)$, the number of vertices and edges in the graph respectively. As there can only be up to four edges for each vertex, this is $O(n)$.

Sparse graph The same complexity applies to the sparse graph, as it effectively runs a BFS, but on a significantly smaller subgraph with no need for tracking visited vertices.

Image grid The cell retrieval stage for the range image grid can be considered as finding the grid cell bounds in elevation and azimuth. This of order $O(1)$, as all data points are then immediately available in contiguous memory for each row.

3.1.4.3 Stage 3: Data point distances

Each algorithm results in a different number of data points that need their distances from the central point P_c checked. For example, once the k-d tree cuboid cells are retrieved as above, the data points within them must be checked, in order to filter out all points outside the region. Reducing the number of points to check is the goal of these algorithms. The worst case scenario is a naive algorithm that forgoes the prior two stages, and simply checks the distance to every point in the *entire* scan (i.e. brute force).

It is helpful to consider each algorithm as carving out a different 3D region which contains this subset of points. For example, the k-d tree defines a set of rectangular cuboids. However, the volume of this region is not necessarily reflective of the number of points, as point density in 3D depends on the surface. A better approach is to consider the range image, where the density is approximately uniform in the image coordinates (e.g. elevation/azimuth). Thus it is useful to consider regions of 3D space by their corresponding 2D region in the image space. A smaller 2D area means fewer data points, and so fewer distance computations.

For example, consider a scanned planar surface, and consider a 3D spherical region on this surface. The sensed surface within the sphere can be projected into the range image, resulting in a 2D distorted ellipse of area S. A different 3D region such as a cube would result in a different 2D area C. If C was larger than S, the cube would contain more data points, and this would be true regardless of the range image density. As such, the cube method would have a 'stage 3' that is slower than the sphere method, as more distance computations would be needed. This notion is applied to each region selection algorithm, and is visualised in the right column of Table 3.2.

k-d tree Recall in Section 3.1.1 the ability for k-d trees to skip data points in cuboids already completely within the sphere. In effect, only cuboids along the border of the sphere must have their constituent data points checked. This hints at favourable scaling. Once again, consider a planar surface of data points. The 3D region carved out by the cuboids along the boundary of the 3D sphere, projected into the 2D image, will give a region similar to a blocky hollow circle (see Table 3.2). Consider this 2D region to have an area K. Depending on the tree structure, it is possible for K to be less than S.

Graph For the graph selection method, all points within the sphere have their distance computed during the BFS, plus the direct neighbours just outside of the sphere. This results in an encompassed 2D area slightly larger than S.

Sparse Graph For the sparse graph, just four lines radiate from the centre, each line being a graph traversal in a given direction. As a result, significantly fewer distance computations are needed, at the cost of an incomplete, sparse selection.

Image grid For the range image grid selection method, the 2D region is simply a square, as the algorithm operates in this space already. It is larger than S.

Finally, note that this section is considering the scaling of algorithms as the number of points within the region increases. Thus, as long as the above ratios of 2D area are not significantly affected by n, this 'stage 3' operation always scales by $O(n)$. However, the algorithms differ by a constant factor, approximated by the 2D areas (shown in Table 3.2). If one method only requires half the number of data distance calculations, it can be twice as fast in this stage.

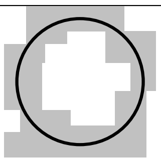
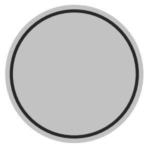
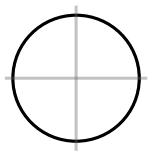
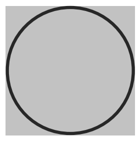
Algorithm	Initialisation	Subset/cell retrieval	Approx. area of data distance computations
k-d tree	$O(n \log n)$	$O(\log n)$	
Graph	$O(n)$	$O(n)$	
Sparse graph	$O(n)$	$O(n)$	
Image grid	$O(n)$	$O(1)$	

Table 3.2 – Comparing the time complexity of each stage of each selection algorithm. Note that the final stage always scales at $O(n)$, but differs between each algorithm by a constant factor, approximated by the shown grey area, against the black circle representing a spherical boundary.

3.1.4.4 Summary

For a full spherical region of points, the k-d tree ultimately scales the best, theoretically and in practice. With increasing data density, the benefit of only needing to compute data point distances on the boundaries of the sphere could overcome the additional computations of tree traversal and cuboid vertex distances. The higher initialisation time is insignificant when over a few thousand region selection queries are needed.

The graph selection method performs the worst experimentally (Table 3.1), likely due to the relatively expensive BFS algorithm. The sparse, four-neighbour graph selection requires no BFS and only a few L2 norm operations, resulting in significant time savings. While the selected points are not a complete, dense selection of all points in a sphere, the sample distribution is sufficient for surface normal computation, as described in the next section.

The range image has a fast initialisation time, and a very fast and constant 'retrieval' time. This comes at the expense of more L2 norm computations on extraneous points captured in the grid. Nevertheless, this combination produces the fastest computation time for a full spherical selection of points on Velodyne data. As such, this is the method of choice later in this thesis, when requiring a full neighbourhood.

3.1.5 Conclusion

Several methods for selecting neighbourhoods of 3D points were described, beyond the standard k-d tree. The bearing graph provided a minimum scale of analysis, as well as a fast subsample of up to four neighbouring points. The range image exploits the regularity of data in azimuth and elevation, providing the fastest dense selection of points in a spherical region, although the time complexity favours the k-d tree for higher density range images. Given such a region of points, properties like surface normals and shape descriptors can be computed.

3.2 Surface Normals

Surface normals are computed from sensed points, in an operation similar to fitting a plane to each local region. This provides an initial level of abstraction towards a geometric surface. They are commonly used in segmentation [58] and shape analysis (Section 4.1.3). This section presents methods of surface normal computation in the context of noisy, sparse range data, contributing some minor improvements.

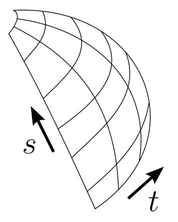


Figure 3.7

The surface normal can be defined in terms of a surface S, where each 3D point on S is parameterized by curvilinear coordinates $\mathbf{x}(s,t)$ (Figure 3.7). The surface normal is then the cross product of partial derivatives:

$$\mathbf{n} = \frac{\delta x}{\delta s} \times \frac{\delta x}{\delta t} \tag{3.1}$$

In 3D sensing, the surface is unknown, and the estimation of the normal at each 3D point provides a first order approximation of the surface. This requires the selection of a set of surrounding points, as presented in the previous section. The non-ideal noisy range data in field robotics distinguishes the surface normal estimation problem from similar tasks in 3D modelling and graphics fields, where data is often more ideal (as shown in Section 2.1.2).

An overview of normal estimation techniques for range sensing is given in [46]. They can either be based on optimisation, such as fitting a plane with least-squares, or based on averaging, wherein the normals of surrounding triangles are averaged (provided such structure is available). Badino et al. [2] computed surface normals on Velodyne data in a highly efficient manner by working directly in the range image space. However, as foreground cannot easily be separated from the background in this method, the surface normals on the boundaries of objects were distorted. Moosmann et al. [58] computed surface normals using an averaging method on the graph structure discussed in Section 3.1.2, but using the nearest neighbours in the graph makes this method subject to noise.

Based on the conclusions of Klasing et al. [46], the fastest algorithm depends on the region selection data structure available. This section analyses two commonly used surface normal algorithms applied to Velodyne data in terms of the selection structures discussed previously.

3.2.1 Computation methods

3.2.1.1 Cross product on graph

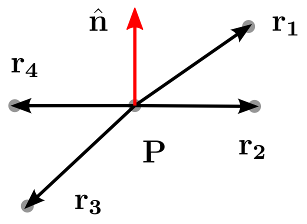


Figure 3.8 – Surface normal $\hat{\mathbf{n}}$ via the average cross product of relative vectors $\mathbf{r_i}$, formed from point $\mathbf{P}$ to neighbouring points in the graph.

$$\mathbf{n} = \frac{\mathbf{r_1} \times \mathbf{r_2}}{\|\mathbf{r_1} \times \mathbf{r_2}\|} + \frac{\mathbf{r_2} \times \mathbf{r_3}}{\|\mathbf{r_2} \times \mathbf{r_3}\|} + \ldots$$

The cross product method allows the use of the graph structure. It was called the *AreaWeighted* algorithm in [46] and applied to Velodyne data in [58]. Relative vectors are formed from the centre point to each surrounding point, and then the cross product is computed on each pair of roughly perpendicular vectors. These are averaged to produce the surface normal at the centre point. This is demonstrated in Figure 3.8.

In [58], the nearest neighbours in the graph were used to compute the surface normal, followed by applying a moving average filter to reduce noise. However, as shown previously in Figure 3.4, errors between lasers are biased, and so averaging is not effective. Instead, this thesis contributes the sparse graph selection algorithm (Section 3.1.2), which allows further neighbours to be used for better noise resistance (see Table 3.4). The resulting surface normals are also blurred with neighbouring values (averaged with a Gaussian weight based on point distance).

A visual comparison is given in Figure 3.10. This method relies on the graph structure, which can have missing links due to non-returns, or overly sparse data. Without at least one connected neighbour in both azimuth and elevation, this method cannot compute a surface normal.

3.2.1.2 Principle Component Analysis

Another method for estimating the surface normal is by computing Principal Component Analysis (PCA) on a spherical selection of points. This involves computing the symmetric covariance matrix C on the set of N points $\{\mathbf{p}_i\}$:

$$\bar{\mathbf{p}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{p}_i \tag{3.2}$$

$$C = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{p}_i - \bar{\mathbf{p}})^T \tag{3.3}$$

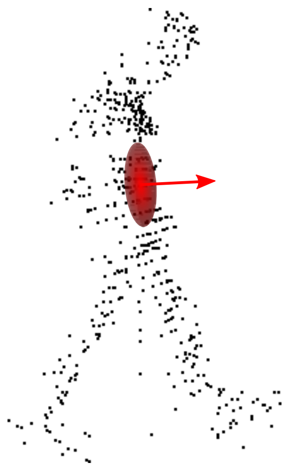


Figure 3.9 – PCA on the middle of a person (side-view), resulting in an ellipsoid and surface normal.

The eigenvalues and eigenvectors are then computed on C. The eigenvector corresponding to the smallest eigenvalue is the surface normal; it is the direction of least variation. This was called the *PlanePCA* method in [46], where it was shown to be equivalent to the maximum likelihood solution of the local plane estimation.

This method effectively fits a 3D Gaussian function to the region, visualised as an ellipsoid, and is shown in Figure 3.9. The eigenvalues provide additional information, and can be used for basic shape classification as in [48]. This is utilised for keypoint detection in Section 3.3.2.

Graph scale selection

Statistical features such as PCA can give invalid results when computed on neighbourhoods that are not representative of the local geometry, as shown in Figure 3.3. In the cross product method, neighbours were chosen regardless of hard limits of distance (in particular, the sparse graph selection algorithm allows neighbours to be further than the specified distance if none would otherwise exist). As PCA requires a spherical region of points, the minimum radius of this region can be determined from the graph, as described in Section 3.1.2.2. Alternately, for a chosen scale of analysis, all points which have an insufficient neighbourhood sampling at this scale can be disregarded.

For a given radius r and sensed point p, if p lacks a neighbour in elevation or azimuth within distance r, it is disregarded. This provides a consistent scale of analysis with regards to varying point density, at the expense of missing surface normals.

3.2.2 Comparison

Average computation times for each stage of the surface normal algorithms are shown in Table 3.3. The cross product method is twice as fast as PCA, with more points receiving surface normals. The main time expenditure of PCA is in region selection, as it requires all points within the region to be selected (using the image grid algorithm). The cross product method only requires four neighbours from the faster sparse graph selection method. Also, the scale constraint of PCA results in sparser regions not receiving surface normals, although these could be computed at larger scales.

A visual comparison of normals is shown in Figure 3.10, where PCA shows smoother, less noisy results. A quantitative comparison of error on a planar surface is shown in Table 3.4, showing PCA to be most accurate. It is worth noting that, for the cross product approach on nearest neighbours, the effective scale of the surface normal computation is much smaller, perhaps providing more detail at the expense of more noise.

In summary, PCA provides smooth, accurate surface normals, but requires more computation to produce. If the scale of analysis is fixed, not all data points can receive surface normals. The cross product approach is faster and provides surface normals at more points, but is noisier. PCA also provides additional shape information beyond the surface normal, which is explored further in the next section.

Algorithm / stage	Time (ms)	Order
Cross product method		
4-neighbour selection	15.26	$O(n)$
Cross product	17.17	$O(1)$
Gaussian blur	25.77	$O(1)$
Total	58.21	
(91.49% of scanned points)		
PCA method		
Valid radius check	2.03	$O(1)$
Image sphere selection	70.59	$O(n)$
PCA	43.27	$O(n)$
Total	115.90	
(70.08% of scanned points)		

Table 3.3 – Average times per scan for surface normal computation; times are broken down according to each step in the process. The percentage of scanned points that received surface normals is also shown. The order refers to the time complexity of one single surface normal computation, where n refers to the number of points in a region. A region radius of 0.3 m was used, with an average of 53 846 points per scan. As previously, algorithms were run on a Core2Duo E8400 3 GHz machine, and implemented in C++.

Algorithm	mean angular error (rad)
Cross product (nearest neighbours)	0.3172
Cross product (neighbours 0.2 m away)	0.1275
PCA (0.2 m radius)	0.0531

Table 3.4 – Mean angular error of surface normals on a close planar surface (shown in Figure 3.4a.

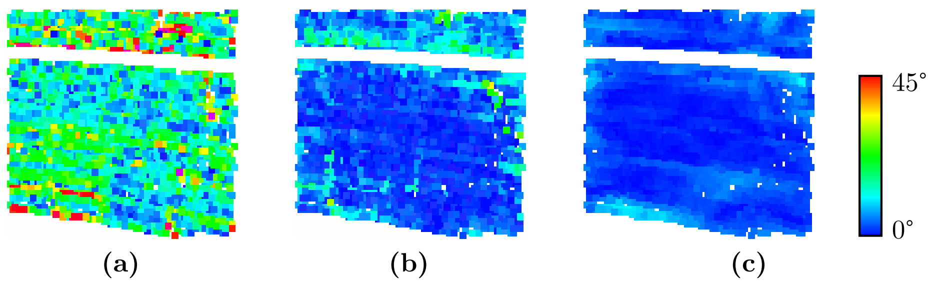


Figure 3.10 – Surface normals computed on the planar region in Figure 3.4a, with each point coloured by the angular difference of the surface normal to a globally-fitted plane. (a) Cross product method, using closest neighbours (and smoothed), whereas (b) uses neighbours 0.2 m away (and smoothed). (c) PCA. Note: the gap in the plane is due to a large angular difference between two lasers in the Velodyne sensor used.

3.3 Keypoints

When comparing two local regions, the surface normal can help to orient them into a common reference frame prior to comparison. This section addresses the problem of finding keypoints, which generally define a set of stable positions and orientations, allowing local regions to be extracted and compared with others. These are calculated from the data, so that they reoccur at the same pose relative to salient shapes. This approach is commonly used in colour vision (e.g. SIFT) to first define a reduced set of points in the data, as well as a direction at each point for alignment. Features are then computed on these regions. Reducing an object to a small set of consistent features can greatly assist feature matching and classification algorithms, which often do not scale well with the number of data points.

For example, consider the task of finding locally matching shapes between two objects A and B, which are scanned to produce n and m points respectively. These objects can be at any arbitrary rotation or position with respect to one another. One way to find matches is to split each point cloud up into separate regions, and compare each pair of regions. Suppose each region is a sphere of fixed radius, centred on each data point. This results in n and m regions, with $n \times m$ comparisons.

Clearly, this process is not favourable to scaling, and yet an even more significant problem remains to be addressed- that of region *orientation*. Consider simply overlaying the regions and visually comparing them. As each point cloud can be at any orientation, and any given *shape* can be at any orientation, there is an issue of alignment. Even given two regions of exactly the same shape, the two must be oriented correctly in order to match. Aligning the surface normals of both regions is a common step, although often more is required, depending on the method of comparison.

Regions are typically compared by forming a descriptor of the region, a feature vector which encodes a notion of shape. The distance of this vector to other feature vectors allows regions to be compared, as introduced in Section 2.3. The specifics of shape descriptors is the topic of the next chapter. While not all shape descriptors have the same requirements of alignment, many use the concept of spatial bins, where the region is separated into a regular grid of smaller regions. If this grid is not aligned correctly, the descriptor will be different, and a match will not be found.

While colour vision keypoints require a single 2D orientation, 3D sensing requires two 3D orientations (i.e. two perpendicular vectors, with the third perpendicular vector of the axes implicitly defined). The difficulty in defining 3D orientation is a key factor in designing a feature. The design of many features is motivated by this aspect. This section explores existing keypoint algorithms, and contributes a method that focuses on orientation stability.

3.3.1 Existing approaches

For a 3D orientation, one vector is typically formed from the surface normal, but a second, perpendicular vector must also be defined. Without a further constraint, the local axes is free to 'spin about' the surface normal. One solution is to form a descriptor that only requires the one orientation vector, such as in the Spin Image (as introduced earlier in Section 2.3.3), where all data in the free 'spin dimension' is accumulated, losing some spatial information.

In colour vision, intensity derivative extrema are used to locate and orient keypoints, and so similar approaches can be considered for 3D data. The first derivative of a surface is the surface normal, the second derivative is known as the *principal curvature*. This is an intrinsic property of the surface, in that its value does not depend on the position or orientation of the surface. A number of 3D keypoint algorithms are based on using the principle curvature [54, 94, 39], however, they focus on denser, cleaner data from 3D modelling. In [84], an operator based on curvature was used, which found multi-scale interest points on unstructured point clouds. It accounted for variable sampling density, and was applied to a sparsely scanned car, segmenting it into broad regions of differing curvature.

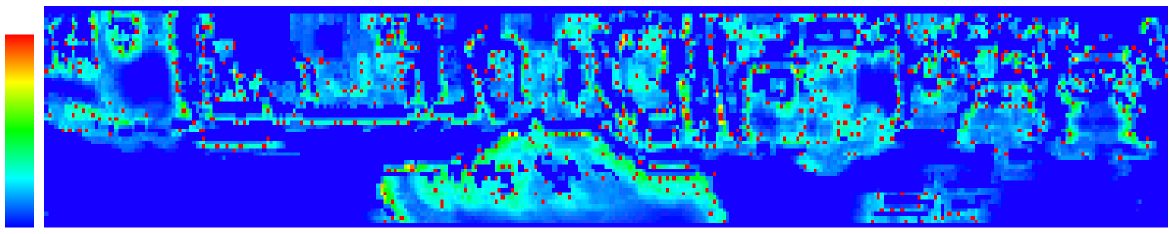
However, applying curvature-based algorithms to sparse, noisy range image data can be problematic. A method for computing principal curvature on Velodyne data is given in Appendix A, and results are visualised in Figure A.1. With sparse range image data, few regions have more than the bare minimum six required points to compute curvature, making it subject to noise. There are only a few points with significant curvature. In order to detect more subtle curvature, smaller support regions must be used, which is only possible on well-sampled, nearby surfaces.

Unless an object has large corrugations, there are only so many large curvatures before the object wraps around and occludes itself. In a range image, any surface with a normal greater than 90° from the viewing ray[3] must be self-occluded and therefore not visible. Large changes in surface normal inherently result in this condition. Curvature at such occlusions cannot be reliably computed, as only half the points in the region are visible (and may appear quite flat, depending on the sharpness of the curvature). Furthermore, the surface normals at these points are not likely to be stable, being on regions with highly varying surface normals by definition. Minor changes in the limited sample of data can change the surface normal estimate significantly.

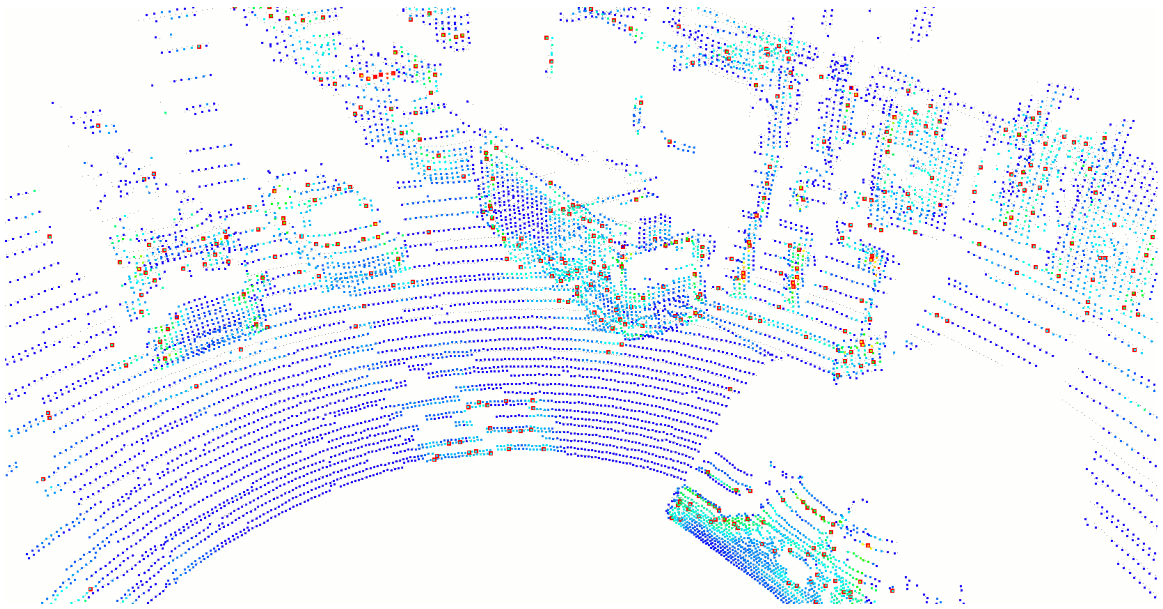
A recent 3D keypoint algorithm called the Normal Aligned Radial Feature (NARF) [76] seeks to find stable surface normals *nearby* object boundaries and regions of high principal curvature. An interest value is defined at each point, with maxima typically found near, but not on, object borders and highly curved regions. Orientation is determined from surface normals and either a nearby 2D image boundary direction or the principal curvature. NARF interest values and resulting keypoints applied to Velodyne data are shown in Figure 3.11.

This algorithm assumes surface normals are stable when principal curvature is low. In sparse range image data, the surface normals themselves are often not well defined, with not enough information to additionally compute principal curvature robustly. Nevertheless, NARF also uses the boundaries of objects in the image, and appears to produce keypoints on Velodyne data as specified, near object boundaries and strong curves. NARF keypoints, and the associated features, are compared with the contributed keypoint algorithm of this thesis in Section 4.4. While NARF does provide a significantly reduced set of keypoints, the resulting classification performance is reduced, as will be shown in Section 4.4.

[3]The relative vector from the sensor to the surface



(a) Image



(b) 3D

Figure 3.11 – NARF keypoints (large red) on a Velodyne scan, coloured by the NARF interest value.

3.3.2 Keypoints at Stable Orientations

Rather than trying to define a well-localised set of keypoints (such as on sharp curves), this section focuses on a method for detecting regions where a single orientation such as a surface normal can be robustly extracted. In this way, locations which provide good orientation alignment are used as keypoints. While well-localised keypoints may be fewer in number, this work argues that there is insufficient data in a sparse range image to do this robustly. More in-depth analysis (e.g. complex features) may be capable of defining well-localised regions, but these often require the orientation alignment that keypoints provide in the first place.

In this thesis, regions with stable surface normals are found by computing a measure of

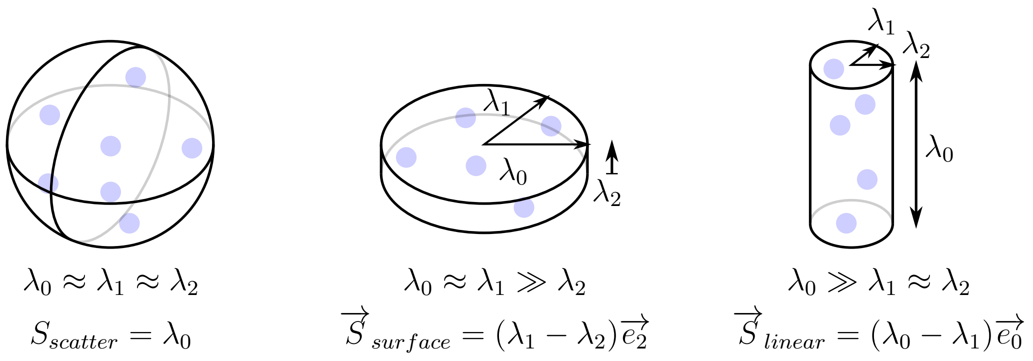

λ_1
λ_2
λ_0
λ_1
λ_0
λ_2
$\lambda_0 \approx \lambda_1 \approx \lambda_2$
$S_{scatter} = \lambda_0$
$\lambda_0 \approx \lambda_1 \gg \lambda_2$
$\overrightarrow{S}_{surface} = (\lambda_1 - \lambda_2)\overrightarrow{e_2}$
$\lambda_0 \gg \lambda_1 \approx \lambda_2$
$\overrightarrow{S}_{linear} = (\lambda_0 - \lambda_1)\overrightarrow{e_0}$

Figure 3.12 – PCA metrics from [48], measuring how scattered, flat or linear a region is. Eigenvectors $\overrightarrow{e_0}, \overrightarrow{e_1}, \overrightarrow{e_2}$ are shown by arrows, with corresponding eigenvalues ordered by increasing value $\lambda_0 \geq \lambda_1 \geq \lambda_2$.

surface normal quality. In [48], PCA was computed on local regions, and the relative eigenvalues were used to compute a measure of how flat, linear or scattered a region was (shown in Figure 3.12). This thesis uses these metrics to extract keypoints at (1) highly flat regions, where the surface normal is well defined, and (2) highly linear regions, such as on poles, where the linear direction is well defined. Due to the central role of PCA, these are referred to as ‘PCA keypoints’. As the eigenvalues vary by the size of the spherical region of points, this thesis normalised $S_{surface}$ and S_{linear} by the norm of the eigenvalues $|(\lambda_0, \lambda_1, \lambda_2)|$. Two thresholds then define flat and linear regions, which were empirically tuned to attain keypoints on visually flat and linear regions.

Poles often have no well defined surface normal, and can sometimes have only a single row of vertical points (Figure 3.13). PCA keypoints allow distinctive regions such as these to be considered in feature extraction and comparison, rather than disregarding them.

While PCA keypoints have a well defined single vector, a second vector is often needed, in order to define a 3D set of axes for alignment. The aim is to define a local *frame*, or $(\mathbf{x_L}, \mathbf{y_L}, \mathbf{z_L})$ set of axes, with two axes explicitly defined, and the remaining axis

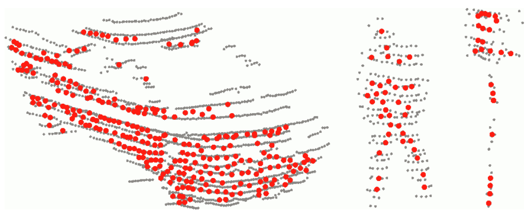


Figure 3.13 – Keypoints at flat or linear regions on a car, pedestrian and sign.

implicit. As stated previously, a second axis is not easy to define from the data alone, and so some additional assumptions are made here. In this section, the primary axis refers to the well-defined eigenvector from PCA (surface normal or linear direction), and the secondary axis refers to the remaining explicitly defined axis that will be outlined here.

3.3.2.1 Flat regions

For flat regions, this second axis is defined by taking into account the vertical direction (i.e. gravity direction), so that it faces as downwards as possible. While this means the axes are not fully invariant to relative pose (the object could roll or pitch), in many contexts, full invariance may not be needed. A mobile robot will typically face the right way up, or at least know which direction is down, and most objects in an urban setting have a preferred vertical orientation. For example, cars and trees are almost always the 'right way up'. In this work, as the sensor is always oriented correctly, the vertical direction is defined as the sensor frame z axis.

To compute this downward-facing axis, the vector $(0, 0, -1)$ is projected onto the plane defined by the surface normal $\hat{\mathbf{n}}$ (which equals the eigenvector $\mathbf{e_2}$). With $\mathbf{I}_3$ as the 3×3 identity matrix:

$$\mathbf{a} = (\mathbf{I}_3 - \hat{\mathbf{n}} \cdot \hat{\mathbf{n}}^T)[0, 0, -1]^T \tag{3.4}$$

$$\mathbf{b} = \hat{\mathbf{n}} \times \hat{\mathbf{a}} \tag{3.5}$$

$$F_S = (\hat{\mathbf{a}}, \hat{\mathbf{b}}, \hat{\mathbf{n}})^T \tag{3.6}$$

In this local frame F_S, the surface normal becomes the local $\mathbf{z_L}$ axis, and the local $\mathbf{x_L}$ axes faces towards the ground (Figure 3.14a). With the region oriented in this

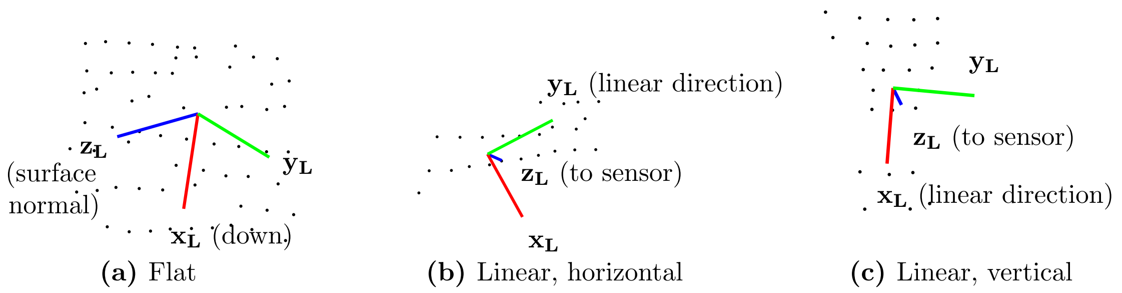

$\mathbf{z_L}$ (surface normal)
$\mathbf{y_L}$
$\mathbf{x_L}$ (down)
(a) Flat
$\mathbf{y_L}$ (linear direction)
$\mathbf{z_L}$ (to sensor)
$\mathbf{x_L}$
(b) Linear, horizontal
$\mathbf{y_L}$
$\mathbf{z_L}$ (to sensor)
$\mathbf{x_L}$ (linear direction)
(c) Linear, vertical

Figure 3.14 – PCA local axes on differently-shaped regions of points. The local $\mathbf{x_L}$ axis faces downwards, and $\mathbf{z_L}$ faces away from the surface.

manner, an algorithm 'looking along' the $\mathbf{z_L}$ axis would see a perpendicular surface, and would face the right way up. Features such as the line image and others to be described in Chapter 4 summarise a local image, formed by looking at the point cloud from this frame.

Finally, there is the case where the surface normal is highly vertical, and so the $\mathbf{x_L}$ axis cannot 'point downwards'. In this case, the eigenvector $\mathbf{e_0}$ corresponding to the largest eigenvalue is used for the $\mathbf{x_L}$ axis[4].

3.3.2.2 Linear regions

For highly linear regions, where the surface normal is not well defined, the linear direction forms the primary axis. The secondary axis is resolved by aligning it towards the sensor origin. This vector is used as the local $\mathbf{z_L}$ axis. This is because, lacking a better source of alignment, the resulting frame will have a similar 'view' of the surface as the sensor. If the whole region is highly linear, the shape may be symmetric about the linear direction anyway.

In addition, the linear direction $\mathbf{e_0}$ must be resolved as either positive or negative. This was not an issue for the surface normal, as it has an unambigous direction, pointing away from the surface[5]. To resolve this, if $\mathbf{e_0}$ is mostly vertical (less than 45° from the

[4]This is not necessarily a stable, well-defined direction for flat regions, and so is not used for all flat regions.

[5]Due to the nature of the range image, no surfaces can be 'facing away' from the sensor, and so the surface normal must always form an angle of less than 90° with the vector from the sensor origin

vertical axis), it is faced downwards, and becomes the local $\mathbf{x_L}$ axis. Note that $\mathbf{x_L}$ also faces downwards in the case of flat regions.

In that case that $\mathbf{e_0}$ is mostly horizontal, the guiding principle is to get $\mathbf{x_L}$ to still face downwards. To do this, $\mathbf{e_0}$ is defined as the local $\mathbf{y_L}$. The direction (left or right relative to the sensor) is chosen such that $\mathbf{x_L}$ faces downwards (i.e. by the right hand rule). Consistently downward-facing $\mathbf{x_L}$ axes are shown in Figures 3.14b, 3.14c and 3.15c.

Thus, for a surface normal $\mathbf{n}$ at point $\mathbf{p}$:

$$\mathbf{c} = (\mathbf{I}_3 - \hat{\mathbf{e_0}} \cdot \hat{\mathbf{e_0}}^T) \cdot (-\mathbf{p}) \tag{3.7}$$

$$\mathbf{d} = \hat{\mathbf{c}} \times \mathbf{e_0} \tag{3.8}$$

If mostly vertical: $\|\mathbf{e_0} \cdot [0, 0, 1]^T\| > \cos(45°)$ then:

$$F_L = (\mathbf{e_0}, \hat{\mathbf{d}}, \hat{\mathbf{c}})^T \tag{3.9}$$

Else:

$$F_L = (\hat{\mathbf{d}}, \mathbf{e_0}, \hat{\mathbf{c}})^T \tag{3.10}$$

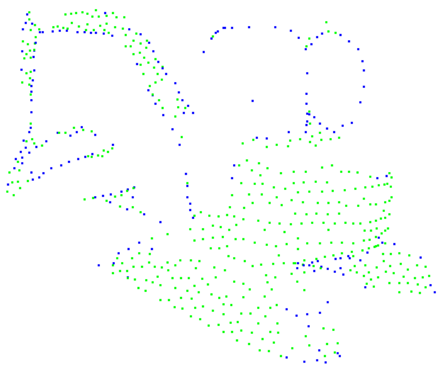
As such, keypoints on flat surface regions have a local frame F_S, and keypoints on thin linear regions have a local frame F_L.

3.3.2.3 Discussion

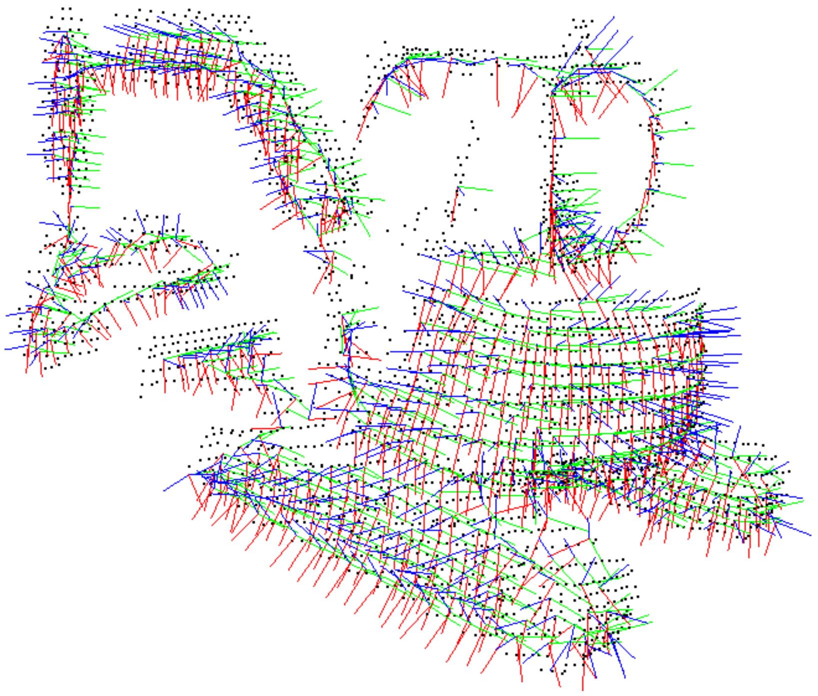
A detailed step-by-step description of the algorithm is given in Appendix B, and the axes on flat and linear regions computed on an excavator are shown in Figure 3.15. These keypoints on flat and linear regions are not necessarily at local extrema ‘interest points’, as in SIFT and NARF features, they simply provide a stable local orientation for repeatable features. On some objects, the sampling is sparse enough that further reduction is not required, and in other cases subsampling can reduce the number of keypoints if required.

to the sensed point. Calculated surface normals that violate this are flipped.

(a) Excavator point cloud



(b) Flat (green), linear (blue) regions



(c) Axes at each keypoint (red/green/blue = x/y/z)

Figure 3.15 – PCA keypoints and local axes on an excavator.

The primary weakness of this algorithm is on objects with large, flat surfaces such as buildings, where the large number of keypoints lead to a high degree of redundancy in subsequent descriptors. This can lead to scaling issues later on in classification, discussed in Chapter 5. One option is to heavily subsample large, flat regions, detected by PCA computed at larger scales. Another approach is to reduce the number of points *after* feature computation, by removing redundant sets of nearby and highly similar features. This has the benefit of not arbitrarily losing shape information, which is possible when subsampling keypoints prior to detailed larger-scale analysis. Depending on the speed of feature computation and the specific classification algorithms used, these may not be serious issues, and are left as future work.

In order to evaluate the effectiveness of this keypoint algorithm, it is compared with NARF keypoints on the task of object classification in the next chapter. While metrics exist for comparing keypoint repeatability [72], it is difficult to quantify how much information is lost by disregarding many points in favour of a sparse, repeatable set. Keypoint reduction for object classification draws a fine line between reducing the number of regions to match, and ensuring enough informative regions are captured such that different classes can be distinguished. Given the nature of the sparse range image, this thesis argues that even more sparse, localised keypoints are not easily determined without in-depth shape analysis, which is what the subsequent feature extraction stage seeks to do. Instead, as such features require an orientation, PCA keypoints provide the locations where orientation is likely to be stable.

3.3.3 Summary

Keypoints define positions and orientations about which features are computed and compared. Existing approaches to keypoints rely on curvature extrema, which are not well defined on sparse range images with inevitable environmental and self occlusion. NARF keypoints also consider 2D object borders, but use surface normals without considering how well defined they are. A primary goal of keypoints is to produce an orientation, and so the contributed PCA keypoints are defined by stable orientations, with surface normals on flat regions and linear directions on thin regions. These are compared in the next chapter in the context of object classification.

3.4 Conclusion

This chapter outlined processing steps for progressive abstraction from raw point clouds to simple surface properties and aligned regions of interest. The approach ensures that the sampling of points in a given region is sufficient for the computation of surface normals. PCA provides simple shape information, from which aligned keypoints are defined. Building upon this abstraction, the next chapter considers larger regions, describing how to capture the more complex and distinguishing shape information necessary for object classification.

Chapter 4

Capturing Complex Shape

As introduced in Section 2.3, local features are used to summarise the appearance of a region of the data, converting it into a vector of numbers which can be easily compared and matched to other regions. In vision, features may describe some aspect of the local colour, intensity or visual texture, whereas when processing 3D sensor data, features typically summarise the local geometry in some way.

In sparse range images, inferring and capturing this geometry is difficult due to variable density and occlusion. This chapter contributes a novel feature, called the line image [62], which addresses these issues by explicitly reasoning about occupancy. An object dataset was collected to evaluate the performance of this feature on the task of object classification. The dataset consists of 14 classes of labelled objects scanned with a Velodyne in an urban setting. Objects were classified with a simple k-nearest neighbours (k-NN) classifier. The line image showed improved performance at smaller scales of analysis compared to existing features.

4.1 Invariance and Distinguishability

Two desirable properties of features are: (1) the ability to capture distinguishing information, and (2) to do so in a consistent manner, such that the same sensed shape receives the same feature values. This section will describe some existing features from the literature in the context of these properties. The limitations of these features when applied to sparse range image data motivates the line image feature.

4.1.1 Viewpoint Invariance

To achieve a consistent representation, viewpoint invariance is desired. Consider a scanned object, where a particular region of it is converted into a feature vector. If the sensor is subsequently moved, and the process repeated, a viewpoint invariant feature would be very similar to the original. The sensor and objects can move about in the world, and so being able to match and recognise a given region regardless of pose is crucial. In classification, this allows an object to be learnt from a single viewpoint, instead of requiring many different poses to be learnt.

At best, this creates a one-to-one mapping from a feature vector to a certain local 3D shape, rather than a many-to-one mapping. This is important for any statistical inference involving local shapes. Consider the latter, non-ideal case where many different feature vectors from different viewpoints indicate a single underlying shape. This is shown in Figure 4.1. In supervised classification, the goal is to infer relationships between shapes and classes. However, shapes are not observed, only feature vectors. A given shape may occur often in a class, but in this non-ideal case, the evidence is split across disjoint instances. Thus, if there are n feature vectors for the same shape, then determining the same degree of correlation certainty requires n times more observations in the form of labelled training data. The problem can be even worse when inferring relationships between two different shapes, which underlies topic models in Section 5.2.

3D sensing has the potential for greater viewpoint invariance than is possible with 2D visual imagery, as the 3D geometry is inherently object-centric. However, the geometry must be inferred from the sensed 3D point cloud, which does vary with viewpoint. Changes in the position of a sensor with respect to the environment results in changes to the distribution of 3D points in the scene. An example of this variation

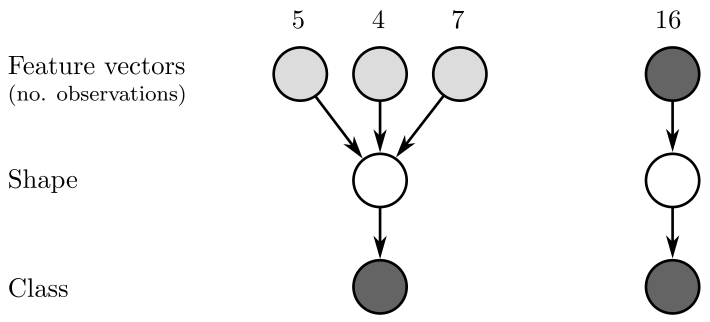


Figure 4.1 – If many different feature vectors represent a single shape (left), the evidence connecting features to classes is split up. A viewpoint invariant feature (right) is able to accrue more evidence from the same amount of training data.

is shown in Figure 4.2, where a fixed location is observed from several positions. Two primary changes are density and occlusion. Density depends on the distance to the surface, and the relative angle of the surface to the sensor, both of which change with viewpoint. Some regions become very sparse, with large gaps between points. An object's silhouette is where it occludes itself, and this also changes with viewpoint. Objects also occlude others as they cast shadows in the 3D data. These effects lead to issues when trying to capture the shape of a region of points, as points appear or disappear depending on viewpoint. Existing features are not robust to these issues (described in further detail in Section 4.1.3).

4.1.2 Occupancy

A more consistent representation of the range image data considers the notion of occupancy, where there are three types of space: occupied, empty and unknown. Occupied space is detected by points on a sensed surface. Empty space exists along the ray between the sensor and the sensed surface point. All other space is unknown, as it has not been sensed. This approach is used in occupancy grid mapping [82], where the world is partitioned into a grid, and ray tracing is used to identify known-empty cells. Without this concept of occupancy, sparse sampling and occlusion lead to confusion between what is observed empty space and what is unknown space. The space between sparse 3D points is unknown space, as is the region in the shadow of a foreground object. Observed empty space, on the other hand, surrounds an object,

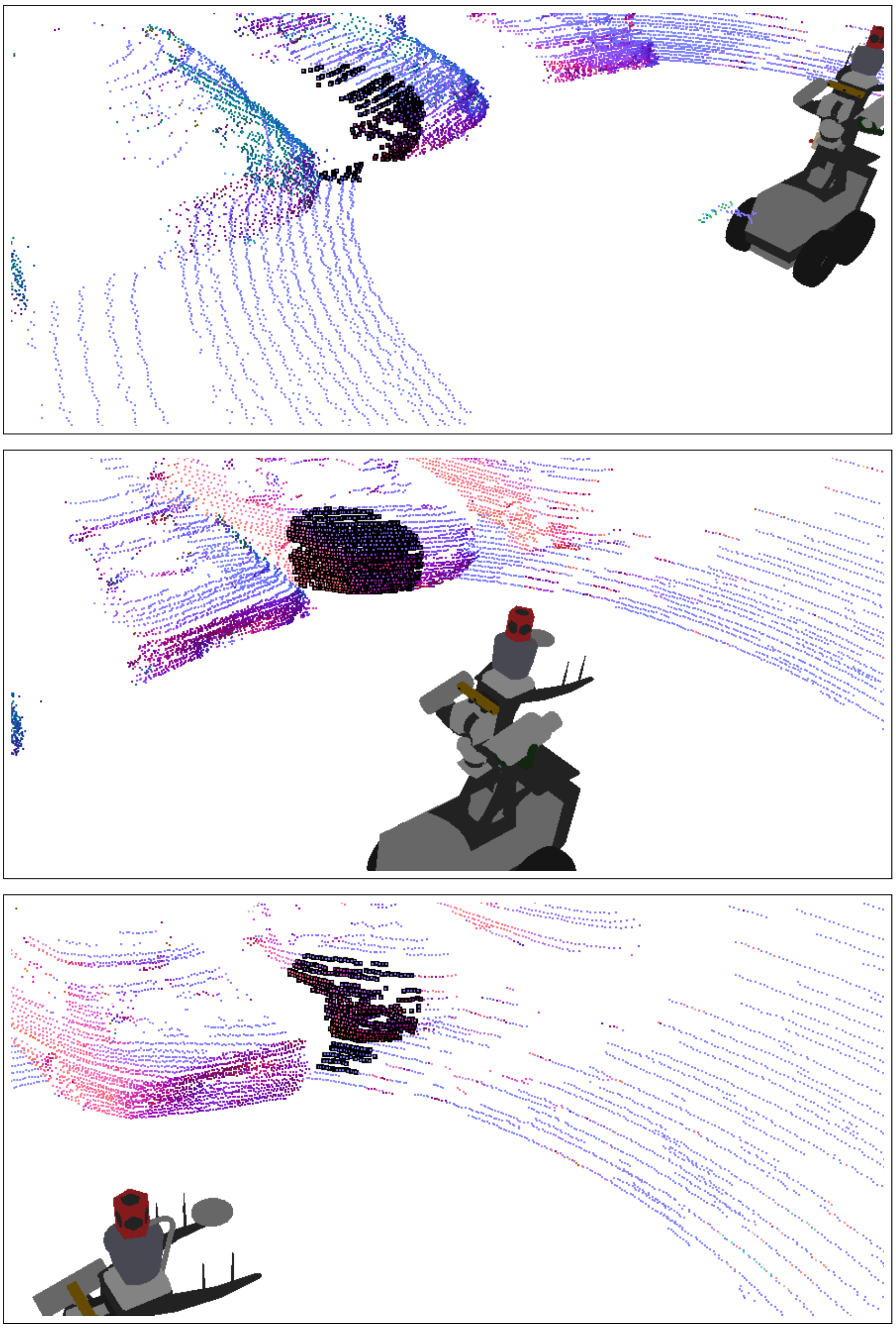


Figure 4.2 – The effect of viewpoint changes on a point cloud. The sensor vehicle was driven through a car park, and the point clouds aligned into a single frame of reference. A fixed region on a car corner was selected (black). The distribution of points changes greatly with viewpoint.

defining geometry by indicating where a surface does not exist.

Empty space is a potentially useful source of information. Capturing as much distinguishing information as possible is important for sparse range imaging, where there is limited sensing resolution and subsequent geometric detail. Consider an object silhouette, which is a 2D outlining shape that forms useful information in computer vision. Along a silhouette, observed empty space surrounds an object, defining a viewpoint-dependent 2D boundary. Ideally, this geometric information should be captured, but the object also occludes itself here. Capturing shape information along silhouettes in a viewpoint invariant manner requires that empty and occluded space be captured and distinguished. While occluded space is effectively missing data that will change with viewpoint, empty space is invariant, distinguishing information.

Current features in the literature do not explicitly encode both unknown and empty spaces, which can lead to a decreased robustness in dynamic outdoor sensing conditions where occlusion is prevalent and there is limited resolution. In order to consistently deal with occlusion while capturing as much information as possible, this chapter contributes the ‘line image’ feature. It detects occupancy by using the range image, providing a representation of the surface, empty space and unknown space. Upon comparison with another region, the unknown portions act as ‘wild cards’ in the matching process; they are permitted to match to anything. This makes the feature highly invariant to viewpoint changes and occlusion, while capturing salient information about the empty spaces within a region.

4.1.3 Existing Features

An overview of existing features was given in Section 2.3; three features will now be described in further detail, as they are used in the experiments in Section 4.4. In particular, the effects of variable point density and occlusion will be discussed. In existing features, occlusions are either ignored or confused with empty space. Most descriptors inherently presume space with no points is empty space, and the violation of this assumption due to occlusion and non uniform sampling causes a lack of robustness.

4.1.3.1 Spin Image

The spin image [43] requires a central point and orienting surface normal. Surrounding points are transformed into perpendicular and parallel components (Figure 2.10). This can be visualised by placing a grid or image at the centre point, and spinning it about the surface normal, accumulating points in each cell. Several spin images are shown in Figure 4.3. This representation only requires a single, central surface normal rather than a full set of 3D axes, which can be difficult to define (Section 3.3).

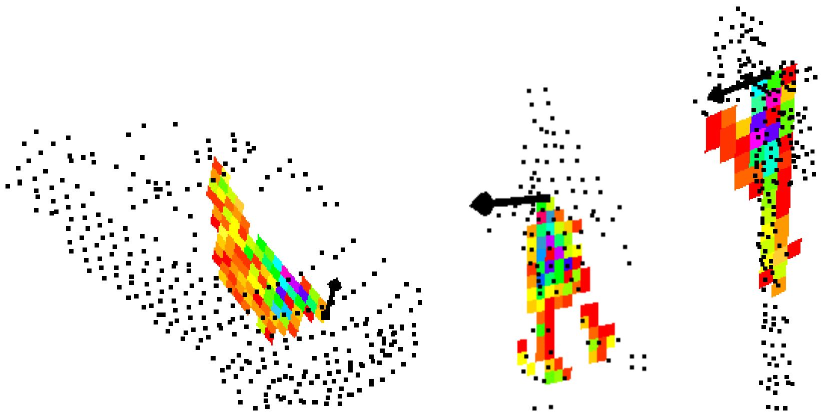


Figure 4.3 – Spin images on a car, pedestrian and traffic sign. The large black arrow is the central surface normal, with the image aligned to it. The image is swept around the arrow, accumulating points. Empty cells are not shown.

To compare two spin images, the linear correlation coefficient R is computed. With two spin images P and Q, with N bins:

$$R(P,Q) = \frac{N\sum p_i q_i - \sum p_i \sum q_i}{\sqrt{(N\sum p_i^2 - (\sum p_i{}^2))(N\sum q_i^2 - (\sum q_i{}^2))}} \tag{4.1}$$

In order to handle data with occlusions, a second similarity measure C (Equation (4.2)) was provided. In this measure, the correlation coefficient only includes occupied bins, treating empty bins as potentially occluded, and effectively allowing them to match to anything. Thus N becomes the number of bins that are occupied and mutually overlapping between two spin images. Using fewer bins results in a lower confidence in the correlation coefficient, and so this is penalised by a second term, scaled by a constant λ.

$$C(P,Q) = (\operatorname{arctanh} R(P,Q))^2 - \lambda \left(\frac{1}{N-3} \right) \tag{4.2}$$

While this accounts for occlusion, it does so by treating *empty* space as occluded space and ignores both. As such, only the relative values of occupied bins are actually used to distinguish and match regions. Removing empty bins weakens the ability to discriminate shapes, as empty space can match to occupied space and not be penalised. In addition, bins encapsulating both occupied space (sensed points) and unknown space can still change. This is because if unknown space is observed, revealing more points, the bin count will increase.

Another problem is the density requirements of the spin image. Spin images generated from regions of different point densities can be compared, as the linear correlation coefficient accounts for this. However, the point density within the region that the spin image encompasses must be consistent. As the spin image infers shape from the density of points in bins, varying the density of points in a non-uniform manner will impact the representation. This is problematic, as the density in a range image varies with surface orientation. Additionally, the size of bins must be above the point density, or bins passing through the surface will not collect any points. These can be addressed by first computing a triangular mesh over the points. The spin image computation then involves the integral of each triangle across the swept volume of each pixel. However, generating a clean surface mesh on a Velodyne scan is problematic due to sparsity and noise.

4.1.3.2 Fast Point Feature Histograms (FPFH)

The Fast Point Feature Histogram (FPFH) [71] is a recent feature that describes the local variation in surface normals using a histogram. It builds on previous iterations of Point Feature Histogram features [69], and more variations exist [70]. The FPFH requires a surface normal at each point.

To compute it on a given region, consider the central query point p_c and a spherical neighbourhood of points $\mathbf{p}$ around it. Each neighbour and the central point form a pair of points p_i and p_j, with a line between them[1]. These two points also have two corresponding surface normals n_i and n_j. The angles formed between these surface normals and the connecting line are captured in three feature values, detailed in Table 4.1.

Table 4.1 – Computation of the three features comprising the SPFH, as in [71]

A coordinate frame is defined based on the two points p_i,p_j and their surface normals n_i,n_j.

$$\begin{aligned} u &= n_i \\ v &= (p_j - p_i) \times u \\ w &= u \times v \end{aligned}$$

The difference between surface normals and point positions are then captured in three features:

$$\begin{aligned} \alpha &= v \cdot n_j \\ \phi &= (u \cdot (p_j - p_i))/\|p_j - p_i\| \\ \theta &= \arctan \frac{w \cdot n_j}{u \cdot n_j} \end{aligned}$$

These values are computed across all such point pairs, and are accumulated into three respective histograms. These histograms are then normalised by the number of points to achieve density invariance. They are then concatenated together to form the *Simplified* Point Feature Histogram (SPFH).

FPFH then goes a step further, and considers neighbourhoods of SPFH. Note this means that SPFH must first be computed about every point in the target region (or an even subsample). Once again, consider a central point p_c, and surrounding points $\mathbf{p}$, this time each with their own histogram. All these histograms are summed, weighted by the 3D distance to the central point, to finally create the FPFH. Thus, for each

[1] p_i is the point with the smaller angle between its surface normal and the line connecting the two points.

neighbouring point p_k and corresponding distance ω_k:

$$\mathrm{FPFH}(p) = \mathrm{SPF}(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k}\cdot \mathrm{SPF}(p_k) \tag{4.3}$$

An implementation exists in the Point Cloud Library (PCL) [68], where the final feature comprises of 33 dimensions, or 3 histograms of 11 bins. In order to match FPFH features, the Histogram Intersection Kernel [71] is used. With two features f_1, f_2 with N dimensions, the distance is simply the sum of the element-wise minimum:

$$d(f_1, f_2) = \sum_{i=1}^{N}\min(f_1^i, f_2^i) \tag{4.4}$$

While this feature is invariant to constant density changes, a variation of density within the region of analysis will affect the histogram. In most cases, an occlusion will also change the histogram.

4.1.3.3 Normal Aligned Radial Feature (NARF)

The Normal Aligned Radial Feature (NARF) [76] seeks to capture empty space, such that the outer shape of objects can be described. However, it does does not consider unknown space. NARF also has a corresponding keypoint formulation, described earlier in Section 3.3.1, and an implementation is available in PCL.

To compute it about a central point, the spherical region of surrounding points is first selected. These are transformed into a coordinate frame relative to the central

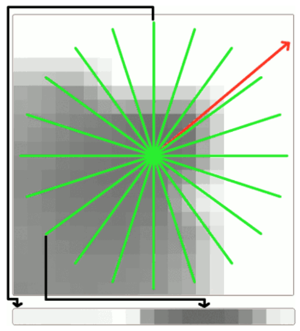


Figure 4.4 – From [76], a normal-aligned image patch of a chair corner, with each green beam tracing out pixels, accumulated into a single dimension of the NARF feature vector. The red arrow is the dominant direction, defining the starting position for the rotationally invariant version.

point's surface normal, such that the normal vector becomes the z-axis. The x,y point coordinates are then placed into an 'image' of 2D cells, creating a normal-aligned image, effectively looking along the normal at the surface. The image patch is blurred, and then a star-shaped set of beams (Figure 4.4) are placed on the image. Each beam traces over pixels, whose depth values are accumulated in a weighted function, resulting in a single value per beam. Empty pixels are given a maximum negative depth, resulting in a distinguishing beam value. Each beam value corresponds to a dimension in the NARF feature. To produce a rotationally-invariant feature, the largest beam value defines an aligning orientation. Alternately, the vertical orientation can be used.

In terms of viewpoint invariance, the underlying normal-aligned image patch (which NARF summarises) has several issues. First, empty space is defined as a lack of points, which considers visibly-empty space and unknown space as the same. As such, occluding foreground objects and missing sensor data are registered as edges[2]. Also, in sparse sensing, the local 3D point density may be too low for the chosen image patch resolution, requiring interpolation.

Finally, consider a region encompassing a silhouette edge, which NARF seeks to capture. The normal-aligned image patch effectively 'looks along' the surface normal, which is offset from the sensor viewing angle. At any such offset angle, the image patch is actually looking around the occluding edge, registering the unknown space as empty. Were the sensor position to change, this unknown space may change to become a surface, resulting in the image patch changing. This depends on the object; the region near a sharp edge on a building would have the same normal-aligned image patch regardless of viewpoint. However, curved surfaces with a 'receding' silhouette would have a changing patch. This demonstrates a conflict in normal-aligned 3D features: the normal-aligned frame is desired for viewpoint invariance, but the sensory data is affected by the sensor's view. The normal-aligned frame can 'look' at unknown regions. This motivates the explicit use of occupancy in a 3D feature.

[2]NARF keypoints distinguish between edges from foreground occlusion and real object silhouette edges, focusing keypoints near the latter. However, this does not preclude resulting regions from being partly occluded, particularly at larger scales.

4.1.4 Summary

In summary, variable density and occlusion can cause problems with existing features. Occupancy is treated in two ways in the literature: (1) Unknown space is assumed to be empty space, which is not a good assumption, as it will change with viewpoint/occlusion. (2) Any space with no data (empty space and unknown space) is disregarded during matching, as in spin images, resulting in a loss of information. There are also general problems with variable and low point density, which can be viewed as a case of unknown space.

The contribution of this chapter is the line image feature, which addresses these issues by explicitly considering occupancy. Unknown regions from low density or occlusion are distinguished from empty regions, with both stored in the feature. Unknown parts in the feature can then match to anything, acting as wild cards. The empty space provides distinguishing information such as along object silhouettes. This aims to improve viewpoint invariance and distinguishability, resulting in more robust region matching in sparse range image data.

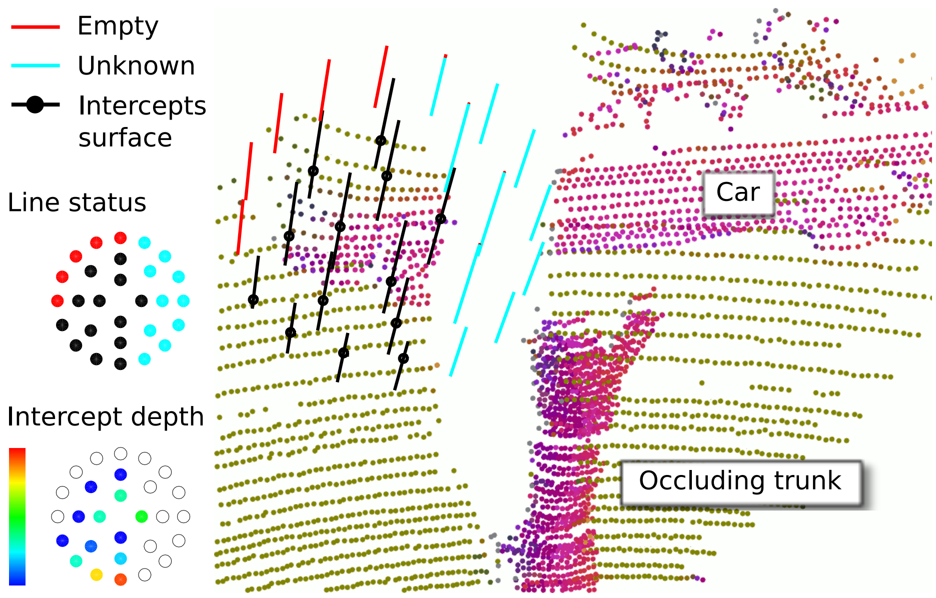


Figure 4.5 – The line image feature computed on the corner of a car, which has been partly occluded by a tree trunk. The status of each line, and the depth of surface intercept, is shown on the left. The point cloud is coloured by surface normal for visualisation.

4.2 The Line Image

This section describes the components of the line image feature [62]. The core concept is in placing a set of 3D line segments in space and detecting whether they become occluded or intercept a surface. This is shown in Figure 4.5. Each line generates two values: (1) A discrete 'status' label s_i indicating whether the line was only in empty space, went into unknown space, or intercepted a surface. (2) A depth value d_i indicates where the unknown space or surface was found.

For a specific point in space, the line image feature $\mathbf{f}$ is represented by two equally sized vectors $\mathbf{d}$ and $\mathbf{s}$:

$$\begin{aligned} \mathbf{f} &= (\mathbf{d}, \mathbf{s}) \\ \mathbf{d} &= (d_1, d_2, \dots, d_n), \quad d_i \in \Re \\ \mathbf{s} &= (s_1, s_2, \dots, s_n), \quad s_i \in \{intercept, occlusion, empty\} \end{aligned} \tag{4.5}$$

for a feature containing n lines. Computing the distance between two features $\mathbf{f}_1$,

$\mathbf{f}_2$ (detailed in Section 4.2.4) is a function of the Root-mean-square error (RMSE) between $\mathbf{d}_1$ and $\mathbf{d}_2$, excluding entries corresponding to occlusions.

Generating this feature involves inspecting the occupancy along each line segment, which is done efficiently using the range image, detailed in Section 4.2.1. Detecting where line segments intercept a surface is then detailed (4.2.2). Multiple line segments are combined into a full line image feature (4.2.3), and matched to other line images (4.2.4). Then, in Section 4.4, the line image is compared to the existing features discussed previously on the task of object classification.

4.2.1 Detecting Occupancy Along a Line

In range sensing, to detect whether a region of 3D space is empty, occupied or unknown, ray tracing is often employed, as mentioned in Section 2.1.1. Each laser return defines a ray from the sensor, through free space to an occupied point in the world. Typically, the sensory data is accumulated in a voxel grid. If a ray intersects and passes through a voxel, that voxel is deemed to be empty. If no rays enter the voxel, it is in an unknown state. This process is computationally intensive, particularly if higher resolution voxel grids are desired, and the data rate is high. This is the case for the Velodyne, with 1.3 million rays per second.

4.2.1.1 Range image occupancy

For 3D data organised in a range image, there is an alternative method, as each 2D point in the image represents a ray. For clarity, 2D will mean the range image coordinate space, with azimuth and elevation in this case (depending on the sensor, a camera model may be used instead). Angular distance thus refers to the 2D distance between points on an image. Sensed points will refer to the data obtained from the range image sensor, as opposed to synthetic points used during computation.

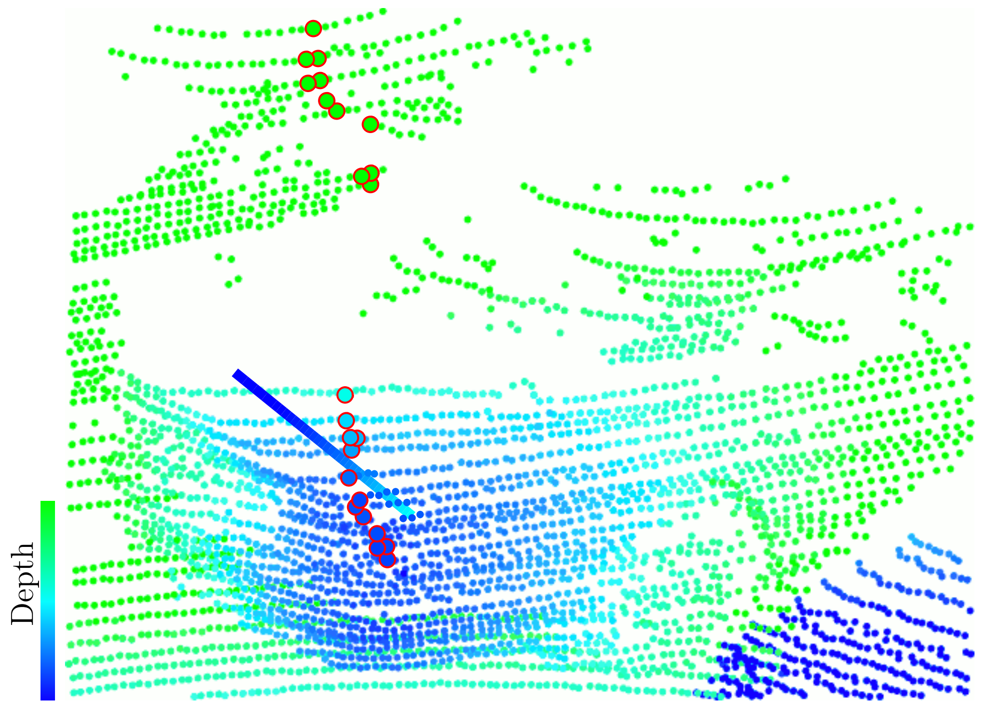
The occupancy at an arbitrary synthetic 3D query point p_q^{3D} can be determined by transforming it into the range image, and simply checking if it is ‘in front’ or ‘behind’ the image data, from the point of view of the sensor. Consider the query point's image coordinates p_q^{2D} and depth d_q, and the nearest sensed point in the image p_s^{2D}, with depth d_s. If the query point is behind the sensed point, is it occluded, and in unknown space ($d_q > d_s$). If the query point is in front of the sensed point, it is in observed empty space ($d_q < d_s$). If these points are very near each other ($d_q \approx d_s$), the query point may lie near a surface (this will be elaborated upon in the next section). Finally, in the case that the range image has portions of missing data, for instance from non returns or sensor failures, there may not be a nearby sensed point in the 2D image space. In this case, the query point is in unknown space. These cases cover detecting empty, unknown and occupied space in a range image, all using a single spherical transform, a nearest (2D) neighbour look-up, and a range comparison.

4.2.1.2 Selecting points along a line

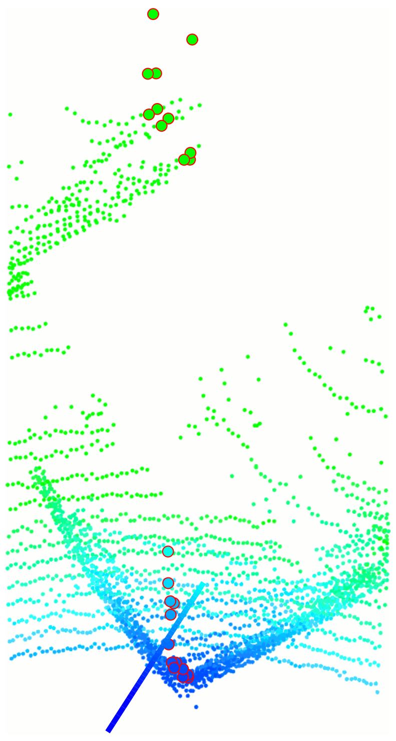
This concept can be extended to a region in 3D space defined by a line. To determine the occupancy along a line segment, first, sensed points will be selected if they have a corresponding ray passing near the line. The range image is used to facilitate this, shown in Figure 4.6. If the line segment is sufficiently short and far away, it can be approximated in the range image by a 2D line. If a given sensed point is close enough to the 2D line, it is selected. The point is connected to the line at its nearest approach, producing a corresponding point on the line P_{line}. This process is detailed in Algorithm 2.

In order to efficiently select points, the range image grid is used, described earlier in Section 3.1.3. This stores 2D points into a grid, with each cell containing multiple points. In particular, cells are not square, and vary in size. While cells are evenly spaced in azimuth, each of the 64 rows are assigned to each laser scan line, resulting in uneven elevation 'heights' of each row. For instance, the upper 32 lasers are angled closer together in elevation, resulting in the upper 32 image grid rows being shorter.

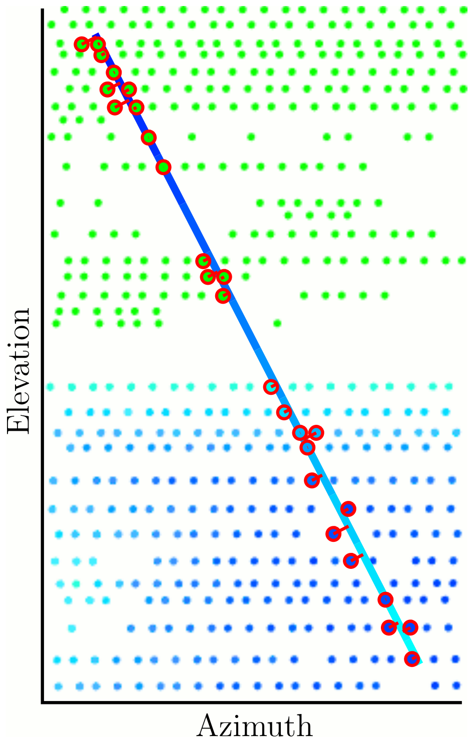
The process of selecting points is detailed in Algorithm 2. First, the cells along the line are traced by determining the line's position at each cell boundary. Surrounding cells



(a)



(b)



(c)

Figure 4.6 – Detecting occupancy along a line. (a) A 3D point cloud of a car, with points coloured by depth, and a line segment which intercepts the car. This is viewed from above in (b). (c) The same scene in the range image. The large, red-outlined points near the line segment in the range image are also shown in (a). These points quantify that the line segment is in front of the scene at the top, and behind the scene at the bottom.

within the small angular distance t_s are also selected. The 2D sensed points within these cells are projected onto the 2D line. Those further than t_s are removed, with the remaining points sorted by position along the line. This produces two equal-length lists of points: P_{sensed} and P_{line} (the projected points on the line).

4.2.1.3 Line occupancy transitions

These points determine the occupancy along the line. In the line image, a line is typically positioned to begin above a surface, and moves into and behind a surface Traversing along the line in this order, (iterating through P_{sensed} and P_{line} in line 3 of Algorithm 3), each sensed point is compared with its corresponding line point, determining the line's occupancy up to that position. The state of the line thus begins as 'empty', and is changed to unknown or intercepted as the line is traversed. This process is shown in Algorithm 3, with a corresponding flow diagram in Figure 4.7 detailing how the state of the line changes.

The final state of the line is set to s_i, the output status of the line. While a line may pass through alternating regions of empty space, unknown space, and even several surfaces, a single value is desired for simplicity. Similarly, the position of the first surface intercept is stored in d_i. If the line enters unknown space, the last position of where it was in empty space is stored in d_i. In this case, the line becomes empty up to d_i, at which point is becomes unknown for the remainder of the line[3]. This makes it more distinguishable when matching line images (Section 4.2.4), as the 'empty space' portion cannot match to a surface intercept.

To begin with (state = empty, Figure 4.7a, or line 6 of Algorithm 3), the algorithm checks if a surface intercept is nearby, detailed in the next section. If there was no intercept, the algorithm checks for unknown space. This occurs from (1) a lack of sensory data, or (2) occlusion. In the first case, some regions of a range image can have no data (e.g. due to laser non-returns), resulting in a significant gap in the points along a line. If such a gap occurs (line 12 of Algorithm 3) the line is considered

[3]Unless a surface intercept is found.

unknown from the last observed point before the gap. The second case of occlusion is detected by comparing the depth value of the sensed point to the depth value on the line. If the line is in front of the sensed point, the line is in empty space, else it is in unknown space.

This process continues as each point along the line is inspected. Once 'unknown', the line cannot subsequently become empty if it encounters empty space, as the earlier unknown space may contain a surface. However, it is allowed to transition to intercepted, so as not to miss capturing the informative surface; this is elaborated upon in Section 4.2.6. When the line registers an intercept, some additional processing is done to ensure the best intercept is found, detailed in the next section, after which the line traversal is ended.

```
Input  : Line segment $L^{3D} \in (L^{3D}_{start}, L^{3D}_{end})$
Output : Sensed points $P_{sensed}$, and their nearest points on the line $P_{line}$
1  $L^{2D}_{start}, L^{2D}_{end}$ = 3DTo2D($L^{3D}$);
2  ClipToImageBoundary($L^{2D}_{start}, L^{2D}_{end}$);
3  $L^{grid}_{start}, L^{grid}_{end}$ = 2DToGrid($L^{2D}_{start}, L^{2D}_{end}$);
4  cells $\in \emptyset$;
5  for $(x,y) = L^{grid}_{start}$ to $L^{grid}_{end}$ do
6      append to cells : (x,y) and directly neighbouring cells;
7  $P$ = GetSensedPointsFromGrid($cells$);
8  $P_{sensed}, P_{line}, D_{line} \in \emptyset$;
9  for $p_{si} \in P$ do
10     distance, $p_{li}$ = ProjectOnLine($p_{si}, L^{2D}$);
11     if $(distance < t_s) \wedge (p_{li}$ inside $L^{2D})$ then
12         append to $P_{sensed}$ : $p_{si}$;
13         append to $P_{line}$ : $p_{li}$;
14         append to $D_{line}$ : $\|p_{li} - L^{2D}_{start}\|$;
15 Sort $P_{sensed}$ and $P_{line}$ by $D_{line}$;
16 Return $P_{sensed}$, $P_{line}$ ;
```

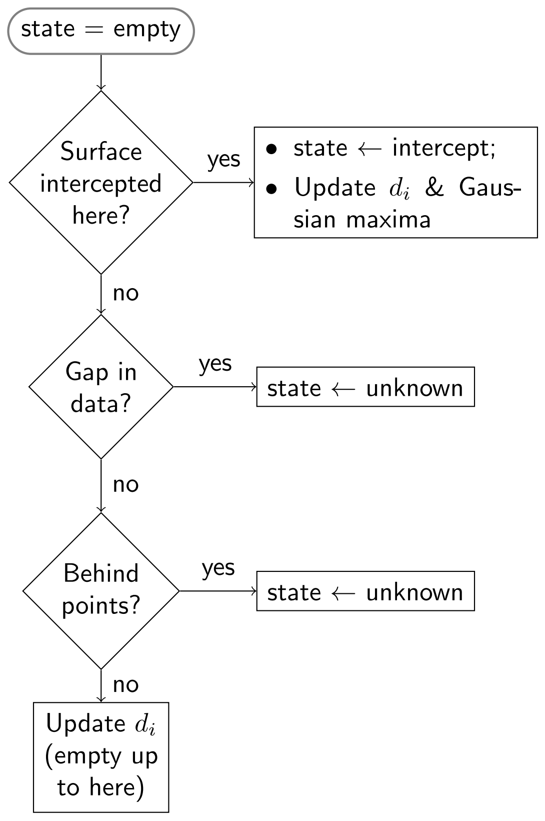
Algorithm 2: `PointsOnLine`, shown visually in Figure 4.6. A 3D line segment is transformed into the range image grid, and the corresponding sensed points are selected and ordered.

Table 4.2 – Parameters and variables used in the line image, and in Algorithms 2 to 4. Values were set to reflect the density of Velodyne data.

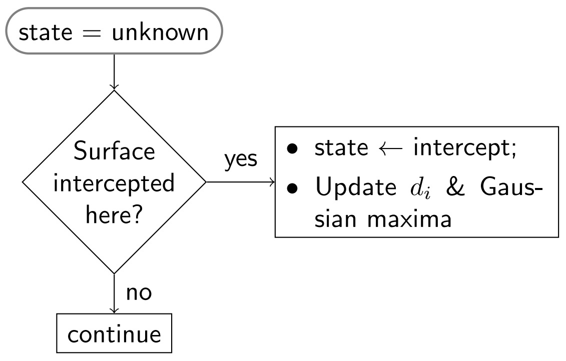
Variable	Description	Value
Line parameters		
t_s	Points are selected if they are closer than this 2D distance from the line segment.	0.28°
t_a	The line is unknown if there is no data along the line for this (2D) angular length.	2°
R_{PCA}	PCA is computed on a spherical region of points at this size.	0.3 m
t_p	A region of points is represented by a 3D Gaussian. The line segment passes through, obtaining a maximum value. It must be higher than this threshold to be deemed a surface intercept.	0.2
t_{length}	Once a surface intercept is found, continue along the line for this short distance (m), looking for a better one (larger Gaussian value).	0.8 m
Line arrangement parameters		
R_{disk}	Outer radius of circular disk pattern of lines. Each ring of lines is positioned at a radius, evenly spaced up to the outer radius. This defines the scale of analysis.	
$N_{sections}$	The number of lines in each radial section, beginning from the innermost ring.	{8,16,32, 32,64,64}
Algorithm variables		
P_{sensed}	Sensed points near the line (red in Figure 4.6).	
P_{line}	Points on the line closest (in 2D) to P_{sensed}.	
$\mathcal{N}(p; \mu, \Sigma, \Lambda)$	3D Gaussian function evaluated at p, with parameters: mean μ, eigenvalues Σ and eigenvectors Λ.	

```
Input : line segment $L$
Output: $s_i$, $d_i$
$P_{sensed}, P_{line} \leftarrow$ PointsOnLine($L$);
$s_i \leftarrow$ empty;
for $p_{si} \in P_{sensed}$ , $p_{li} \in P_{line}$ do
    switch $s_i$ do
        $isIntercepted$, $length$, $g \leftarrow$ SurfaceIntercept($p_{si}$, $L$);
        case $empty$
            if $isIntercepted$ then
                $s_i \leftarrow$ intercept;
                $d_i \leftarrow length$;
                $g_{best} \leftarrow g$;
                continue;
            if $\|$ 3DTo2D($p_{li} - p_{(li-1)}$) $\| > t_a$ then            // Gap in data.
                $s_i \leftarrow$ unknown;
            if $\|p_{si}\| < \|p_{li}\|$ then                    // Line is behind point.
                $s_i \leftarrow$ unknown;
            else                                                 // Still in empty space.
                $d_i \leftarrow$ LineLength($p_{li}$);      // Line was last visible here.
            continue;
        case $unknown$
            if $isIntercepted$ then
                $s_i \leftarrow$ intercept;
                $d_i \leftarrow length$;
                $g_{best} \leftarrow g$;
            continue;
        case $intercept$                               // Was already intercepted.
            if $\|$ LineLength($p_{li}$) $-d_i\| > t_{length}$ then
                break 2;           // Too far from best existing intercept
            if $isIntercepted \wedge (g > g_{best})$ then
                $d_i \leftarrow length$;
                $g_{best} \leftarrow g$;
            continue;
Return $s_i$, $d_i$ ;
```

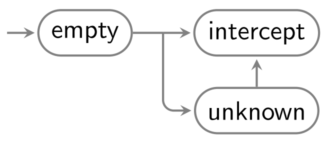
Algorithm 3: Computing s_i and d_i for a line segment. Also see the flow diagrams in Figure 4.7, which provide an equivalent description of the case structure within the for loop.



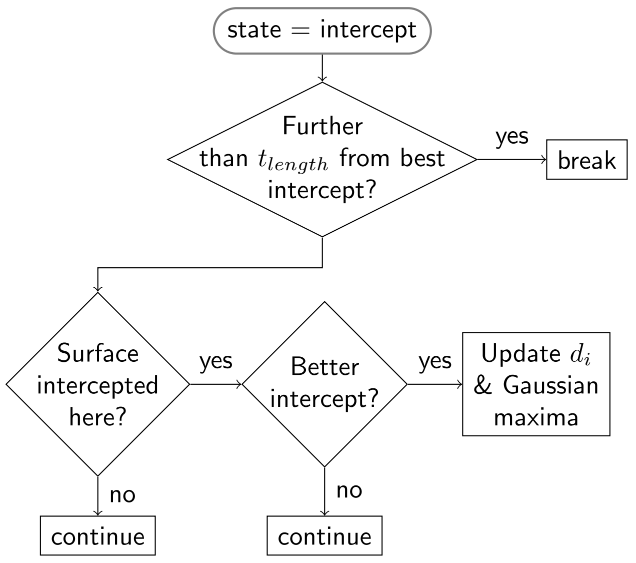
(a) state = empty



(c) state = unknown



(b) State transitions


state = intercept
Further than t_{length} from best intercept?
yes
break
Surface intercepted here?
yes
Better intercept?
yes
Update d_i & Gaussian maxima
no
continue
no
continue

(d) state = intercept

Figure 4.7 – When traversing along a line, each sensed point is inspected (for loop in Algorithm 3), changing the state of the line. The state begins as empty (typically above the surface), changing to unknown if there is a large gap in the data, or a point is far in front of the line. If an intercept is detected, the line continues to be traversed for a certain length, to ensure the best intercept is found.

4.2.2 Detecting Surface Intercepts

This section describes the task of detecting whether the line intercepts a surface and where. This necessitates some form of surface model to interpolate the surface at the line. While the bearing graph described in Section 3.1.2 can be converted into a triangular mesh, it is not very robust. Any noise or incorrectly connected or removed edges between points can result in an incorrect surface. Triangulation in general is avoided in this work due to the sparsity and noise of the points from a Velodyne scan. In addition, thin poles can have only a single line of points, not enough for triangulation.

This work uses a local Gaussian representation, fitted by computing PCA at each point, as described earlier in Section Section 3.2.1.2. Therefore, before computing the line image, PCA must be computed on all regions of the point cloud. In particular, if the PCA keypoint algorithm from Section 3.3.2 is used, no additional computation is required. An example of such a set of Gaussians is shown in Figure 4.8.

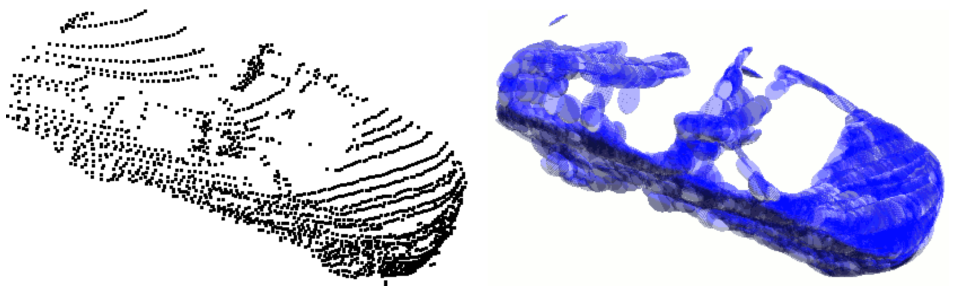


Figure 4.8 – (a) A scan of a car. (b) Ellipsoids representing local Gaussians, used in surface interpolation.

To compute the surface intercept, recall that each point along the line is inspected in Figure 4.7. Each point also has an associated Gaussian function. Therefore, for a specific point, the maximum value of the corresponding Gaussian is determined along the line[4]. If this value is above a threshold, it is considered an intercept. This method is illustrated in Figure 4.9, and in Algorithm 4. The local Gaussian function represents

[4]This involves a simple transform of the line coordinates followed by a projection

noisy regions of points, approximating simple planar or linear shapes. As before, the point density of the region is checked to ensure there is a sufficient distribution of points (Section 3.2.1.2).

As there are often several sensed points surrounding the line at the intercept, each with an associated Gaussian, there may be several detected intercepts within close proximity. In this case, the largest Gaussian function value across all intercepts is chosen as the best one. In order to prevent a truly separate, second surface from overriding the first, the line is only traversed a further 3D distance t_{length} m.

This approach presents a balance between simple, fast methods such as nearest-neighbour interpolation, and more intensive surface modelling techniques, such as fitting splines. It is worth noting that, in the case that the surface model fails to interpolate a given surface, the line image handles this gracefully. The line will miss the intercept, but end up behind the points, registering unknown space instead. The penalty is a lower amount of surface information captured in the feature. With the PCA method, this often occurs at large ranges, when the density of points is below the chosen scale of analysis.

```
  Input  : Sensed point $P$, line segment $L$
  Output : $isIntercepted$, $length$
1 if PointDensityLow( $P$, $R_{PCA}$) then
2     return $False$;
3 neighbours = Sphere( $P$, $R_{PCA}$);
4 $\mu, \Sigma, \Lambda$ = PCA(neighbours);
5 $p = \arg\max_p \mathcal{N}(p; \mu, \Sigma, \Lambda)$  subject to  $p \in L$;
6 $g = \mathcal{N}(p; P_\mu, \Sigma, \Lambda)$ if $g > t_p$ then
7     return $True$, LineLength($p$), $g$;
8 return $False$;
```

Algorithm 4: `SurfaceIntercept`

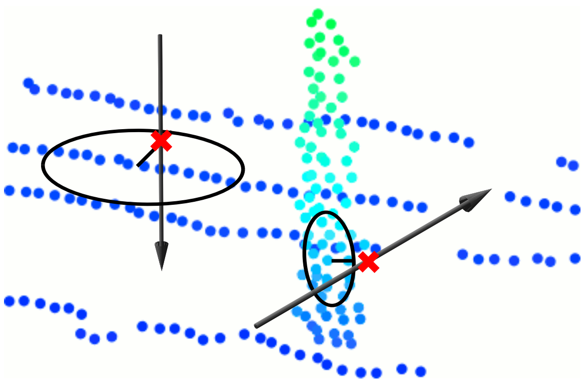
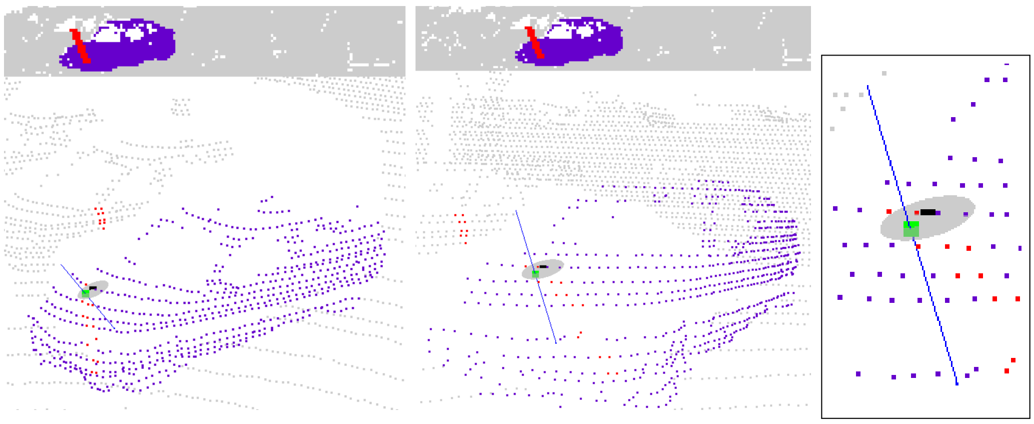


Figure 4.9 – Detecting the surface intercept for two lines (points coloured by height). The Gaussian distributions (shown as ellipses) represent the local point cloud near the line. The maximum value of the Gaussian is found along each line, shown by a red cross. The line passing very near a trunk (right) is outside the threshold, so it does not intercept the surface. The line passing through the sparse ground (left) is within the threshold, providing a surface intercept.


(a)

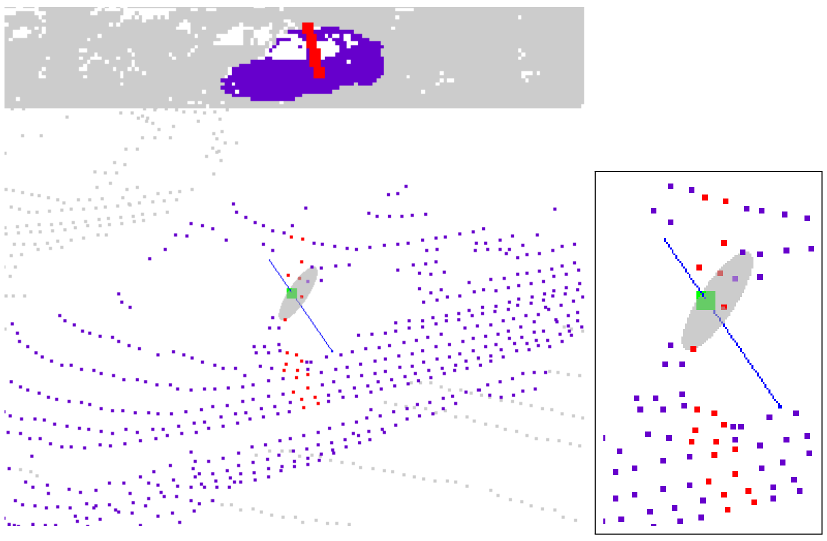
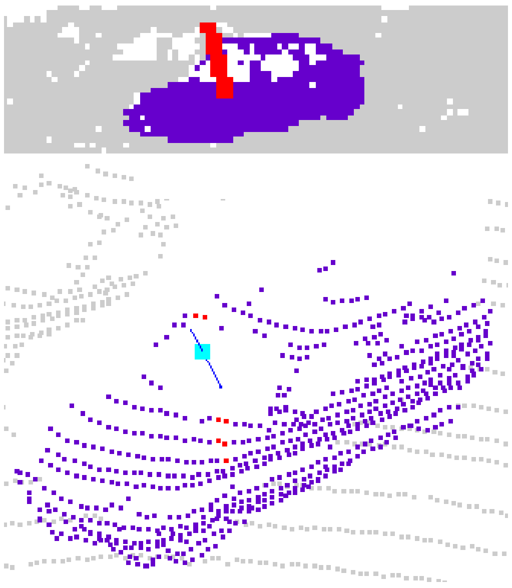
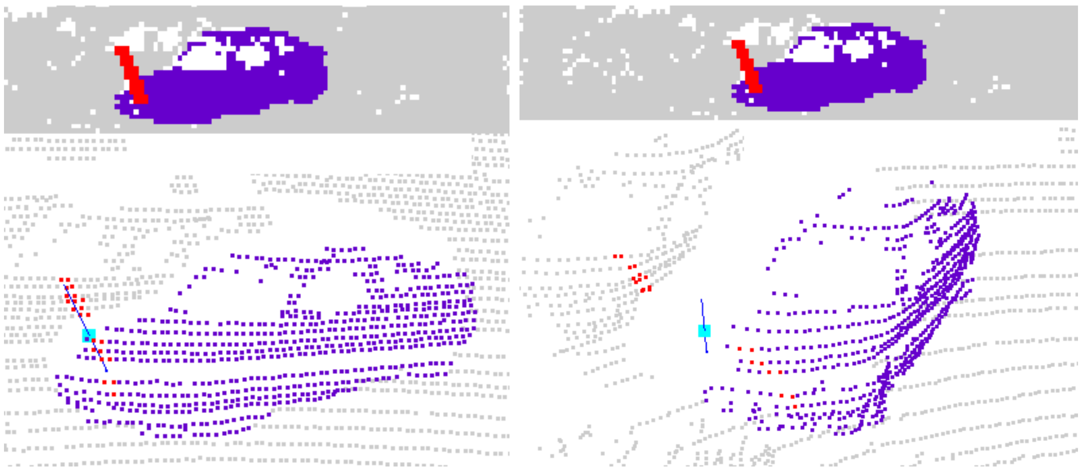

(b)

Figure 4.10 – Occupancy line traversals in various situations about a car. In each image, the range image grid is shown above, with the car points/cells coloured purple. The blue 3D line, oriented from top to bottom, defines the red cells in the image grid, with the associated 3D points also coloured red. These are traversed to find an intercept (large green square). The local Gaussian resulting in this intercept is shown as a grey ellipse. (a) An intercept on the car bonnet (with two different views and a zoomed version). (b) An intercept on the thin door frame.



(a)



(b)

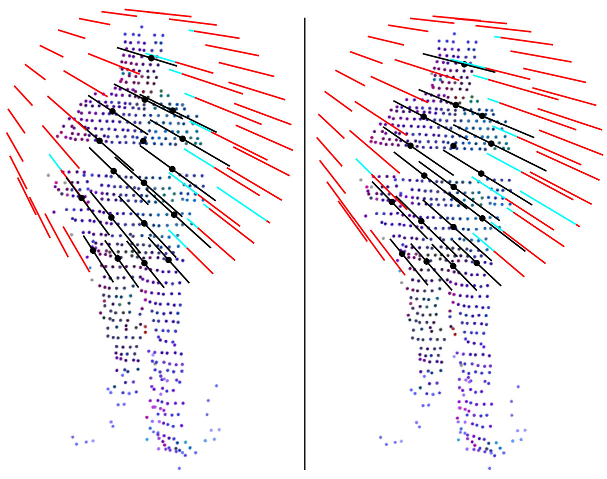
Figure 4.11 – (a) The line encounters the windshield, which gives no laser returns, resulting in missing data. The cyan square indicates the last location the line was in front of the data; it is in unknown space from this point onwards. (b) The line goes behind the car, again noting where it encountered unknown space.

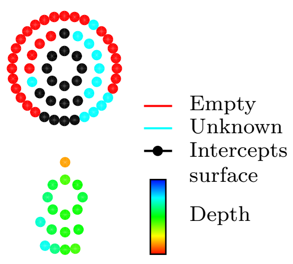
4.2.3 Creating a Detailed Feature

A given line segment can now probe a region of space, detecting emptiness, unknown space or a surface. These operations are demonstrated in Figures 4.10 and 4.11. The line image feature is composed of a set of these lines, describing a local region. First, a central point must be defined, along with a local orientation frame (described earlier in Section Section 3.3.2), where, for instance, the surface normal forms the z coordinate. Lines are then positioned parallel to the local z axis, arranged evenly through the region. This results in a representation similar to a 2½-D height map, with addition occupancy information, shown in Figure 4.5.

Lines are organised in concentric rings about the central point. This arrangement allows for different spin orientations to be compared, detailed in Section 4.2.4. In this work, to produce a detailed feature, 6 rings were used, each one evenly spaced up to R_{disk}. From the innermost ring outwards, the number of lines were $8, 16, 32, 32, 64, 64$, which is 216 in total[5]. This number is a compromise between computational speed and sufficient sampling resolution. The underlying raw points have a limited resolution,

[5]Some figures in this thesis show a reduced number of lines for clarity


(a)



(b)

Figure 4.12 – (a) A line image computed on a person, with large black surface intercepts. This is a cross-eyed autostereogram; to see in 3D, cross your eyes until you see three images, then focus on the centre one. (b) A graphic of the status and depth vectors. Note that there are only sets of $8, 16, 32$ lines here for clarity.

making very high sampling redundant.

The feature was also restricted to describing a spherical region, with lines beginning and ending at the spherical boundary of radius $1.2 \times R_{disk}$ m (this permits the outer ring of lines some length). Note that this arrangement of lines is not critical; a more traditional square image is also possible.

4.2.4 Comparing Line Images

With occlusion measured in the feature, it can be accounted for when determining if two line images $\mathbf{f}$ and $\mathbf{f}'$ are similar (e.g. Figure 4.13). A measure of distance r, similar to the root-mean-square error (RMSE), is generated from the visible (i.e. not unknown) portions of the features. An additional value v measures the amount of visibility, giving a degree of confidence in r. These can be combined in different manners, depending on the aims of matching and classification.

4.2.4.1 Visible difference and mutual visibility

Each line l_i is compared with the corresponding line l'_i in $\mathbf{f}'$, to compute the measures r and v from Equation (4.6). In the simplest case, if $\mathbf{f}$ and $\mathbf{f}'$ contained surface intercepts only, an appropriate measure of distance would be the RMSE of $\mathbf{d}$ and $\mathbf{d}'$. However, adding occlusion and empty space requires an additional logical step. Each line has three states: intercepted at a point, empty, or unknown from a point onwards. The amount they contribute to the comparison value r depends on how much their respective states conflict.

If a surface intercept on l_i falls within an unknown region of l'_i, it is not appropriate to increase the distance measure, as there is no conflict in information. Therefore, the line is not counted in the RMSE calculation, shown in Table 4.3 and Equation (4.6). However, the information available to calculate a distance between $\mathbf{f}$ and $\mathbf{f}'$ has reduced. The loss of surface intercepts to occlusion is reflected in v (Equation (4.7)), which is the percentage of mutually visible points. As an extreme example, if one line image

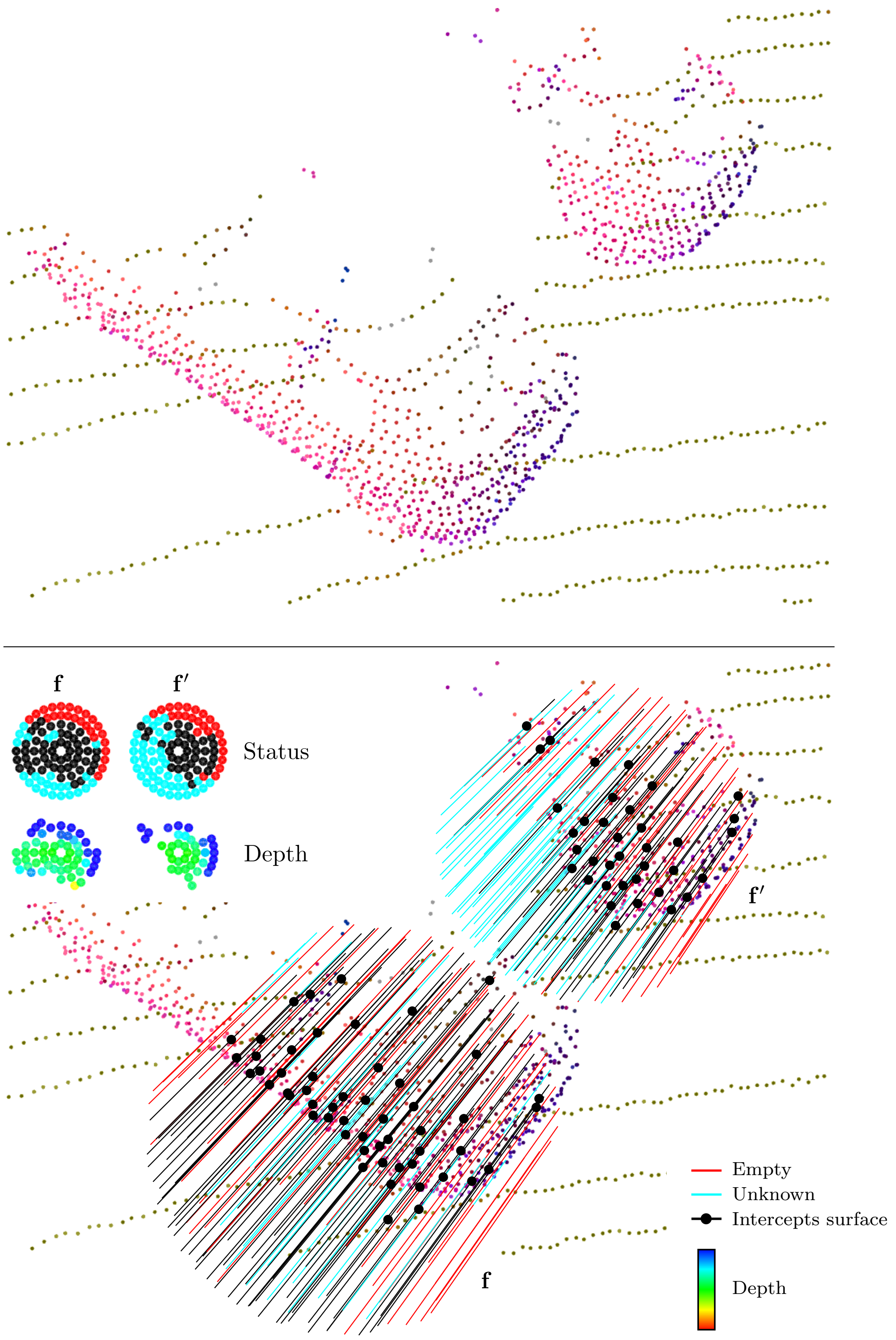


Figure 4.13 – Matching two line images on cars, with the right car (**f′**) partly occluded.

contains mostly unknown lines, another line image may only match well if the majority of the surface intercepts fall in the unknown space. The two features are consistent, but the amount of information we have is low. This provides a measure of confidence in r, which may change greatly if unknown regions become observed.

Another scenario is when a surface intercept on l_i is compared with an empty line l'_i, meriting a large increase in the distance measure. To make empty space distinguishable, the maximum error for these lines are added to the RMSE function (i.e. the whole length). A similar case is when the line l'_i is empty up to point, from which it becomes unknown. If l_i's intercept depth occurs in the empty portion, it is penalised in the same manner.

These cases are summarised in Table 4.3. The value r is the RMSE, not including unknown points, and heavily penalising points in empty space. The value v is the percentage of mutually visible points.

Table 4.3 – Different line types and the effects of matching them. This shows the error values e_i and the increment to n_r (how many valid error values), which are used in Equation (4.6). Similarly, the increment to n_v (number of mutually visible points) is shown, which is used in Equation (4.7).

Line states

—●—	intercepted at d_i
🟥	all empty
🟥🟦	empty up to d_i, then unknown

l_i, l'_i	error e_i	inc. to n_r	inc. to n_v	effect
—●— ——●—	$d_i - d'_i$	1	2	r changes
—●— 🟥	$\|l_i\|$	1	1	r increases
—●— 🟥🟥🟦	$\|l_i\|$	1	1	r increases
—●— 🟥🟦🟦	-	0	0	v decreases
🟥 🟥	0	1	0	r decreases
🟥🟦🟦 🟥🟥🟦	-	0	0	

Each line l_i has the following:

$$\begin{aligned}
s_i &= \text{line status: intercept, empty, unknown} \\
d_i &= \text{location on line where intercept/occlusion occurred} \\
|l_i| &= \text{line length} \\
e_i &= \text{line error, defined in the table above}
\end{aligned}$$

With the resulting distance measures $\{r, v\}$:

$$r = \sqrt{\frac{\sum e_i^2}{n_r}} \tag{4.6}$$

$$\begin{aligned}
v &= \frac{\text{No. points in } \mathbf{f} \text{ visible in } \mathbf{f}' + \text{vice versa}}{\text{No. points in } \mathbf{f} \text{ and } \mathbf{f}'} \\
&= \frac{n_v}{|\{S\,|s_i = \text{intercept}\}\,| + |\{S'\,|s'_i = \text{intercept}\}\,|}
\end{aligned} \tag{4.7}$$

Asymmetric distance

The visibility measure v measures how much surface was lost to unknown space during matching. A slight variation considers only how much surface was lost in 'one direction', from $\mathbf{f}$ to $\mathbf{f}'$.

$$v_a(\mathbf{f}, \mathbf{f}') = \frac{\text{No. points in } \mathbf{f} \text{ visible in } \mathbf{f}'}{\text{No. points in } \mathbf{f}} \tag{4.8}$$

In other words, how much surface from $\mathbf{f}$ was actually used in matching. This *asymmetric* measure permits missing portions to match to more complete instances, but not vice versa. This can be advantageous, as the previous symmetric measure would only permit half-visible features to match best to other half-visible features. This formulation was motivated by the task of clustering, discussed later in Chapter 5, where exemplars with greater visibility were desired. However, it was found to produce better results in k-NN classification as well, discussed in the upcoming Section 4.4.

4.2.4.2 Combined distance measure

The two values $\{r, v\}$ may be used separately, or combined into a single distance measure. A common task is to compare a target feature with a set of features in memory. If a distribution of possible matches is desired, matches with a low distance r may be sought regardless of their visibility v. If only the best matches were desired, as is the case in nearest neighbour classifiers, then matches with a low r and high v would be sought. This draws parallels to the spin image similarity metric (Equation (4.2)), where the number of empty bins is similar to v. In this case, as r is in metres, v is scaled by the radius of the feature (Equation (4.9)), producing the distance measure m. Alternately, standard deviations of the r values across a dataset could be used as scaling factors for v.

$$m = r - v_a \cdot R_{disk} \tag{4.9}$$

4.2.4.3 Alignment Constraints

The ring arrangement of lines was chosen to allow for an alternate method of comparing line images without needing alignment in spin. Recall that, while the surface normal[6] provides a z axis for the lines, another axis is required to constrain the 'spin', to align two line images. Without this, two line images can still be compared in a similar fashion to convolution, where all spin angles are compared, and a best match is returned. However, this can make line images individually less distinguishable, particularly when a large number of unknown line portions permit a larger number of matches. In practice, the vertical orientation (parallel to the direction of gravity, as in Section 3.3.2) produces accurate matching in an urban setting.

This may seem like a step backwards for viewpoint invariance, as objects that roll or pitch significantly (or sensors which do so) are not matched with non-rotated instances. However, this is a limitation with the simple feature matching used in algorithms such as k-nearest neighbours classification, where the different features (parts) of an object are matched independently. Ideally, any rotation would be permitted, but with some coherence across the parts of an object. Without this constraint, features become much less distinguishable; for example, a person can be constructed out of a combination of improbably rotated and combined car parts. Additionally, without higher level contextual awareness (such as where the ground is, or how likely a car is to be rolled 90°), constraining roll and pitch in this context is the simplest approach. This is further discussed and shown in Section 4.4.

4.2.5 Implementation Details and Assumptions

The line image algorithm was implemented for Velodyne data, but could be applied to other range image sensors. No pixel structure for the range data is assumed, only a 2D layout of some form. The Velodyne sensor uses 64 lasers in a spinning head, each producing a row of points at a particular elevation. The points are not sampled in a precise or regular pattern in azimuth, and the lasers are not evenly spaced in elevation.

[6]or perpendicular to the linear direction as in Section 3.3.2

Table 4.4 – Average computation time per single feature (2 m radius), computed on the Sydney Urban Objects Dataset (Section 4.3). Run on a Core2Duo E8400 3 GHz, single thread C++.

Feature	Generation (ms)
Line Image	1.49
Spin Image	0.096
FPFH	0.345
NARF	0.177

Parts of the range image often have no data due to non-returns. Despite this, the line image can still form a consistent representation of the world, as unexpected gaps in data simply results in a line being marked as unknown, without loss of generality. Thresholds and parameters in this feature (shown in Table 4.2) reflect the expected density of the range image, as well as the scale and detail of the shapes to be captured.

The computation time for an average line image is shown in Table 4.4, along with existing features from the literature. The additional complexity makes it about 15 times slower than spin images.

A key assumption this algorithm makes is that the emitted and sensed ray both follow approximately the same path, and are encapsulated by a single 2D point in the range image. Relatively large distances between emitter and sensor (with respect to the operating distance) will break this assumption. The occupancy of a given point can still be determined, but two range images will be required to inspect the emitted and sensed ray separately. The extra range image (describing the emitted rays) can be computed from the original given the relative pose of the emitter to the sensor. Ultimately, the range image could be substituted for an unstructured set of rays, but searching for rays that pass near a given line segment would be computationally intensive.

Another assumption is that the range image is dense enough to not miss entire surfaces between rays. In the case of LiDAR, rays occupy a certain beam width, with gaps between rays that are technically unknown space. However, these are unlikely to hold anything surprising unless the resolution is low enough to miss surfaces. The threshold t_a accounts for large gaps in the image, but the line image would be less informative if this was set to occur everywhere. Sensing at such extremely low resolutions relative to the scale of the desired objects warrants more thorough occupancy mapping.

4.2.6 Caveats

By requiring a specific surface intercept location for each line, small or thin surfaces that can exist between lines are lost. Structures such as people's limbs or thin tree branches require the set of lines to be dense enough to intercept them regularly. This is in contrast to binning techniques, where a low-resolution feature with large bins can still capture points, if not their fine structure.

However, poles *are* successfully captured, due to the PCA keypoint alignment on thin, linear regions. For instance, a PCA keypoint on a pole provides a local x-axis parallel to the pole. The subsequent arrangement of lines results in lines sampling the surface precisely along this x-axis.

In addition, binning techniques permit points to move about within bins, whereas the line intercept is a relatively rigid registration of the local surface. This may mean the line image is *too* specific, with the distance between similar features increasing rapidly upon slight shape variation or misalignment. This could be detrimental to the ability to generalise from training data in classification. Some form of blurring or summary of the depth variation as in NARF [76] could fix this, although it is then difficult to preserve the occupancy information.

Another issue is the negative effect of small to moderate gaps or occlusions in the data. These result in the line registering unknown space from that point onwards, whereas in most cases, the likelihood of a surface occurring in such a space is very low. For example, if most of the line is in empty space, but encountered a small amount of unknown space early on, it will be marked as largely unknown. Subsequently, in matching line images, surfaces are permitted to match to this excessive unknown space, which was in fact empty space. This issue motivated the transition of the unknown state to the intercepted state, so that surface information was not lost. Ultimately, any feature vector will result in the loss of some information present in the raw data. This chapter argues that occupancy (in this case, unknown space) is important information to preserve for viewpoint invariance, but it is equally important to capture distinguishable information (empty space, intercepts).

Finally, the surface interpolation method could be improved upon. A constant scale of analysis was used in this work (0.3 m), which was a trade off of surface complexity and density in Velodyne data. When the density falls too low (e.g. only one horizontal scan line within 0.3 m), a surface is not generated, and the line image registers only unknown space at the surface points. This reduces the amount of distinguishing information at further ranges. More complex multi-scale interpolation would alleviate this.

4.2.7 Summary

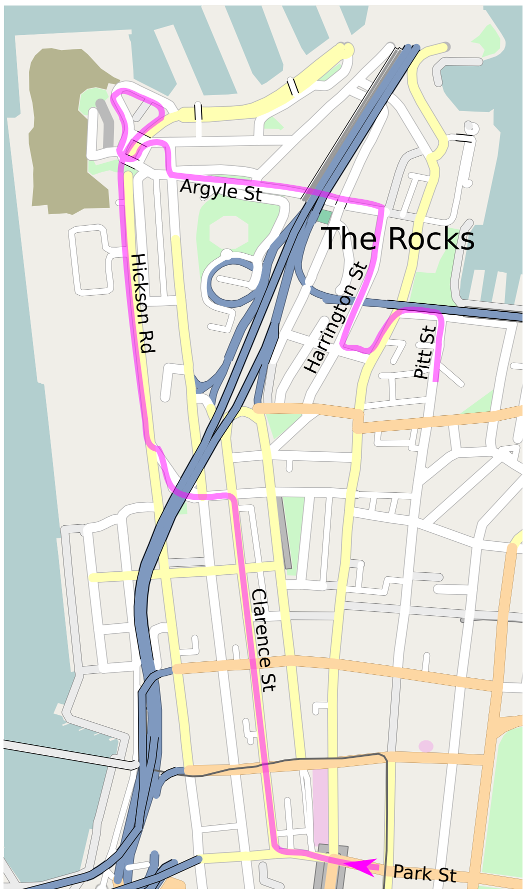
This section contributed the line image, a feature that incorporates occupancy and surface interpolation in order to attain viewpoint invariance. It exploits the 2D structure of the range image, performing line traversals in the range image to select support points to infer occupancy. Surface interpolation is done by simple PCA, producing a 2½-D representation of the surface with additional occupancy information. The unknown portions of each line act as wild cards, allowing anything to match to them, whereas the empty portions contribute distinguishing information. The line image will be applied to a dataset of objects in the following two sections, where it is compared to existing features in the task of classification.

4.3 Sydney Urban Objects Dataset

This dataset [63] contains single Velodyne scans of a variety of common urban road objects, and was collected in order to test matching and classification algorithms. In particular, it aims to provide non-ideal sensing conditions that are representative of practical urban sensing systems, with a large variability in viewpoint and occlusion. It consists of 588 objects across classes of vehicles, pedestrians, signs and trees, collected in the Central Business District of Sydney, Australia. These are shown in Table 4.5.

Objects are labelled with two levels of specification. (1) Using 8 classes (simple class), with each class distinct in shape. (2) Using 14 classes (detailed class), where similar categories such as 4WD's, vans and cars are separate. The 14 classes provides an additional level of difficulty, where subtle shape differences must be captured to distinguish some classes, such as traffic lights and traffic signs. Using two levels of specification can better identify misclassification. For example, misclassifying a 4WD as a car is not as bad as misclassifying a car as a building.

The experimental platform used to collect this data has several sensors, with the primary focus on this dataset being the Velodyne. Other sensors include a GPS-INS, visual cameras (e.g. omnidirectional) and 2D SICK LiDARs, which were also logged but not labelled; these could provide additional sources of information in future. The full sensory data was logged during a drive through the Sydney CBD (Figure 4.14a). This environment contained a large number of dynamic objects (vehicles, pedestrians), with lots of occlusion.



(a)

(b)

Figure 4.14 – Dataset location

Table 4.5 – Object classes

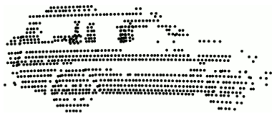
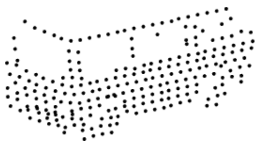
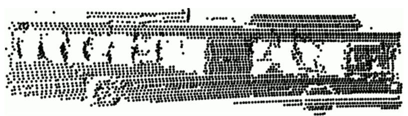
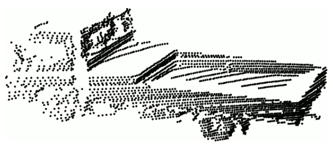
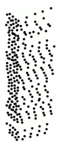
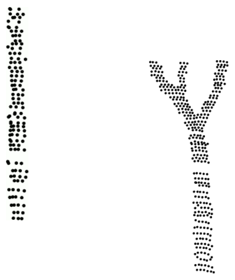
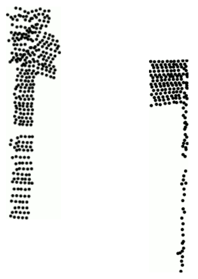
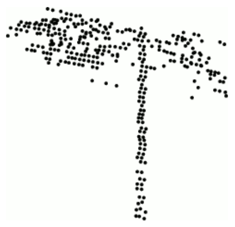
Simple class	Detailed class	No. instances	
Car	4WD (4 wheel drive)	21	
	Car	88	
	Ute (Tray utility vehicle)	16	
	Van	35	
Heavy Vehicle	Bus	16	
	Truck	12	
Pedestrian		152	
Building	Building	20	
Pillar	Pillar	20	

Table 4.5 – (continued)

Simple class	Detailed class	No. instances	
Pole	Pole	21	
	Trunk	55	
Sign	Traffic lights	47	
	Traffic sign	51	
Tree	Tree	34	

4.3.1 Labelling Methodology

From the full set of Velodyne scans, individual objects were manually selected and labelled. Labelling consisted of selecting each point in a given object, using a custom interactive visualisation application (Figure 4.15). Fast region selection routines from Section 3.1.3 allowed the use of a 3D spherical ‘paintbrush’ to select and label sets of points.

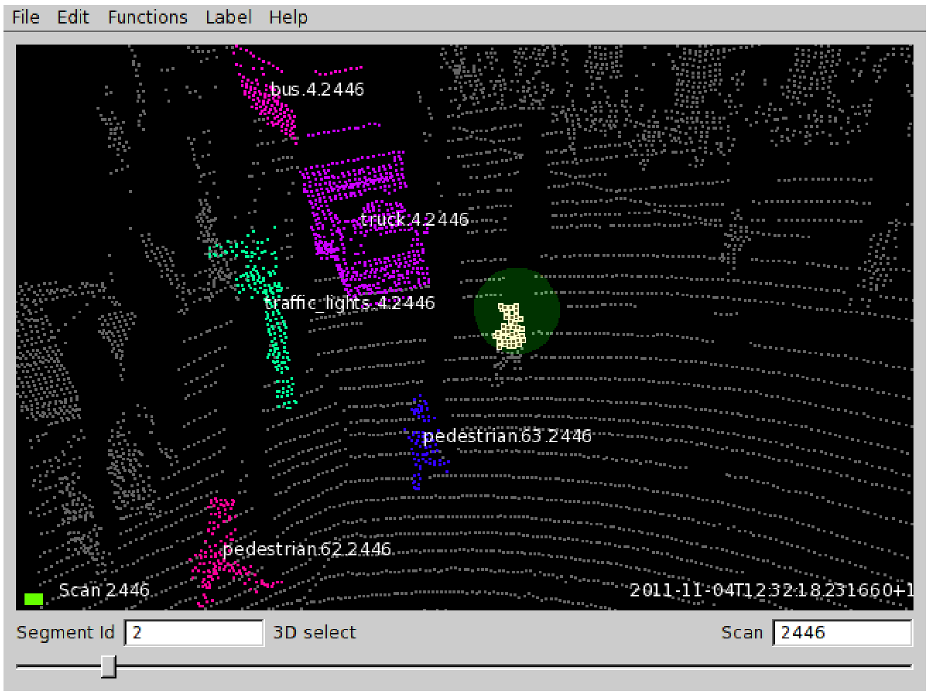


Figure 4.15 – Labelling application

Objects had to be recognised by a human in the point cloud, precluding objects that were too distant (low density) or occluded to identify. However, for some objects, it was difficult to distinguish between classes. For example, poles and some smooth, straight trunks appeared very similar. This motivated their grouping in the 8-class dataset. The limited sensor field of view in elevation also leads to ambiguities between poles and traffic signs, and so traffic signs were only labelled thus if the sign portion was visible. Similarly, trees had to have visible foliage, else they were trunks. Highly cluttered configurations of objects were avoided, such as pedestrians that were too close to others to separate, and bikes resting on poles. Segmentation algorithms are likely to group these, and so being able to classify them is important, but using them in evaluating classification performance is an additional difficulty, and so is avoided in this work.

Nevertheless, many recognisable but only partly visible objects were labelled, as this is predominant in the environment. Traffic regularly occludes other objects, and objects that are too close or tall are clipped by the limited sensor field of view in elevation. A combination of close range, highly visible objects and distant or occluded objects were chosen, with a variety of orientations. In order to get a reasonable number of object instances for each class, scans were chosen for the objects they contained, not at random. The dataset is not an unbiased sample of objects in an urban environment; rather it provides a variety of object instances, encompassing different class variations, poses and quality.

The resulting 588 objects were split into four groups, called *folds* for performing cross-validation in classification. To evaluate a supervised classification technique, cross-validation is a method of splitting a dataset into *training* and *testing* sets. For instance, in four-fold cross-validation, the classifier is first trained on the first fold, with the remaining three folds forming the test data to be classified. This is repeated with the second fold as training data, and so on. This allows all the data to be used in training and testing the classifier.

As some labelled scans were taken at relatively close positions along the driven route, care was taken to ensure that different folds did not share the same underlying, scanned object (else a classifier would be trained and tested on the same object). Using the logged GPS localisation, objects were transformed into the global reference frame, positioning them into the map shown in Figure 4.14a. Distinctly separate regions of objects were then assigned to one of the four folds. Additionally, for a given class, the number of objects in each fold was set to be approximately equal.

4.3.2 Variation in Range and Occlusion

To show the variation in occlusion and range in the dataset, this section estimates the degree of occlusion based on several assumptions. A measure of occlusion can help identify low quality objects in the dataset, and could be used to evaluate a classifier's robustness to occlusion.

First, consider the polar area of an object in a range image. This area varies as the range to the object changes, and occlusions occur. The relationship between object range R, width w, height h and polar area is shown in Equation (4.10).

$$\text{polar area} = \arctan\frac{w}{R}\arctan\frac{h}{R} \tag{4.10}$$

A simple measure of polar area is simply the number of points in an object, as a range image sensor samples points approximately evenly in polar space. This means that, for a given object at any range R, with no occlusion, c in the following equation is constant:

$$c = \frac{\arctan^2\frac{1}{R}}{\text{nPoints}} \tag{4.11}$$

Increasing levels of occlusion will increase c, and so this becomes a measure of occlusion.

In order to compare different objects in the same class, a simplifying assumption is made: the visible surface area of objects across a given class is constant. The polar area also varies with object rotation; consider for example a frontal view of a car vs a side view. While this effect can be significant for some objects, it can be considered as a type of self-occlusion. Without additional annotations to the dataset, it is difficult to account for, and so is ignored for simplicity. In classes where these two assumptions do not hold, c becomes less informative. For instance, buildings and trees will rarely have the same visible surface area, whereas cars and pedestrians are more homogeneous.

The variation in range, number of points, and the associated relative c values are shown in Figure 4.16. The dataset contains a wide variation of ranges (up to 40 m) and occlusion levels.

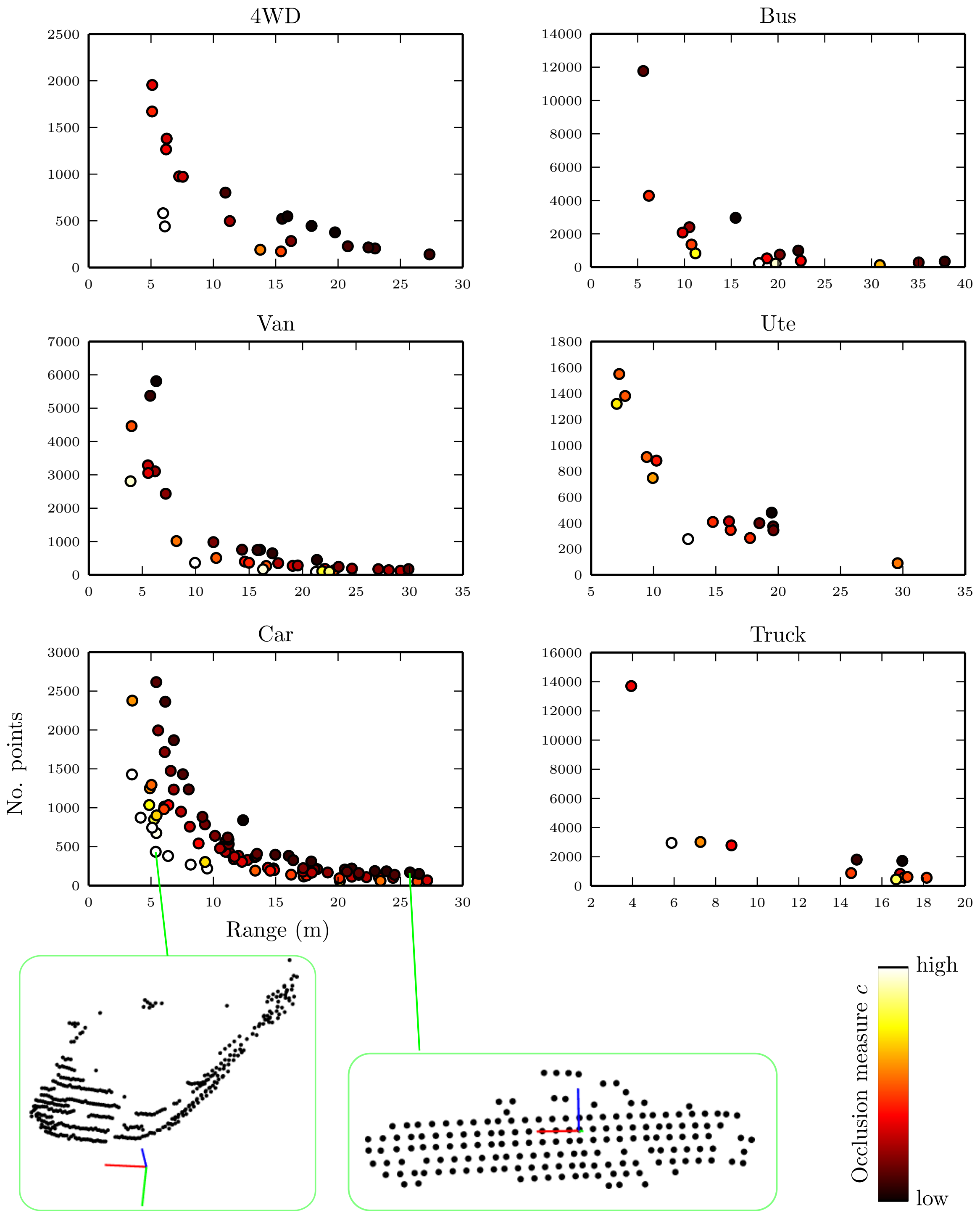
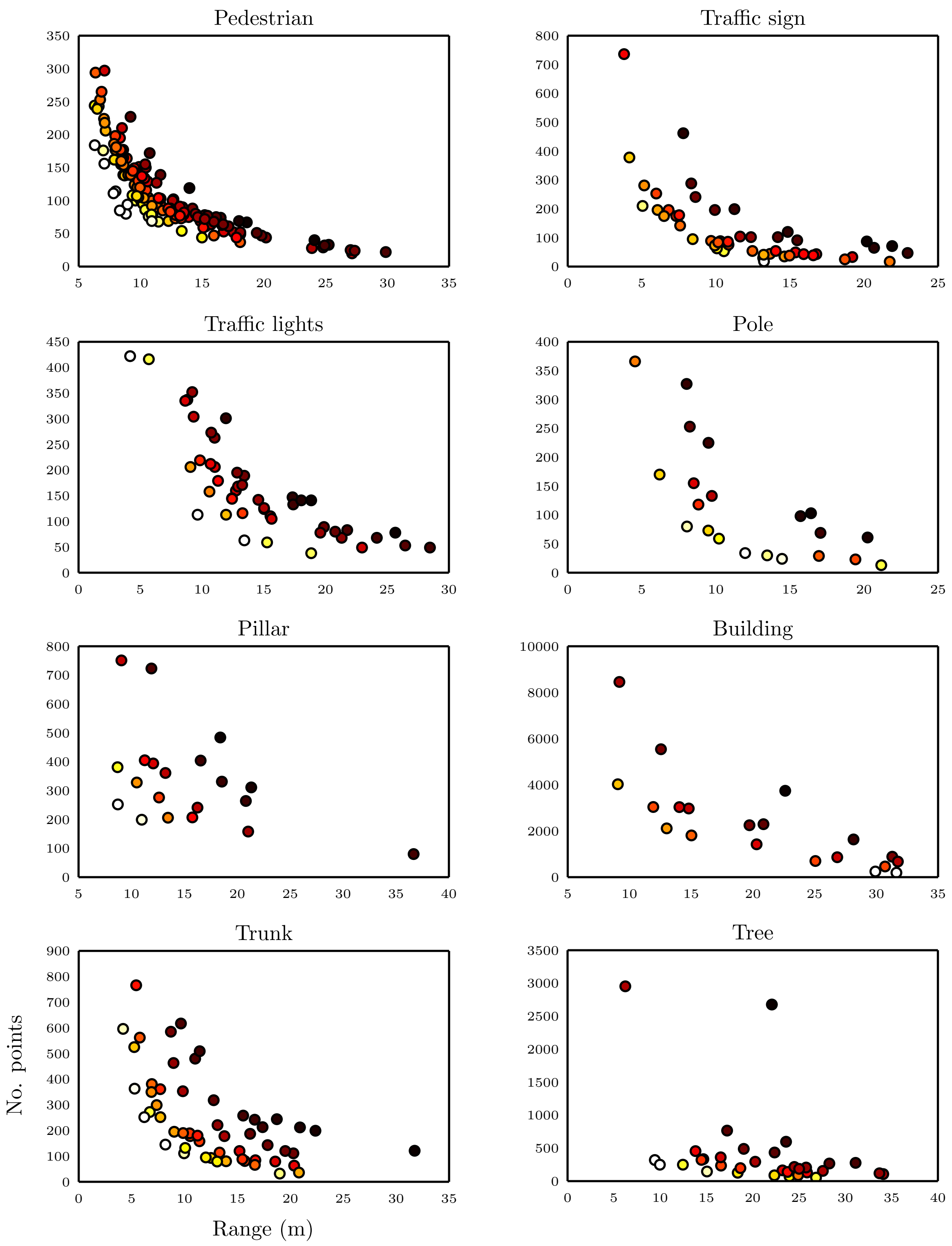


Figure 4.16 – The number of points vs range for each object. Higher levels of occlusion (lighter) occur towards the axes, where relatively fewer points are visible for a given range. A highly occluded, but nearby car (bottom right) has a high c value. A non-occluded, distant car (bottom middle) has a low c value.


Pedestrian
Traffic sign
Traffic lights
Pole
Pillar
Building
Trunk
Tree
No. points
Range (m)

4.4 Classification With *k*-NN

In order to compare the efficacy of the line image, it was applied to classifying objects in the above dataset, along with the three previously discussed local features: spin images [43], FPFH [71] and NARF [76]. A k-nearest neighbours classifier was used as a benchmark for comparing features. While there are many more advanced methods to classify objects from a set of local features, this was used as the simplest method for comparing different feature types. The next chapter focuses on classification using more complex methods.

This section will first describe the classification experiments performed, and the metrics used to evaluate them. Implementation details for each feature are discussed, followed by the experimental results and discussion.

4.4.1 Experimental Method

For each feature type, objects were classified using the same process. Keypoints were first extracted on each object, using the PCA keypoint formulation described in Section 3.3.2. Features were computed about each keypoint, resulting in many features for each object. As a separate experiment, NARF features were also computed at NARF keypoints, a much reduced set. The average number of keypoints for each class is shown in Table 4.6, with computation times in Table 4.7. An exception is FPFH, which requires computation at every point in its formulation.

All features were computed at the same scale, denoted here as the radius. For the spin image, the image length was considered the radius, as it corresponds to the radius of the cylindrical region it sweeps. Furthermore, to remove issues with clutter, only points on the segmented object were used for feature computation (although surrounding points were used for occupancy detection in the line image, with any surface intercepts outside of the target object counted as unknown space).

The objects were split into testing and training sets, with four-fold cross validation. Given the labelled training set of features, the k nearest neighbours for each test feature was found[7]. To determine a single object class, each of its features provided k votes, and the class with the most votes was used as the final result.

As each local feature is analysed independently, it must represent a shape which discriminates its class, necessitating analysis at different scales for each class. To investigate how features performed at different scales, each object was classified multiple times, each time using a different radius for the feature size.

Finally, recall that the Sydney dataset had two levels of specification, in that some objects were given an additional, more specific label. The '8 classes' set of labels considers relatively distinct categories, whereas the '14 classes' set of labels specifies vans, utes, trucks etc. Classification was done on the 8-class set and the 14-class set (i.e. the same data, but with different labels), with results in the following Section 4.4.4 for both.

Table 4.6 – Average number of keypoints (and subsequent feature vectors) for each class.

Class	Av. points per object	Av. PCA keypoints per object	Av. NARF keypoints per object
4WD	660	235	24
Building	2318	1257	62
Bus	1847	603	59
Car	582	183	21
Pedestrian	111	36	5
Pillar	338	181	14
Pole	116	25	8
Traffic lights	164	49	9
Traffic sign	127	28	7
Tree	400	129	21
Truck	2489	678	61
Trunk	243	66	12
Ute	637	261	21
Van	1152	322	33

[7] $k = 3$ was used, which gave the best performance in preliminary results

Table 4.7 – Total keypoint computation time across the whole dataset.

Keypoint algorithm	Total in dataset	Total computation time (s)
PCA	100 761	19.41
NARF	10 210	10.25
All points	288 536	

4.4.2 Classification Metrics

Several metrics for classification performance are considered. All objects begin with a *ground truth* class label, and after classification, receive an *inferred* class label. Compiling these two sets of labels into a 2D histogram forms the *confusion matrix*. Each row considers all the instances of a given, ground truth class. Each subsequent column then indicates how many objects in this class were classified as each inferred class. For example, see Table 4.10. The confusion matrix shows which classes were often confused as others.

In order to summarise the results into a single number for easy comparison, the F_1 [65] and normalised mutual information (NMI) [77] measures are considered.

4.4.2.1 F_1 Measure

To compute the F_1 measure, first consider a single class. The multi-class confusion matrix can be reduced into a binary confusion matrix, summarised in Table 4.8.

Table 4.8 – Binary confusion matrix, for a given class of objects.

	Classified as this class	Classified as not this class
Is actually in this class	True positives (tp)	False negatives (fn)
Not actually in this class	False positives (fp)	True negatives (tn)

The elements of this matrix are the counts of objects that were actually in this class or not, and were classified correctly or not. These counts define precision and recall:

$$\text{precision} = \frac{tp}{tp + fp} \tag{4.12}$$

$$\text{recall} = \frac{tp}{tp + fn} \tag{4.13}$$

Precision measures how often the predicted class labels were actually correct. Recall measures how many of the actual objects in that class were identified. Thus, a classifier can be highly precise for a given class (e.g. when it does identify a car, it is often correct), but with low recall (it fails to identify many cars).

The F_1 measure combines precision and recall, and varies from 0 (worst) to 1.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.14}$$

It is computed on each class separately. To summarise the results across all classes, these are averaged together.

4.4.2.2 Normalised Mutual Information

The NMI also ranges from 0 (worst) to 1, and is a measure from information theory that captures the agreement between ground truth category X and *any discrete assignment* Y.

It is defined as follows, with I as the mutual information, and H as the entropy.

$$\text{NMI}(X, Y) = \frac{I(X;Y)}{\sqrt{H(X)H(Y)}} \tag{4.15}$$

It is computed for each class separately, using the binary classification matrix as above. It is also computed across the entire confusion matrix for a single, multi-class measure.

This measure is used in evaluating unsupervised classification and clustering, which is relevant to techniques discussed later in Section 5.2. In this chapter, supervised classification is performed, and so X and Y both refer to the same class labels.

However, if for instance, all utes were classified as 4WDs, and vice versa, the NMI would remain high. Permuting the columns of the confusion matrix does not affect the NMI.

Just as the F_1 measure incorporates precision and recall, the NMI incorporates the *homogeneity* and *completeness* of the assignments, which are two desirable properties defined in [66] (the NMI is the harmonic mean of the homogeneity and completeness, and is equivalent to the v-measure [66], shown in [3]). To describe these, imagine the act of classification as placing coloured balls into buckets. The colour represents the ground truth X, and the bucket, the inferred class label Y. While in this chapter, there are the same number of buckets as colours, this need not be the case. After classification, if each bucket Y only contained balls of a single colour, the results satisfy homogeneity (however, two buckets may have the same colour). Inversely, if all balls of the same colour ended up sharing the same bucket, the results satisfy completeness (however, that bucket may contain other colours).

In supervised classification, the NMI provides a different measure of performance to the F_1 measure. For example, if all vans were consistently classified as cars, the multi-class NMI would not penalise this as much as the average F_1 measure. This is because only the completeness will be reduced, whereas the precision and recall would be zero for that class. This is relevant for the Sydney dataset (Section 4.3), where similar, related classes in the 14-class set (e.g. trunks, poles, signs; and cars, vans and utes) may end up being inferred as a single class. Alternately, if all objects were classified arbitrarily as buildings, F_1 would be non-zero for that class (as recall would be high, and precision low but non-zero), whereas the NMI would be zero.

4.4.3 Implementation Details

Line Image

For the line image, this experiment used the distance measure m with the assymetric v_a visibility measure (Equations (4.8) and (4.9)). In addition, a match threshold of 0.75 on v_a was used to remove matches which require a high degree of occlusion, which was found to improve performance.

Spin Image

The implementation of spin images followed the method in [43], which involved binning the scanned points and using bilinear interpolation. The spin image was found to perform significantly worse using the occlusion-handling similarity metric (Equation (4.2)), and so the results shown here use the original correlation coefficient (Equation (4.1)). As for resolution, a 16×16 spin image was found to perform better than a lower resolution 8×8 image.

FPFH

The FPFH implementation from PCL v1.6 [68] was used. FPFH requires surface normals at all points, and so the bearing graph and cross product method was chosen (Section 3.2.1.1) to produce as many surface normals as possible. FPFH requires that the PFH be computed first at all points[8] on an object, precluding it from being computed only at keypoints. Forming a histogram over a dense set of surface normals also somewhat negates the need for sparser, stable orientations. The number of data points could be reduced based on a measure of local uniqueness as in [71], but was left as the full set in this experiment.

NARF

PCL was also used here, however an issue with applying Velodyne data to this implementation is the uneven spacing between scan lines in elevation. The image resolution was set to $0.5°$, but at least one row of pixels had no data, as a result of the slightly larger gap between the top and bottom sets of 32 lasers. The internal interpolation resulted in a slight bump along this gap, which in turn created a line of NARF keypoints near this artifact. Similarly, the image required square pixels, necessitating a loss in many points along the more densely sampled azimuth. The keypoint parameters were modified as to attain as many points as reasonably possible along object edges and corners, as this was found to improve results.

[8]with valid surface normals

Orientation Constraints

Most large urban objects have a vertical orientation, so keypoints were additionally constrained to only match to other keypoints with a similar sized z component in orientation. This permits objects to change in position and yaw, but not significantly in roll and pitch, as discussed earlier in Section 4.2.4.3. This improved results across all features.

Computation Times

Computation times for the whole dataset at a radius of 2 m are shown in Table 4.9. Note that FPFH was computed at all points (2.8 times as many features), and so took longer. k-NN scales by $O(n^2)$, and so the resulting time spent computing distances far outweighs the feature generation times. This motivates more efficient classification methods. The line image distance metric takes about twice as long as the spin image, which in turn is twice as long as NARF (and FPFH, if adjusted for the increased points).

Table 4.9 – Total computation time on full dataset, at 2 m radius, across all folds.

Feature	Generation (s)	k-NN (s)
Line Image	150.4	2009
Spin Image	9.7	817
FPFH	99.6	2770
NARF, PCA keypoints	17.9	466
NARF, NARF keypoints	2.1	17

4.4.4 Results

Beginning with the 8-class set, classification metrics for each class, at each radius, are shown in Figures 4.16 and 4.17. A summary of the best values across all classes is shown in Table 4.11. The confusion matrix for line images at their best is shown in Table 4.10. These are repeated for the more specialised 14-class dataset. See Appendix C for additional confusion matrices and metrics for each feature.

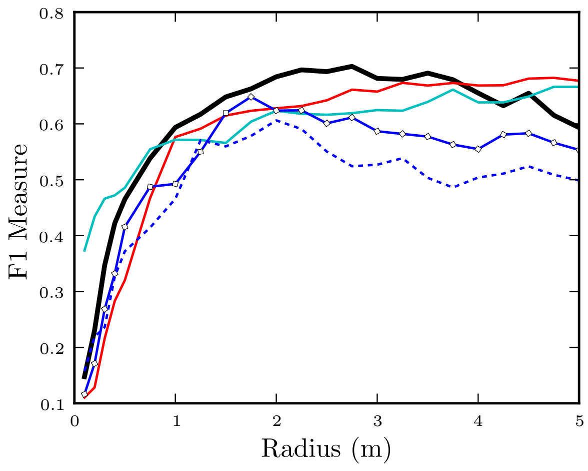
4.4.4.1 8 Classes

Table 4.10 – The confusion matrix for the line image at 2.75 m radius, showing how many instances from each true class (rows) were classified as each inferred class (columns).

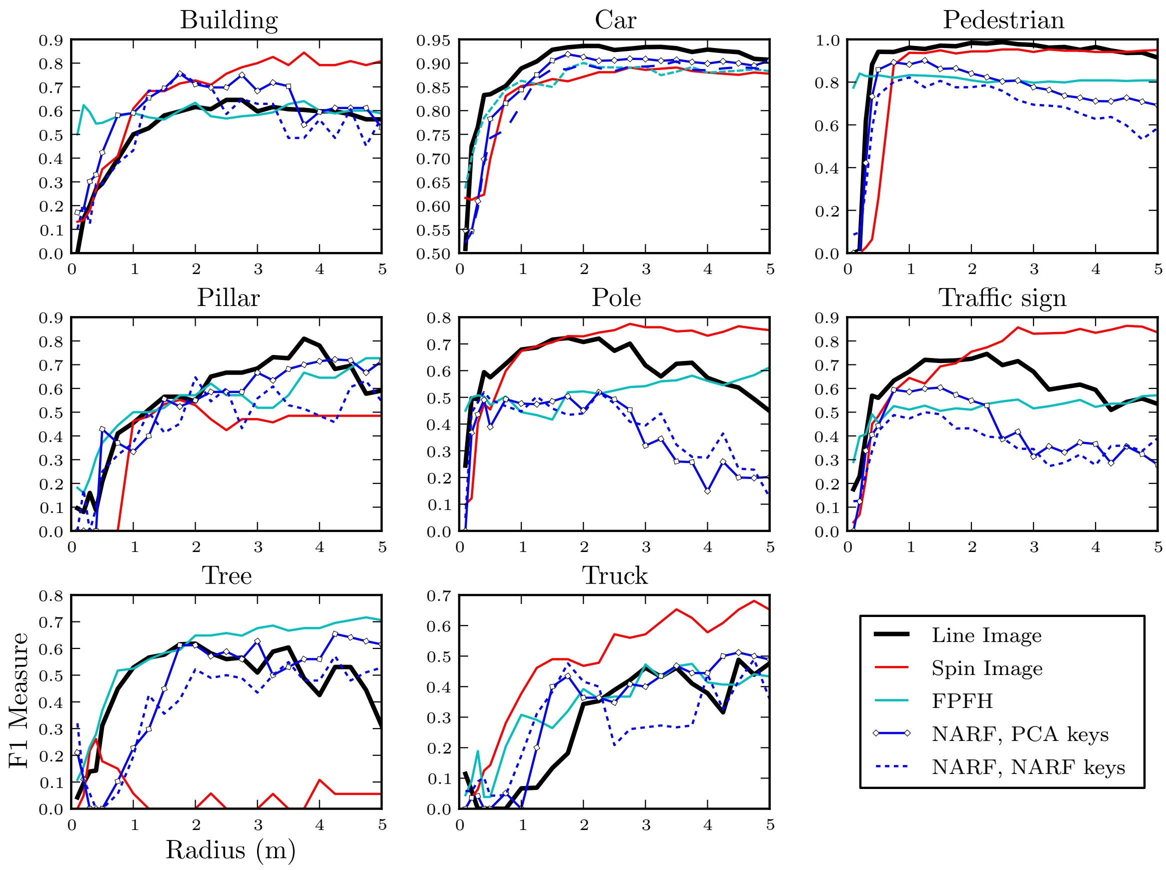
Truth \ Inferred	Building	Car	Pedestrian	Pillar	Pole	Traffic sign	Tree	Truck
Building	20	0	0	0	0	0	0	0
Car	5	154	0	0	0	0	0	1
Pedestrian	0	0	151	0	1	0	0	0
Pillar	2	0	0	14	4	0	0	0
Pole	1	0	2	5	61	5	2	0
Traffic sign	0	5	3	3	25	59	2	1
Tree	2	5	1	0	7	3	15	1
Truck	12	7	0	0	0	0	0	8

Table 4.11 – 8 Classes: Maximum metrics for each feature.

Feature	Max F_1	Radius	Max NMI	Radius	Max Accuracy	Radius
Line Image	0.7	2.75	0.71	2.0	0.72	2.75
Spin Image	0.68	4.75	0.7	3.25	0.69	4.75
FPFH	0.67	4.75	0.53	4.75	0.67	5.0
NARF, PCA keys	0.65	1.75	0.62	1.75	0.64	1.75
NARF, NARF keys	0.61	2.0	0.55	2.25	0.58	2.0

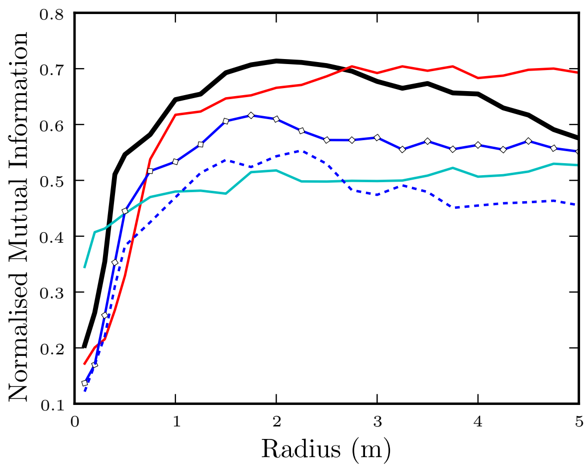


(a) F$_1$ measure, class average

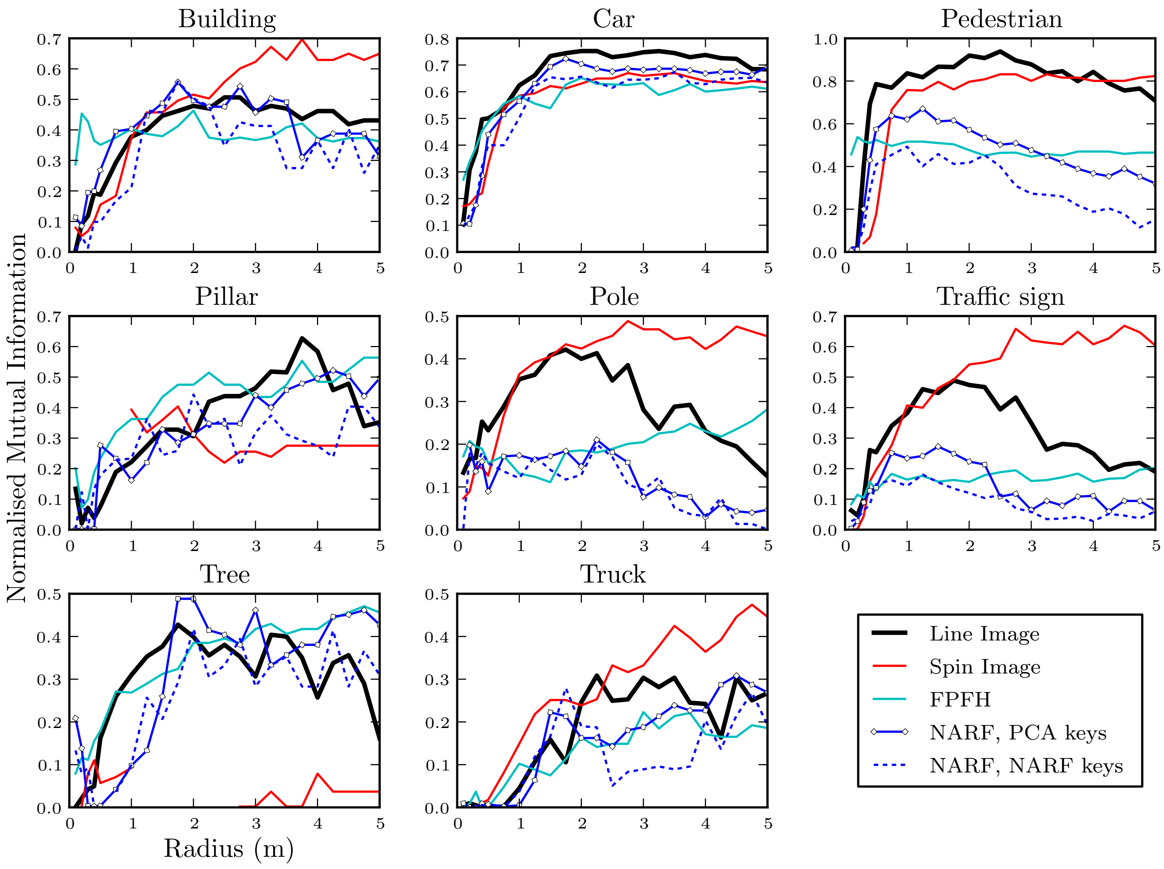


(b) F$_1$ measure per class

Figure 4.16 – F$_1$ measure, across different support radii for each feature.



(a) NMI on confusion matrix



(b) NMI per class

Figure 4.17 – Normalised Mutual Information (NMI), across different support radii for each feature.

4.4.4.2 Specialised 14 Classes

Table 4.12 – The confusion matrix for the expanded 14-class dataset, using the line image at 1.75 m radius. Truth = row, inferred = column.

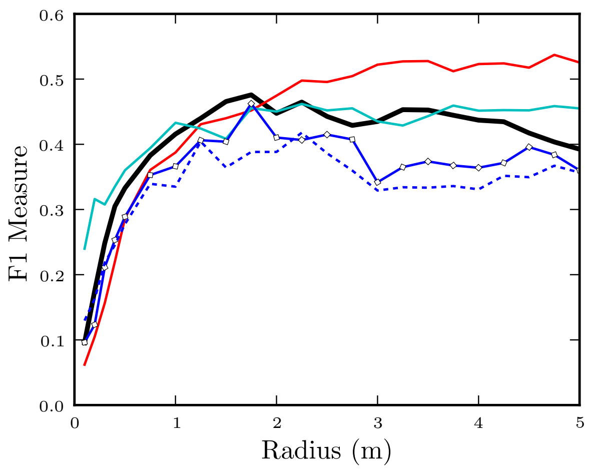
Truth \ Inferred	4wd	Building	Bus	Car	Pedestrian	Pillar	Pole	Traffic lights	Traffic sign	Tree	Truck	Trunk	Ute	Van
4wd	0	0	0	21	0	0	0	0	0	0	0	0	0	0
Building	0	20	0	0	0	0	0	0	0	0	0	0	0	0
Bus	0	10	2	0	1	0	0	0	0	0	1	0	0	2
Car	2	1	0	85	0	0	0	0	0	0	0	0	0	0
Pedestrian	0	0	0	0	151	1	0	0	0	0	0	0	0	0
Pillar	0	4	0	0	0	13	0	0	0	0	0	3	0	0
Pole	0	0	0	0	0	0	7	1	0	0	0	13	0	0
Traffic lights	0	0	0	1	2	3	1	28	1	0	0	11	0	0
Traffic sign	0	0	0	1	8	3	2	3	25	1	0	8	0	0
Tree	0	3	0	0	2	1	0	1	0	20	0	7	0	0
Truck	0	8	0	2	0	0	0	0	0	0	1	0	0	1
Trunk	0	1	0	0	0	7	5	1	0	2	0	39	0	0
Ute	0	3	0	11	0	0	0	0	0	0	0	0	0	2
Van	0	9	0	12	0	0	0	0	0	0	0	0	0	14

Table 4.13 – Per-class metrics, 14 classes, line image at 1.75 m.

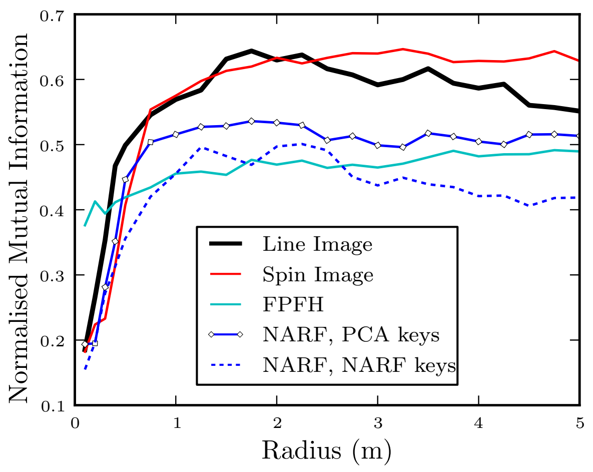
Class	Precision	Recall	F_1	Often confused as
4wd	0.0	0.0	0.0	Car (100%)
Building	0.34	1.0	0.51	
Bus	1.0	0.13	0.22	Building (63%)
Car	0.64	0.97	0.77	
Pedestrian	0.92	0.99	0.96	
Pillar	0.46	0.65	0.54	Building (20%)
Pole	0.47	0.33	0.39	Trunk (62%)
Traffic lights	0.82	0.6	0.69	Trunk (23%)
Traffic sign	0.96	0.49	0.65	Trunk (16%)
Tree	0.87	0.59	0.7	Trunk (21%)
Truck	0.5	0.08	0.14	Building (67%)
Trunk	0.48	0.71	0.57	Pillar (13%)
Ute	0.0	0.0	0.0	Car (69%)
Van	0.74	0.4	0.52	Car (34%)

Table 4.14 – Specialised 14 Classes: Maximum metrics for each feature.

Feature	Max F_1	Radius	Max NMI	Radius	Max Accuracy	Radius
Line Image	0.48	1.75	0.64	1.75	0.5	1.75
Spin Image	0.54	4.75	0.65	3.25	0.53	4.75
FPFH	0.46	2.25	0.49	4.75	0.47	2.25
NARF, PCA keys	0.46	1.75	0.54	1.75	0.47	1.75
NARF, NARF keys	0.42	2.25	0.5	2.25	0.42	2.25

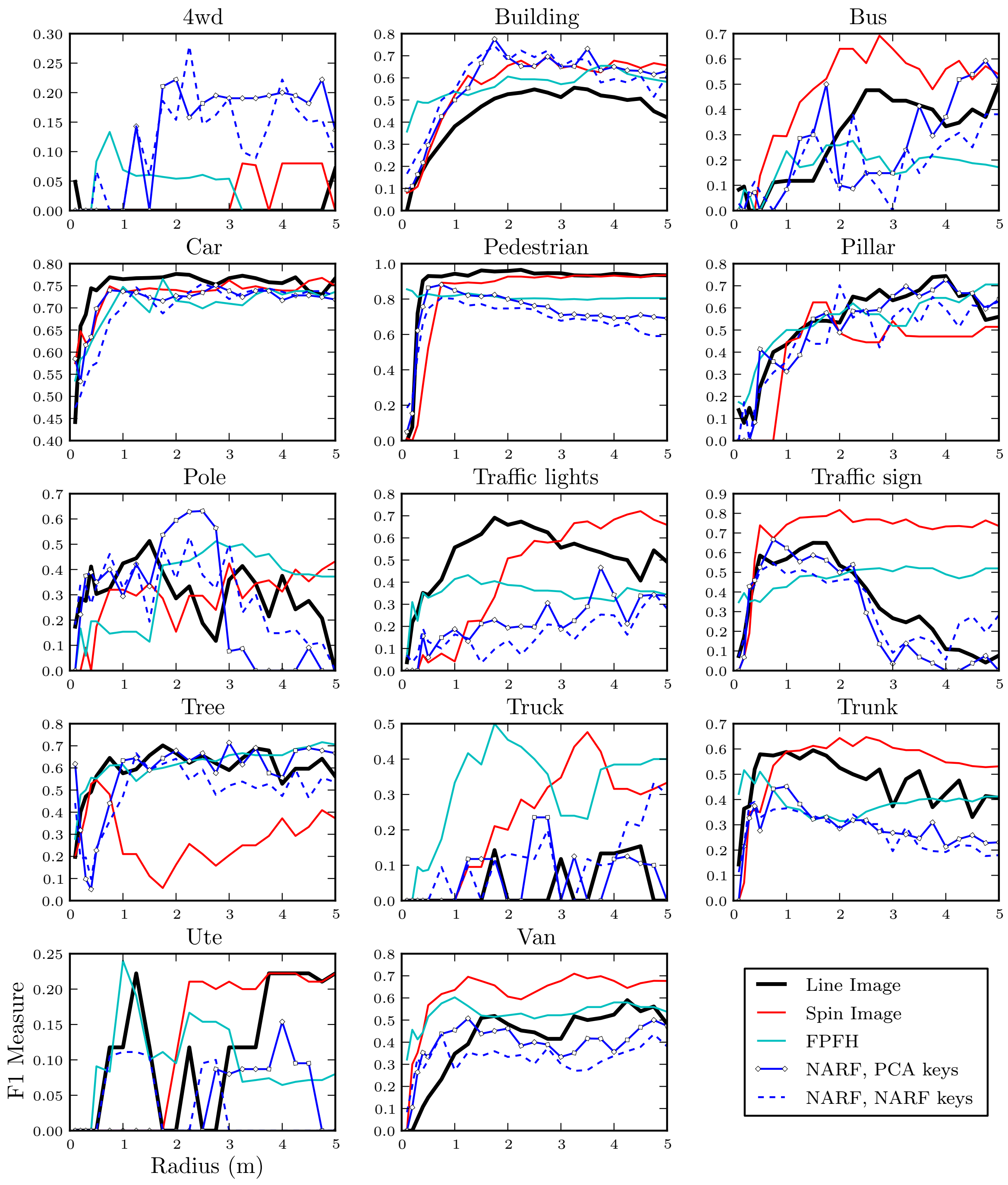


(a) F_1 measure, class average



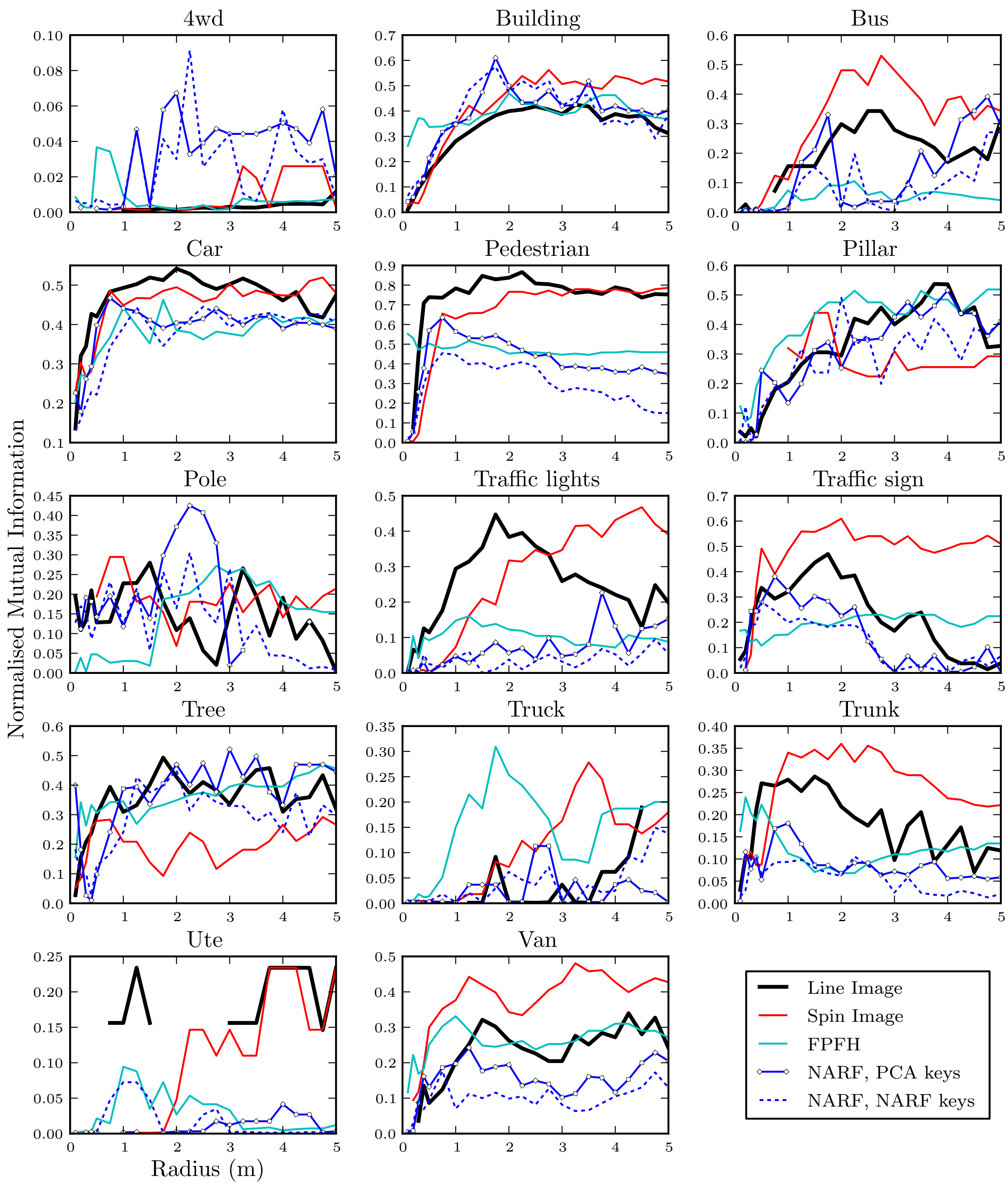
(b) NMI on confusion matrix

Figure 4.18 – Class average F_1 measure, and NMI on confusion matrix (14 Classes).



(a) F_1 measure per class

Figure 4.19 – F_1 measure, across different support radii for each feature (14 Classes).



(a) NMI per class

Figure 4.20 – Normalised Mutual Information (NMI), across different support radii for each feature (14 Classes).

4.4.5 Discussion

The variation in performance across each class and scale indicates the strengths and weaknesses of each feature. Each object class contains a different set of distinguishing shapes, best captured by certain features at certain scales.

Scale

The line image's best performance is approximately the same as the spin image's best, but at a smaller radius of 2 m, compared with the spin image's 3.25 m. This indicates that the line image can distinguish different classes based on smaller, constituent parts, rather than larger regions encompassing most of the object. At these smaller scales, the surface interpolation and detection of distinguishing empty space enables more of the geometry to be captured. For the spin image, without interpolation, the density of some regions may be below the size of the bins at small scales, resulting in poorer performance. Alternately, at larger scales, the line image degrades in performance, where the spin image peaks. This could be due to fewer lines in the line image intersecting the target object, as well as the distance metric enforcing those lines to be rigidly matched. In contrast, large bin sizes in spin images may allow a greater variation of shape, as points can move more freely within each bin.

Note that at large scales (spherical regions of 4 m radius), the region encompassed by the feature is, in many cases, the entire object, negating the concept of classification by parts and the potential benefits this brings (see Section 2.3.3). Nevertheless, such 'global' scales perform best for some features. A general issue with the simple k-NN method of classification is that small, independent parts have a limited class specificity on their own, even if as much geometric information is captured as possible. This becomes more significant at the smallest scales (less than 0.5 m), where most feature metrics drop towards zero. Large or global object representations are highly specific, but lack the ability to generalise beyond limited training data. In this experiment, the former may have been more important than the latter. This motivates other approaches that combine object parts to produce higher specificity, detailed in the next chapter.

Line Image

The line image performed better than others on cars, pedestrians and pillars, and relatively well on trees, trunks and traffic lights. These objects have distinguishing silhouettes, and so the corresponding empty space surrounding them was captured and used in classification.

However, line images confused most buses and trucks with buildings, whereas spin images were more successful here. These objects are best distinguished at large scales, with right-angle edges and corners. At smaller scales, the component parts of these classes are very similar. Again, k-NN considers these parts independently, and so it is plausible to construct a building out of pieces of a bus's geometry.

Line images may not be as successful as spin images in capturing the details of these types of shapes. Consider two surfaces joined in a right angle edge, such as on the back of a bus (Figure 4.21). A line image captures a 2½-D view of the region, aligned to a

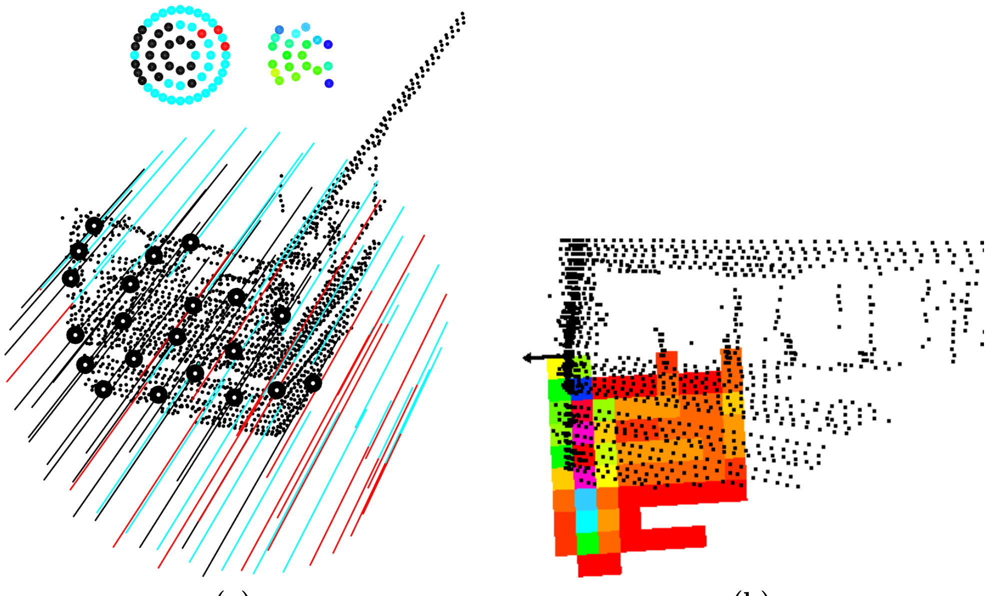


Figure 4.21 – (a) A line image on the rear surface of a bus. The right side surface is not intercepted, with only some empty space captured in the non-intercepting lines. (b) A spin image on the same location, with the bus facing side-on. The image 'spins' about the black arrow, spinning out of the page. The rear of the bus is collated into a line of bins, with the side face captured along a square section of bins. Spin images may capture more shape information in this case.

keypoint on one of the flat surfaces. Half the lines will only intercept the perpendicular surface, with the other surface facing parallel to the remaining lines. These lines only capture some of the empty space in front of the parallel surface. Contrast this with a spin image, which bins all points in the region, not limited to a 2½-D view (albeit collapsing information along bins). The spin image may be a stronger representation for this type of geometry.

The expanded vehicle classes (ute, 4WD, van) were all confused with cars, and vans with buildings. Utes and 4WDs were not very distinguishable with any feature. These classes may be too similar to distinguish with k-NN, requiring a stronger classifier to pick out the few shapes which differ. Likewise, the similarly shaped classes of poles, traffic signs, trees and trunks had some confusion across all features.

Spin Image

Spin images did well on buildings, traffic signs and other thin structures. Due to the alignment of PCA keypoints on thin regions, the spin image 'spins about' the long axis. In the case of poles, this results in a distinguishing line of bins being occupied. A wider traffic sign on top falls into bins away from this line. As such, these structures fall into bins in relatively informative ways.

Spin images did not perform well on trees, potentially due to the scattered, unstructured internal foliage. Lines in line images often do not reach a tree's internals, instead intercepting the outer foliage as a surface, or else becoming occluded.

FPFH

FPFH peaked at larger scales, where a sufficiently distinguishing variation in surface normals is captured. Trees performed very well; the highly variable overall shape of trees may be best captured by FPFH's reduction into a non-spatial histogram. Trucks and pillars also performed well, having distinct angular structures which translate well into a histogram of angles. However, it had lower performance on linear structures such as traffic signs/lights, and pedestrians. Such structures are not well-described by their surface normals. However, if linear orientations were incorporated in a similar histogram, these might be better distinguished.

NARF

NARF, similar to line images, peaked at a small, 1.75 m radius. NARF performed well on trees and buildings, but not on trucks, traffic signs and pedestrians.

Using NARF keypoints resulted in a drop in performance, although given that there were ten times fewer keypoints, the boost in computation speed may make this worthwhile. Pedestrians received only 5 NARF keypoints on average, resulting in a relatively large drop of 0.2 in NMI. Given that larger objects such as buildings and trees were reduced to fewer keypoints without significant performance loss, NARF keypoints may be best applied selectively based on object size.

Specific object failures

Some distant objects failed at the PCA keypoint level, where as few as 5 keypoints were extracted. The low density of points meant that the fixed radius PCA computation did not have enough points to proceed. This could be fixed with a more adaptive selection of scale to compute PCA.

Occlusion of key identifying structures caused some failures. On one large, box-like set of traffic lights, the associated pole was occluded, and so was classified as a car. The same occurred for a car-sized tree canopy with the trunk occluded. Incorporating additional simple object properties such as distance from the ground could alleviate this.

Summary

Each feature captured different aspects of geometry, with different features being well suited to certain classes containing favourable distinguishing structures. Line images were very successful at smaller scales, with only large buildings and trucks being less accurately classified. Spin images were equally good, but required larger scales encompassing the whole object. Many sets of similar classes such as vehicles, as well as poles, signs and trunks, were consistently confused. With the highest 8-class NMI at 0.71, the dataset proved very challenging, leaving much room for improvement with stronger classifiers or still better features.

4.4.6 Conclusion

This chapter presented the line image feature as a way of representing and matching regions in sparse 3D range image data under realistic conditions of variable density and occlusion. Occupancy detection within the feature allowed occluded, or unknown, spaces to be matched with anything, while empty space and surface intercepts provided a strong description of shape.

The line image was compared with existing features on the task of object classification, using a collected dataset of urban objects. With a simple k-NN approach, objects were classified using each feature type, with different scales of support radii compared. Overall, line images performed equally well to spin images, but did so at smaller scales.

The good performance of line images at smaller scales is encouraging, as small regions of an object can be matched rather than matching the whole object at once. This is advantageous for identifying partially observed objects. This also allows classes with a large variation in global shape to be accurately identified by smaller structures. Ultimately, the k-NN approach is limited in that each part must be relatively class-specific, and so object parts are disadvantaged compared to using the whole object. There is also no clear way to combine different scales of analysis. This motivates other approaches to combining object parts together for classification. The next chapter explores techniques to achieve this, utilising the line image's ability to capture local shape to learn recurring object parts, followed by more detailed inference on these parts.

Chapter 5

Classification from Parts

This chapter focuses on applying the line image feature to more advanced classification methods, involving clustering and topic modelling. As outlined in Section 2.4, classification from parts has two aspects to consider: (1) performing inference on the parts (each represented by a feature vector), followed by (2) classifying the object from these parts. In the previous chapter, a k-nearest neighbours classifier was used on each feature, with the votes from all feature vectors in an object pooled to classify the object. However this approach is limited due to the following issues:

- There is no clear way to combine multiple scales of analysis.
- The relative importance of each part is not considered.
- The structure requires parts to be reasonably class-specific.
- Parts are classified independent of other parts in the object.
- The method scales poorly at $O(n^2)$.

An alternative approach is through clustering, which discretises the feature space into specific, repeatable parts. This allows the distribution of features to be modelled, enabling more advanced object classification methods. In particular, topic modelling is a recent approach that shows success in classifying visual images from component features, and so is examined here.

This chapter makes the following contributions:

(1) The use of Affinity Propagation (AP) clustering on line images to generate exemplars for each class.
(2) The application of supervised Latent Dirichlet Allocation (sLDA) to classify objects from multiple scales of line images.

It will be shown that this approach results in significant improvements over k-NN classification, with an F_1 score of 0.82, compared with 0.7.

5.1 Clustering

This section outlines an approach to clustering line images, presenting results on the Sydney dataset in Section 4.3. These are then used in classification in the next Section 5.2.

So far in the processing pipeline, a feature vector has been extracted from each region in the point cloud, allowing shapes to be compared with a measure of distance. However, classification techniques often go beyond the pairwise matching of k-NN. Instead, feature vectors are considered as points in a high dimensional feature space, and the distribution of these points is modelled. For example, modelling the distribution of points over each class enables Naive Bayes classification, where new data points are assigned a probability distribution over classes. Note that, in this section, *points* will refer to feature vectors in feature space.

One approach to define a distribution in the feature space is through quantization, or clustering. This process discretises the feature space, with each region defined by a cluster. A new feature is then assigned to the closest cluster[1], allowing statistics to be collected. The clusters provide a space upon which to do inference. In essence, an object is reduced into a set of previously observed shape instances.

However, the line image is not a simple vector in euclidean space, and so this requires some consideration. As a data point, it consists of a standard continuous vector d,

[1]Multiple closest matches can be used to more accurately localise within feature space in a method called sparse coding.

and an additional 'mask' s. The measure of distance described in Section 4.2.4 does not fulfil the triangle inequality, which is a requirement for these features to be in a metric space. Many algorithms presume a Euclidean space, or at least a metric space. Without a metric space, comparing more than two data points becomes non-intuitive.

To demonstrate the lack of a triangle inequality, consider two different line images A and B representing differently-shaped regions, that are completely visible with no unknown regions present. Now consider a third line image C, with unknown regions organised in such a way that the RMSE to both fully-visible line images is zero. That is to say, the partially-visible region in C could in fact be either A or B. In this situation, the triangle inequality does not hold for the RMSE measure r (Equation (4.6)), as $r(A, B) > r(A, C) + r(B, C)$. Penalising for unknown regions in the distance measure m (Equation (4.9)) will increase the right hand side, but does not guarantee the triangle inequality. In the context of clustering, if point a is close to cluster c, and point b is close to c, in a non-metric space, it does not follow that a is near b. Also note that, if the number of unknown lines in c were subsequently reduced, the similarity of a and b could be increasingly inferred.

5.1.1 Clustering techniques

With this is mind, two general methods to cluster the line image are explored.

(1) The (non-metric) distance measure m, from Equation (4.9), is used to define a similarity matrix, upon which exemplar-based clustering algorithms are performed, such as affinity propagation.

(2) The line image is interpreted as an instance of incomplete data in a metric space, where the task of clustering is to infer the missing portions from support data.

The first approach is then implemented in the next section.

5.1.1.1 Exemplars from the Similarity Matrix

Affinity Propagation [31] (AP) is a relatively recent exemplar-based clustering method, and is discussed in this section. To motivate this approach, consider the simplest clustering algorithm of k-means. An immediate issue is that there is no clear way to compute the mean of a set of line images without ignoring the empty and unknown portions. Instead, exemplar-based approaches are considered, where an existing data point is chosen to represent each cluster. Exemplar clustering, and affinity propagation in particular, has benefits over inferred models (such as a Gaussian mixture model), in that detail is not lost to an oversimplified model, and the difficulty of inferring more complex models is avoided [32]. Additionally, a non-metric similarity measure can be used [25]. Affinity propagation has benefits over other exemplar clustering techniques such as k-medoids, as it considers all data points simultaneously without an initial starting set.

It requires a similarity matrix $s(i,k)$, which indicates how suitable point k is as an exemplar for point i. The entries can be any similarity measure, or the negative of a distance measure. The matrix can also be sparse to save on computation time and memory. The diagonal entries $s(k,k)$ indicate how suitable point k is as an exemplar prior to clustering; these are called the preference values.

The algorithm functions by passing messages between data points i and k. The two messages are the responsibility $r(i,k)$, and the availability $a(i,k)$. The responsibility considers the exemplar assignment problem from the point of view of the data point i. When considering the pair $\{i,k\}$, the update rule determines how suitable k is to be an exemplar for i, given all other possible exemplars k'. Initialising $a(i,k)=0$, it is computed by:

$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k} \{a(i,k') + s(i,k')\} \tag{5.1}$$

The availability then considers the problem from the point of view of the exemplar k, pooling the responsibility values that each data point i has assigned to k. It is computed by:

$$a(i,k) \leftarrow \min \left\{ 0, \quad r(k,k) + \sum_{i' \notin \{i,k\}} \max\{0, r(i',k)\} \right\} \tag{5.2}$$

The self-availability $a(k,k)$ is separately computed:

$$a(k,k) \leftarrow \sum_{i' \neq k} \max\{0, r(i',k)\} \tag{5.3}$$

After computing r and a for all pairs, the assigned exemplar for point i is:

$$\text{assignment}(i) = \arg\max_k \{a(i,k) + r(i,k)\} \tag{5.4}$$

Affinity propagation is run by iteratively computing the responsibility and availability messages. Each update is damped by a constant λ, with the message x set to $\lambda x_t + (1-\lambda)x_{t-1}$. It is run until a convergence criteria is met, which can be when the output exemplar points remain constant for several iterations.

The number of exemplars is controlled by varying the preference values, with smaller numbers resulting in fewer exemplars. Preferences can be individually set, or all set to the same value, such as the median value of the similarity matrix. Several additions to this algorithm exist, such as adaptive AP [89] for scanning preference and damping values, and hierarchical AP [35].

Applying affinity propagation to missing data presents the following problem: a data point with lots of unknown space can match well to many other data points. This could result in points with the most unknown space being chosen as exemplars, simply because the most data points match to them. As discussed earlier, having a large amount of unknown space in an exemplar can group two very different points in the same cluster. Missing data can be penalised by the mutual visibility v, however, this prevents partially visible points from supporting matching, but more-visible instances. Thankfully, affinity propagation can use asymmetric similarity measures, and so the asymmetric visibility v_a can be used (Equation (4.9)). Combined into the measure m,

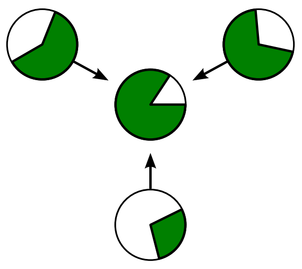


Figure 5.1 – Ideally, data points with less visibility (more unknown space) would support matching instances that are more visible (shaded) as exemplars.

this allows a largely missing data point to match strongly to a more complete version (Figure 5.1), but not the other way around.

Additionally, the self-similarity or preference value of each data point could be scaled by the degree of unknown space in a line image. However, it is often the case that the geometry is coupled to the degree of visibility as measured. For instance, flat regions on walls are often fully visible, whereas line images on pedestrians invariably have unknown space. Such complex shapes, with an aligning surface normal not facing the sensor, tend to self-occlude. As a result, the preferences were set to a constant value, purely to control the final number of clusters. Experiments and results from affinity propagation are further discussed in Section 5.1.2.

5.1.1.2 Modelling Incomplete Data

An alternative approach to clustering is to consider the line image as a data point in a metric space, but with some dimensions partly missing. This is known as missing or incomplete data, commonly encountered in surveys [73] and bioinformatics [90]. For example, a line image would be a single vector in a metric space with values $(0.3, 0.4, ?, 0.1, ?, 0)$, where question marks indicate missing dimensions. This could be made completely Euclidean if empty space was assigned, for instance, a maximum negative value. Missing data is typically imputed by a model, and so has relevance in clustering which seeks to model the different regions in the feature space.

Clustering is done in an iterative fashion, alternately fitting the model and updating the clusters. This approach can allow k-means clustering when coupled with a model such as PCA, as in [41], or fuzzy c-means clustering as in [93]. Alternately, instead of clustering data points and inferring the missing dimensions, they can be marginalised away during logistic regression classification [91].

Fitting a model while inferring missing dimensions fits into the framework of the Expectation Maximisation (EM) algorithm. For example, in a Gaussian mixture model (GMM), EM alternately infers latent variables (the responsibilities of each Gaussian for explaining each data point) and Gaussian parameters [8]. The above missing data elements can also be considered as latent variables, and inferred in the process of EM. However, for the GMM in particular, the number of free parameters in the model becomes an issue. With 216 dimensions in the line image, the covariance of size 216×216 presents many parameters to fit, particularly when a reasonable number of clusters are desired per class, as the aim is in modelling all an object's parts with clusters.

In summary, this approach is promising because it specifically reasons about unknown space as a value to be inferred, as opposed to matching heuristics that penalise unknown space in the exemplar based approach. However, it requires an underlying local model for each cluster, without too many parameters. This approach was not pursued, but does present an interesting line of future work.

5.1.2 Experiments

Affinity propagation clustering was used on the object dataset introduced previously (Section 4.3), in order to perform classification with topic models (Section 5.2). The goal was to extract a representative number of clusters at several scales of analysis. As such, line images were generated at radii of 0.5, 1 and 2 m, at PCA keypoints as in Section 4.4. This resulted in a set of $n = 100\,761$ line images for each scale, across the whole dataset. Initially, each set of n features were clustered. However, two issues resulted:

(1) Classes with the most keypoints dominated, with most exemplars extracted from buildings.

(2) The memory requirements were high, as three $n \times n$ matrices[2] were required (for the similarities, availabilities and responsibilities).

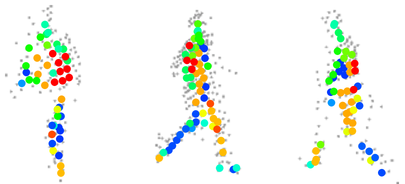
The first issue is, in part, a problem with doing clustering independently from classification. The clustering process does not consider how well each class is represented by the exemplars it produces. This was tackled by clustering each class separately, acquiring class-specific exemplars. As fewer input data points were clustered at any one time, this resulted in much smaller matrices, reducing memory and computation time requirements. The cost of this approach is that it cannot be applied to strictly unsupervised classification, as class labels are required in this prior step of clustering.

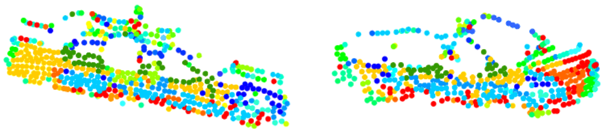
The number of clusters was controlled by varying the preference values of the similarity matrix, resulting in a set of clusters C for each class, and for each preference value. As the number of data points also influences the number of clusters, classes with a large number of keypoints received a large number of clusters at the same preference value, compared to other classes. In order to balance the classes, cluster sets C of approximately equal size were selected from each class. Roughly 10 clusters per class resulted in approximately 500 clusters in total, across all three scales. In order to classify objects using a four-fold cross validation, clustering was performed only on training data for each fold (i.e. it was performed four times).

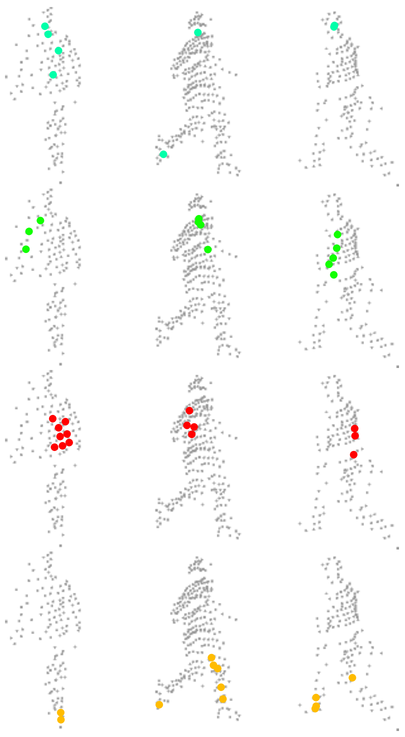
5.1.3 Results

Each feature vector from the dataset was matched to the final set of cluster exemplars. Many clusters appear to be repeatably localised on a given class of objects (Figure 5.2). The distribution of associations across classes is shown in Figure 5.3, showing each class has some degree of representation by the chosen exemplars. Exemplars are not necessarily class specific. The desired outcome at this stage is that objects are represented by a collection of shared parts. Classes are subsequently inferred by the statistics of parts, detailed in the next section.

[2]Sparse matrices could be used, but this requires an additional heuristic to determine which similarities are too small.


(a) Keypoints on pedestrians


(c) Keypoints on cars (no raw points)


(b) Isolated clusters

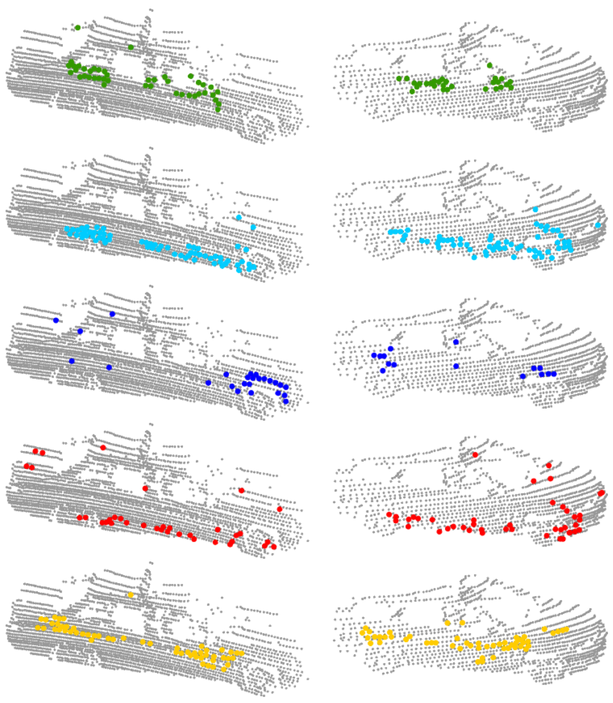

(d) Isolated clusters

Figure 5.2 – (a) Clusters on pedestrians, with each keypoint coloured by cluster association. Small grey points are the raw sensor data. (b) Separate cluster associations, showing clusters approximately matching to heads, shoulders, torsos, legs. (c) (d) The same on cars (with different colour-to-cluster mappings from pedestrians).

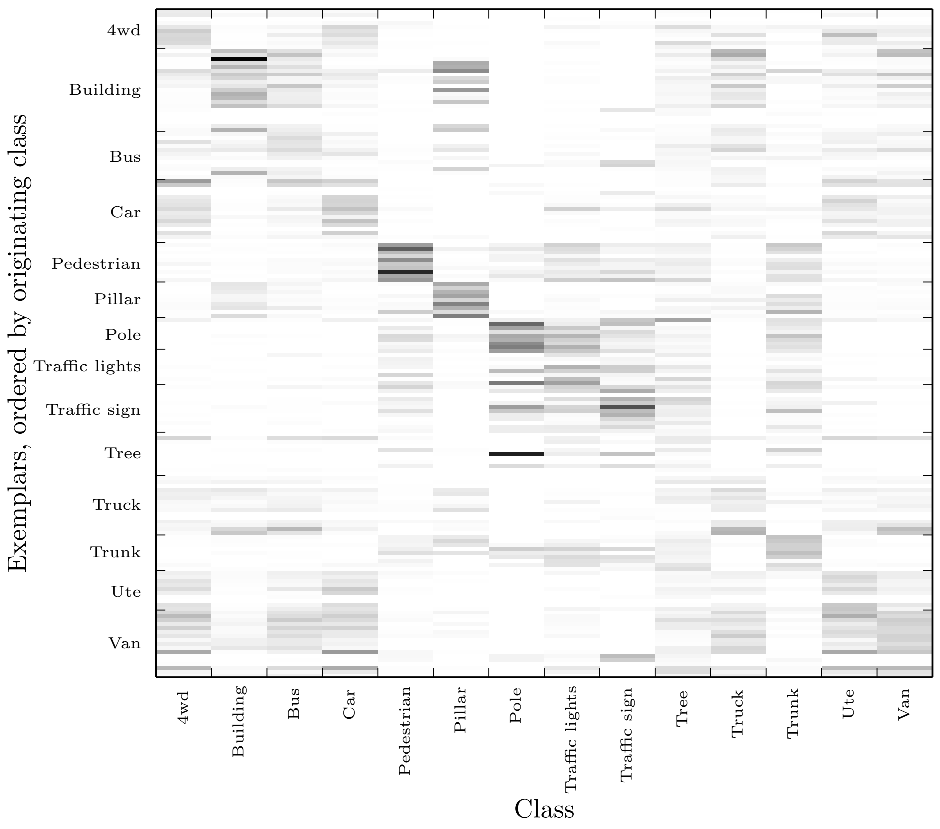


Figure 5.3 – Probability of each cluster (row) given each class (column), i.e. columns sum to 1. Clusters are arranged by originating class, showing a faint diagonal. While some clusters are class-specific, many are shared across several classes. Typical similarities can be observed, such as clusters present in cars also present in 4WDs, utes and vans. This is for a 1 m radius feature.

5.1.4 Summary

Affinity propagation was applied to line images, with the asymmetric distance measure allowing less visible instances of shapes to support more visible ones. Imbalances in the number of data points in each class necessitated per-class clustering, followed by selecting the preference values which provided an approximately equal number of clusters from each class. The resulting clusters were specific to different regions of objects, and had a degree of class specificity and class coverage. These clusters will be used to support classification in the next section.

5.2 Classification with Topic Models

This section applies topic modelling to classify 3D objects using line images and the above clustering results. Topic models were introduced earlier in Section 2.4.2, and provide a generative model that captures some underlying statistical relationships between the parts of an object, allowing for richer classification. Topic models are used in computer vision [28, 18, 74], and have recently been applied to 3D point clouds [27] using a modified spin image.

To re-iterate the basics of topic modelling from Section 2.4.2, objects are comprised of a set of feature vectors, with each instance capturing the local shape of an object region. Each instance is associated with a cluster, allowing it to be summarised by the cluster I.D., or codeword. Each object, also referred to as a document in this context, is summarised by a set of these codewords, in this case from a vocabulary of approximately 500 codewords (the total number of clusters from the previous section). This set of codewords is known as a 'bag of words' representation [21].

Now consider the simplest form of topic model: Latent Dirichlet Allocation [10] (LDA). This model describes how an object is generated. Consider a fixed number of K topics:

(1) Each topic β_k is a distribution over the vocabulary of words.

(2) Each word w in a document d is generated from a topic ($z_{d,n}$ is the assignment of word n in document d to a topic).

(3) Each document has a distribution over topics θ_d.

In the process of training the model (shown in graphical form in Figure 2.11), the distributions that comprise topics and documents are encouraged to be sparse. This results in topics loosely capturing co-occurring shapes within objects.

5.2.1 Multi-class Supervised LDA

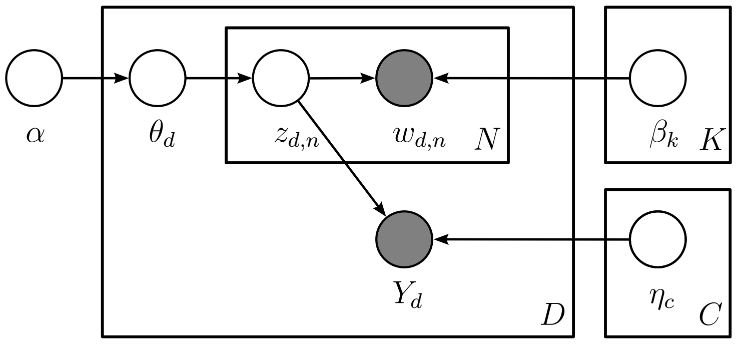
LDA is an unsupervised classification method, where each topic can be considered as a class. However, this is not strictly necessary; topics are simply co-occurring codewords (parts). A larger number of topics than classes can be used, and the distribution over topics can be used as a feature vector. This (single) vector can then be used with standard supervised classifiers. However, a better approach which integrates class labels into the graphical model itself is in supervised LDA [11] (sLDA). Each document has an associated response variable that the model is to predict, given labelled training instances. The response variable was originally formulated to be generally applicable to any type of data, such as predicting movie ratings from reviews [11]. This was extended to the multi-class case in [87] for image classification, resulting in topics which best predict classes.

This section focuses on using multi-class sLDA for 3D data, which results in classification scores 10% better than from the best k-NN classification in Section 4.4. This is the first work to apply sLDA to 3D data, to the best knowledge of the author. It provides a framework for combining multiple scales of local features, while finding correlations between features which best distinguish class.

5.2.1.1 Model

This section provides a review of multi-class sLDA, as described in [87][3]. An object (document) d is generated by drawing items from the Dirichlet (Dir) and Multinomial (Mult) distributions, following the process shown in graphical form in Figure 5.4:

[3] Note that the model in [87] also included an additional step for handling image annotations, which is not used here.



$w_{d,n}$: the n^{th} word from document d.
$z_{d,n}$: the topic which generated the corresponding word.
θ_d : topic proportions for document d.
Y_d : the document's class label.
$\beta_{1:K}$: the K topics (distributions over words).
$\eta_{1:C}$: the C class-specific topic frequency coefficients.
α : Dirichlet hyperparameter.

Figure 5.4 – Graphical model of sLDA.

(1) Draw the topic proportions $\theta_d \sim \mathrm{Dir}(\alpha)$.

(2) For each keypoint on the object:

  (1) Draw a topic assignment $z_n|\theta \sim \mathrm{Mult}(\theta)$.

  (2) Draw a codeword $w_n|z_n \sim \mathrm{Mult}(\beta_{z_n})$.

(3) Draw class label $c|z_{1:N} \sim \mathrm{softmax}(\bar{z}, \eta)$, with:

$$\bar{z} = \frac{1}{N}\sum_n z_n \tag{5.5}$$

$$\mathrm{softmax}(\bar{z}, \eta) = p(c|\bar{z}, \eta) = \frac{\exp(\eta_c^T \bar{z})}{\sum_l \exp(\eta_l^T \bar{z})} \tag{5.6}$$

In determining the class label for an object, $\bar{z}$ is a vector of the frequencies of each topic, and $\eta_{1:C}$ the weights. These weights are estimated when training the model, and so classes in sLDA consist of regions in the topic frequency space.

Note that sLDA uses the empirical frequencies of word-topic assignments, rather than the topic proportions θ_d. As described in [11], this is because, while θ_d ought to describe these frequencies, an issue occurs if θ_d is additionally used to explain classes. When estimating the model, some topics can end up explaining classes *instead* of words. Such topics would not assist inferring the class given the words. Instead, with $\bar{z}$, topics must explain both words and classes during estimation.

5.2.1.2 Training

Training seeks to maximise the likelihood of the words in an object, and the object's class label. From the point of view of the class, a given object has a point in the topic frequency space, whose position is controlled by the set of word-topic assignments z_n. The boundaries between class regions are also controlled by the topic coefficients $\eta_{1:C}$. The goal is to move this point and the boundaries to correctly separate the classes. From the point of view of the words, each word is assigned a topic by z_n, which maximises the likelihood of that word. However, only a few topics per document are permitted through θ, due to the Dirichlet prior and in order to maximise the log-likelihood. Likewise, topics can only contain a few words through β.

The process of training the model was followed from [87]; a brief overview will be given here. Training consists of estimating the parameters, which includes the topics $\beta_{1:K}$ and the class coefficients $\eta_{1:C}$. In order to do this, the latent (per-document) variables must also be estimated, across the labelled training dataset. These consist of the topic proportions per object θ_d and each word's topic assignment $z_{d,n}$. The latent variables are estimated by computing the conditional distribution of the latent variables, given an initial set of model parameters and a labelled dataset. This is approximated using variational inference.

Variational inference allows the true posterior distribution to be approximated by a set of simpler, tractable ones. The process minimises the KL-divergence between the true and approximate distributions. In this case, a factorised distribution [8] is used (also known as the mean-field method) to simplify computation. This considers the approximating distribution q for each latent variable as independent. Each q is then optimised in turn, holding the other distributions constant, iteratively refining the approximation. The factorised distribution is

$$q(\theta, \mathbf{z}) = q(\theta|\gamma) \prod_{n=1}^{N} q(z_n|\phi_n) \tag{5.7}$$

where the parameters fitted are γ, a Dirichlet distribution over topics, and ϕ_n, a multinomial over topics.

Once this variational posterior is computed, the model parameters $\beta_{1:K}$ and $\eta_{1:C}$ are estimated. EM is used, where the E-step consists of inferring the above variational posterior for the latent variables. The M-step then estimates the parameters, and these steps are repeated until convergence. For the full formulation, see [87].

5.2.1.3 Classification

Once training is complete, test objects can then be classified. This involves inferring the class labels given an object's words. First, the latent variables are estimated as in training. Second, the probability $p(c|\bar{z}, \eta)$ (Equation (5.6)) is approximated as in [87], with the softmax function becoming a simple maximum. With ϕ_n as the variational approximation of z_n, replacing $\bar{z}$:

$$c = \arg\max_c \eta_c^T \bar{\phi} \tag{5.8}$$

5.2.2 Experiments

The implementation of multi-class sLDA available from [88] was used. As before, the Sydney dataset (Section 4.3) was split into four folds. The previous Section 5.1 yielded several sets of cluster exemplars, with a set:

- for each fold (clustered on the associated training data),
- for each scale of 0.5, 1 and 2 m radius.

For a given fold and scale, the line images from each object were assigned the closest exemplar, or codeword. This resulted in a given object having a set of codewords, which was summarised as a histogram over the vocabulary (i.e. the counts of each word in the object). The is the bag of words model.

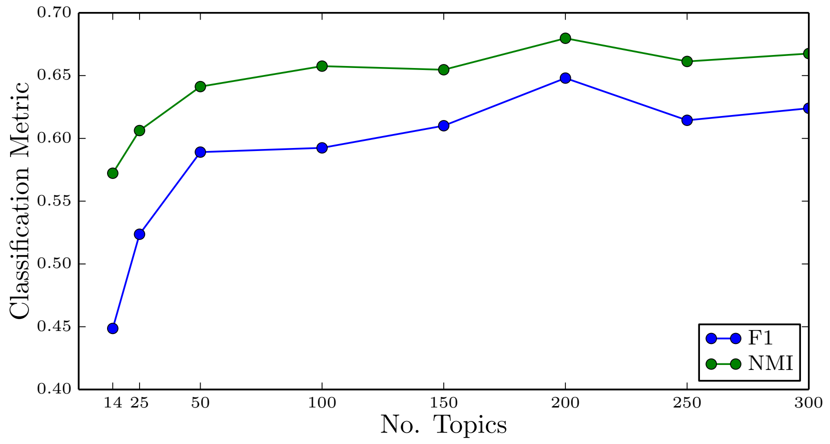
The training objects were then used to train the model, with sLDA estimating the topics $\beta_{1:K}$ and per-class topic frequency coefficients $\eta_{1:C}$. These were then used to infer the classes of the test objects.

The number of topics was varied, with results shown in Figure 5.5a. The number of clusters per class was also varied, shown in Figure 5.5c.

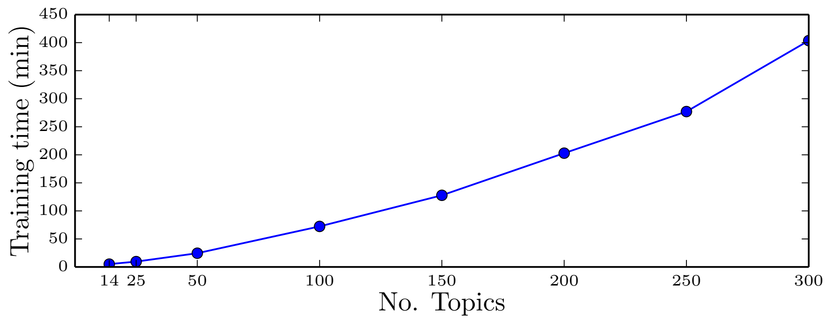
5.2.3 Results

The results from varying the number of topics are shown below. The metrics are from the 14-class dataset. With 200 topics providing the best result, the confusion matrices and class-specific metrics are then examined. This is shown first with the 8-class dataset in Section 5.2.3.1, and repeating again for the 14-class dataset in Section 5.2.3.2. In addition, the matrices underlying the 'code-word to topic' and 'topic to class' mapping are visualised in Figures 5.7 and 5.8, and further discussed in Section 5.2.4.

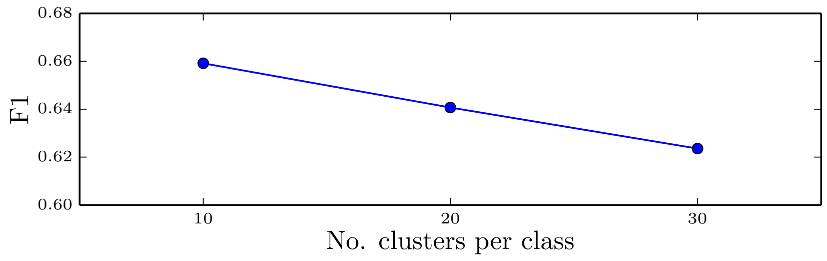
Note that, in this chapter, the clustering, training and classification are performed on the 14-class set only. The 8-class results are purely a post-classification step grouping the results of vans, 4WDs etc.



(a) Classification metrics with different numbers of topics.



(b) Training time vs number of topics.



(c) F_1 score with an increasing number of clusters, 200 topics.

Figure 5.5 – Metrics on the full 14-class dataset, with a varying number of topics. 200 topics produced the best result, at the cost of a long training time (although testing takes less than a minute). Increasing the number of clusters per class had a small negative effect on performance.

5.2.3.1 8 Classes

Table 5.1 – Metrics for the 8 classes classified with sLDA, compared with the best from *k*-NN.

Metric	sLDA	*k*-NN
Average class F_1 measure	0.82	0.7
NMI	0.76	0.71
Average class accuracy	0.82	0.72

Table 5.2 – The confusion matrix for sLDA.

Truth \ Inferred	Building	Car	Pedestrian	Pillar	Pole	Traffic sign	Tree	Truck
Building	18	0	0	0	0	0	1	1
Car	0	155	0	0	0	0	1	4
Pedestrian	0	0	151	0	1	0	0	0
Pillar	0	0	0	16	4	0	0	0
Pole	0	1	2	6	49	15	3	0
Traffic sign	0	1	2	0	13	79	2	1
Tree	0	0	1	0	2	2	29	0
Truck	1	9	0	0	0	1	0	17

Table 5.3 – Per-class metrics, 8 classes.

Class	Precision	Recall	F_1	Often confused as
Building	0.95	0.9	0.92	
Car	0.93	0.97	0.95	
Pedestrian	0.97	0.99	0.98	
Pillar	0.73	0.8	0.76	Pole (20%)
Pole	0.71	0.64	0.68	Traffic sign (20%)
Traffic sign	0.81	0.81	0.81	Pole (13%)
Tree	0.81	0.85	0.83	
Truck	0.74	0.61	0.67	Car (32%)

5.2.3.2 Specialised 14 Classes

Table 5.4 – Metrics for the specialised 14 classes classified with sLDA, compared with the best from k-NN.

Metric	sLDA	k-NN
Average class F_1 measure	0.65	0.48
NMI	0.68	0.64
Average class accuracy	0.63	0.5

Table 5.5 – The confusion matrix for sLDA.

Truth \ Inferred	4wd	Building	Bus	Car	Pedestrian	Pillar	Pole	Traffic lights	Traffic sign	Tree	Truck	Trunk	Ute	Van
4wd	4	0	0	16	0	0	0	0	0	0	0	0	0	1
Building	0	18	1	0	0	0	0	0	0	1	0	0	0	0
Bus	0	0	11	3	0	0	0	1	0	0	0	0	0	1
Car	4	0	0	81	0	0	0	0	0	1	0	0	2	0
Pedestrian	0	0	0	0	151	0	0	0	0	0	0	1	0	0
Pillar	0	0	0	0	0	16	1	0	0	0	0	3	0	0
Pole	0	0	0	0	0	0	8	1	4	0	0	8	0	0
Traffic lights	0	0	0	0	1	0	0	33	5	2	0	6	0	0
Traffic sign	0	0	0	0	1	0	1	7	34	0	1	6	1	0
Tree	0	0	0	0	1	0	0	1	1	29	0	2	0	0
Truck	0	1	1	3	0	0	0	0	0	0	5	0	1	1
Trunk	0	0	0	1	2	6	1	8	2	3	0	32	0	0
Ute	0	0	0	11	0	0	0	0	0	0	0	0	5	0
Van	2	0	2	14	0	0	0	0	0	0	2	0	1	14

Table 5.6 – Per-class metrics, 14 classes.

Class	Precision	Recall	F_1	Often confused as
4WD	0.4	0.19	0.26	Car (76%)
Building	0.95	0.9	0.92	
Bus	0.73	0.69	0.71	Car (19%)
Car	0.63	0.92	0.75	
Pedestrian	0.97	0.99	0.98	
Pillar	0.73	0.8	0.76	Trunk (15%)
Pole	0.73	0.38	0.5	Trunk (38%)
Traffic lights	0.65	0.7	0.67	Trunk (13%)
Traffic sign	0.74	0.67	0.7	Traffic lights (14%)
Tree	0.81	0.85	0.83	
Truck	0.63	0.42	0.5	Car (25%)
Trunk	0.55	0.58	0.57	Traffic lights (15%)
Ute	0.5	0.31	0.38	Car (69%)
Van	0.82	0.4	0.54	Car (40%)

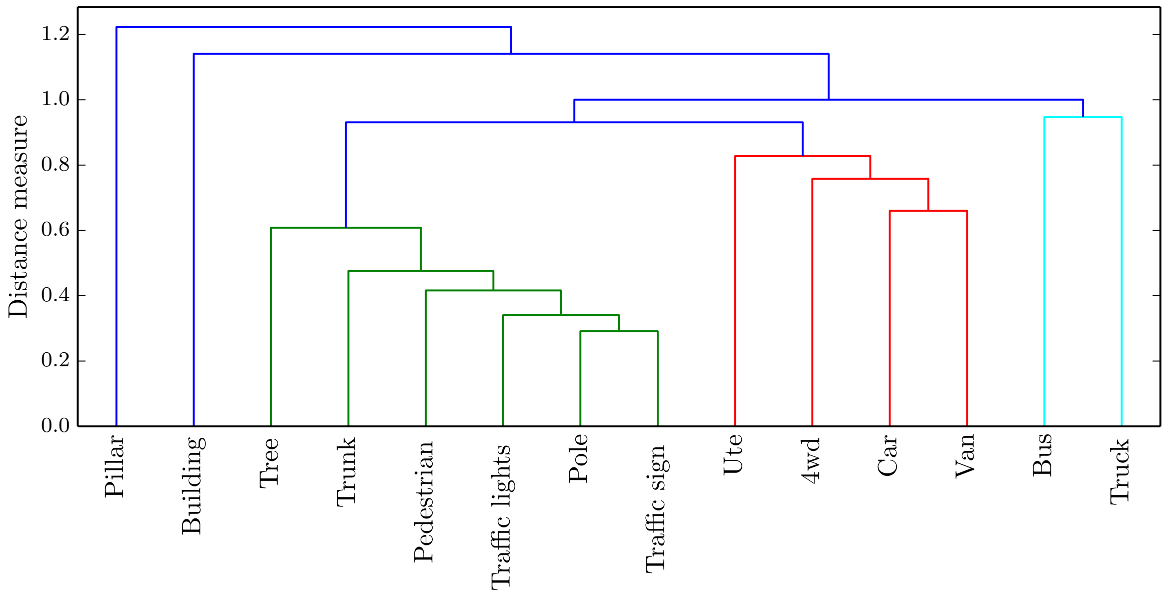


Figure 5.6 – A dendrogram showing similarities between classes. This was built by taking the average distribution of topics for each class, and applying hierarchical clustering. This results in intuitive groupings such as between vehicles, and signs/poles.

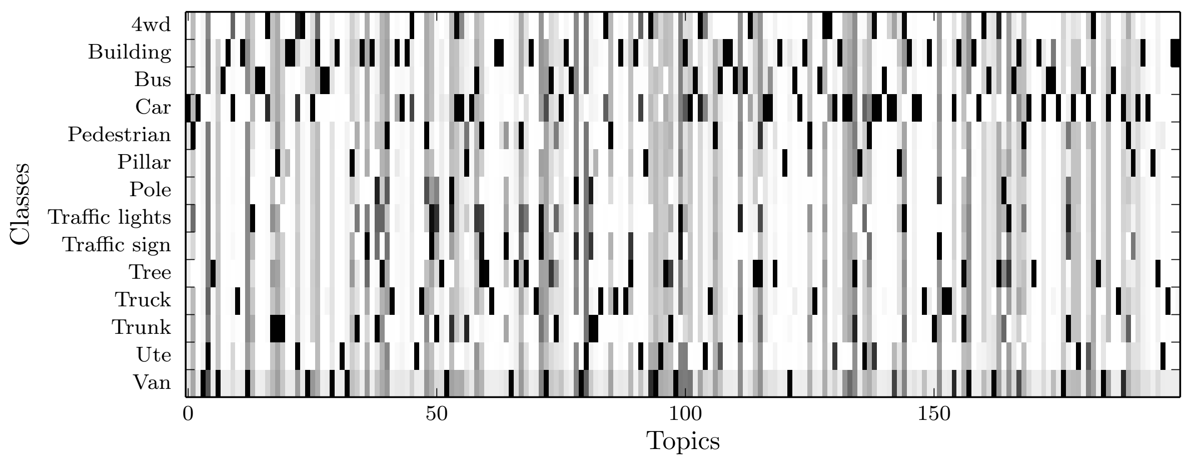


Figure 5.7 – The per-class, per-topic coefficients $\eta_{1:C}$ (darker is higher). To classify an object (Equation (5.8)), each class (row in the above) receives a score, computed as follows. Consider the object's topic frequencies $\bar{z}$, a vector of equal length to a row in the above figure. These are multiplied with the class/row of η above, and summed to produce the class score. The class with the maximum score is chosen as the output. As such, if an object contains a topic that falls in a dark-shaded cell for a given class, that class becomes more likely. (In actuality, there are only 13 rows of η, with the final class defined by a score of less than 0; η can contain negative elements. The above visualisation considers the last row as all zeros, and normalises each column to range from 0 to 1.)

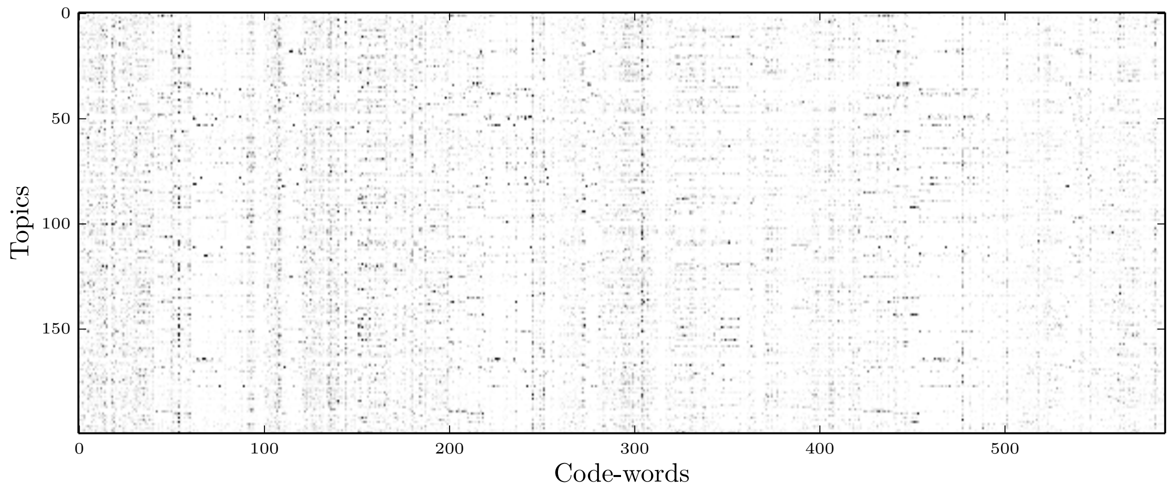


Figure 5.8 – The distribution of codewords (columns) over each topic (rows), so each row sums to one. Darker corresponds to more probability. Each topic (row) encodes the co-occurrance of shaded codewords.

5.2.4 Discussion

Using only 14 topics results in poor performance; this case treats each topic as representing one class. There is significant improvement at 50 topics, continuing to increase up to 200 topics. Beyond 200 topics, overfitting occurs and performance slowly decreases. In comparison, visual sLDA in [87] was performed on 8 classes of whole images in particular settings or activities, where 100 topics was found to be optimal. Figure 5.7 shows how topics relate to classes, with some topics shared between classes, and some that are class-specific. With 200 sets of co-occurring shapes, there is a large latent space to separate classes. Using this number of topics reinforces the interpretation of topics as subcomponents of a class, and not directly as classes themselves. Using too few topics limits the ways two classes can differ. Using too many topics results in overfitting the model, impacting its ability to generalise to new data. The optimal number of topics may depend on the size and variability of the dataset, with more classes warranting more topics.

To examine what each topic encodes, the distribution of words over topics is shown in Figure 5.8. Words are sparse, with approximately 5 to 10 words of significant probability in each topic. Thus, each topic reflects roughly 5 parts that are likely to occur together in an object. Some words are more common across topics, seen as vertical bands. These occur regardless of other shapes, and so have a greater degree of statistical independence.

Using more exemplars for each class had a small negative effect. It may be that splitting the small dataset up into an even more verbose dictionary of codewords reduces the ability to find salient topics. As clustering is performed in isolation to training the model, the exemplars chosen are not optimal.

This approach produced significant gains over k-NN (Tables 5.1 and 5.4), with 10% gains in accuracy metrics in both 8-class and 14-class datasets. Looking at the 14-class results, the NMI increases from k-NN's 0.64 to sLDA's 0.68, a less significant gain than the F_1 increase from 0.48 to 0.65. Recall from Section 4.4.2 that NMI permits two classes to be grouped, whereas the F_1 measure is strict in considering misclassified objects. This indicates that the classes previously being misclassified were being consistently placed in another class. With sLDA, they are now being distinguished. One issue with the k-NN results was the large number of false-positive buildings, particularly with buses, trucks and vans (Table 4.12). This may be because flat regions are common to these classes, and there are many more flat parts on buildings, resulting in many k-NN votes for this class. In contrast, sLDA results in high precision for buildings. It considers sets of regions rather than independent pooling, improving the ability to distinguish these objects.

While training can take several hours for a large number of topics (Figure 5.5b), classification is very fast, taking only 46 s for all folds of the dataset for the 200 topics. This excludes feature generation and matching to exemplars, although there are only approximately 500 exemplars whose distances must be computed. This contrasts with the half-hour classification time of k-NN.

Specialised classes such as 4WDs, utes and vans are still not clearly separated from cars. However, buses, traffic signs and trees perform much better. Some insight into the similarities of classes is shown in Figure 5.6, where they are arranged in a dendrogram, formed from hierarchical clustering on the average topic vector of each class. Two possible approaches could further improve these subtle classes: (1) Clustering techniques that identify class-specific exemplars. (2) Hierarchical classification such as in [74], in order to acknowledge the commonalities that group classes like vehicles, and focus on more subtle differences.

5.2.5 Conclusion

This section applied multi-class supervised LDA to 3D object classification. Using a dictionary of codewords from the previous clustering stage, each object was represented as a set of codewords. This allowed multiple scales of analysis to be combined, as each scale simply provided another set of codewords. The model then learned topics, or co-occurring sets of codewords, with each object summarised by the frequencies of each topic. These topic frequencies were used to distinguish object classes. The best

F_1 score of 0.82 was achieved by using 200 topics, a significant gain over the k-NN approach of the previous chapter. As such, this approach provides a principled way of combining local features for object classification.

Chapter 6

Conclusion

This thesis has presented an approach for performing object classification on sparse range images, with an evaluation on urban field data from a Velodyne LiDAR. The emphasis has been on the following.

- Addressing the effects of sparsity and noise in the data during low level operations such as surface normal computation.
- Forming invariant and distinguishing local shape representations, allowing object parts to be matched.
- Using these local shape representations to classify objects.

Without an effective local shape representation, it becomes difficult for classification algorithms to generalise beyond the pose and shape of training instances.

This chapter summarises the proposed solutions presented in this thesis, and suggests future directions for this work.

6.1 Summary of Contributions

The processing steps proposed in this thesis are combined into a pipeline in Figure 6.1. These are grouped as: (1) abstracting from raw data to basic geometric information,

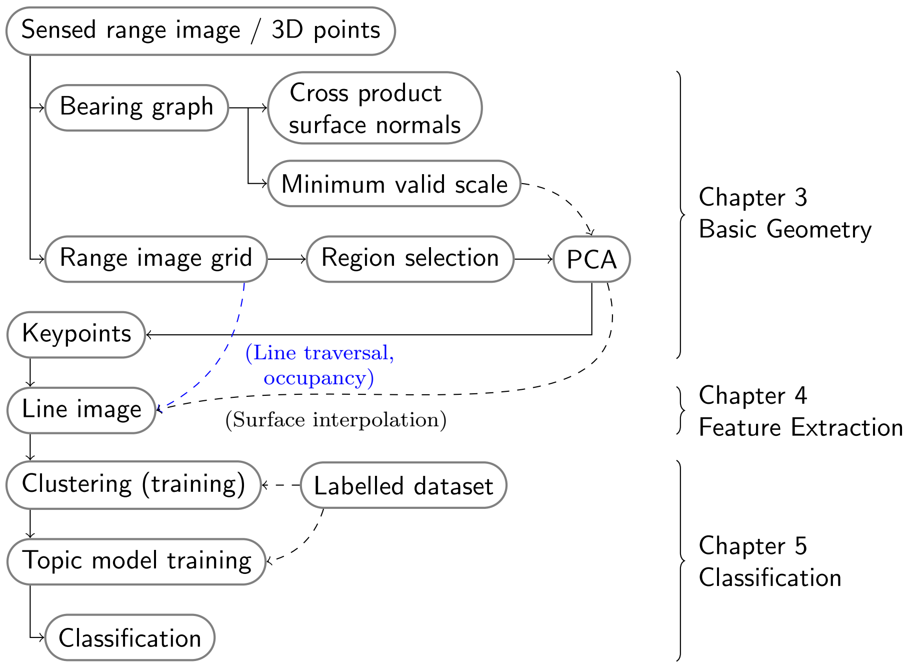


Figure 6.1 – The presented pipeline from sensory data to classification, linked to each relevant section.

(2) the line image feature for capturing a complex, local shape representation, and (3) using sets of line images for classifying objects.

6.1.1 Low Level Shape Analysis

Chapter 3 presented several processing steps from raw point clouds to simple surface regions and their properties. It examined region selection and the various data structures supporting this, followed by surface normal computation. Keypoints were examined, which define specific locations and orientations for features. Contributions included:

- Using the range image for efficient spherical region selection.
- Using the bearing graph for defining a minimum valid scale of analysis.
- Performing traversal on the bearing graph to select further neighbours for surface normal computation via the cross product method.
- Keypoints derived from PCA eigenvalues, providing locations at stable orientations.

6.1.2 The Line Image

Chapter 4 presented the line image feature. This feature incorporated occupancy information in order to capture distinguishing information while remaining invariant to changes in viewpoint. It involved a set of parallel 3D lines aligned to the surface, probing for the depth of the interpolated surface. Using the range image, each line was considered as being ‘in front’ or ‘behind’ the sensed points, allowing lines to register occlusion (unknown space) and empty space. Missing data in the range image was likewise registered as unknown space. This produced a local 2½-D representation of occupancy. The distance measure between two line images permitted unknown space to match to anything, while penalising violations of observed empty space.

The Sydney Urban Objects Dataset was introduced, providing a means to compare the line image with several existing features on the task of object classification. Using a simple k-NN classifier, different features and support scales were compared. Line images performed equivalently to spin images, but at smaller scales of analysis, encompassing parts rather than whole objects.

To summarise, the contributions of this chapter were:

- Efficient occupancy detection along a 3D line using joint reasoning in the range image and 3D point cloud.
- An approximate, fast surface interpolation method using PCA.
- The line image formulation using the above, with a distance metric to allow regions to be compared consistently.
- The Sydney dataset, consisting of 588 urban objects across 14 classes, scanned with a Velodyne.
- A feature comparison study with Spin Images, NARF and FPFH, applied to object classification with a k-NN classifier.

6.1.3 Classifying from Local Features

The k-NN approach assumes each feature is independent, which can mean reduced performance for smaller scale, part-sized shapes. In order to use more advanced classification techniques, clustering was examined in Section 5.1, allowing the feature space to be reduced into a set of clusters. Affinity propagation was used to cluster line images with an asymmetric distance measure, allowing less visible instances of line images to support more visible ones.

Using these clusters as a dictionary of codewords, Section 5.2 examined the use of multi-class supervised LDA for 3D object classification. Each line image in an object became a codeword (its nearest cluster exemplar), reducing an object into a bag of words. This representation permitted multiple scales of line images to be combined. Given the labelled training dataset, the topic model then learned topics, which encoded co-occurring sets of codewords. The classification process involved inferring each object's topics using the learnt model, and using the frequency of topics to predict the class. This process was faster than k-NN, and produced a higher F_1 score of 0.82.

In summary, this chapter's contributions were:

- An approach to clustering partially-missing data, using an asymmetric distance metric and affinity propagation.
- The application of multi-class supervised latent Dirichlet allocation to 3D object classification, using multiple scales of analysis.

6.2 Future Research

There are potential improvements or extensions for each section of this thesis. These involve keypoints, surface models, and improvements to the line image and associated distance measures. In terms of machine learning, there is the potential for incorporating more information from the line image and geometry, as well as better models for clustering and classification.

6.2.1 Keypoint and Surface Definition

A large issue with the PCA keypoint algorithm is that large, flat surfaces received many points, which increased the computation burden and redundancy. One way around this would be to compute PCA at an additional, larger scale, and subsample *large* flat areas more heavily.

A general issue with defining keypoints from simple analyses such as PCA and curvature is that informative regions can be disregarded before any complex shape analysis is performed. A flat region is informative for distinguishing class, but many overlapping flat regions are redundant. To solve this, keypoints as sparse, well-localised positions could be revisited *after* clustering. Features such as line images would first be computed as before, at all locations as defined by simple PCA keypoints. Then, these features would be matched to a set of exemplars. For each region, the feature with the best match to an exemplar would be kept, and the rest discarded. This way, instead of a given region of points producing a set of the same, redundant codewords, a single codeword is produced. Final keypoints are aligned to where an exemplar was found to match. Such a scheme would not lose information naively, while producing fewer, localised keypoints.

Another area of 'low level' future work is in surface interpolation, which the line image relies on. PCA is currently only computed at a single scale, with areas of very low density not receiving any interpolated surface. Instead, multiple scales could be computed, as determined from the bearing graph. An alternative is surface modelling in the range image space. For instance, a Gaussian process could yield more effective results, if an efficient formulation was used.

6.2.2 Line Image Formulation

Several other aspects of the line image could be modified and improved, as touched upon in Section 4.2.6. The line image still requires much more computation power than spin images. Two aspects that influence this are the number of lines, and the length of lines. Using many lines provides more detail, and too few lines can lead to some smaller surfaces being missed altogether. One modification would be to use fewer lines, but to analyse a larger region with each line, effectively turning lines into cylinders. Rather than looking for where a specific line intercepts a surface, a cylinder can encompass a surface patch. Additional surface properties could also be captured, such as PCA eigenvalues.

Long lines result in more points to select, sort and traverse. The line length in the current formulation is set by the overall spherical size of analysis, but the surface of interest[1] and surrounding volume only occupies a small portion of the line. Long lines can also encounter unknown space, or even other surfaces, before encountering the visible surface of interest. This is difficult to improve upon without making prior assumptions of where the relative surface depth may be. As a result, the line formulation searches all depths. A rough surface model over the spherical region could define a reduced space to inspect.

Another option is to disregard lines altogether, and detect surface depths using another method (e.g. simple projection). In order to capture the occupancy at locations without Locations without a surface could then be assigned an 'inspection depth', based on a neighbouring surface depth. This defines a 3D point whose occupancy can be inspected with the range image, in order to capture things like empty surrounding space, or occluded space.

6.2.3 Distances and Clustering

The distance measure between line images could be improved. The current measure involves matching heuristics that could be based on something more principled. If line images were modelled as incomplete data, as suggested in Section 5.1.1.2, a distance measure based on information theory may be applicable.

In addition, the line image is essentially a small image patch, and so second-order features of the line image itself could be considered. The challenge, as in clustering, is

[1]This is nominally at the keypoint where the feature is being computed

to incorporate the unknown and empty space. One method would be to use a set of linear filters, similar to Gabor filters in vision. Each filter would describe some aspect of the line image's surface intercepts or empty regions. For example, a filter could be specific to a flat portion, or a transition from surface to empty space. This transforms the line image into a set of filter weights. Unknown portions could be represented by considering a distribution over each weight.

For clustering, the approach of Section 5.1.1.2 could be pursued, where each cluster is modelled, considering the unknown space as missing data to be inferred. Additionally, if clustering was done jointly with classification [92], the clusters chosen could better distinguish each class.

6.2.4 Classifying from Multiple Parts

A natural area of continuation for this work is exploring better machine learning approaches to classification from constituent parts. A simple extension would be permitting each line image to be associated to a distribution of codewords, accounting for the range of potential matches due to unknown portions of the feature.

A significant source of information that is lost in the bag-of-words representation is larger scale structure. The geometric arrangement of codewords could be considered. The geometric consistency of features across two different objects can be used to align and match them [43, 71]. Generalising this to classification could take the form of constellation models [29] or other graphical models [19], although this may require sparse, well localised keypoints. Another approach in topic modelling does this [18] by over-segmenting the image into patches, and then limiting the codewords within each patch to be from a single topic. Topics then reflect words that are spatially nearby.

Segmentation could also be revisited in the context of classification, for example with MRFs, as discussed in Section 2.2.

Other sources of sensory information could be added to the process, for instance, colour imagery of each object. Standard colour vision approaches could generate visual words, which can be added to the dictionary.

6.2.5 Other Applications

One aspect that was not in the scope of this thesis was tracking. In the target scenario of video frame rates in dynamic environments, multiple observations provide an additional source of information. In [51], classification of tracks of objects produced good results. If a specific region was tracked, for instance by localised keypoints or from matching line images, the line images across observations could be merged in a filter. This could permit unknown space to be filled in by subsequent observations.

In the field of planning and next best view prediction, a common objective is to move the robot to a location for better sensory data. A line image allows limitations in sensory data to be connected to semantic uncertainty. For example, there could be a region that, due to occlusion, could belong to a person or a trunk. If this object's class hinged on that region, moving to better observe this target would take precedence.

Bibliography

[1] M. Ankerst, G. Kastenmüller, H. Kriegel, and T. Seidl. 3D shape histograms for similarity search and classification in spatial databases. In *Advances in Spatial Databases*, pages 207–226. Springer, 1999.

[2] H. Badino, D. Huber, Y. Park, and T. Kanade. Fast and accurate computation of surface normals from range images. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3084–3091. IEEE, 2011.

[3] H. Becker. *Identification and characterization of events in social media*. PhD thesis, Columbia University, 2011.

[4] J. L. Bentley. Survey of techniques for fixed radius near neighbor searching. Technical report, Stanford Linear Accelerator Center, Calif. (USA), 1975.

[5] J. L. Bentley, D. F. Stanat, and E. H. Williams. The complexity of finding fixed-radius near neighbors. *Information processing letters*, 6(6):209–212, 1977.

[6] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[7] P. Besl. Triangles as a primary representation. *Object Representation in Computer Vision*, pages 191–206, 1995.

[8] C. M. Bishop et al. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[9] D. M. Blei. Probabilistic topic models. *Communications of the ACM*, 55(4): 77–84, 2012.

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[11] D. Blei and J. McAuliffe. Supervised topic models. *arXiv preprint arXiv:1003.0783*, 2010.

[12] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield. Little ben: The ben franklin racing team's entry in the 2007 darpa urban challenge. *Journal of Field Robotics*, 25(9): 598–614, 2008.

[13] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. Ieee, 2008.

[14] M. Bosse and R. Zlot. Place recognition using regional point descriptors for 3D mapping. In *Field and Service Robotics*, pages 195–204. Springer, 2010.

[15] Y. Boykov and G. Funka-Lea. Graph cuts and efficient nd image segmentation. *International Journal of Computer Vision*, 70(2):109–131, 2006.

[16] D. Bradley, R. Unnikrishnan, and J. Bagnell. Vegetation detection for driving in complex environments. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 503–508. IEEE, 2007.

[17] R. J. Campbell and P. J. Flynn. A survey of free-form object representation and recognition techniques. *Computer Vision and Image Understanding*, 81(2): 166–210, 2001.

[18] L. Cao and L. Fei-Fei. Spatially coherent latent topic model for concurrent segmentation and classification of objects and scenes. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[19] D. J. Crandall and D. P. Huttenlocher. Weakly supervised learning of part-based spatial models for visual object recognition. In *Computer Vision–ECCV 2006*, pages 16–29. Springer, 2006.

[20] P. Csakany and A. Wallace. Representation and classification of 3-D objects. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 33 (4):638–647, 2003.

[21] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, page 22, 2004.

[22] B. Douillard, J. Underwood, V. Vlaskine, A. Quadros, and S. Singh. A pipeline for the segmentation and classication of 3D point clouds. In *International Symposium on Experimental Robotics*, 2010.

[23] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel. On the segmentation of 3D LIDAR point clouds. In *Robotics and Automation, 2011. Proceedings. ICRA'11. IEEE International Conference on*, pages 2798–2805. IEEE, 2011.

[24] B. Douillard, A. Quadros, P. Morton, J. Underwood, and M. DeDeuge. A 3D classifier trained without field samples. In *International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2012.

[25] D. Dueck and B. Frey. Non-metric affinity propagation for unsupervised image categorization. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[26] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics*, 3(1):2–12, 2012.

[27] F. Endres, C. Plagemann, C. Stachniss, and W. Burgard. Unsupervised discovery of object classes from range data using latent dirichlet allocation. In *Robotics: Science and Systems*, page 34, 2009.

[28] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 524–531. IEEE, 2005.

[29] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[30] B. Freedman, A. Shpunt, M. Machline, and Y. Arieli. Depth mapping using projected patterns. 2012. US Patent 8,150,142.

[31] B. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[32] B. J. Frey and D. Dueck. Mixture modeling by affinity propagation. *Advances in neural information processing systems*, 18:379, 2006.

[33] B. Fulkerson, A. Vedaldi, and S. Soatto. Class segmentation and object localization with superpixel neighborhoods. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 670–677. IEEE, 2009.

[34] W. Garage. PR2 user manual. `http://pr2support.willowgarage.com/wiki/PR2%20Manual`. 2012.

[35] I. Givoni, C. Chung, and B. J. Frey. Hierarchical affinity propagation. *arXiv preprint arXiv:1202.3722*, 2012.

[36] C. Glennie and D. Lichti. Static calibration and analysis of the Velodyne HDL-64E S2 for high accuracy mobile scanning. *Remote Sensing*, 2(6): 1610–1624, 2010.

[37] S. Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1):335–346, 1990.

[38] G. Hetzel, B. Leibe, P. Levi, and B. Schiele. 3D object recognition from range images using local feature histograms. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–394. IEEE, 2001.

[39] H. T. Ho and D. Gibbins. Curvature-based approach for multi-scale feature extraction from 3D meshes and unstructured point clouds. *IET computer vision*, 3(4):201–212, 2009.

[40] R. Hoffman and A. K. Jain. Segmentation and classification of range images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 9(5):608–620, 1987.

[41] K. Honda, R. Nonoguchi, A. Notsu, and H. Ichihashi. Pca-guided k-means clustering with incomplete data. In *Fuzzy Systems, IEEE International Conference on*, pages 1710–1714. IEEE, 2011.

[42] B. K. P. Horn. Extended gaussian images. *Proceedings of the IEEE*, 72(12): 1671–1686, 1984.

[43] A. Johnson. *Spin-images: a representation for 3-D surface matching.* PhD thesis, Carnegie Mellon University, 1997.

[44] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 156–164. Eurographics Association, 2003.

[45] K. Khoshelham. Accuracy analysis of kinect depth data. In *ISPRS workshop laser scanning*, volume 38, page 1, 2011.

[46] K. Klasing, D. Althoff, D. Wollherr, and M. Buss. Comparison of surface normal estimation methods for range sensing applications. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3206–3211. IEEE, 2009.

[47] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[48] J. Lalonde, N. Vandapel, D. Huber, and M. Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of Field Robotics*, 23(10):839–861, 2006.

[49] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, et al. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10):727–774, 2008.

[50] J. Levinson and S. Thrun. Unsupervised calibration for multi-beam lasers. In *International Symposium on Experimental Robotics*, 2010.

[51] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 163–168. IEEE, 2011.

[52] D. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[53] W. Maddern, A. Harrison, and P. Newman. Lost in translation (and rotation): Rapid extrinsic calibration for 2D and 3D LIDARs. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3096–3102. IEEE, 2012.

[54] A. Mian, M. Bennamoun, and R. Owens. On the repeatability and quality of keypoints for local feature-based 3D object retrieval from cluttered scenes. *International Journal of Computer Vision*, 89(2-3):348–361, 2010.

[55] I. Miller and M. Campbell. A mixture-model based algorithm for real-time terrain estimation. *Journal of Field Robotics*, 23(9):755–775, 2006.

[56] I. Miller, M. Campbell, D. Huttenlocher, F.-R. Kline, A. Nathan, S. Lupashin, J. Catlin, B. Schimpf, P. Moran, N. Zych, et al. Team cornell's skynet: Robust perception and planning in an urban environment. *Journal of Field Robotics*, 25 (8):493–527, 2008.

[57] 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.

[58] F. Moosmann, O. Pink, and C. Stiller. Segmentation of 3D lidar data in non-flat urban environments using a local convexity criterion. In *Intelligent Vehicles Symposium, 2009 IEEE*, pages 215–220. IEEE, 2009.

[59] M. Muja and D. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications (VISSAPP09)*, pages 331–340, 2009.

[60] D. Munoz, N. Vandapel, and M. Hebert. Directional associative markov network for 3-D point cloud classification. In *Fourth international symposium on 3D data processing, visualization and transmission*. Citeseer, 2008.

[61] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Shape distributions. *ACM Transactions on Graphics (TOG)*, 21(4):807–832, 2002.

[62] A. Quadros, J. Underwood, and B. Douillard. An occlusion-aware feature for range images. In *Robotics and Automation, 2012. Proceedings. ICRA'12. IEEE International Conference on*, pages 4428–4435. IEEE, May 14-18 2012.

[63] A. Quadros, J. Underwood, and B. Douillard. Sydney urban objects dataset. `http://www.acfr.usyd.edu.au/papers/SydneyUrbanObjectsDataset.shtml`. 2013.

[64] X. Ren and J. Malik. Learning a classification model for segmentation. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 10–17. IEEE, 2003.

[65] C. V. Rijsbergen. *Information retrieval.* London: Butterworths, 2nd ed edition, 1979.

[66] A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *EMNLP-CoNLL*, volume 7, pages 410–420, 2007.

[67] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *3dim*, page 145. Published by the IEEE Computer Society, 2001.

[68] R. B. Rusu and S. Cousins. 3D is here: Point cloud library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[69] R. B. Rusu, N. Blodow, Z. C. 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.

[70] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2155–2162. IEEE, 2010.

[71] R. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009.

[72] S. Salti, F. Tombari, and L. D. Stefano. A performance evaluation of 3D keypoint detectors. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 International Conference on*, pages 236–243. IEEE, 2011.

[73] J. Schafer and J. Graham. Missing data: our view of the state of the art. *Psychological methods*, 7(2):147, 2002.

[74] J. Sivic, B. Russell, A. Zisserman, W. Freeman, and A. Efros. Unsupervised discovery of visual object class hierarchies. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[75] S. Skiena. *The algorithm design manual: Text*, volume 1. Springer, 1998.

[76] B. Steder, R. Rusu, K. Konolige, and W. Burgard. Point feature extraction on 3d range scans taking into account object boundaries. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2601–2608. IEEE, 2011.

[77] A. Strehl and J. Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research*, 3: 583–617, 2003.

[78] R. Szeliski. *Computer vision: algorithms and applications*. Springer, 2010.

[79] J. Tangelder and R. Veltkamp. A survey of content based 3D shape retrieval methods. *Multimedia Tools and Applications*, 39(3):441–471, 2008.

[80] A. Teichman, J. Levinson, and S. Thrun. Towards 3D object recognition via classification of arbitrary object tracks. In *International Conference on Robotics and Automation*, 2011.

[81] S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic robotics*, volume 1. MIT press Cambridge, 2005.

[82] S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic robotics*, chapter 9. Volume 1 of Thrun et al. [81], 2005.

[83] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318. ACM, 1994.

[84] R. Unnikrishnan and M. Hebert. Multi-scale interest regions from unorganized point clouds. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008.

[85] 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.

[86] *HDL-64E S2 and S2.1 high definition LiDAR sensor*. Velodyne LIDAR, Inc., rev H edition, November 2012.

[87] C. Wang, D. Blei, and F. Li. Simultaneous image classification and annotation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1903–1910. IEEE, 2009.

[88] C. Wang, D. Blei, and L. Fei-Fei. Supervised latent dirichlet allocation for classification. `http://www.cs.cmu.edu/~chongw/slda/`. 2009.

[89] K. Wang, J. Zhang, D. Li, X. Zhang, and T. Guo. Adaptive affinity propagation clustering. *arXiv preprint arXiv:0805.1096*, 2008.

[90] X. Wang, A. Li, Z. Jiang, and H. Feng. Missing value estimation for dna microarray gene expression data by support vector regression imputation and orthogonal coding scheme. *BMC bioinformatics*, 7(1):32, 2006.

[91] D. Williams, X. Liao, Y. Xue, L. Carin, and B. Krishnapuram. On classification with incomplete data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):427–436, 2007.

[92] L. Yang, R. Jin, R. Sukthankar, and F. Jurie. Unifying discriminative visual codebook generation with classifier training for object category recognition. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[93] D. Zhang and S. Chen. Clustering incomplete data using kernel-based fuzzy c-means algorithm. *Neural Processing Letters*, 18(3):155–162, 2003.

[94] G. Zou, J. Hua, M. Dong, and H. Qin. Surface matching with salient keypoints in geodesic scale space. *Computer Animation and Virtual Worlds*, 19(3-4): 399–410, 2008.

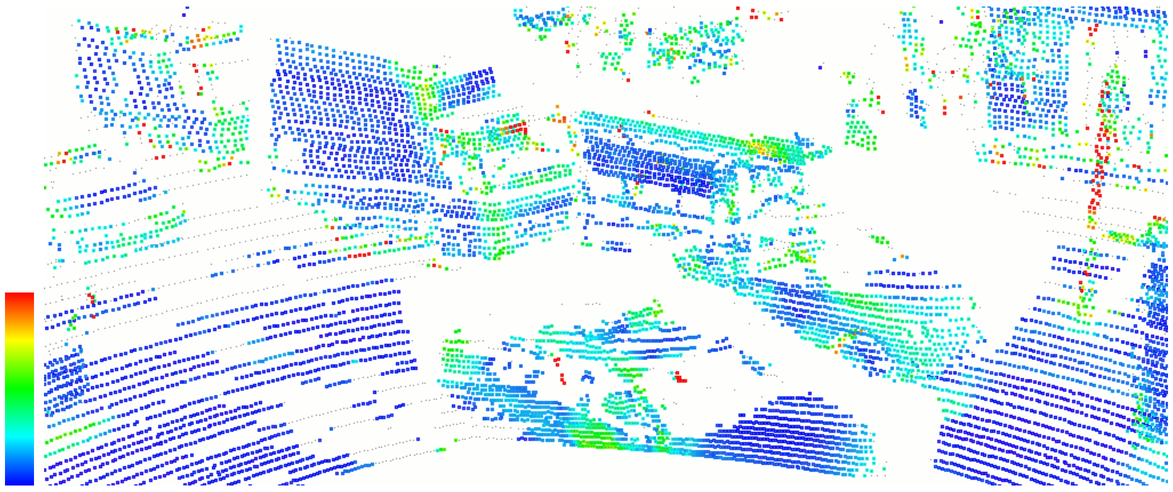
Appendix A

Principal Curvature

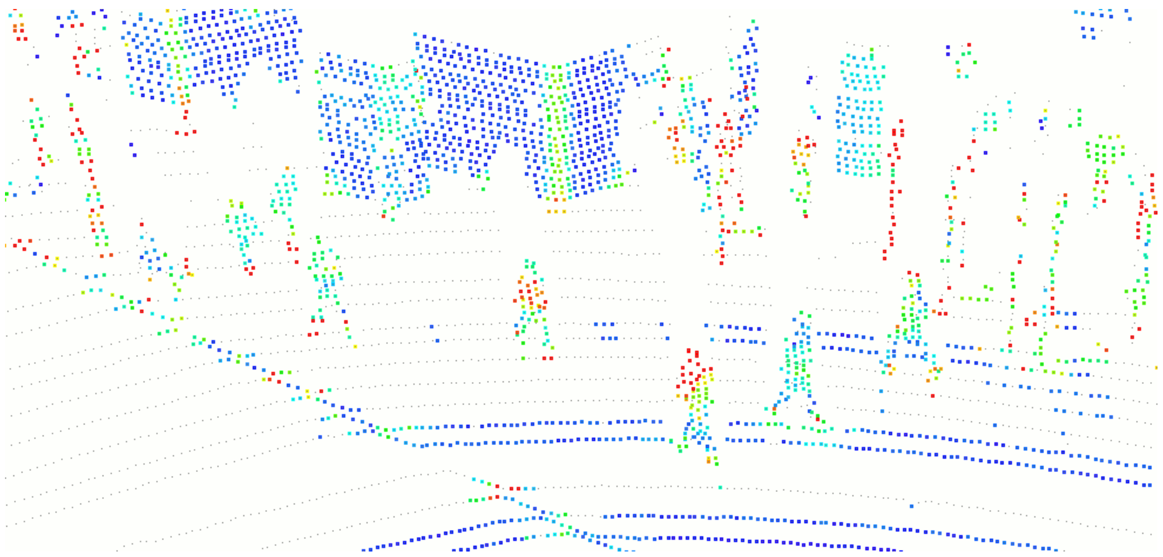
Surface normals can be interpreted as the derivative of the surface, and the second order derivative is called the principal curvature. It consists of the vector p1, tangential to the surface along which the surface curvature is the greatest, and p2, the perpendicular vector along which the curvature is the least (diagram).

The method called *QuadTransSVD* in [46] was used. This involves computing the surface normals with PCA as in Section 3.2.1.2, then tranforming the points to a local frame defined with the z coordinate parallel to the surface normal. A quadratic surface is then fitted to the data. This procedure requires six or more neighbours.

More efficient but noisy approaches are possible using the graph structure to define curvature.



(a) Dense, well sampled regions



(b) Sparser region

Figure A.1 – Principal curvature calculated on a Velodyne scan.

Appendix B

PCA Keypoint Algorithm

The keypoint extraction algorithm for a given Velodyne scan is as follows:

(1) A bearing graph structure is computed, as in [58], shown in Figure 3.2.

(2) Region selection. A spherical region of points (of a given radius) is selected about each point p, using the range image grid method in Section 3.1.3 for speed.

(3) Minimum valid scale filtering. Regions with an insufficient sample of points at this radius are disregarded, as described in Section 3.1.2.2.

(4) PCA is computed on each region. This provides a mean point (the average 3D position of all points in the region), eigenvalues and eigenvectors for each region.

(5) Surfaceness and linearness are computed using the eigenvalues of each region, as in [48], shown in Figure 3.12. Highly flat and linear regions are chosen using two thresholds.

(6) For these regions, the mean point is used as the keypoint position.

(7) The frame orientation is computed, as outlined in Sections 3.3.2.1 and 3.3.2.2.

(8) Final sub-sampling. Many mean points are often in close proximity, so they are subsampled. An example of the resulting keypoints is shown in Figure 3.13.

Appendix C

Additional *k*-NN Results

C.1 Spin Image, 3.25 m

Table C.1 – Confusion matrix for Spin Image, 3.25 m.

Truth \ Inferred	4wd	Building	Bus	Car	Pedestrian	Pillar	Pole	Traffic lights	Traffic sign	Tree	Truck	Trunk	Ute	Van
4wd	1	0	0	20	0	0	0	0	0	0	0	0	0	0
Building	0	20	0	0	0	0	0	0	0	0	0	0	0	0
Bus	0	5	7	1	0	0	0	0	0	0	2	0	1	0
Car	3	0	0	84	1	0	0	0	0	0	0	0	0	0
Pedestrian	0	0	0	0	152	0	0	0	0	0	0	0	0	0
Pillar	0	0	0	2	1	9	0	0	0	0	3	3	0	2
Pole	0	0	0	0	0	0	4	1	1	0	0	15	0	0
Traffic lights	0	0	0	0	7	2	0	27	4	0	0	7	0	0
Traffic sign	0	0	0	0	8	0	0	4	35	0	0	4	0	0
Tree	0	7	0	9	3	0	0	2	0	5	0	8	0	0
Truck	0	5	0	0	0	0	0	0	0	0	5	0	1	1
Trunk	0	3	0	0	2	7	3	0	0	1	0	39	0	0
Ute	0	0	0	12	0	0	0	0	0	0	0	0	2	2
Van	0	1	1	10	0	0	0	0	0	0	1	0	0	22

Table C.2 – Per-class metrics, Spin Image, 3.25 m.

Class	Precision	Recall	F_1	Often confused as
4wd	0.25	0.05	0.08	Car (95%)
Building	0.49	1.0	0.66	
Bus	0.88	0.44	0.58	Building (31%)
Car	0.61	0.95	0.74	
Pedestrian	0.87	1.0	0.93	
Pillar	0.5	0.45	0.47	
Pole	0.57	0.19	0.29	Trunk (71%)
Traffic lights	0.79	0.57	0.67	
Traffic sign	0.88	0.69	0.77	
Tree	0.83	0.15	0.25	Car (26%) Trunk (24%) Building (21%)
Truck	0.45	0.42	0.43	Building (42%)
Trunk	0.51	0.71	0.6	
Ute	0.5	0.13	0.2	Car (75%)
Van	0.81	0.63	0.71	Car (29%)

C.2 FPFH, 4.75 m

Table C.3 – Confusion matrix for FPFH, 4.75 m.

Truth \ Inferred	4wd	Building	Bus	Car	Pedestrian	Pillar	Pole	Traffic lights	Traffic sign	Tree	Truck	Trunk	Ute	Van
4wd	0	0	0	10	1	0	0	0	0	2	2	0	2	4
Building	0	16	3	0	0	0	0	0	0	0	1	0	0	0
Bus	1	3	3	1	1	0	0	1	0	0	2	0	1	3
Car	11	0	1	66	1	1	0	0	0	0	0	0	2	6
Pedestrian	0	0	1	2	120	0	3	12	5	1	0	7	1	0
Pillar	0	6	0	0	0	12	0	1	1	0	0	0	0	0
Pole	0	0	0	0	1	0	8	2	4	0	0	6	0	0
Traffic lights	0	0	1	0	11	0	0	19	7	2	0	5	2	0
Traffic sign	0	2	1	0	1	1	3	8	26	3	0	6	0	0
Tree	0	1	0	0	2	0	0	4	1	24	0	2	0	0
Truck	0	5	0	0	0	0	0	0	0	0	5	0	2	0
Trunk	0	0	0	1	8	0	8	10	5	1	0	21	1	0
Ute	3	0	3	3	0	0	0	2	0	0	3	0	1	1
Van	0	1	4	11	0	0	0	0	0	0	0	0	0	19

Table C.4 – Per-class metrics, FPFH, 4.75 m.

Class	Precision	Recall	F_1	Often confused as
4wd	0.0	0.0	0.0	Car (48%)
Building	0.47	0.8	0.59	
Bus	0.18	0.19	0.18	
Car	0.7	0.75	0.73	
Pedestrian	0.82	0.79	0.81	
Pillar	0.86	0.6	0.71	Building (30%)
Pole	0.36	0.38	0.37	Trunk (29%)
Traffic lights	0.32	0.4	0.36	Pedestrian (23%)
Traffic sign	0.53	0.51	0.52	
Tree	0.73	0.71	0.72	
Truck	0.38	0.42	0.4	Building (42%)
Trunk	0.45	0.38	0.41	
Ute	0.08	0.06	0.07	
Van	0.58	0.54	0.56	Car (31%)

C.3 NARF, PCA Keypoints, 2 m

Table C.5 – Confusion matrix for NARF (PCA keypoints), 2 m.

Truth \ Inferred	4wd	Building	Bus	Car	Pedestrian	Pillar	Pole	Traffic lights	Traffic sign	Tree	Truck	Trunk	Ute	Van
4wd	4	0	0	13	0	0	0	0	0	1	0	0	0	3
Building	0	18	0	0	0	0	0	0	0	0	2	0	0	0
Bus	0	11	1	1	0	0	0	0	0	0	1	0	0	2
Car	6	0	0	72	0	0	0	1	0	0	0	0	3	6
Pedestrian	0	0	0	1	150	0	0	0	1	0	0	0	0	0
Pillar	0	0	0	0	2	11	0	0	0	0	1	6	0	0
Pole	0	0	0	0	2	1	11	1	3	0	0	3	0	0
Traffic lights	0	0	0	0	26	4	1	6	4	0	0	6	0	0
Traffic sign	0	0	0	0	20	1	1	3	22	0	0	4	0	0
Tree	0	0	0	1	2	0	0	2	0	19	0	4	0	6
Truck	0	3	2	4	0	0	0	0	0	0	0	0	0	3
Trunk	0	0	0	0	21	8	3	2	7	0	0	13	1	0
Ute	2	0	0	8	0	0	0	0	0	1	0	0	0	5
Van	3	0	1	10	0	0	0	0	0	1	1	0	1	18

Table C.6 – Per-class metrics, NARF (PCA keypoints), 2 m.

Class	Precision	Recall	F_1	Often confused as
4wd	0.27	0.19	0.22	Car (62%)
Building	0.56	0.9	0.69	
Bus	0.25	0.06	0.1	Building (69%)
Car	0.65	0.82	0.73	
Pedestrian	0.67	0.99	0.8	
Pillar	0.44	0.55	0.49	Trunk (30%)
Pole	0.69	0.52	0.59	
Traffic lights	0.4	0.13	0.19	Pedestrian (55%)
Traffic sign	0.59	0.43	0.5	Pedestrian (39%)
Tree	0.86	0.56	0.68	
Truck	0.0	0.0	0.0	Car (33%) Van (25%) Building (25%)
Trunk	0.36	0.24	0.29	Pedestrian (38%)
Ute	0.0	0.0	0.0	Car (50%) Van (31%)
Van	0.42	0.51	0.46	Car (29%)

C.4 NARF, NARF Keypoints, 2 m

Table C.7 – Confusion matrix for NARF (NARF keypoints), 2 m.

Truth \ Inferred	4wd	Building	Bus	Car	Pedestrian	Pillar	Pole	Traffic lights	Traffic sign	Tree	Truck	Trunk	Ute	Van
4wd	3	0	0	12	0	0	0	0	0	0	0	0	1	5
Building	0	17	2	0	0	0	0	0	0	1	0	0	0	0
Bus	1	9	1	0	1	0	0	0	0	0	0	0	0	4
Car	8	0	0	69	2	0	0	0	0	0	0	0	0	9
Pedestrian	0	0	0	0	143	0	1	1	3	0	0	2	0	0
Pillar	0	0	0	0	5	13	0	1	0	0	0	1	0	0
Pole	0	0	0	0	3	0	6	0	6	0	0	4	0	0
Traffic lights	0	0	0	0	32	0	1	4	5	0	0	5	0	0
Traffic sign	0	0	0	0	15	0	4	2	20	0	0	9	0	0
Tree	0	1	1	1	2	0	0	2	0	17	0	7	0	3
Truck	1	3	1	3	0	0	0	0	0	0	1	0	0	3
Trunk	0	0	1	0	29	4	2	1	3	1	0	14	0	0
Ute	2	0	0	8	0	0	0	0	0	0	0	0	0	6
Van	3	0	3	12	1	0	0	0	0	0	2	0	1	13

Table C.8 – Per-class metrics, NARF (NARF keypoints), 2 m.

Class	Precision	Recall	F_1	Often confused as
4wd	0.17	0.14	0.15	Car (57%) Van (24%)
Building	0.57	0.85	0.68	
Bus	0.11	0.06	0.08	Building (56%) Van (25%)
Car	0.66	0.78	0.72	
Pedestrian	0.61	0.95	0.75	
Pillar	0.76	0.65	0.7	Pedestrian (25%)
Pole	0.43	0.32	0.36	Traffic sign (32%) Trunk (21%)
Traffic lights	0.36	0.09	0.14	Pedestrian (68%)
Traffic sign	0.54	0.4	0.46	Pedestrian (30%)
Tree	0.89	0.5	0.64	Trunk (21%)
Truck	0.33	0.08	0.13	Van (25%) Car (25%) Building (25%)
Trunk	0.33	0.25	0.29	Pedestrian (53%)
Ute	0.0	0.0	0.0	Car (50%) Van (38%)
Van	0.3	0.37	0.33	Car (34%)