Interpreting the Structure of Single Images by Learning from Examples

Osian Haines

A dissertation submitted to the University of Bristol in accordance with the requirements for the degree of Doctor of Philosophy in the Faculty of Engineering, Visual Information Laboratory.

October 2013

52000 words

Abstract

One of the central problems in computer vision is the interpretation of the content of a single image. A particularly interesting example of this is the extraction of the underlying 3D structure apparent in an image, which is especially challenging due to the ambiguity introduced by having no depth information. Nevertheless, knowledge of the regular and predictable nature of the 3D world imposes constraints upon images, which can be used to recover basic structural information.

Our work is inspired by the human visual system, which appears to have little difficulty in interpreting complex scenes from only a single viewpoint. Humans are thought to rely heavily on learned prior knowledge for this. As such we take a machine learning approach, to learn the relationship between appearance and scene structure from training examples.

This thesis investigates this challenging area by focusing on the task of plane detection, which is important since planes are a ubiquitous feature of human-made environments. We develop a new plane detection method, which works by learning from labelled training data, and can find planes and estimate their orientation. This is done from a single image, without relying on explicit geometric information, nor requiring depth.

This is achieved by first introducing a method to identify whether an individual image region is planar or not, and if so to estimate its orientation with respect to the camera. This is done by describing the image region using basic feature descriptors, and classifying against training data. This forms the core of our plane detector, since by applying it repeatedly to overlapping image regions we can estimate plane likelihood across the image, which is used to segment it into individual planar and non-planar regions. We evaluate both these algorithms against known ground truth, giving good results, and compare to prior work.

We also demonstrate an application of this plane detection algorithm, showing how it is useful for visual odometry (localisation of a camera in an unknown environment). This is done by enhancing a planar visual odometry system to detect planes from one frame, thus being able to quickly initialise planes in appropriate locations, avoiding a search over the whole image. This enables rapid extraction of structured maps while exploring, and may increase accuracy over the baseline system.

Declaration

I declare that the work in this dissertation was carried out in accordance with the Regulations of the University of Bristol. The work is original, except where indicated by special reference in the text, and no part of the dissertation has been submitted for any other academic award.

Any views expressed in the dissertation are those of the author and in no way represent those of the University of Bristol.

The dissertation has not been presented to any other University for examination either in the United Kingdom or overseas.

SIGNED: DATE:

Acknowledgements

I would like to begin by thanking my supervisor Andrew Calway, for all his guidance and sage advice over the years, and making sure I got through the PhD. I would also like to thank Neill Campbell and Nishan Canagarajah, for very useful and essential comments, helping to keep me on the right track. My special thanks go to José Martínez-Carranza and Sion Hannuna, who have given me so much help and support throughout the PhD – I would not have been able to do this without them. Thanks also to my former supervisors at Cardiff, Dave Marshall and Paul Rosin, for setting me on the research path in the first place.

I owe a large amount of gratitude to Mam, Dad and Nia, for being supportive throughout the PhD, always being there when needed, for their encouragement and understanding, and not minding all the missed birthdays and events.

I am enormously grateful to my fantastic team of proofreaders, whose insightful comments and invaluable suggestions made this thesis into what it is. They are, in no particular order: David Hanwell, Austin Gregg-Smith, José Martínez-Carranza, Oliver Moolan-Feroze, Toby Perrett, Rob Frampton, John McGonigle, Sion Hannuna, Nia Haines, and Jack Greenhalgh.

I would like to thank all of my friends in the lab, and in Bristol generally, who have made the past five years so enjoyable, and without whose friendship and support the PhD would have been a very different experience. This includes, but is not limited to: Elena, Andrew, Dima, Adeline, Lizzy, Phil, Tom, John, Anthony, Alex, Tim, Kat, Louise, Teesid, and of course all of the proofreaders already mentioned above.

Thanks also the British Machine Vision Association, whose helping hand has shaped the PhD in various ways, including the excellent computer vision summer school, experiences at the BMVC conferences, and essential financial help for attending conferences toward the end of the PhD.

Finally, thanks to Cathryn, for everything.

Publications

The work described in this thesis has been presented in the following publications:

1. Visual mapping using learned structural priors (Haines, Martínez-Carranza and Calway, International Conference on Robotics and Automation 2013) [59]

2. Detecting planes and estimating their orientation from a single image (Haines and Calway, British Machine Vision Conference 2012) [57]

3. Estimating planar structure in single images by learning from examples (Haines and Calway, International Conference on Pattern Recognition Applications and Methods 2012) [58]

4. Recognising planes in a single image (Haines and Calway, submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence)

For Cathryn

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

A key problem in computer vision is the processing, perception and understanding of individual images. This is an area which includes heavily researched tasks such as object recognition, image segmentation, face detection and text recognition, but can also involve trying to interpret the underlying structure of the scene depicted by an image. This is a particularly interesting and challenging problem, and includes tasks such as identifying the 3D relationships of objects, gauging depth without parallax, segmenting an image into structural regions, and even creating 3D models of a scene. However, in part due to the difficulty and ill-posed nature of such tasks, it is an area which – compared to object recognition, for example – has not been so widely studied.

One reason why perceiving structure from one image is difficult is that unlike many tasks in image processing, it must deal with the fact that depth is ambiguous. Short of using laser scanners, structured light or time-of-flight sensors, an individual image will not record absolute depth; and with no parallax information (larger apparent motion in the image for closer objects, in a moving camera or an image pair), it is impossible to distinguish even relative depths. Furthermore, when relying on the ambiguous and potentially low resolution pixel information, it remains difficult even to tell which regions belong to continuous surfaces – an important problem in image segmentation – making extraction of structure or surface information challenging.

Nevertheless, recent work has shown that substantial progress can be made in perceiving 3D structure by exploiting various features of single images: they may be ambiguous, but there is still sufficient information to begin perceiving the structures represented. This involves making use of cues such as vanishing points and rectilinear structure, shading and texture patterns, or relating appearance directly to structure.

Motivated by the initial success of such techniques, and driven by the potential benefits that single-image perception will bring, this thesis focuses on investigating new methods of interpreting the 3D structure of a single image. We believe this is a very interesting task theoretically, since despite the considerable difficulties involved, some kinds of single image structure perception do indeed seem to be possible. Recent developments in this area also highlight that it is of great practical interest, with applications in 3D reconstruction [66, 113], object recognition [65], wide-baseline matching [93], stereopsis [111] and robot navigation [73, 92], amongst others.

Furthermore, this is an interesting topic because of its relationship to biological vision systems, which as we discuss below seem to have little difficulty in interpreting complex structure from single viewpoints. This is important, since it shows that a sufficiently advanced algorithm can make good use of the very rich information available in an image; and suggests that reliable and detailed perception from even a single image must be in principle possible. The means by which biological systems are thought to do this even hint at possible ways of solving the problem, and motivates striving for a more general solution than existing methods. In particular, humans' ability to take very limited information, and by relating it to past visual experiences generate a seemingly complete model of the real world, is something we believe we can learn much from, as we discuss in depth below.

1.1 Perception of Single Images

In general, extracting the 3D structure from a 2D image is a very difficult problem, because there is insufficient information recorded in an image. There is no way of recovering depth information directly from the image pixels, due to the nature of image formation. Assuming a pin-hole camera model, all points along one ray from the camera centre, toward some location in space, will be projected to the same image location (Figure 1.1), which means that from the image there is no way to work backwards to find where on that line the original point was.

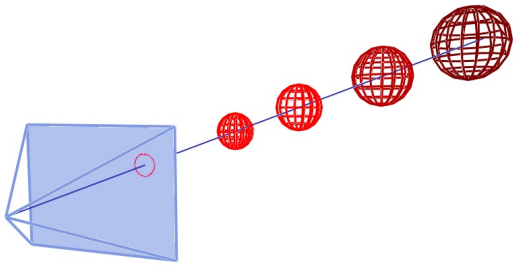
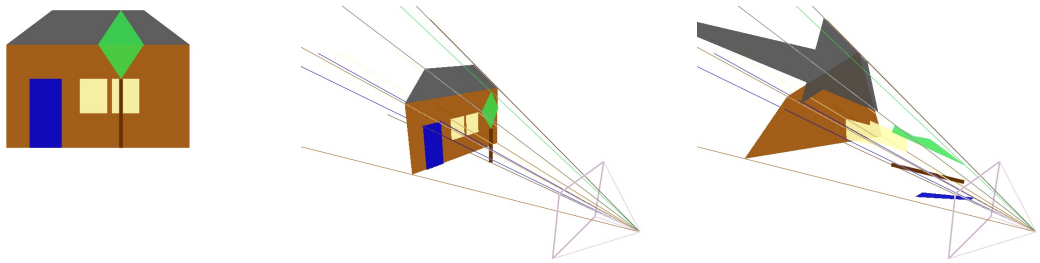


Figure 1.1: *This illustrates the ambiguity of projecting 3D points to a 2D image. The circle in the image plane shows the projection of the red sphere, but any one of the spheres along the ray through the camera centre will project to the same location. We have shown the image plane in front of the camera centre for visual clarity.*

Because of this there could potentially be an infinite number of different 3D scenes which lead to the same 2D image. For example, a configuration of irregular quadrilaterals, appropriately placed and shaded, may falsely give the appearance of a street scene or a building façade, when viewed from a particular vantage point. At the extreme, one may even be looking at a picture of a picture, and have no way of knowing that all depths are in fact equal (this is resolved as soon as multi-view information becomes available).

However, while this argument seems to imply that any extraction of structure without 3D information is not possible, it is evident that although any of a number of 3D configurations are technically valid, some are much more *likely* than others. While we cannot for definite say that we are looking at a building, say, as opposed to a contrived collection of shapes which happen to give that appearance after projection, the former interpretation is much more plausible. We illustrate this in Figure 1.2, where out of two possible 3D configurations for a 2D image, one is much more realistic. Pictures, most of the time, depict actual physical objects which obey physical laws, and tend to be arranged in characteristic ways (especially if human-made).

This observation – that the real world is quite predictable and structured – is what makes any kind of structure perception from a single image possible, by finding the most plausible configuration of the world out of all possible alternatives [133], based on what we can assume about the world in general. Thus the notion of prior knowledge, either explicitly encoded or learned beforehand, becomes essential for making sense of otherwise confusing image data.



(a) (b) (c)

Figure 1.2: *From even this very simple cartoon image of a house (a), it is possible to recognise what it is, and from this imagine a likely 3D configuration (b), despite the fact that, from a single image, the ambiguity of depth means there are an infinite number of possible configurations, such as (c), which project to an identical image. Such contrived collections of shapes are arguably much less likely in the real world.*

The different ways in which generic knowledge about the world is used leads to the various computer vision approaches to detecting structure in single images. For example, parallel world lines are imaged as lines which converge at a point in the image, and can be used to recover the orientation of a plane; and the predictable deformation of texture due to projection is enough to recover some of the viewing information. We go into more detail about these possibilities in the following chapter, but for now it suffices to say that, despite the difficulty of the problem, a number of attempts have been made toward addressing it, with considerable success.

1.2 Motivation and Applications

The task of extracting structure from single images is interesting for several reasons. As stated above, it is a difficult and ill-posed problem, since the information in one image cannot resolve the issue of depths; and yet by exploiting low-level cues or learning about characteristic structures, it is possible to recover some of this lost information. It is also an interesting task to attempt since it has some biological relevance. Because it is something humans have little difficulty with, this suggests ways of addressing the problem, and gives us some baseline with which to ultimately compare. An algorithm attempting to emulate the psychology of vision may also shed some light on unknown details of how this is achieved, if it is sufficiently physiologically accurate, although this is not a route we intend to investigate.

From a practical point of view, the potential of being able to interpret single images would have a number of interesting applications. For example, being able to perceive structure in a single image would allow quick, approximate 3D reconstructions to be created, using only one image as input [4, 66]. For some tasks, a more sophisticated understanding than knowledge of the structures themselves would not be necessary — that is, the geometry alone is sufficient. For example, reconstructing rough 3D models when limited data are available (historical images, for example, or more prosaically to visualise holiday photos), or to extend the range and coverage of multi-view reconstructions [113].

Alternatively, a deeper understanding of the visual elements would make possible a variety of interesting applications. For example, perception of the underlying structural elements or their relationships would be useful in reconstructing 3D models where insufficient data are available to see the intersections of all surfaces or retrieve the depth of all pixels [113]. It would also be useful for providing context for other tasks, such as object recognition, acting as a prior on likely object location in the image [65].

Knowledge of structure would also be very useful to tasks such as mapping and robot navigation. Usually, when exploring a new and unknown environment, all that can be sensed is the location of individual points from different positions, from which a 3D point cloud can be derived. If however there was a way of gleaning knowledge of the structure apparent in the image, this could be used to more quickly create a map of the environment, and build a richer representation. Indeed, knowledge of higher level structure (obtained by means other than single-image perception) has been very useful in simplifying map representations and reducing the computation burden of maintaining the map [47, 88]. We speculate that being able to more quickly derive higher-level structures would bring further benefits, in terms of scene representation and faster initialisation of features. This is something we come back to in Chapter 7.

1.3 Human Vision

As discussed above, interpreting general structure from single images is a significant challenge, and one which is currently far from being solved. On the other hand, we argue that this must in principle be possible, due to the obvious success of human (and other animal) vision systems. To a human observer, it appears immediately obvious when viewing a scene (and crucially, even an *image* of the scene, void of stereo or parallax cues) what it is, with an interpretation of the underlying structure good enough to make

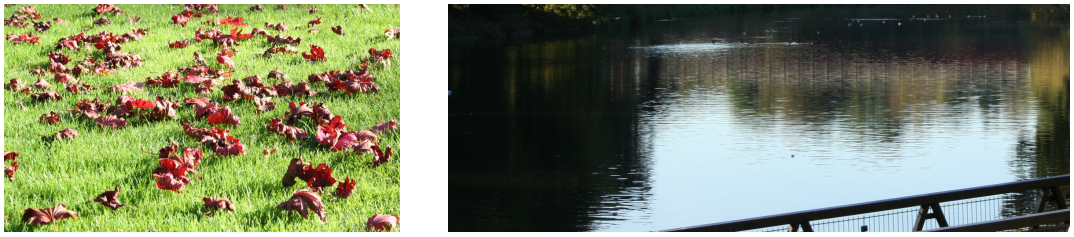
(a) (b) (c)

Figure 1.3: *Examples of images which can be interpreted without difficulty by a human, despite the complex shapes and clutter.*

judgements about relative depths and spatial relationships [51]. Despite the complexity of the task, this even appears subjectively to happen so fast as to be instantaneous [124], and in reality cannot take much more than a fraction of a second.

The human vision system is remarkably adept at interpreting a wide range of types of scene, and this does not appear to depend on calculation from any particular low-level features, such as lines or gradients. While these might seem to be useful cues, humans are quite capable of perceiving more complex scenes where such features are absent or unreliable. Indeed, it is difficult to articulate precisely why one sees a scene as one does, other than simply that it looks like what it is, or looks like something similar seen before. This suggests that an important part of human vision may be the use of learned prior experience to interpret new scenes, which is why complex scenes are so 'obvious' as to what they contain.

For example, consider the image of Figure 1.3a. This is clearly a street scene, comprising a ground plane with two opposing, parallel walls. The geometric structure is quite apparent, in that there are a number of parallel lines (pointing toward a common vanishing point near the centre of the image). It is plausible that this structure is what allows the shape to be so easily perceived, and indeed this has been the foundation of many geometry-based approaches to single-image perception [73, 93]. However, consider Figure 1.3b, which shows a similar configuration of streets and walls, and while it remains obvious to a human what this is, there are considerably fewer converging lines, and the walls show a more irregular appearance.



(a) (b)

Figure 1.4: *These images further highlight the powers of human perception. The grassy ground scattered with leaves can easily be seen as sloping away from the viewer; and the reflections in a river are seen to be of trees, despite the lack of tree-like colour or texture.*

A more extreme example is shown in Figure 1.3c, where the main structures are obscured by balconies and other obtrusions, and people occlude the ground plane. This would be difficult for any algorithm attempting to recover the scene structure based on explicit geometric constructs, or for general image segmentation — and yet the human brain still sees it with ease. This example in particular is suggestive that perception is a top-down process of interpretation, rather than building up from low-level cues, and depends upon a wealth of visual experience for it to make sense.

Finally, we demonstrate how humans can perceive structural content in images with limited or distorted information. In Figure 1.4a, a grassy plane is shown, strewn with fallen leaves. Despite the lack of any other objects for context or scaling, nor a uniform size of leaves (and where cues from foreshortening of texture are quite weak), it is still possible to see the relative orientation of the ground. Another interesting example where the information available in the image itself is ambiguous is in Figure 1.4b, taken from a picture of a riverside. One can see without much difficulty that this depicts trees reflected in water. However, there is very little in the reflections corresponding to the features generally indicative of trees. The branching structure is completely invisible, the overall shape is not particularly tree like, and even the colour is not a reliable cue. As before, this gives the impression that these objects and structures are perceptible by virtue of prior experience with similar scenes, and knowing that trees reflected in water tend to have this kind of distorted shape. The alternative, of recovering shapes from the image, accounting for the distortion produced by the uneven reflections, and matching them to some kind of tree prototype, seems rather unlikely (similar effects are discussed in [51]).

Of course, despite these abilities, a human cannot guess reliable depth estimates nor

report the precise orientation of surfaces in a scene. While they may have the impression that they can visualise a full model of the scene, this is rarely the case, and much of the actual details of structures remain hidden. Humans tend to be good at perceiving the general layout of a scene and its content, without being able to describe its constituent parts in detail (the number of branches on a tree or the relative distances of separated objects, for example). Then again, the fact that such details are not even necessary for a human to understand the gross layout is very interesting, hinting that the ability to describe a scene in such detail is a step too far. This is an important point for an algorithm attempting to follow the success of human vision since it suggests that attempting to fully recover the scene structure or depth may not be necessary for a number of tasks, and a limited perception ability can be sufficient.

These insights into the powers of human vision are what inspire us to take on the challenge of using computer vision techniques to extract structure from single images, and suggests that a learning-based approach is a good way to address this. In the next section we take a deeper look at the details of human perception, focusing on how learning and prior knowledge are thought to play a crucial role.

1.4 On the Psychology of Perception

Having described how the impressive feats of human perception inspire us to build a learning-based system, we now consider how humans are believed to be achieve this. The mechanics of human vision – the optics of the eye, the transmission of visual stimuli to the brain, and so on – are of little interest here, being less germane to our discussion than the interpretation of these signals once they arrive. In this section we give a brief overview of current theories of human perception, focusing on how prior knowledge forms a key part of the way humans so easily perceive the world.

An important figure in the psychology of human perception is James J. Gibson, whose theory of how humans perceive pictures [50] focused on the idea that light rays emanating from either a scene or a picture convey information about its contents, the interpretation of which allows one to reconstruct what is being viewed. As such, vision can be considered a process of picking up information about the world, and actively interpreting it, rather than passive observation. This contrasted strongly with previous theories, which claimed for example that the light rays emanating from a picture, if identical to those from a real object, would give rise to the same perception.

One aspect of his theory is that what humans internally perceive are perceptual and temporal invariants [49], encompassing all views of an object, not just the face currently visible. A consequence is that in order to perceive an object as a whole there must be some model in the mind of what it is likely to look like, since from any particular view, the information about an object is incomplete, and must be supplemented by already acquired internal knowledge to make sense. Thus, vision is a process of applying stored knowledge about the world, in order to make sense of partial information. This certainly implies some kind of learning is necessary for the process, even in cases of totally novel objects.

Similarly Ernst Gombrich, a contemporary of Gibson, likened an image to a trace left behind by some object or event, that must be interpreted to recover its subject [51]. He described this as requiring a "well stocked mind", again clearly implying that learning is important for perception. It follows that no picture can be properly interpreted on its own, as being a collection of related 3D surfaces, without prior knowledge of how such structures generally relate to each other, and how they are typically depicted in images. For example, seeing an image of a building front as anything other than a jumble of shapes is not possible without knowing what such shapes tend to represent.

These theories suggest that even though humans have the impression they perceive a real and detailed description of the world, it is based on incomplete and inaccurate information (due to occlusion, distance, limited acuity and simply being unable to see everything at once), and prior beliefs will fill in the mental gaps. This inference and synthesis is automatic and unconscious, and as Gombrich said, it is almost impossible to see with an 'innocent eye', meaning perceive only what the eyes are receiving, without colouring with subjective interpretations.

1.4.1 Hypotheses and Illusions

These ideas were developed further by Richard Gregory, reacting to what he perceived as Gibson's ignorance of the role of 'perceptual intelligence'. This was defined as knowledge applied to perception, as opposed to 'conceptual intelligence' (the knowledge of specific facts). In this paradigm, perception is analogous to hypothesis generation [53], so that vision can be considered a process of generating perceptual hypotheses (explanations of the world derived from incomplete data received from the eyes), and comparing them to reality. Recognition of objects and scenes equates to the formation of working hypotheses

Figure 1.5: *The hollow face illusion: even when this face mask is viewed from behind (right) the human brain more easily interprets it as being convex. This suggests that prior experience has a strong impact on perception (figure taken from [54]).*

about the world, when the information directly available is insufficient to give a complete description.

This also explains why familiar, predictable objects are easier to see: the mind more readily forms hypotheses to cover things about which it has more prior experience, and requires more evidence to believe the validity of an unusual visual stimulus. One example of this is the hollow face illusion [54], shown in Figure 1.5, where despite evidence from motion and stereo cues, the hollow interior of a face mask is mistakenly perceived as a convex shape. This happens because such a configuration agrees much better with conventional experience — in everyday life, faces tend to be convex. Interestingly, the illusion is much more pronounced when the face is the right way up, further supporting the theory that it is its familiarity as a recognised object which leads to the misinterpretation. This is a striking example of where the patterns and heuristics, learned in order to quickly generate valid hypotheses, can sometimes lead one astray, when presented with unusual data.

This is explored further by Gregory in his discussion of optical illusions [54], phenomena which offer an interesting insight into human perception, and show that people are easily fooled by unexpected stimuli. Not all optical illusions are of this type, for example the Café wall illusion, which is due to neural signals becoming confused by adjacent parallel lines [55]. But many are due to misapplication of general rules, presumably

learned from experience. One such example is the Ames window illusion [54], where the strong expectation for a window shape to be rectangular – when it is actually a skewed quadrilateral – makes it appear to change direction as it rotates.

1.4.2 Human Vision Through Learning

These examples and others strongly suggest that there is an important contribution to human vision from learned prior information, and that one's previous experience with the world is used to make sense of it. Humans cannot see without projecting outward what they expect to see, in order to form hypotheses about the world. This allows quick interpretation of scenes from incomplete and ambiguous data, but also leads to formation of incorrect beliefs when presented with unusual stimuli. In turn this suggests that a successful approach to interpreting information from single images would also benefit from attempting to learn from past experiences, enabling fast hypothesis generation from incomplete data — even if in doing so we are not directly emulating the details of human vision. Indeed, as Gregory states in the introduction to [54], taking advantage of prior knowledge should be crucial to machine vision, and the failure to recognise this may account for the lack of progress to date (as of 1997).

With these insights in mind, we next introduce possibilities for perceiving structure in single images by learning from prior experience. It is important to note, however, that we are not claiming that the above overview is a comprehensive or complete description of human vision. Nor do we claim any further biological relevance for our algorithm. We do not attempt to model ocular or neurological function, but resort to typical image processing and machine learning techniques; we merely assert that the potential learning-based paradigm of human vision is a starting point for thinking of solutions to the problem. Given the success of human vision at arbitrarily difficult tasks, an approach in a similar vein, based on using machine learning to learn from prior visual experience, appears to offer great promise.

1.5 Machine Learning for Image Interpretation

Learning from past experience therefore appears to be a promising and biologically plausible means of interpreting novel images. The use of machine learning is further motivated

by the difficulty of creating models manually for tasks this complex. As previous work has shown (such as [73, 93, 107] as reviewed in Chapter 2), explicitly defining how visual cues can be used to infer structure in general scenes is difficult. For example, shape from texture methods make assumptions on the type of texture being viewed, and from a statistical analysis of visible patterns attempt to directly calculate surface slant and tilt [44, 133]. However, this means that they can work well only in a limited range of scenarios.

Alternatively, when it is difficult to specify the details of the model, but we know what form its inputs and outputs should take, it may be more straightforward to learn the model. Indeed, this is a common approach to complex problems, where describing the system fully is tedious or impossible. Almost all object recognition algorithms, for example, are based on automatically learning what characterises certain objects [69], rather than attempting to explicitly describe the features which can be used to identify and distinguish them. We must proceed with caution, since this still supposes that the world is well behaved, and we should accept that almost any algorithm (including sometimes human vision, as discussed above) would be fooled by sufficiently complicated or contrived configurations of stimuli. Thus the desire to make use of heuristics and ever more general assumptions is tempered by the need to deal with unusual situations.

We are also motivated by other recent works, which use machine learning to deduce the 3D structure of images. This includes the work of Saxena et al. [113], who learn the relationship between depth maps (gathered by a laser range sensor) and images, allowing good pixel-wise depth maps to be estimated for new, previously unseen images. Another example is Hoiem et al. [66], who use classification of image segments into geometric classes, representing different types of structure and their orientation, to build simple 3D interpretations of images. Both these examples, which use quite different types of scene structure, have very practical applications, suggesting that machine learning is a realistic way to tackle the problem — and so next we investigate possible directions to take, with regards what kind of structure perception we intend to achieve.

1.6 Directions

There are several possible ways in which we might explore the perception of structure in a single image. As stated above, one of the primary difficulties of single image interpretation is that no depth information is available. Once depth can be observed, it is

then a fairly simple matter to create a 3D point cloud, from which other reconstruction or segmentation algorithms can work [25, 32]. Therefore one of the most obvious ways to approach single-image perception would be to try to recover the depth. Clearly this cannot be done by geometric means, but as the work of Torralba and Oliva [127] and Sudderth et al. [121] have shown, exploiting familiar objects, or configurations of scene elements, can allow some rudimentary depth estimates to be recovered. This is related to the discussion above, in that this relies on the knowledge that some relationships of depths to image appearance are much more likely than others.

The most sophisticated algorithm to date for perceiving depth from a single image is by Saxena et al. [113], where detailed depth maps can be extracted from images of general outdoor scenes. This work has shown that very good depth estimates can be obtained in the absence of any stereo or multi-view cues (and even used in conjunction with them for better depth map generation). Since this work has succeeded in extracting rather good depth maps, we leave the issue of depth detection, and consider other types of structure.

As we discussed in the context of human vision, it may not be necessary to know detailed depth information in order to recover the shape, and this suggests that we could attempt to recover the shape of general objects without needing to learn the relationship with depth. Indeed, some progress has been made in this direction using shape from shading and texture [41, 98, 122], in which the shape of even deformable objects can be recovered from one image. However, this is a very difficult problem in general, due to the complexities of shapes in the real world, and the fact that lighting and reflectance introduce significant challenges — thus it may be difficult to extend to more general situations.

A simplification of the above is to assume the world is composed of a range of simple shapes – such as cubes and spheres – and attempt to recover them from images. This is an interesting approach, since it relies on the fact that certain kinds of volumetric primitive are likely to appear, and we can insert them into our model based on limited data (i.e. it is not necessary to see all sides of a cuboid). This is the motivation for early works such as Roberts' 'blocks world' [109], in which primitive volumes are used to approximate the structure of a scene; and a more recent development [56] based on first estimating the scene layout [66] and fitting primitives using physical constraints. While this approach limits the resulting model to only fairly simple volumetric primitives, many human-made scenes do tend to consist of such shapes (especially at medium-sized scales), and so a rough approximation to the scene is plausible.

A rather different approach would be to attempt to segment the image, so that segments correspond to the continuous parts of the scene, and by assigning each a semantic label via classification, to begin to perceive the overall structure. Following such a segmentation, we could potentially combine the information from the entire image into a coherent whole, using random fields for example. This bears some similarity to the work of [66], in which the segments are assigned to geometric classes, which are sufficient to recover the overall scene layout. A related idea is to associate elements detected across the image to create a coherent whole using a grammar based interpretation [3, 74], in which the prior information encoded in the grammatical rules enforces the overall structure.

The above possibilities offer potentially powerful ways to get structure from a single image. For practical purposes, in order to make a start at extracting more basic structures, we will consider a somewhat more restricted – though still very general – alternative, based on two of the key ideas above. First, we might further simplify the volume primitives concept to use a smaller class of more basic shapes, which should be easier to find from a single image, at the same time as being able to represent a wider variety of types of scene. The most basic primitive we could use like this are planes. Planar surfaces are very common and so can compactly represent a diverse range of images. If we consider human-made environments – arguably where the majority of computer vision algorithms will see most use – these are often comprised of planar structures. The fact that planes can compactly represent structure means they have seen much use in tasks such as 3D reconstruction and robot navigation.

While shapes such as cubes and spheres may be distinctive due to their shape in an image, the same cannot be said of planes. Planes project simply to a quadrilateral, assuming there is no occlusion; and while quadrilateral shapes have been used to find planes [94] this is limited to fairly simple environments with obvious line structure. It is here that the second idea becomes relevant: that using classification of surfaces can inform us about structure. Specifically, while the shape of planes may not be sufficiently informative, their appearance is often distinctive, due to the way planar structures are constructed and used. For example, building façades and walls have an easily recognisable appearance, and this allows them to be quickly recognised by humans as being planar even if they have no obvious outline. Moreover, the appearance of a plane is related to its orientation with respect to the viewer. Its surface will appear different depending which way it is facing, due to effects such as foreshortening and texture compression, which suggests we should be able to exploit this in order to predict orientation.

For these reasons, planar structures appear to be a good place to begin in order to

extract structure from a single image. Therefore, to investigate single-image perception in a feasible manner, we look at the potential of recovering planar structures in images of urban scenes, by learning from their appearance and how this relates to structure.

1.7 Plane Detection

Planes are important structures in computer vision, partly because they are amongst the simplest possible objects and are easy to represent geometrically. A 3D plane needs only four parameters to completely specify it, usually expressed as a normal vector in $\mathbb{R}^3$ and a distance (plus a few more to describe spatial extent if appropriate). Because of this, they are often easy to extract from data. Only three 3D points are required to hypothesise a plane in a set of points [5], enabling planar structure recovery from even reasonably sparse point clouds [46]. Alternatively, a minimum of only four point correspondences between two images are required to create a homography [62], which defines a planar relationship between two images, making plane detection possible from pairs [42] or sequences [136] of images.

Planes are also important because they are ubiquitous in urban environments, making up a significant portion of both indoor and outdoor urban scenes. Their status as one of the most basic geometric primitives makes them a defining feature of human-made environments, and therefore an efficient and compact way of representing structure, giving realistic and semantically meaningful models while remaining simple. As such, they have been shown to be very useful in tasks including wide baseline matching [73], object recognition [65], 3D mesh reconstruction [25], robot navigation [47] and augmented reality [16].

To investigate the potential of detecting and using planes from single image information, we have developed an algorithm capable of detecting planar structures in single images, and estimating their orientation with respect to the camera. This uses general feature descriptors and standard methods for image representation, combined with a classifier and regressor to learn from a large set of training data. One of the key points about our method is that it does not depend upon any particular geometric or textural feature (such as vanishing points and image gradients), and so is not constrained to a particular type of scene. Rather, it exploits the fact that appearance of an image is related to scene structure, and that learning from relevant cues in a set of training images can be used to interpret new images. We show how this can be used in a variety of environments,

and that it is useful for a visual odometry task.

The work can be divided into three main sections: developing the basic algorithm to represent image regions, recognise planes and estimate orientation; using this for detecting planes in an image, giving groupings of image points into planar and non-planar regions; and an example application, showing how plane detection can provide useful prior information for a plane-based visual mapping system.

1.8 Thesis Overview

After this introductory chapter concludes by summarising our main contributions, Chapter 2 presents some background to this area and reviews related work, including details of geometric, texture, and learning-based approaches to single image structure perception.

In Chapter 3 we introduce our method for **plane recognition**, which can identify planes from non-planes and estimate their orientation, example results of which are shown in Figure 1.6. This chapter describes how regions of an image are represented, using a collection of feature descriptors and a visual bag of words model, enhanced with spatial information. We gather and annotate a large training set of examples, and use this to train a classifier, so that new image regions may be recognised as planar or not; then, for those which are planar, their orientation can be estimated by training a regression algorithm. The important point about this chapter is that it deals only with known, pre-specified image regions. This algorithm cannot find planes in an image, but can identify them assuming a region of interest has been defined.

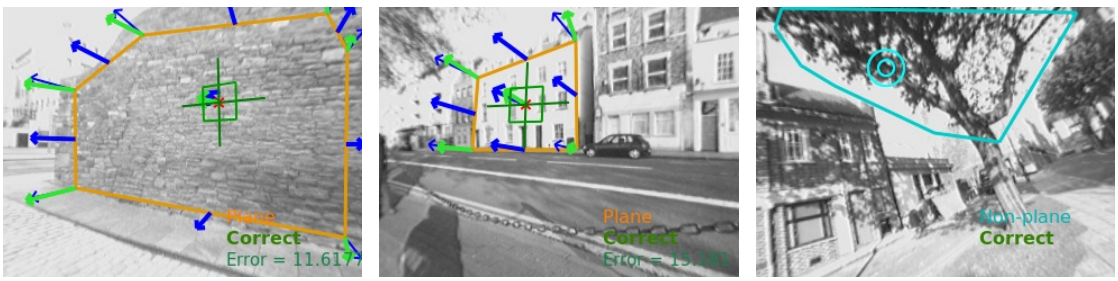


Figure 1.6: *Examples of our plane recognition algorithm, which for manually delineated regions such as these, can identify whether they are planar or not, and estimate their orientation with respect to the camera.*

Chapter 4 thoroughly evaluates the plane recognition algorithm, investigating the effects of feature representation, vocabulary size, use of synthetic data, and other design choices and parameters, by running cross-validation on our training set. We also experiment with an alternative classification technique, leading to useful insights into how and why the algorithm works. We evaluate the algorithm against an independent test set of images, showing it can generalise to a new environment.

The plane recognition algorithm is adapted for use as part of a full **plane detection** algorithm in Chapter 5, which is able to find the planes themselves, in a previously unseen image; and for these detected planes, estimate their orientation. Since the boundaries between planes are unknown, the plane recognition algorithm is applied repeatedly across the image, to find the most likely locations of planes. This allows planar regions to be segmented from each other, as well as separating planes with different orientations. The result is an algorithm capable of detecting multiple planes from a single image, each with an orientation estimate, using no multi-view or depth information nor explicit geometric features: this is the primary novel contribution of this thesis. Example results can be seen in Figure 1.7.

Figure 1.7: *Examples of our plane detection algorithm, in a variety of environments, showing how it can find planes from amongst non-planes, and estimate their orientation.*

We then evaluate this detection algorithm in Chapter 6, showing the results of experiments to investigate the effect of parameters such as the size of regions used for the recognition step, or sensitivity of plane segmentation to different orientations. Such experiments are used to select the optimal parameters, before again testing our algorithm on an independent, ground-truth-labelled test set, showing good results for plane detection in urban environments. We also compare our algorithm to similar work, showing side-by-side comparison for plane detection. Our method compares favourably, showing superior performance in some situations and higher accuracy overall.

Finally, Chapter 7 presents an example application of the plane detection algorithm.

Planar structures have been shown to be very useful in tasks such as simultaneous localisation and mapping and visual odometry, for simplifying the map representation and producing higher-level, easier to interpret scene representations [47, 89, 132]. We show that our plane detector can be beneficial in a plane-based visual odometry task, by giving prior information on the location and orientation of planes. This means planes may be initialised quickly and accurately, before even the accumulation of parallax necessary for multi-view methods. We show that this enables fast building of structured maps of urban environments, and may improve the accuracy by reducing drift. We end with Chapter 8, which concludes the thesis with a summary of the work and a discussion of possible future directions.

1.9 Contributions

To summarise, these are the main contributions of this thesis:

- We introduce a method for determining whether individual regions of an image are planar or not, according to their appearance (represented with a variety of basic descriptors), based on learning from training data.
- We show that it is possible, for these planar regions, to estimate their 3D orientation with respect to the viewer, based only on their appearance in one image.
- The plane recognition algorithm can be used as part of a plane detection algorithm, which can recover the location and extent of planar structure in one image.
- This plane detection algorithm can estimate the orientation of (multiple) planes in an image — showing that there is an important link between appearance and structure, and that this is sufficient to recover the gross scene structures.
- We show that both these methods work well in a variety of scenes, and investigate the effects of various parameters on the algorithms' performance.
- Our method is shown to perform similarly to a state of the art method on our dataset.
- We demonstrate an example application of this plane detection method, by applying it to monocular visual odometry, which as discussed above could benefit from being able to quickly see structure without requiring parallax.

CHAPTER 2

Background

In this chapter we discuss related works on single image perception, focusing on those which consider the problem of detecting planar structure, or estimating surface orientations. These can be broadly divided into two main categories: firstly, those which directly use the geometric or textural properties of the image to infer structure. These may be further divided into those which calculate directly from visible geometric entities, and those that use a statistical approach based on image features. Secondly, there exist methods which use machine learning, to learn the relationship between appearance and scene structure.

As we discussed in Chapter 1, even when only a single image is available it is usually possible to infer something about the 3D structure it represents, by considering what is most likely, given the information available. Various methods have been able to glean information about surface orientation, relative depth, curvature or material properties, for example, even though none of these are measurable from image pixels themselves. This is due to a number of cues present in images, especially those of human-made, regular environments, such as vanishing points, rectilinear structure, spectral properties, colours and textures, lines and edges, gradient and shading, and previously learned objects.

The use of such cues amounts to deliberately using prior knowledge applied to the scene. Here prior knowledge means information that is assumed to be true in general, and can be used to extrapolate beyond what is directly observed. This can be contrasted with specific scene knowledge [2, 102], which can also help to recover structure when limited or incomplete observations are available. However, this is limited to working in specific locations.

Over the following sections we review work on single image perception which makes use of a variety of prior knowledge, represented in different ways and corresponding to different types of assumption about the viewed scene. These are organised by the way they make assumptions about the image, and which properties of 3D scenes and 2D projections they exploit in order to achieve reconstruction or image understanding.

2.1 Vanishing Points

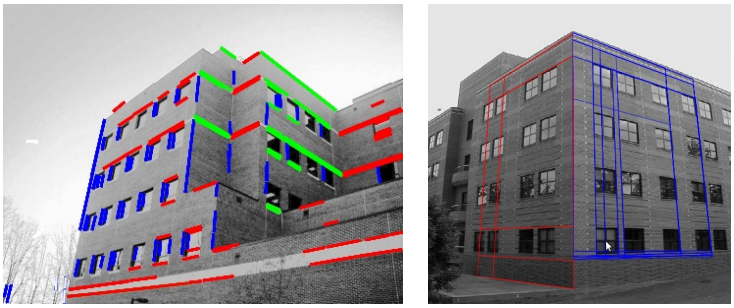
Vanishing points are defined as the points in the image where parallel lines appear to meet. They lie on the plane at infinity, a special construct of projective space that is invariant to translations of the camera, thus putting constraints on the geometry of the scene. Vanishing points are especially useful in urban, human-made scenes, in which pairs of parallel lines are ubiquitous, and this means that simple methods can often be used to recover 3D information. A detailed explanation of vanishing points and the plane at infinity can be found in chapters 3 and 6 of [62].

A powerful demonstration of how vanishing points and vanishing lines can be used is by Criminisi et al. [26], who describe how the vanishing line of a plane and a vanishing point in a single image can be used to make measurements of relative lengths and areas. If just one absolute distance measurement is known, this can be used to calculate other distances, which is useful in measuring the height of people for forensic investigations, for example. As the authors state, such geometric configurations are readily available in structured scenes, though they do not discuss how such information could be extracted automatically.

Another example of using vanishing points, where they are automatically detected, is the work of Košecká and Zhang [73], in which the dominant planar surfaces of a scene are recovered, with the aim of using them to match between widely separated images. The method is based on the assumption that there are three primary mutually orthogonal

directions present in the scene, i.e. this is a 'Manhattan-like' environment, having a ground plane and mutually perpendicular walls. Assuming that there are indeed three dominant orientations, the task is thus to find the three vanishing points of the image. This is done by using the intersections of detected line segments to vote for vanishing points in a Hough accumulator. However, due to noise and clutter (lines which do not correspond to any of the orientations), this is not straightforward. The solution is to use expectation maximisation (EM) to simultaneously estimate the location of the vanishing points in the image, and the probability of each line belonging to each vanishing point; the assignment of lines to vanishing points updates the vanishing points' positions, which in turn alters the assignment, and this iterates until convergence. An example result, where viable line segments are coloured according to their assigned vanishing direction, is shown in Figure 2.1a.

To find actual rectangular surfaces from this, two pairs of lines, corresponding to two different vanishing points, are used to hypothesise a rectangle (a quadrilateral in the image). These are verified using the distribution of gradient orientations within the image region, which should contain two separate dominant orientations. An example of a set of 3D rectangles detected in one image is shown in Figure 2.1b. These can subsequently be used for wide-baseline matching to another image.



(a) **(b)**

Figure 2.1: *Illustration of the method of Košecká and Zhang [73], showing the lines coloured by their assignment to the three orthogonal vanishing directions (a), and a resulting set of planar rectangles found on an image (b). Images are adapted from [73].*

This method has shown promising results, in both indoor and outdoor scenes, and the authors mention that it would be useful for robot navigation applications. Its main downside, however, is that it is limited to scenes with this kind of dominant rectangular structure. As such, planes which are perpendicular to the ground, but are oriented differently from the rest of the planes, would not be easily detected, and may cause problems for the EM algorithm. Furthermore, the method relies on there being sufficiently many

good lines which can be extracted and survive the clustering, which may not be the case when there is background clutter or spurious edges produced by textured scenes.

A related method is presented by Mičušík et al. [93], where line segments are used to directly infer rectangular shapes, which in turn inform the structure of the scene. It differs from [73] in that it avoids the high computational complexity of exhaustively considering all line pairs to hypothesise rectangles. Rectangle detection is treated as a labelling problem on the set of suitable lines, using a Markov random field. By using two orthogonal directions at a time, labels are assigned to each line segment to indicate the vanishing direction to which it points, and which part of a rectangle it is (for example the left or right side); from this, individual rectangles can be effortlessly extracted.

These methods highlight an important issue when using vanishing points and other monocular cues. While they are very useful for interpreting a single image, they can also provide useful information when multiple images are present, by using single image cues to help in wide-baseline matching for example. Similar conclusions are drawn in work we review below [111], where a primarily single-image method is shown to be beneficial for stereo vision by exploiting complementary information.

An alternative way of using vanishing points is to consider the effect that sets of converging parallel lines will have on the spectral properties of the image. For example, rather than detecting lines directly in the spatial domain, Ribeiro and Hancock [107] use the power spectrum of the image, where spectra which are strongly peaked describe a linear structure. From this, properties of the texture and the underlying surface can be recovered, such as its orientation. In their later work [108], spectral moments of the image's Fourier transform are used to find vanishing lines via a Hough-like accumulator. Such methods are dependent on the texture being isotropic and homogeneous, such that any observed distortions are due to the effects of projection rather than the patterns themselves. These restrictions are quite limiting, and so only a subset of scenes – namely those with regular grid-like structures, such as brick walls and tiled roofs – are appropriate for use with this method.

2.2 Shape from Texture

An alternative to using vanishing points, which has received much attention in the literature, is known as shape from texture (this is part of a loose collection of methods known

collectively as 'shape-from-X', which includes recovery of 3D information from features such as shading, defocus and zoom [35, 75, 116]). Such an approach is appealing, since it makes use of quite different types of information from the rectangle-based methods above. In those, texture is usually an inconvenience, whereas many real environments have multiple textured surfaces, especially outdoors.

Early work on shape from texture was motivated by the insights of Gibson into human vision [48], specifically that humans extract information about surface orientation from apparent gradients. However, this model was not shown to be reliable for real textures [133]; and due to making necessary assumptions about the homogeneity and isotropy of texture (conditions that, while unrealistic, allow surface orientation to be computed directly) methods developed based on these ideas fall short of being able to recover structure from real images.

Work by Witkin [133] allows some of these assumptions to be relaxed in order to better portray real-world scenes. Rather than requiring the textures to be homogeneous or uniform, the only constraint is that the way in which textures appear non-uniform does not mimic perspective projection — which is in general true, though of course there will be exceptional cases. Witkin's approach is based on the understanding that for one image there will be a potentially infinite number of plausible reconstructions due to the rules of projective geometry, and so the task is to find the 'best', or most likely, from amongst all possible alternatives. This goal can be expressed in a maximum-likelihood framework.

The method works by representing texture as a distribution of short line segments, and assumes that for a general surface texture their orientations will be distributed uniformly. When projected, however, the orientations will change, aligning with the axis of tilt, so there is a direct relationship between the angular distribution of line segments (which can be measured), and the slant and tilt of the image. An iterative method is used to find the most likely surface orientation given the observed angles, in a maximum likelihood approach. This is developed by Gårding [44], who proposes a fast and efficient one-step method to estimate the orientation directly from the observations, with comparable results. An example of applying the latter method is shown in Figure 2.2, which illustrates the estimated orientation at the four image quadrants.

However, shape from texture techniques such as these again face the problem that textural cues can be ambiguous or misleading, and an incorrect slant and tilt would be estimated for textures with persistent elongated features, for example rock fissures or

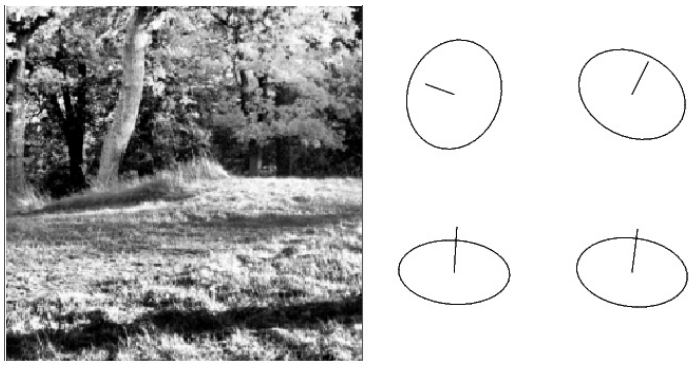


Figure 2.2: *An example result from the shape from texture algorithm of Gårding [44], applied to an image of an outdoor scene. Orientation is approximately recovered for the four quadrants of the image (there is no notion of plane detection here, only slant/tilt estimation). The image was adapted from [44].*

fence-posts. Moreover, while the accuracy of [44] was shown to be good for simulated textures of known orientation, no ground truth was available for evaluation on real images. While the presented results are qualitatively good, it was not possible to accurately assess its applicability to real images.

A further issue with shape from texture methods is that generally they deal only with estimating the orientation of given regions (as in the image above), or even of the whole image; the detection of appropriate planar surfaces, and their segmentation from non-planar regions, is not addressed, to our knowledge.

2.3 Learning from Images

The approaches discussed above use a well defined geometric model, relating features of appearance (vanishing points, texture, and so on) to 3D structure. They have proven to be successful in cases where the model is valid – where such structure is visible and the key assumptions hold – but are not applicable more generally. This is because it is very difficult to explicitly specify a general model to interpret 3D scenes, and so a natural alternative is to learn the model from known data. This leads us on to the next important class of methods, to which ours belongs: those which use techniques from machine learning to solve the problem of single-image perception.

An interesting approach is presented in a set of papers by Torralba and Oliva, where general properties are extracted from images in order to describe various aspects of

the scene [99, 100, 127, 128]. They introduce the concept of the 'scene envelope' [99], which combines various descriptions of images, such as whether they are natural or artificial locations, depict close or far structures, or are of indoor or outdoor scenes. By estimating where along each of these axes an image falls a qualitative description can be generated automatically (for example 'Flat view of a man-made urban environment, vertically structured' [99]). This is achieved by representing each image by its 2D Fourier transform, motivated by the tendency for the coarse spectral properties of the image to depend upon the type and configuration of the scene. An indoor scene, for example, will produce sharp vertical and horizontal features, which is evident when viewed in the frequency domain.

This work is further developed for scene depth estimation [127], where the overall depth of an image is predicted. This is based on an important observation, that while it is possible for an object to be at any size and at any depth, there are certain characteristic sizes for objects, which can be taken advantage of (a concept illustrated by Figure 2.3). For example, buildings are much more likely to be large and distant than small and close (as in a toy model); conversely something that looks like a desktop is very unlikely to be scenery several kilometres away. Again, this relates to the key point that we desire the most likely interpretation. Like the methods mentioned above, this method is based on the frequency characteristics of the image. Image properties are represented using local and global wavelet energy, with which depth is estimated by learning a parametric model using expectation maximisation. While this is a very interesting approach to recovering depth, the main downside is that it gives only the overall depth of the image. There is no distinction between near or far objects within one image, so it cannot be used to infer any detailed scene structure.

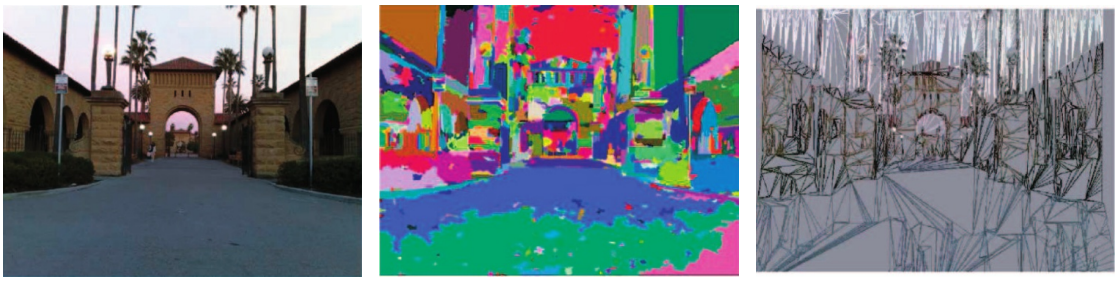
Figure 2.3: *An illustration from Torralba and Oliva [127], making the point that some objects tend to only appear at characteristic scales, which means they can be used to recover approximate scene depth (image taken from [127]).*

2.3.1 Learning Depth Maps

A more sophisticated approach by Saxena et al. [113] estimates whole-image depth maps for single images of general outdoor scenes. This is based on learning the relationship between image features and depth, using training sets consisting of images with associated ground truth depth maps acquired using a custom built laser scanner unit. The central premise is that by encoding image elements using a range of descriptors at multiple scales, estimates of depth can be obtained.

In more detail, they first segment the image into superpixels (see Figure 2.4b), being local homogeneous clusters of pixels, for which a battery of features are computed (including local texture features and shape features). As well as using features from individual superpixels, they combine features from neighbouring superpixels in order to encode contextual information, because the orientation or depth of individual locations is strongly influenced by that of their surroundings. In addition to these features, they extract features to represent edge information to estimate the location of occlusion boundaries.

These features allow them to estimate both relative and absolute depth, as well as local orientation. The latter is useful in evaluating whether there should be a boundary between adjacent superpixels. This is interesting, since it implies they are exploiting planarity — indeed, they make the assumption that each superpixel can be considered locally planar. This is a reasonable assumption when they are small. They justify this by analogy with the meshes used in computer graphics, where complex shapes are built from a tessellation of triangles. They explicitly build such meshes using the depth and orientation estimates, an example of which is shown in Figure 2.4c.



(a) **(b)** **(c)**

Figure 2.4: *The depth map estimation algorithm of Saxena et al. [113] takes a single image (a) as input, and begins by segmenting to superpixels (b). By estimating the orientation and depth of each locally planar facet, they produce a 3D mesh representation (c). Images are taken from [113].*

Figure 2.5: *This shows some typical outputs of depth maps (bottom row) estimated by Saxena et al. [113] from single images (top row), where yellow is the closest and cyan the farthest. These examples showcase the ability of the algorithm to recover the overall structure of the scene with good accuracy (images taken from Saxena et al. [113]).*

Following the success of human vision, which easily integrates information from multiple cues over the whole image, they attempt to relate information at one location to all other locations. This is done by formulating the depth estimation problem in a probabilistic model, using a Markov random field, with the aim of capturing connected, coplanar and collinear structures, and to combine all the information to get a consistent overall depth map. Example results of the algorithm are shown in Figure 2.5.

This has a number of interesting applications, such as using the resulting depth map to create simple, partially realistic single-image reconstructions. These are sufficient to create virtual fly-throughs, by using the connected mesh of locally planar regions, and these can semi-realistically simulate a scene from a photograph from new angles. Such reconstructions were shown to rival even those of Hoiem et al. [64] discussed below. Interestingly, while estimating the orientation of the planar facets is a necessary step of the algorithm, to relate the depth of adjacent segments, they do not actually produce a set of planes. Instead they focus on polygonal mesh representations, and while it might be possible to attempt to extract planar surfaces from such models, their results hint that it would not be trivial to do so.

They also show that their depth estimates can be combined for creating multi-view reconstructions from images which are normally too widely separated for this to be possible (with too little overlap to allow reliable reconstruction using traditional structure-from-motion, for example). In [111] this algorithm is also shown to be beneficial for stereo

vision, as another means of estimating depth. This is interesting since even when two images are available, from which depth can be calculated directly, a single-image depth estimation still provides valuable additional information, by exploiting complementary cues. A simplified version of the algorithm was even used to guide an autonomous vehicle over unknown terrain [92]. These last two illustrate the value of using perception from single images in multi-view scenarios, an idea we come back to in Chapter 7.

2.3.2 Geometric Classification

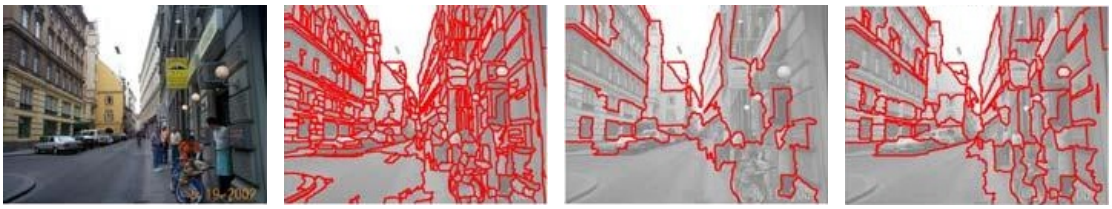
The work of Hoiem et al. [66] has the goal of interpreting the overall scene layout from a single image. This is motivated by the observation that in a great many images, almost all pixels correspond to either the ground, the sky, or some kind of upright surface, and so if these three classes can be distinguished a large portion of images can be described in terms of their rough geometry. This notion of 'geometric classification' is the core of their method, in which regions of the image are assigned to one of three main classes — namely support (ground), sky, and vertical, of which the latter is further subdivided into left, right, and forward facing planes, or otherwise porous or solid.

While the method is not explicitly aimed at plane detection, it is an implicit part of understanding the general structure of scenes, as the image is being effectively partitioned into planar (ground, left, right, forward) and non-planar (sky, solid, porous) regions. It is important to note, however, that plane orientation is limited by quantisation into four discrete classes. No finer resolution on surface orientation can be obtained than 'left' and so on.

Classification is achieved using a large variety of features, including colour (summary statistics and histograms), filter bank responses to represent texture, and image location. Larger-scale features are also used, including line intersections, shape information (such as area of segments), and even vanishing point information. These cues are used in the various stages of classification, using boosted decision trees, where the logistic regression version of Adaboost chosen for the weak learners is able to select the most informative features from those available. This automatic selection out of all possible features is one of the interesting properties of the method, in that it is not even necessary to tell it what to learn from — although this is at the expense of extracting many potentially redundant feature descriptors. The classifiers are trained on manually segmented training data, where segments have been labelled with their true class.

The central problem is that in an unknown image, it is not known how to group the pixels, and so more complex features than simple local statistics cannot be extracted. Thus features such as vanishing points cannot be included until initial segments have been hypothesised, whose fidelity in turn depends upon such features. Their solution is to gradually build up support, from the level of pixels to superpixels (over-segmented image regions) to segments (groups of superpixels), which are combined in order to create a putative segmentation of the image. An initial estimate of structure (a particular grouping of superpixels) is used to extract features, which are then used in order to update the classifications, to create a better representation of the structure and a coherent labelling of all the superpixels.

The superpixels are found by using Felzenszwalb and Huttenlocher's graph-cut segmentation method [36] to separate the image into a number (usually around 500) of small, approximately homogeneous regions, from which local features can be extracted. These are the atomic elements, and are much more convenient to work with than pixels themselves. Segments are formed by grouping these superpixels together, combining information probabilistically from multiple segmentations of different granularity (see Figure 2.6). Since it is not feasible to try all possible segmentations of superpixels, they sample a smaller representative set.



(a) **(b)** **(c)** **(d)**

Figure 2.6: *For some image (a), this shows the superpixels extracted (b), and two segmentations at different granularities (c,d, with 15 and 80 segments respectively). Images adapted from [66].*

There are two main steps in evaluating whether a segment is good and should be retained. First, they use a pairwise-likelihood classification on pairs of adjacent superpixels, which after being trained on ground truth data is able to estimate the probability of them having the same label. This provides evidence as to whether the two should belong in the same eventual segment, or straddle a boundary. Next there is an estimate of segment homogeneity, which is computed using all the superpixels assigned to a segment, and is in turn used to estimate the likelihood of its label. To get the label likelihoods for individual superpixels, they marginalise over all sampled segments in which the superpixel lies.

They show how these likelihoods can be combined in various ways, from a single max-margin estimate of labels, to more complex models involving a Markov random field or simulated annealing. The final result is a segmentation of the image into homogeneously labelled sets of superpixels, each with a classification to one of the three main labels, and a sub-classification into the vertical sub-classes where appropriate. Some examples from their results are shown in Figure 2.7.

Figure 2.7: *Examples outputs of Hoiem et al. [66], for given input images. The red, green and blue regions denote respectively vertical, ground and sky classes. The vertical segments are labelled with symbols, showing the orientation (arrows) or non-planar (circle and cross) subclass to which they are assigned. Images adapted from [66].*

The resulting coarse scene layout can help in a number of tasks. It can enable simple 3D reconstruction of a scene from one image [64], termed a 'pop-up' representation, based on folding the scene along what are estimated to be the ground-vertical intersections. Its use in 3D reconstruction is taken further by Gupta and Efros [56], who use the scene layout as the first step in a blocks-world-like approach to recovering volumetric descriptions of outdoor scenes.

Scene layout estimation is also used as a cue for object recognition [65], because knowing the general layout of a scene gives a helpful indication of where various objects are most likely to appear, which saves time during detection. For example, detecting the location and orientation of a road in a street scene helps predict the location and scale of pedestrians (i.e. connected to the road and around two metres in height), thus discarding a large range of useless search locations. This is interesting as it shows how prior scene knowledge can be beneficial beyond reconstructing a scene model.

However, this method has a few shortcomings. Firstly from an algorithmic standpoint, its sampling approach to finding the best segmentation is not deterministic, and so quite different 3D structures will be obtained for arbitrarily similar images. The iterative nature of the segmentation, in which the features are extracted from a structure estimate which is not yet optimal, might also be problematic, for example falling into local minima where the true segmentation cannot be found because the most appropriate cues cannot

be exploited.

In terms of scene understanding, there are a few more drawbacks. The method rests on the not unreasonable assumption that the camera is level (i.e. there is no roll), which for most photographs is true, although cannot be guaranteed in a robot navigation application, for example (indeed, such images were manually removed from their dataset before processing). Moreover, the horizon needs to be somewhere within the image, which again may be rather limiting. While the assumptions it makes are much less restrictive than those of, say, shape from texture, it is still limited in that the scene needs to be well represented by the discrete set of classes available.

In terms of plane detection, the quantisation of orientation (left-facing, right-facing and so on) is the biggest downside, since it limits its ability to distinguish planes from one another if they are similar in orientation, and would give ambiguous results for planes facing obliquely to the camera; as well as the obvious limit to accuracy attainable for any individual plane. Thus while the algorithm shows impressive performance in a range of images, extracting structure from arbitrarily placed cameras could be a problem. Nevertheless, the fact that a coarse representation gives a very good sense of scene structure, and is useful for the variety of tasks mentioned above, is reassuring, and paves the way for other learning-based single-image perception techniques.

2.4 Summary

This chapter has reviewed a range of methods for extracting the structure from single images, giving examples of methods using either direct calculation from image features or inference via machine learning. While these have been successful to an extent, a number of shortcomings remain. As we have discussed, those which rely on texture gradients or vanishing points are not applicable to general scenes where such structures are not present. Indeed, as Gregory [54] suggested, their shortcomings arise because they fail to take into account the contribution of learned prior knowledge to vision, and thus are unable to deal with novel or unexpected situations.

It is encouraging therefore that several methods using machine learning have been developed; however, these also have a few problems. Torralba and Oliva et al.'s work focuses on global scene properties, which is a level of understanding too coarse for most interesting applications. Saxena et al. successfully show that depth maps can be extracted

from single images (so long as comprehensive and accurate ground truth is available for training), then used to build rough 3D models. While this does reflect the underlying scene structure, it does not explicitly find higher-level structures with which the scene can be represented, nor determine what kind of structure the superpixels represent. We emphasise that this work has made significant progress in terms of interpreting single images, but must conclude that it does not wholly address the central issue with which this thesis is concerned — namely, the interpretation and understanding of scenes from one image. That is, they can estimate depth reliably, but do not distinguish different types of scene element, nor divide structures as to their identity (the mesh represents the whole scene, with no knowledge of planes or other objects). By contrast, the plane detection algorithm we present distinguishes planar and non-planar regions, and though it does not classify planes according to what they actually are, is a first step toward understanding the different structures that make up the scene, potentially enabling more well-informed augmentation or interaction.

Hoiem et al. on the other hand focus almost exclusively on classifying parts of the image into geometric classes, bridging the gap between semantic understanding and 3D reconstruction, and combining techniques from both machine learning and single view metrology. However, because orientations are coarsely quantised it means that the recovered 3D models lack specificity, being unable to distinguish similarly oriented planes which fall into the same category; and any reconstruction is ultimately limited to the fidelity of the initial superpixel extraction. This limitation is not only a practical inconvenience, but suggests that the available prior knowledge contained in the training set could be exploited more thoroughly. Moreover, their requirements that the camera be roughly aligned with the ground plane, and the use of vanishing point information as a cue, suggest they are not making use of fully general information. In particular, they tend to need a fairly stable kind of environment, with a visible and horizontal ground plane at the base of the image. While this is a common type of scene, what we are aiming for is a more general treatment of prior information.

On the other hand, its ability to cope with cartoon images and even paintings show it is much more flexible than typical single-image methods, being able to extract very general cues which are much more than merely the presence of lines or shapes. This method, more than any others, shows the potential of machine learning methods to make sense of single images. In this thesis we attempt to further develop these ideas, to show how planar structures can be extracted from single images.

CHAPTER 3

Plane Recognition

This chapter introduces our plane recognition algorithm, whose purpose is to determine whether an image region is planar or not, and estimate its 3D orientation. As it stands, this requires an appropriately delineated region of interest in the image to be given as input, and does not deal with the issue of finding or segmenting such regions from the whole image; however we stress that this technique, while limited, will form an essential component of what is to come.

As we discussed in Chapter 2, many approaches to single image plane detection have used geometric or textural cues, for example vanishing points [73] or the characteristic deformation of textures [44]. We aim to go beyond these somewhat restrictive paradigms, and develop an approach which aims to be more general, and is applicable to a wider range of scenes. Our approach is inspired by the way humans appear to be able to easily comprehend new scenes, by exploiting prior visual experiences [50, 51, 54, 101]. Therefore we take a machine learning approach, to learn the relationship between image appearance and 3D scene structure, using a collection of manually labelled examples.

To be able to learn, our algorithm requires the collection and annotation of a large set of training data; representation of training and test data in an appropriate manner; and the training of classification and regression algorithms to predict class and orientation

respectively for new regions. Over the course of this chapter we develop these concepts in detail.

3.1 Overview

The objective of our plane recognition algorithm is as follows: for a given, pre-segmented area of an image (generally referred to as the 'image region'), to classify it as being planar or non-planar, and if it is deemed to be planar, to estimate its orientation with respect to the camera coordinate system. For now, we are assuming that an appropriate region of the image is given as input.

The basic principle of our method is to learn the relationship between appearance and structure in a single image. Thus, the assumption upon which the whole method is founded is that there is a consistent relationship between how an image region looks and its 3D geometry. While this statement may appear trivial, it is not necessarily true: appearances can deceive, and the properties of surfaces may not be all they seem. Indeed, exploiting the simplest of relationships (again, like vanishing points) between appearance and structure in a direct manner is what can lead to the failure of existing methods, when such assumptions are violated. However, we believe that in general image appearance is a very useful cue to the identity and orientation of image regions, and we show that this is sufficient and generally reliable as a means of predicting planar structure.

To do this, we gather a large set of training examples, manually annotated with their class (it is to be understood that whenever we refer to 'class' and 'classification', it is to the distinction of plane and non-plane, rather than material or object identity) and their 3D orientation (expressed relative to the camera, since we do not have any global coordinate frame, and represented as a normalised vector in $\mathbb{R}^3$, pointing toward the viewer), where appropriate. These examples are represented using general features, rather than task-specific entities such as vanishing points. Image regions are represented using a collection of gradient orientation and colour descriptors, calculated about salient points. These are combined into a bag of words representation, which is projected to a low dimensional space by using a variant of latent semantic analysis, before enhancement with spatial distribution information using 'spatiograms' [10]. With these data and the regions' given target labels, we train a classifier and regressor (often referred to as simply 'the classifiers'). Using the same representation for new, previously unseen, test regions, the classifiers are used to predict their class and orientation.

In the following sections, we describe in detail each of these steps, with a discussion of the methods involved. A full evaluation of the algorithm, with an investigation of the various parameters and design choices, is presented in the next chapter.

3.2 Training Data

We gather a large set of example data, with which to train the classifiers. From the raw image data we manually choose the most relevant image regions, and mark them up with ground truth labels for class and orientation, then use these to synthetically generate more data with geometric transformations.

3.2.1 Data Collection

Using a standard webcam (Unibrain Fire-i)[1] connected to a laptop, we gathered video sequences from outdoor urban scenes. These are of size 320×240 pixels, and are corrected for radial distortion introduced by a wide-angle lens[2]. For development and validation of the plane recognition algorithm, we collected two datasets, one taken in an area surrounding the University of Bristol, for training and validating the algorithm; and a second retained as an independent test set, taken in a similar but separate urban location.

To create our training set, we select a subset of video frames, which show typical or interesting planar and non-planar structures. In each, we manually mark up the region of interest, by specifying points that form its convex hull. This means that we are using training data corresponding to either purely planar or non-planar regions (there are no mixed regions). This is the case for both training and testing data.

To train the classifiers (see Section 3.4), we need ground truth labels. Plane class is easy to assign, by labelling each region as to whether it is planar or not. Specifying the true orientation of planar regions is a little more complicated, since the actual orientation is of course not calculable from the image itself, so instead we use an interactive method based on vanishing points as illustrated in Figure 3.1 [26, 62]. Four points corresponding to the

[1] www.unibrain.com/products/fire-i-digital-camera/

[2] Using the Caltech calibration toolkit for MATLAB, from www.vision.caltech.edu/bouguetj/calib_doc

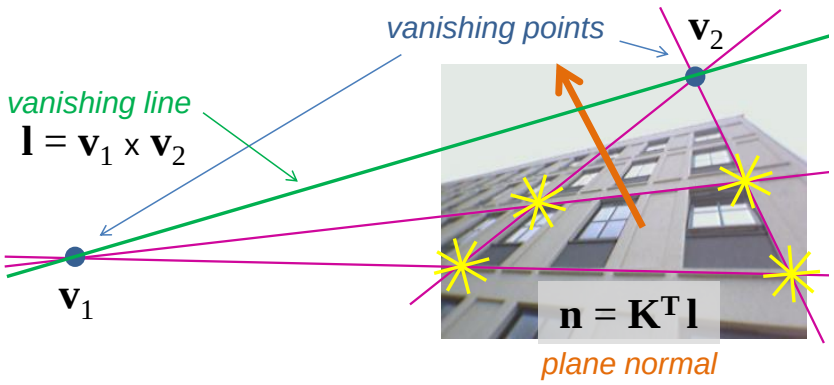


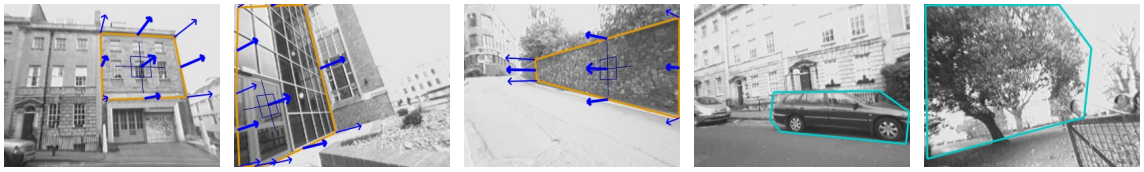
Figure 3.1: *Illustration of how the ground truth orientation is obtained, using manually selected corners of a planar rectangle.*

corners of a rectangle lying on the plane in 3D are marked up by hand, and the pairs of opposing edges are extended until they meet to give vanishing points in two orthogonal directions. These are denoted $\mathbf{v}_1$ and $\mathbf{v}_2$, and are expressed in homogeneous coordinates (i.e. extended to 3D vectors by appending a 1). Joining these two points in the image by a line gives the vanishing line, which is represented by a 3-vector $\mathbf{l} = \mathbf{v}_1 \times \mathbf{v}_2$. The plane which passes through the vanishing line and the camera centre is parallel to the scene plane described by the rectangle, and its normal can be obtained from $\mathbf{n} = \mathbf{K}^{\mathrm{T}}\mathbf{l}$, where $\mathbf{K}$ is the 3×3 intrinsic camera calibration matrix [62] (and a superscripted 'T' denotes the matrix transpose) . Examples of training data, both planar, showing the 'true' orientation, and non-planar, are shown in Figure 3.2.

3.2.2 Reflection and Warping

In order to increase the size of our training set, we synthetically generate new training examples. First, we reflect all the images about the vertical axis, since a reflection of an image can be considered equally physically valid. This immediately doubles the size of our training set, and also removes any bias for left or right facing regions.

We also generate examples of planes with different orientations, by simulating the view as seen by a camera in different poses. This is done by considering the original image to be viewed by a camera at location $[\mathbf{I}|\mathbf{0}]$, where $\mathbf{I}$ is the 3×3 identity matrix (no rotation) and $\mathbf{0}$ is the 3D zero vector (the origin); and then approximating the image seen from a camera at a new viewpoint $[\mathbf{R}|\mathbf{t}]$ (rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ with respect to the original view) by deriving a planar homography relating the image of the



(a) (b) (c) (d) (e)

Figure 3.2: *Examples of our manually outlined and annotated training data, showing examples of both planes (orange boundary, with orientation vectors) and non-planes (cyan boundary).*

plane in both views. The homography linking the two views is calculated by

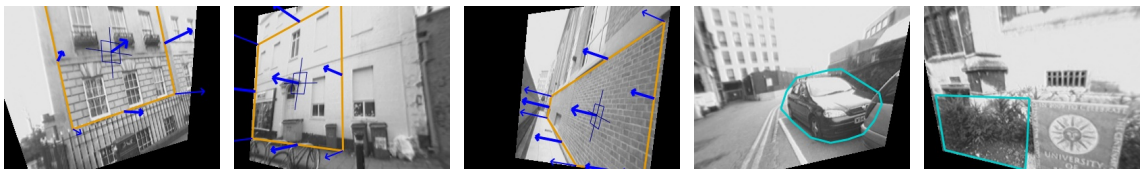
$$\mathbf{H} = \mathbf{R} + \frac{\mathbf{t}\mathbf{n}^{\mathrm{T}}}{d} \tag{3.1}$$

where $\mathbf{n}$ is the normal of the plane and d is the perpendicular distance to the plane (all defined up to scale, which means without loss of generality we set $d = 1$). We use this homography to warp the original image, to approximate the view from the new viewpoint. To generate the pose $[\mathbf{R}|\mathbf{t}]$ of the hypothetical camera, we use rotations of angle θ_γ about the three coordinate axes $\gamma \in \{x, y, z\}$, each represented by a rotation matrix $\mathbf{R}_\gamma$, the product of which gives us the final rotation matrix

$$\mathbf{R} = \mathbf{R}_x\mathbf{R}_y\mathbf{R}_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

We calculate the translation by $\mathbf{t} = -\mathbf{R}\mathbf{D} + \mathbf{D}$, where $\mathbf{D}$ is a unit vector in the direction of the point on the plane around which to rotate (calculated as $\mathbf{D} = \mathbf{K}^{-1}\mathbf{m}$ where $\mathbf{m}$ is a 2D point on the plane, usually the centroid, expressed in homogeneous coordinates). After warping the image, the normal vector for this warped plane is $\mathbf{R}\mathbf{n}$.

To generate new $[\mathbf{R}|\mathbf{t}]$ pairs we step through angles in x and y in increments of 15°, up to ±30° in both directions. While we could easily step in finer increments, or also apply rotations about the z-axis (which amounts to rotating the image), this quickly makes



(a) (b) (c) (d) (e)

Figure 3.3: *Training data after warping using a homography to approximate new viewpoints.*

training sets too large to deal with. We also omit examples which are very distorted, by checking whether the ratio of the eigenvalues of the resulting regions are within a certain range, to ensure that the regions are not stretched too much. Warping is also applied to the non-planar examples, but since these have no orientation specified, we use a normal pointing toward the camera, to generate the homography, under the assumption that the scene is approximately front-on. Warping these is not strictly necessary, but we do so to increase the quantity of non-planar examples available, and to ensure that the number of planar and non-planar examples remain comparable. Examples of warped images are shown in Figure 3.3.

3.3 Image Representation

We describe the visual information in the image regions of interest using local image descriptors. These represent the local gradient and colour properties, and are calculated for patches about a set of salient points. This means each region is described by a large and variable number of descriptors, using each individual point. In order to create more concise descriptions of whole regions, we employ the visual bag of words model, which represents the region according to a vocabulary; further reduction is achieved by discovering a set of underlying latent topics, compressing the bag of words information to a lower dimensional space. Finally we enhance the topic-based representation with spatial distribution information, an important step since the spatial configuration of various image features is relevant to identifying planes. This is done using spatial histograms, or 'spatiograms'. Each step is explained in more detail in the following sections.

3.3.1 Salient Points

Only salient points are used, in order to reduce the amount of information we must deal with, and to focus only on those areas of the image which contain features relevant to our task. For example, it would be wasteful to represent image areas devoid of any texture as these contribute very little to the interpretation of the scene.

Many alternatives exist for evaluating the saliency of points in an image. One popular choice is the FAST detector [110], which detects corner-like features. We experimented with this, and found it to produce not unreasonable results (see Section 4.1.2); however this detects features at only a single scale, ignoring the fact that image features tend to occur over a range of different scales [80]. Since generally there is no way to know *a priori* at which scale features will occur, it is necessary to analyse features at all possible scales [79].

To achieve this we use the difference of Gaussians detector to detect salient points, which is well known as the first stage in computing the SIFT feature descriptor [81]. As well as a 2D location for interest points, this gives the scale at which the feature is detected. This is to be interpreted as the approximate size, in pixels, of the image content which leads the point to be considered salient.

3.3.2 Features

Feature descriptors are created about each salient point in the image. The patch sizes used for building the descriptors come from the points' scales, since the scale values represent the regions of the image around the points which are considered salient. The basic features we use to describe the image regions capture both texture and colour information, these being amongst the most important basic visual characteristics visible in images.

Texture information is captured using histograms of local gradient orientation; such descriptors have been successfully used in a number of applications, including object recognition and pedestrian detection [28, 82]. However one important difference between our descriptors and something like SIFT is that we do not aim to be invariant. While robustness to various image transformations and deformations is beneficial for reliably detecting objects, this would actually be a disadvantage to us. We are aiming to actually

recover orientation, rather than be invariant to its effects, and so removing its effects from the descriptor would be counter-productive.

To build the histograms, the local orientation at each pixel is obtained by convolving the image with the mask $[1, 0, -1]$ in the x and y directions separately, to approximate the first derivatives of the image. This gives the gradient values G_x and G_y, for the horizontal and vertical directions respectively, which can be used to obtain the angle θ and magnitude m of the local gradient orientation:

$$\begin{aligned} \theta &= \tan^{-1}\left(\frac{G_y}{G_x}\right) \\ m &= \sqrt{G_x^2 + G_y^2} \end{aligned} \tag{3.3}$$

The gradient orientation and magnitude are calculated for each pixel within the image patch in question, and gradient histograms are built by summing the magnitudes over orientations, quantised into 12 bins covering the range $[0, \pi)$. Our descriptors are actually formed from four such histograms, one for each quadrant of the image patch, then concatenated into one 48D descriptor vector. This is done in order to incorporate some larger scale information, and we illustrate it in Figure 3.4. This construction is motivated by histograms of oriented gradient (HOG) features [28], but omits the normalisation over multiple block sizes, and has only four non-overlapping cells.

The importance of colour information for geometric classification was demonstrated by Hoiem et al. [66], and is used here as it may disambiguate otherwise difficult examples, such as rough walls and foliage. To encode colour, we use RGB histograms, which are formed by concatenating intensity histograms built from each of the three colour channels

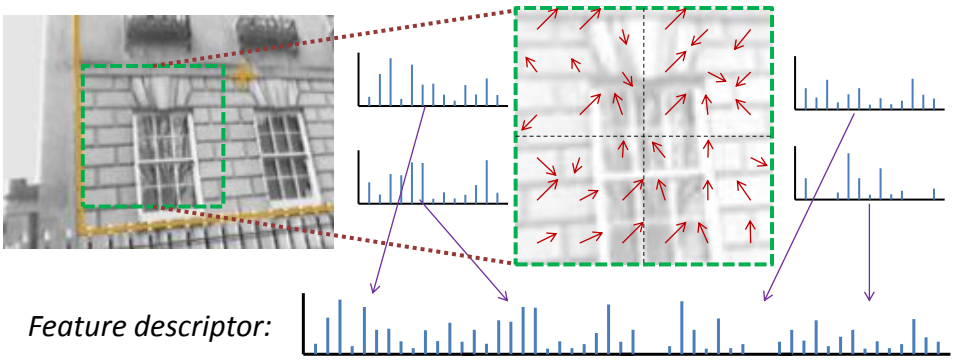


Figure 3.4: *An illustration showing how we create the quadrant-based gradient histogram descriptor. An image patch is divided into quadrants, and a separate orientation histogram created for each, which are concatenated.*

of the image, each with 20 bins, to form a 60D descriptor.

We use both types of feature together for plane classification; however, it is unlikely that colour information would be beneficial for estimating orientation of planar surfaces. Therefore, we use only gradient features for the orientation estimation step; the means by which we use separate combinations of features for the two tasks is described below.

3.3.3 Bag of Words

Each image region has a pair of descriptors for every salient point, which may number in the tens or hundreds, making it a rather rich but inefficient means of description. Moreover, each region will have a different number of points, making comparison problematic. This is addressed by accumulating the information for whole regions in an efficient manner using the bag of words model.

The bag of words model was originally developed in the text retrieval literature, where documents are represented simply by relative counts of words occurring in them; this somewhat naïve approach has achieved much success in tasks such as document classification or retrieval [77, 115]. When applied to computer vision tasks, the main difference that must be accounted for is that there is no immediately obvious analogue to words in images, and so a set of 'visual words' are created. This is done by clustering a large set of example feature vectors, in order to find a small number of well distributed points in feature space (the cluster centres). Feature vectors are then described according to their relationship to these points, by replacing the vectors by the ID of the word (cluster centre) to which they are closest. Thus, rather than a large collection of descriptor vectors, an image region can be represented simply as a histogram of word occurrences over this vocabulary. This has been shown to be an effective way of representing large amounts of visual information [134]. In what follows, we use 'word' (or 'term') to refer to such a cluster centre, and 'document' is synonymous with image.

We create two separate codebooks (sets of clustered features), for gradient and colour. These are formed by clustering a large set of features harvested from training images using K-means, with K clusters. Clustering takes up to several minutes per codebook, but must only be done once, prior to training (codebooks can be re-used for different training sets, assuming there is not too much difference in the features used).

Word histograms are represented as K-dimensional vectors $\mathbf{w}$, with elements denoted

w^k, for $k = 1, \ldots, K$. Each histogram bin is $w^k = |\Lambda^k|$ where Λ^k is the set of points which quantised to word k (i.e. w^k is the count of occurrences of word k in the document). To reduce the impact of commonly occurring words (words that appear in all documents are not very informative), we apply term frequency–inverse document frequency (tf-idf) weighting as described in [84]. We denote word vectors after weighting by $\mathbf{w}'$, and when we refer to word histograms hereafter we mean those weighted by tf-idf, unless otherwise stated.

As discussed above, we use two different types of feature vector, of different dimensionality. As such we need to create separate codebooks for each, for which the clustering and creation of histograms is independent. The result is that each training region has two histograms of word counts, for its gradient and colour features.

3.3.4 Topics

The bag of words model, as it stands, has a few problems. Firstly, there is no association between different words, so two words representing similar appearance would be deemed as dissimilar as any other pair of words. Secondly, as the vocabularies become large, the word histograms will become increasingly sparse, making comparison between regions unreliable as the words they contain are less likely to coincide.

The answer again comes from the text retrieval literature, where similar problems (word synonymy and high dimensionality) are prevalent. This idea is to make use of an underlying latent space amongst the words in a corpus, which can be thought of as representing 'topics', which should roughly correspond to a single semantic concept.

It is not realistic to expect each document to correspond to precisely one latent topic, and so a document is represented by a distribution over topics. Since the number of topics is generally much less than the number of words, this means topic analysis achieves dimensionality reduction. Documents are represented as a weighted sum over topics, instead of over words, and ideally synonyms are implicitly taken care of by being mapped to the same semantic concept.

A variety of methods exist for discovering latent topics, which all take as input a 'term-document' matrix. This is a matrix $\mathbf{W} = [\mathbf{w}'_0, \mathbf{w}'_1, \ldots, \mathbf{w}'_M]$, where each column is the weighted word vector for each of M documents, so each row corresponds to a word k.

3.3.4.1 Latent Semantic Analysis

The simplest of these methods is known as latent semantic analysis (LSA) [30], which discovers latent topics by factorising the term-document matrix $\mathbf{W}$, using the singular value decomposition (SVD), into $\mathbf{W} = \mathbf{UDV}^{\mathrm{T}}$. Here $\mathbf{U}$ is a $K \times K$ matrix, $\mathbf{V}$ is $M \times M$, and $\mathbf{D}$ is the diagonal matrix of singular values. The reduced dimensionality form is obtained by truncating the SVD, retaining only the top T singular values (where T is the desired number of topics), so that $\mathbf{W} \approx \mathbf{U}_t\mathbf{D}_t\mathbf{V}_t^{\mathrm{T}}$, where $\mathbf{U}_t$ is $K \times T$ and $\mathbf{V}_t$ is $M \times T$. The rows of matrix $\mathbf{V}_t$ are the reduced dimensionality description – the topic vectors $\mathbf{t}_m$ – for each document m.

An advantage of LSA is that it is very simple to use, partly because topic vectors for the image regions in the training set are extracted directly from $\mathbf{V}_t$. Topic vectors for new test image regions are calculated simply by projecting their word histograms into the topic space, using $\mathbf{t}_j = \mathbf{D}_t^{-1}\mathbf{U}_t^{\mathrm{T}}\mathbf{w}'_j$. However, LSA suffers from an important problem, due to its use of SVD: the topic weights (i.e. the elements of the vectors $\mathbf{t}_j$) may be negative. This makes interpretation of the topic representation difficult (what does it mean for a document to have a negative amount of a certain topic?), but more importantly the presence of negative weights will become a problem when we come to take weighted means of points' topics (see Section 3.3.5), where all topic weights would need to be non-negative.

A later development called probabilistic latent semantic analysis (pLSA) [63] does preserve the non-negativity of all components (because they are expressed as probabilities), but its use of expectation maximisation to both find the factorisation, and get the topic vectors for new data, is infeasibly slow for our purposes. Fortunately, a class of methods known as non-negative matrix factorisation has been shown, under some conditions, to be equivalent to pLSA [31, 45].

3.3.4.2 Non-negative Matrix Factorisation

As the name implies, non-negative matrix factorisation (NMF) [76] is a method for factorising a matrix (in this case, the term-document matrix $\mathbf{W}$) into two factor matrices, with reduced dimensionality, where all the terms are positive or zero. The aim is to find the best low-rank approximation $\mathbf{W} \approx \mathbf{BT}$ to the original matrix with non-negative terms. If $\mathbf{W}$ is known to have only non-negative entries (as is the case here, since its

entries are word counts) so will $\mathbf{B}$ and $\mathbf{T}$. Here, $\mathbf{T}$ is $T \times M$ and will contain the topic vectors corresponding to the columns of $\mathbf{W}$; and $\mathbf{B}$ can be interpreted as the basis of the topic space (of size $K \times T$, the number of words and topics respectively). There are no closed form solutions to this problem, but Lee and Seung [76] describe an iterative algorithm, which can be started from a random initialisation of the factors.

As with LSA, topic vectors for training regions are simply the columns of $\mathbf{T}$. Unfortunately, although we can re-arrange the above equation to obtain $\mathbf{t}_j = \mathbf{B}^\dagger \mathbf{w}'_j$ (where $\mathbf{B}^\dagger$ is the Moore-Penrose pseudoinverse) to get a low-dimensional approximation for test vectors $\mathbf{w}'_j$, there is no non-negativity constraint on $\mathbf{B}^\dagger$. This means that the resulting topic vectors $\mathbf{t}_j$ too will contain negative elements, and so we have the same problem as with LSA.

The problem arises from using the pseudoinverse. It is generally calculated using SVD, which as we saw above does not maintain non-negativity. One way to ensure that the inverse of $\mathbf{B}$ is non-negative is to make $\mathbf{B}$ orthogonal, since for a semi-orthogonal matrix (non-square) its pseudoinverse is also its transpose. This leads us to the methods of orthogonal non-negative matrix factorisation.

3.3.4.3 Orthogonal Non-negative Matrix Factorisation

The objective of orthogonal non-negative matrix factorisation (ONMF) [18] is to factorise as above, with the added constraint that $\mathbf{B}^{\mathrm{T}}\mathbf{B} = \mathbf{I}$. Left-multiplying $\mathbf{W} = \mathbf{BT}$ by $\mathbf{B}^{\mathrm{T}}$ we get $\mathbf{B}^{\mathrm{T}}\mathbf{W} = \mathbf{B}^{\mathrm{T}}\mathbf{BT} = \mathbf{T}$, and so it is now valid to project a word vector $\mathbf{w}'_j$ to the topic space by $\mathbf{t}_j = \mathbf{B}^{\mathrm{T}}\mathbf{w}'_j$, such that the topic vector contains only non-negative elements. In practice, these factors are found by a slight modification of the method of NMF [135], and so the algorithms used for ONMF are:

$$\mathbf{B}_{nt} \longleftarrow \mathbf{B}_{nt} \frac{(\mathbf{W}\mathbf{T}^{\mathrm{T}})_{nt}}{(\mathbf{B}\mathbf{T}\mathbf{W}^{\mathrm{T}}\mathbf{B})_{nt}} \qquad \mathbf{T}_{tm} \longleftarrow \mathbf{T}_{tm} \frac{(\mathbf{B}^{\mathrm{T}}\mathbf{W})_{tm}}{(\mathbf{B}^{\mathrm{T}}\mathbf{B}\mathbf{T})_{tm}} \tag{3.4}$$

The result is a factorisation algorithm which directly gives the topic vectors for each training example used in creating the term-document matrix, and a fast linear method for finding the topic vector of any new test word histogram, simply by multiplying with the transpose of the topic basis matrix $\mathbf{B}$.

(a) *The words in the image, each with a unique colour*

(b) *Points contributing to Topic 16*

(c) *Points contributing to Topic 5*

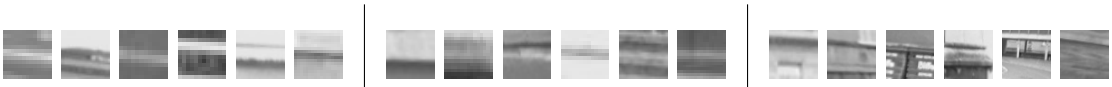
Figure 3.5: *For a given planar region (a), on which we have drawn coloured points to indicate the different words, we can also show the weighting of these points to different topics, namely Topics 16 (b) and 5 (c), corresponding to those visualised below.*

3.3.4.4 Topic Visualisation

So far, in considering the use of topics analysis, we have treated it simply as a means of dimensionality reduction. However, just as topics should represent underlying semantic topics in text documents, they should also represent similarities across our visual words. Indeed, it is partly in order capture otherwise overlooked similarities between different words that we have used topic analysis, and so it would interesting to check whether this is actually the case.

First, consider the example plane region in Figure 3.5a, taken from our training set. In the left image, we have shown all the words, each with a unique colour – showing no particular structure. The two images to its right show only those points corresponding to certain topics (Topic 16 and Topic 5, from a gradient-only space of 20 topics), via the words that the features quantise to, where the opacity of the points represent the extent to which the word contributes to that topic. It appears that words contributing to Topic 16 tend to occur on the tops and bottoms of windows, while those for Topic 5 lie within the windows, suggesting that they may be picking out particular types of structure.

We expand upon this, by more directly demonstrating what the topics represent. Visualising words and topics can be rather difficult, since both exist in high dimensional spaces, and because topics represent a distribution over all the words, not simply a reduced selection of important words. Nevertheless, it should be the case that the regions which quantise to the words which contribute most strongly to a given topic will look more similar to each other than other pairs of words.



(a) *Word 336* **(b)** *Word 84* **(c)** *Word 4*

Figure 3.6: *A selection of patches quantising to the top three words for Topic 16, demonstrating different kinds of horizontal edge.*

(a) *Word 376* **(b)** *Word 141* **(c)** *Word 51*

Figure 3.7: *Patches representing the top three words for Topic 5, which appear to correspond to different kinds of grid pattern (or, with a combination of vertical and horizontal edges; a few apparent outliers are shown toward the right, although these retain a similar pattern of edges).*

For the image region above, we looked at the histogram of topics, and found that the two highest weighted were Topics 16 and 5 (those shown above). Rather than showing these topic directly, we can find which words contribute most strongly to those topics. For Topic 16, the highest weighted words (using the word-topic weights from $\mathbf{B}$) were 336, 84 and 4; and for Topic 5, the words with the highest weights were 376, 141 and 51. Again, these numbers have little meaning on their own, being simply indices within the unordered set of 400 gradient words we used.

We show example image patches for each of these words, for those two topics, in Figures 3.6 and 3.7. These patches were extracted from the images originally used to create the codebooks (the patches are of different sizes due to the use of multi-scale salient point detection, but have been resized for display). If the clustering has been performed correctly there should be similarities between different examples of the same word; and according to the idea behind latent topic analysis, we should also find that the words assigned to the same topic are similar to each other.

This does indeed appear to be the case. The three words shown for Topic 16 represent various types of horizontal features. Because we use gradient orientation histograms as features, the position of these horizontal edges need not remain constant, nor the direction of intensity change. Similarly, Topic 5 appears to correspond to grid-like features, such as window panes and tiles. The fact that these groups of words have been placed together within topics suggests topic analysis is performing as desired, in that it is able to link words with similar conceptual features which would usually – in a bag of words model – be considered as different as any other pair of words. Not only does this grouping suggest the correct words have been associated together, but it agrees with the location

of the topics as visualised in Figure 3.5, which are concentrated on horizontal window edges and grid-like window panes respectively.

We acknowledge that these few examples do not conclusively show that this happens consistently, but emphasise that our method does not depend on it: it is sufficient that ONMF performs dimensionality reduction in the usual sense of the word. Nevertheless, it is reassuring to see such structure spontaneously emerge from ONMF, and for this to be reflected in the topics found in a typical image from our training set.

3.3.4.5 Combining Features

The above discussion assumes there is only one set of words and documents, and does not deal with our need to use different vocabularies for different tasks. Fortunately, ONMF makes it easy to combine the information from gradient and colour words. This is done by concatenating the two term-document matrices for the corpus, so that effectively each document has a word vector of length $2K$ (assuming the two vocabularies have the same number of words). This concatenated matrix is the input to ONMF, which means the resulting topic space is over the joint distribution of gradient and colour words, so should encode correlations between the two types of visual word. Generally we double the number of topics, to ensure that using both together retains the details from either used individually.

When regressing the orientation of planes, colour information is not needed. We run ONMF again to create a second topic space, using a term-document matrix built only from gradient words, and only using planar image regions. This means that there are two topic spaces, a larger one containing gradient and colour information from all regions, and a smaller one, of lower dimensionality, using only the gradient information from the planar regions.

3.3.5 Spatiograms

The topic histograms defined above represent image regions compactly; however, we found classification and regression accuracy to be somewhat disappointing using these alone (see our experiments in Section 4.1.1). A feature of the underlying bag of words model is that all spatial information is discarded. This also applies to the latent topic

representation. Although this has not hampered performance in tasks such as object recognition, for our application the relative spatial position of features are likely to be important. For example, some basic visual primitives may imply a different orientation depending on their relative position.

It is possible to include spatial information by representing pairwise co-occurrence of words [9], by tiling overlapping windows [131], or by using the constellation or star models [37, 38]. While the latter are effective, they are very computationally expensive, and scale poorly in terms of the number of points and categories. Instead, we accomplish this by using spatiograms. These are generalisations of histograms, able to include information about higher-order moments. These were introduced by Birchfield and Rangarajan [10] in order to improve the performance of histogram-based tracking, where regions with different appearance but similar colours are too easily confused.

We use second-order spatiograms, which as well as the occurrence count of each bin, also encode the mean and covariance of points contributing to that bin. These represent the spatial distribution of topics (or words), and they replace the topic histograms above (not the gradient or colour histogram descriptors). While spatiograms have been useful for representing intensity, colour and terrain distribution information [10, 52, 83], to our knowledge they have not previously been used with a bag of words model.

We first describe spatiograms over words, in order to introduce the idea. A word spatiogram $\mathbf{s}^{\mathsf{word}}$, over K words, is defined as a set of K triplets $\mathbf{s}_k^{\mathsf{word}} = (h_k^{\mathsf{word}}, \boldsymbol{\mu}_k^{\mathsf{word}}, \boldsymbol{\Sigma}_k^{\mathsf{word}})$, where the histogram values $h_k^{\mathsf{word}} = w'^k$ are the elements of the word histogram as above, and the mean and (unbiased) covariance are defined as

$$\boldsymbol{\mu}_k^{\mathsf{word}} = \frac{1}{|\Lambda^k|} \sum_{i \in \Lambda^k} \mathbf{v}_i \qquad \boldsymbol{\Sigma}_k^{\mathsf{word}} = \frac{1}{|\Lambda^k| - 1} \sum_{i \in \Lambda^k} \mathbf{v}_i^k \mathbf{v}_i^{k\mathrm{T}} \tag{3.5}$$

where $\mathbf{v}_i$ is the 2D coordinate of point i and $\mathbf{v}_i^k = \mathbf{v}_i - \boldsymbol{\mu}_k^{\mathsf{word}}$, and as above Λ^k is the subset of points whose feature vectors quantise to word k.

To represent 2D point positions within spatiograms, we normalise them with respect to the image region, leaving us with a set of points with zero mean, rather than being coordinates in the image. This gives us a translation invariant region descriptor. Without this normalisation, a similar patch appearing in different image locations would have

a different spatiogram, and so would have a low similarity score, which may adversely affect classification. The shift is achieved by replacing the $\mathbf{v}_i$ in the equations above with $\mathbf{v}_i^* = \mathbf{v}_i - \frac{1}{N}\sum_{i=1}^{N}\mathbf{v}_i$. Note that by shifting the means, the covariances (and histogram values) are unaffected.

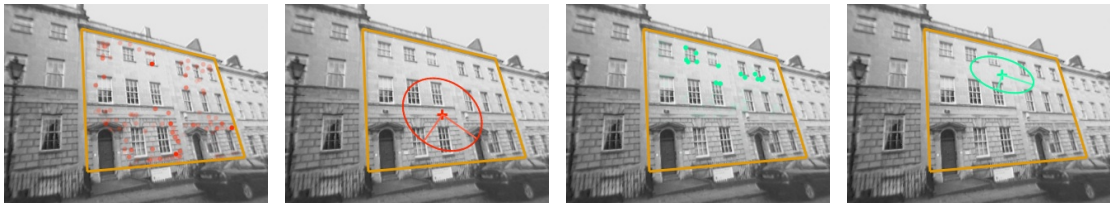
The definition of word spatiograms is fairly straightforward, since every 2D point quantises to exactly one word. Topics are not so simple, since each word has a *distribution* over topics, and so each 2D point contributes some different amount to each topic. Therefore, rather than simply a sum over a subset of points, the mean and covariance will be a weighted mean and a weighted (unbiased) covariance over all the points, according to their contribution to each topic. It is because of this calculation of weighted means that the topic weights cannot be negative, as ensured by our use of ONMF, discussed above.

Rather than a sum over individual points, we can instead express topic spatiograms as a sum over the words, since all points corresponding to the same word contribute the same amount to their respective topics. The resulting topic spatiograms $\mathbf{s}^{\mathsf{topic}}$, defined over T topics, consist of T triplets $\mathbf{s}_t^{\mathsf{topic}} = (h_t^{\mathsf{topic}}, \boldsymbol{\mu}_t^{\mathsf{topic}}, \boldsymbol{\Sigma}_t^{\mathsf{topic}})$. Here, the scalar elements h_t^{topic} are from the region's topic vector as defined above, while the mean and covariance are calculated using

$$\boldsymbol{\mu}_t^{\mathsf{topic}} = \frac{1}{\alpha_t}\sum_{k=1}^{K}\eta_{tk}\boldsymbol{\mu}_k^{\mathsf{word}} \qquad \boldsymbol{\Sigma}_t^{\mathsf{topic}} = \frac{\alpha_t}{\alpha_t^2 - \beta_t}\sum_{k=1}^{K}\frac{\eta_{tk}}{|\Lambda^k|}\sum_{i\in\Lambda^k}\mathbf{v}_i^t\mathbf{v}_i^{t\mathrm{T}} \tag{3.6}$$

where $\mathbf{v}_i^t = \mathbf{v}_i^* - \boldsymbol{\mu}_t^{\mathsf{topic}}$, $\alpha_t = \sum_{k=1}^{K}\eta_{tk}$, and $\beta_t = \sum_{k=1}^{K}\frac{\eta_{tk}^2}{|\Lambda^k|}$. The weights η_{tk} are given by $\eta_{tk} = B_{tk}w'_k$ and reflect both the importance of word k through w'_k and its contribution to topic t via B_{tk} (elements from the topic basis matrix). As in Section 3.3.4.5, we maintain two spatiograms per (planar) image region, one for gradient and colour features, and one for just gradient features. We illustrate the data represented by spatiograms in Figure 3.8, where we have shown an image region overlaid with some of the topics to which the image features contribute (see Figure 3.5 above), as well as the mean and covariance for the topics, which is encoded within the spatiogram.

To use the spatiograms for classification and regression, we use a similarity measure proposed by Ò Conaire et al. [97]. This uses the Battacharyya coefficient to compare spatiogram bins, and a measure of the overlap of Gaussian distributions to compare



(a) (b) (c) (d)

Figure 3.8: *An illustration of what is represented by topic spatiograms. The points show the contribution of individual points to each topic (a,c), and the spatiograms represent the distribution of these contributions (b,d), displayed here as an ellipse showing the covariance, centred on the mean, for individual topics.*

their spatial distribution. For two spatiograms $\mathbf{s}^A$ and $\mathbf{s}^B$ of dimension D, this similarity function is defined as

$$\rho(\mathbf{s}^A, \mathbf{s}^B) = \sum_{d=1}^{D} \sqrt{h_d^A h_d^B} 8\pi |\mathbf{\Sigma}_d^A \mathbf{\Sigma}_d^B|^{\frac{1}{4}} N\big(\boldsymbol{\mu}_d^A; \boldsymbol{\mu}_d^B, 2(\mathbf{\Sigma}_d^A + \mathbf{\Sigma}_d^B)\big) \tag{3.7}$$

where $N(\mathbf{x}; \boldsymbol{\mu}, \mathbf{\Sigma})$ is a Gaussian with mean $\boldsymbol{\mu}$ and covariance matrix $\mathbf{\Sigma}$ evaluated at $\mathbf{x}$. Following [97], we use a diagonal version of the covariance matrices since it simplifies the calculation.

3.4 Classification

After compactly representing the image regions with spatiograms, they are used to train a classifier. We use the relevance vector machine (RVM) [125], which is a sparse kernel method, conceptually similar to the more well-known support vector machine (SVM). We choose this classifier because of its sparse use of training data: once it has been trained, only a small subset of the data need to be retained (fewer even than the SVM), making classification very fast even for very large training sets. The RVM gives probabilistic outputs, representing the posterior probability of belonging to either class.

3.4.1 Relevance Vector Machines

The basic model of the RVM is very similar to standard linear regression, in which the output label y for some input vector $\mathbf{x}$ is modelled as a weighted linear combination of

M fixed (potentially non-linear) basis functions [11]:

$$y(\mathbf{x}) = \sum_{i=1}^{M} \omega_i \phi_i(\mathbf{x}) = \boldsymbol{\omega}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}) \tag{3.8}$$

where $\boldsymbol{\omega} = (\omega_i, \ldots, \omega_M)^{\mathrm{T}}$ is the vector of weights, for some number M of basis functions $\boldsymbol{\phi}(\mathbf{x}) = (\phi_i(\mathbf{x}), \ldots, \phi_M(\mathbf{x}))^{\mathrm{T}}$. If we choose the basis functions such that they are given by kernel functions, where there is one kernel function for each of the N training examples (a similar structure to the SVM), (3.8) can be re-written:

$$y(\mathbf{x}) = \sum_{n=1}^{N} \omega_m k(\mathbf{x}, \mathbf{x}_n) + b \tag{3.9}$$

where $\mathbf{x}_n$ are the training data, for $n = 1, \ldots, N$, and b is a bias parameter. The kernel functions take two vectors as input and return some real number. This can be considered as a dot product in some higher dimensional space — indeed, under certain conditions, a kernel can be guaranteed to be equivalent to a dot product after some transformation of the data. This mapping to a higher dimensional space is what gives kernel methods their power, since the data may be more easily separable in another realm. Since the mapping need never be calculated explicitly (it is only ever used within the kernel function), this means the benefits of increased separability can be attained without the computational effort of working in higher dimensions (this is known as the 'kernel trick' [11]).

Training the RVM is done via the kernel matrix $\mathbf{K}$, where each element is the kernel function between two vectors in the training set, $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The matrix is symmetric, which saves some time during computation, but calculating it is still quadratic in the number of data. This becomes a problem when using such kernel methods on very large datasets (the RVM training procedure is cubic, due to matrix inversions [11], although iteratively adding relevance vectors can speed it up somewhat [126]).

While equation 3.9 is similar to standard linear regression, the important difference here is that rather than having a single shared hyperparameter over all the weights, the RVM introduces a separate Gaussian prior for each ω_i controlled by a hyperparameter α_i. During training, many of the α_i tend towards infinity, meaning the posterior probability of the associated weight is sharply peaked at zero — thus the corresponding basis functions

are effectively removed from the model, leading to a significantly sparsified form.

The remaining set of training examples – the 'relevance vectors' – are sufficient for prediction for new test data. These are analogous to the support vectors in the SVM, though generally far fewer in number; indeed, in our experiments we observed an over 95% reduction in training data used, for both classification and regression. Only these data (and their associated weights) need to be stored in order to use the classifier to predict the target value of a new test datum $\mathbf{x}'$. Classification is done through a new kernel matrix $\mathbf{K}'$ (here being a single column), whose elements are calculated as $\mathbf{K}'_r = k(\mathbf{x}_r, \mathbf{x}')$ for R relevance vectors indexed $r = 1, \ldots, R$. The prediction is then calculated simply by matrix multiplication:

$$y(\mathbf{x}') = \boldsymbol{\omega}^{\mathrm{T}}\mathbf{K}' \tag{3.10}$$

where $\boldsymbol{\omega}$ is again the vector of weights. For regression problems, the target value is simply $y(\mathbf{x}')$; whereas for classification, this is transformed by a logistic sigmoid,

$$\begin{aligned} p(\mathbf{x}') &= \sigma(y(\mathbf{x}')) = \sigma(\boldsymbol{\omega}^{\mathrm{T}}\mathbf{K}') \\ \sigma(x) &= \tfrac{1}{1+exp(-x)} \end{aligned} \tag{3.11}$$

This maps the outputs to $p(\mathbf{x}') \in (0, 1)$, which is interpreted as the probability of the test datum belonging to the positive class. Thresholding this at 0.5 (equal probability of either class) gives a binary classification. In our work we used the fast [126] 'Sparse Bayes' implementation of the RVM made available by the authors[3].

The above describes binary classification or single-variable regression in the standard RVM. In order to regress multi-dimensional data – in our case the three components of the normal vectors – we use the multi-variable RVM (MVRVM) developed by Thayananthan et al. [123]; the training procedure is rather different from the above, but we omit details here (see [11]). This regresses over each of the D output dimensions simultaneously, using the same set of relevance vectors for each. Regression is achieved as in (3.10),

[3] www.vectoranomaly.com/downloads/downloads.htm

but now the weights are a $R \times D$ matrix $\mathbf{\Omega}$, with a column for each dimension of the target variable; and so $\mathbf{y}(\mathbf{x}') = \mathbf{\Omega}^{\mathrm{T}}\mathbf{K}'$. For this we adapted code available online[4]. For both classification and regression, we can predict values for many data simultaneously if necessary, simply by adding more columns to $\mathbf{K}'$.

All that remains is to specify the form of kernel function we use. When using word or topic histograms, we experimented with a variety of standard histogram similarity measures, such as the Bhattacharyya coefficient and cosine similarity; for spatiograms, we use various functions of equation 3.7. More details of the different functions can be found in our experiments in Section 4.1.4.

3.5 Summary

This concludes our description of the plane recognition algorithm, for distinguishing planes from non-planes in given image regions and estimating their orientation. To summarise: we represent data for regions by detecting salient points, which are described using gradient and colour features; these are gathered into a bag of words, reduced with topic analysis, and enhanced with spatial information using spatiograms. These data are used to train an RVM classifier and regressor. For a test region, once the spatiogram has been calculated, the RVMs are used to classify it and, if deemed planar, to regress its orientation.

However, it is important to realise that, as successful as this algorithm may be, it is only able to estimate the planarity and orientation for a given region of the image. It is not able to detect planes in a whole image, since there is no way of knowing where the boundaries between regions are, and this is something we shall return to in Chapter 5. Before that, in the next chapter we present a thorough evaluation of this algorithm, both in cross-validation, to investigate the effects of the details discussed above; and an evaluation on an independent test set, showing that it can generalise well to novel environments.

[4] mi.eng.cam.ac.uk/~at315/MVRVM.htm

CHAPTER 4

Plane Recognition Experiments

In this chapter, we present experimental results for our plane recognition algorithm. We show how the techniques described in the previous chapter affect recognition, for example how the inclusion of spatial distribution information makes classification and regression much more accurate, and the benefits of using larger amounts of training data. We describe experiments on an independent set of data, demonstrating that our algorithm works not only on our initial training and validation set, but also in a wider context; and compare to an alternative classifier.

Our experiments on the plane recognition algorithm were conducted as follows: we began with a set of training data (individual, manually segmented regions, not whole images), which we represented using the steps discussed in Section 3.3. The resulting descriptions of the training regions were used to train the RVM classifiers. Only at this point were the test data introduced, so the creation of the latent space and the classifiers was independent of the test data.

4.1 Investigation of Parameters and Settings

For the first set of experiments, we used only our training set, collected from urban locations at the University of Bristol and the surrounding area. This consisted of 556 image regions, each of which was labelled as described in Section 3.2; these were reflected to create our basic dataset of 1112 regions. We also warped these regions, as described in Section 3.2.2, to obtain a total of 7752 regions. The effect of using these extra regions is discussed later.

All of these experiments used five-fold cross-validation on this training set — the train and test folds were kept independent, the only association between them being that the data came from the same physical locations, and that the features used to build the bag of words codebooks could have overlapped both train and test sets. We also ensured that warped versions of a region never appeared in the training set when the original region was in the test set (since this would be a potentially unfairly easy test). All of the results quoted below came from ten independent runs of cross-validation, from which we calculated the mean and standard deviation, which are used to draw error bars. The error bars on the graphs show one standard deviation either side of the mean, over all the runs of cross-validation (i.e. it is the standard deviation of the means from the multiple cross-validation runs, and not derived from the standard deviations of individual cross-validations).

4.1.1 Vocabulary

The first experiment investigated the performance for different vocabulary sizes, for different basic representations of the regions. The vocabulary size is the number of clusters K used in the K-means algorithm, and by testing using different values for K we could directly see what effect this had on plane recognition (instead of choosing the best K according to the distortion measure [11], for example).

This experiment also compared four different representations of the data. First, we used weighted word histograms as described in Section 3.3.3 — this representation does not use the latent topics, nor use spatial information. We compared this to the basic topic histogram representation, where the word vectors have been projected into the topic space to reduce their dimensionality (refer to Section 3.3.4); in these experiments we always used a latent space of 20 topics, but found performance to be robust to different

values. Next we created spatiograms for both word and topic representations, using equations 3.5 and 3.6.

The experiment was conducted as follows: for each different vocabulary size, we created a new codebook by clustering gradient features harvested from a set of around 100 exemplary images. The same set of detected salient points and feature descriptors was used each time (since these are not affected by the vocabulary size); and for each repeat run of cross-validation the same vocabulary was used, to avoid the considerable time it takes to re-run the clustering. This was done for each of the four region descriptions, then the process was repeated for each of the vocabulary sizes. We used only gradient features for this to simplify the experiment — so the discussion of combining vocabularies (Section 3.3.4.5) is not relevant at this point. In this experiment (and those that follow), we used only the original marked-up regions, and their reflections – totalling 1112 regions – since warping was not yet confirmed to be useful, and to keep the experiment reasonably fast. The kernels used for the RVMs were polynomial sums (see Section 4.1.4) of the underlying similarity measure (Bhattacharyya for histograms and spatiogram similarity [97] for spatiograms).

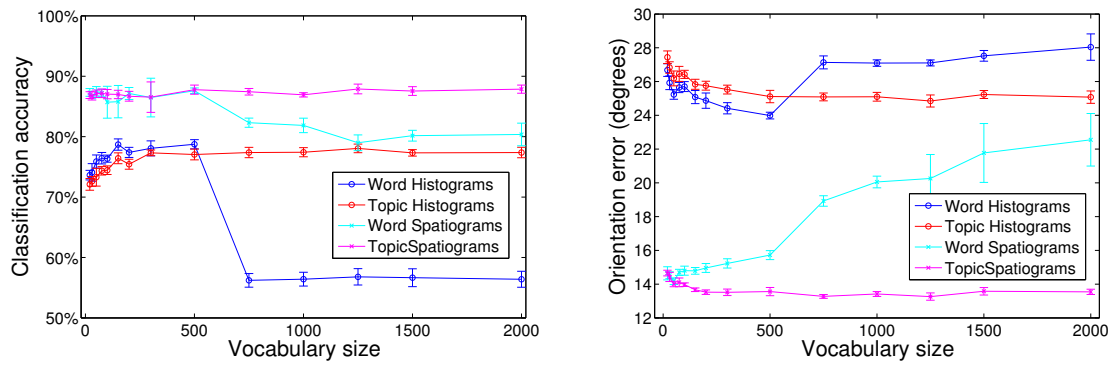


Figure 4.1: *Performance of plane classification and orientation estimation as the size of the vocabulary was changed.*

The results are shown in Figure 4.1, comparing performance for plane classification (a) and orientation regression (b). It is clear that spatiograms outperformed histograms, over all vocabulary sizes, for both error measures. Furthermore, using topics also tended to increase performance compared to using words directly, especially as the vocabulary size increased. As one would expect, there was little benefit in using topics when the number of topics was approximately the same as the number of words, and when using small vocabularies, performance of word spatiograms was as good as using topic spatiograms. However, since this relies on using a small vocabulary it would overly con-

strain the method (and generally larger vocabularies, up to an extent, give improved results [68]); and best performance was only seen when using around 400 words, the difference being much more pronounced for orientation estimation. This experiment confirms the hypothesis advanced in Section 3.3 that using topic analysis plus spatial representation is the best way, of those studied, for representing region information.

4.1.2 Saliency and Scale

In Section 3.3.1 we discussed the choice of saliency detector, and our decision to use a multi-scale detector to make the best use of the information at multiple image scales. We tested this with an experiment, comparing the performance of the plane recognition algorithm when detecting points using either FAST [110] or the difference of Gaussians (DoG) detector [81]. FAST gives points' locations only, and so we tried a selection of possible scales to create the descriptors. We did the same using DoG (i.e. ignoring the scale value and using fixed patch sizes), to directly compare the type of saliency used. Finally, we used the DoG scale information to choose the patch size, in order to create descriptors to cover the whole area deemed to be salient.

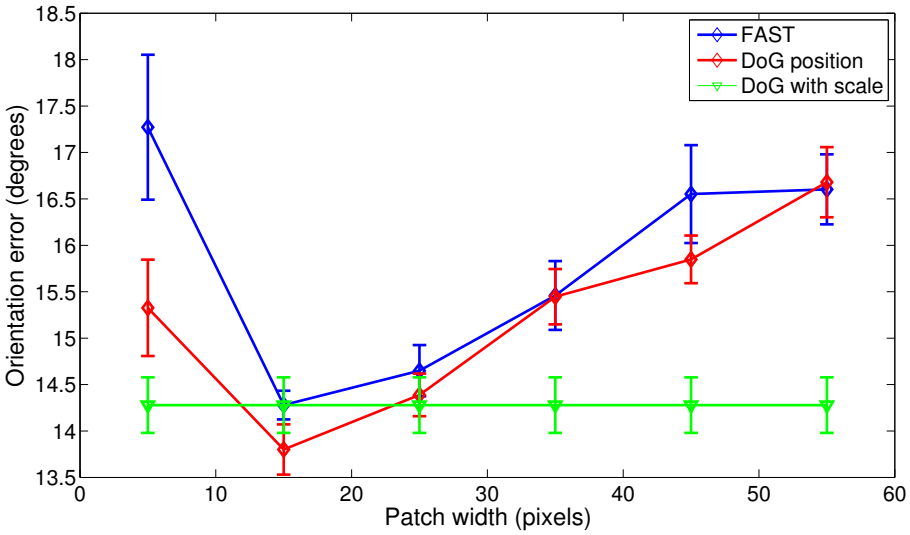


Figure 4.2: *The effect of patch size on orientation accuracy, for different means of saliency detection.*

Our results are shown in Figure 4.2, for evaluation using the mean angular error of orientation regression (we found that different saliency detectors made no significant difference to classification accuracy). It can be seen that in general, DoG with fixed patch sizes out-performed FAST, although at some scales the difference was not significant.

This suggests that the blob-like features detected by DoG might be more appropriate for our plane recognition task than the corner-like features found with FAST. The green line shows the performance when using the scale chosen by DoG (thus the patch size axis has no meaning, hence the straight line), which in most cases was clearly superior to both FAST and DoG with fixed patch sizes.

We note that at a size of 15 pixels, a fixed patch size appeared to out-perform scale selection, and it may be worth investigating further since this would save computational effort. However, we do not feel this one result is enough yet to change our method, as it may be an artefact of these data. We conclude that this experiment broadly supports our reasons for using scale selection, rather than relying on any particular patch size; especially given that we would not know the best scale to use with a particular dataset.

4.1.3 Feature Representation

The next experiment compared performance when using different underlying feature representations, namely the gradient and colour features as described in Section 3.3.2. In these experiments, we used a separate vocabulary for gradient and colour descriptors, using K-means as before (having fixed the number of words, on the basis of the earlier experiment, at 400 words for the gradient vocabulary, and choosing 300 for colour). For either feature descriptor type used in isolation, the testing method was effectively the same; when using both together, we combined the two feature representations by concatenating their word histograms before running ONMF (words were re-numbered as appropriate, so that word i in the colour space became word $K_g + i$ in the concatenated space, where K_g is the number of words in the gradient vocabulary). Since this used around twice as much feature information, we doubled the number of topics to 40 for the concatenated vocabularies, which we found to improve performance somewhat (simply doubling the number of topics for either feature type in isolation showed little difference, so any improvement will be due to the extra features).

These experiments were again conduced using ten runs of five-fold cross-validation, using topic spatiograms, after showing them to be superior in the experiment above. We used the non-warped set of 1112 regions, with the polynomial sum kernel (see below), and the vocabularies were fixed throughout.

Table 4.1 shows the results, which rather interestingly indicate that using colour on its

own gave superior performance to gradient information. This is somewhat surprising given the importance of lines and textural patterns in identifying planar structure [44], although as Hoiem et al. [66] discovered, colour is important for geometric classification. It is also important to remember that we use spatiograms to represent the distribution of colour words, not simply using the mean or histogram of regions' colours directly. Even so, our hypothesis that using both types of feature together is superior is verified, since the concatenated gradient-and-colour descriptors performed better than either in isolation, suggesting that the two are representing complementary information.

	Gradient	Colour	Gradient&Colour
Classification Accuracy (%)	86.5 (1.8)	92.5 (0.5)	93.9 (2.8)
Orientation Error (deg)	13.1 (0.2)	28.4 (0.3)	17.9 (0.7)

Table 4.1: *Comparison of average classification accuracy and orientation error when using gradient and colour features. Standard deviations are shown in parentheses.*

On the other hand, as we expected, colour descriptors fared much worse when estimating orientation, and combining the two feature types offered no improvement. This stands to reason, since the colour of a region – beyond some weak shading information, or the identity of the surface – should not give any indication to its 3D orientation, whereas texture will [107]. Adding colour information would only serve to confuse matters, and so the best approach is simply to use only gradient information for orientation regression.

To summarise, the image representations we use, having verified this by experiment, are computed as follows: gradient and colour features are created for all salient points in all regions, which are used to create term-document matrices for the two vocabularies. These are used to create a combined 40D topic space, encapsulating gradient and colour information, and this forms the classification topic space. Then, a separate term-document matrix is built using only planar regions, using only gradient features, to create a second 20D gradient only topic space, to be used for regression. This means that each region will have two spatiograms, one of 40 dimensions and one of 20 dimensions, for classification and regression respectively.

Data	Name	Function $k(\mathbf{x}_i, \mathbf{x}_j) =$
Histogram	Linear	$\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$
	Euclidean	$\|\mathbf{x}_i - \mathbf{x}_j\|$
	Bhattacharyya	$\sum_d \sqrt{\mathbf{x}_{id}\mathbf{x}_{jd}}$
	Bhattacharyya Polynomial	$\sum_{q=1}^{Q} \left(\sum_d \sqrt{\mathbf{x}_{id}\mathbf{x}_{jd}}\right)^q$
Spatiogram	Spatiogram	$\rho(\mathbf{s}_i, \mathbf{s}_j)$ (see (3.7))
	Gaussian	$\exp(\frac{-\rho(\mathbf{s}_i,\mathbf{s}_j)^p}{2\sigma^2})$
	Polynomial	$\sum_{q=1}^{Q} \rho(\mathbf{s}_i, \mathbf{s}_j)^q$
	Logistic	$\frac{1}{1+\exp(-\rho(\mathbf{s}_i,\mathbf{s}_j))}$
	Weighted Polynomial	$\sum_{q=1}^{Q} w_d \rho(\mathbf{s}_i, \mathbf{s}_j)^q$

Table 4.2: *Description of the kernel functions used by the RVM, for histograms and spatiograms.*

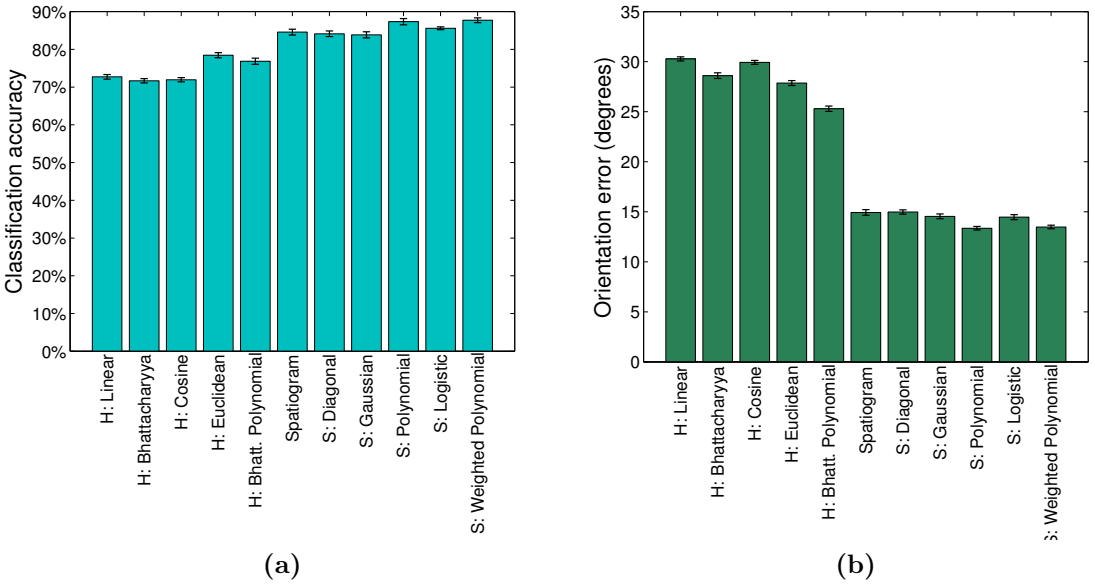
4.1.4 Kernels

Next, we compared the performance of various RVM kernels, for both classification and orientation estimation (this test continued to use only gradient features for ease of interpreting the results). In this experiment, we compared kernels on both histograms and spatiograms.

For histograms, we used various standard comparison functions, including Euclidean, cosine, and Bhattacharyya distances, as well as simply the dot product (linear kernel). For spatiograms, since they do not lie in a vector space, we could only use the spatiogram similarity measure, denoted ρ, from equation 3.7 [97], and functions of it. Variations we used include the original measure, the version with diagonalised covariance, a Gaussian radial basis function, and polynomial functions of the spatiogram similarity. The latter were chosen in order to increase the complexity of the higher dimensionality space and strive for better separability. These functions are described in Table 4.2.

Figure 4.3 shows the results. It is clear that as above the spatiograms out-performed histograms in all cases, though the difference is less pronounced for classification compared to regression. Of the spatiogram kernels, the polynomial function showed superior performance (altering the weights for each degree seemed to make no difference). Therefore we chose the unweighted polynomial sum kernel for subsequent testing (and we set the maximum power to $Q = 4$).

It is also interesting to compare this polynomial sum kernel to the equivalent polynomial sum of the Bhattacharyya coefficient on histograms, which leads to substantially lower



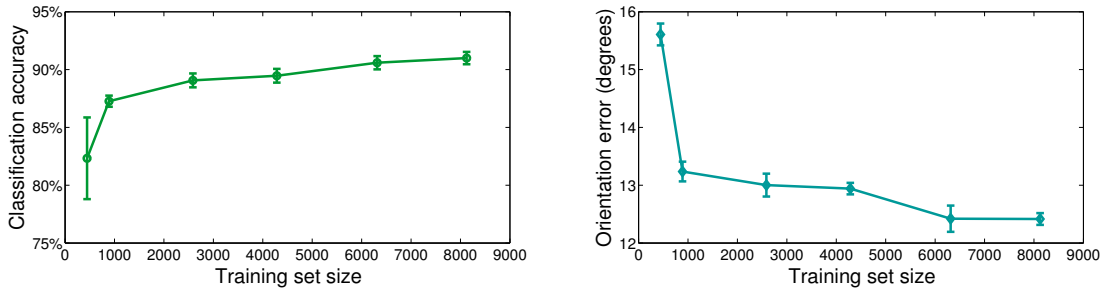
(a) **(b)**

Figure 4.3: *Comparison of different kernel functions, for histogram (first five) and spatiogram (the others) region descriptions. Spatiograms always out-perform histograms, with the polynomial sum kernels proving to be the best.*

performance. This confirms that the superior performance is due to using spatiograms, rather than the polynomial function or Bhattacharyya comparison of histogram bins; while the polynomial function leads to increased performance compared to the regular spatiogram similarity kernel.

4.1.5 Synthetic Data

In Section 3.2.2 we described how, from the initial marked-up training examples, we can synthetically generate many more by reflecting and warping these, to approximate views from different locations. We conducted an experiment to verify that this is actually beneficial to the recognition algorithm (note that all the above experiments were using the marked-up and reflected regions, but not the warped). This was done by again running cross-validation, where the test set was kept fixed (comprising marked-up and reflected regions as before), but for the training set we added progressively more synthetically generated regions. The experiment started with a minimal set consisting of only the marked-up regions, then added the reflected regions, followed by increasing quantities of warped regions, and finally included warped and reflected regions. These were added such that the number of data was increased in almost equal amounts each time.



(a) **(b)**

Figure 4.4: *The effect of adding progressively more synthetically generated training data is to generally increase performance, although the gains diminish as more is added. The best improvement is seen when adding reflected regions (second data point).*

The results of this experiment, on both classification and orientation performance, are shown in Figure 4.4. As expected, adding more data was beneficial for both tasks. However, while performance tended to increase as more training data were used, the gains diminished until adding the final set of data made little difference. This could be because we were only adding permutations of the already existing data, from which no new information could be derived. It is also apparent that the biggest single increase was achieved after adding the first set of synthetic data (second data point) which consisted of the reflected regions. This seems reasonable, since of all the warped data this was the most realistic (minimal geometric distortion) and similar to the test regions.

It seems that synthetically warping data was generally beneficial, and increased performance on the validation set — but as most of the benefit came from reflected images this calls into question how useful or relevant the synthetic warping actually was (especially given there were a very large number of warped regions). This experiment confirmed that using a larger amount of training data was an advantage, and that reflecting our initial set was very helpful, but it may be that including more marked-up data, rather than generating new synthetic regions, would be a better approach.

4.1.6 Spatiogram Analysis

It may be thought that part of the success of spatiograms comes from the way the test regions have been manually segmented, with their shape often being indicative of their actual orientation – for example the height of an upright planar region in the image

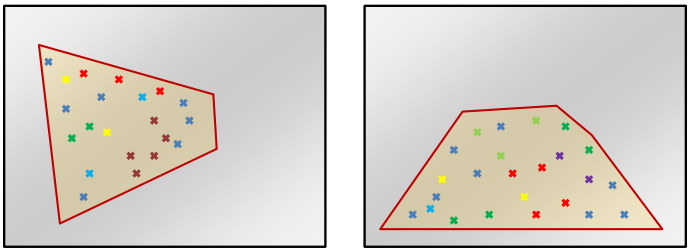


Figure 4.5: *Region shape, even without any visual features, is suggestive of the orientation of manually segmented regions.*

will often diminish as it recedes into the distance. Even without considering the actual features in a region, the shape alone can give an indication of its likely orientation, and such cues are sure to exist when boundaries are outlined by a human. We illustrate this effect in Figure 4.5. This is significant, as unlike histograms, spatiograms use the location of points, which implicitly encodes the region shape. This is a problem, since in 'real' images, such boundary information would not be available; worse, it could bias the classifier into an erroneous orientation simply due to the shape of the region.

To investigate how much of an effect this has on our results, an experiment was carried out where all regions were reduced to being circular in shape (by finding the largest circle which can fit inside the region). We would expect this to reduce performance generally, since there was less information available to the classifier; however, we found that spatiograms still significantly outperformed histograms for orientation regression, as Table 4.3 shows. While circular regions gave worse performance, this was by a similar amount for both representations, and adding spatial information to circular regions continued to boost accuracy. A similar pattern was seen for classification too, where the region shape should not be such a significant cue. To summarise it seems that the region shapes are not a particularly important consideration, confirming our earlier conclusions that spatiograms contribute significantly to the performance of the plane recognition algorithm.

	Histograms	Spatiograms	Cut Hist.	Cut Spat.
Class. Acc. (%)	77.3 (1.0)	87.9 (1.0)	75.3 (1.0)	84.4 (0.9)
Orient. Err. (deg)	24.9 (0.2)	13.3 (0.1)	26.6 (0.2)	17.0 (0.3)

Table 4.3: *Comparison of performance for histograms and spatiograms on regions cut to be uniformly circular, compared to the original shaped regions. Spatiograms are still beneficial, showing this is not only due to the regions' shapes (standard deviation in parentheses).*

4.2 Overall Evaluation

Finally, using the experiments above, we decided upon the following as the best settings to use for testing our algorithm:

- Difference of Gaussians saliency detection, to detect location and scale of salient points.
- Gradient and colour features combined for classification, but gradient only for regression.
- A vocabulary of size 400 and 300 for gradient and colour vocabularies respectively.
- Latent topic analysis, to reduce the dimensionality of words to 20-40 topics.
- Spatiograms as opposed to histograms, to encode spatial distribution information.
- Polynomial sum kernel of the spatiogram similarity measure within the RVM.
- Augmented training set with reflected and warped regions.

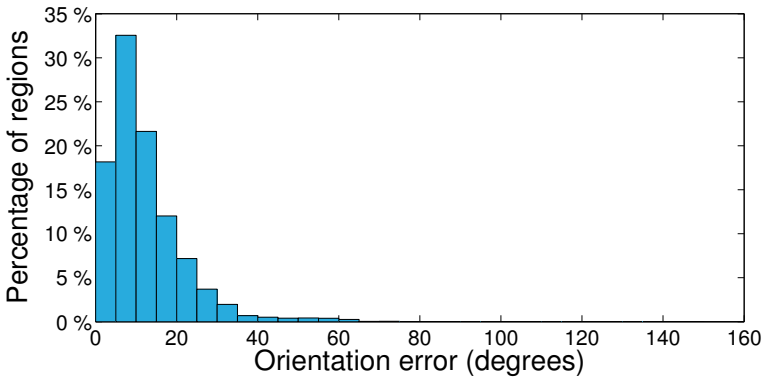


Figure 4.6: *Distribution of orientation errors for cross-validation, showing that the majority of errors were below 15°.*

Using the above settings, we ran a final set of cross-validation runs on the full dataset, with reflected and warped regions, comprising 7752 regions. We used the full set for training but only the marked-up and reflected regions for testing. We observed a mean classification accuracy of 95% (standard deviation $\sigma = 0.49\%$) and a mean orientation error of 12.3° ($\sigma = 0.16°$), over the ten runs. To illustrate how the angular errors were distributed, we show a histogram of orientation errors in Figure 4.6. Although some are large, a significant number are under 15° (72%) and under 20° (84%). This is a very

encouraging result, as even with a mean as low as 12.3°, the errors are not normally distributed, with an obvious tendency towards lower orientation errors.

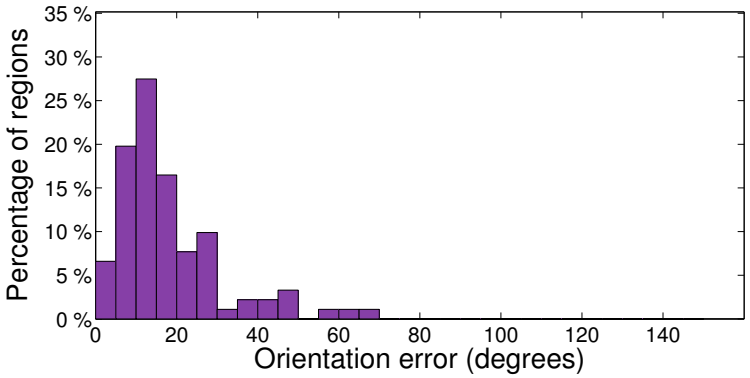
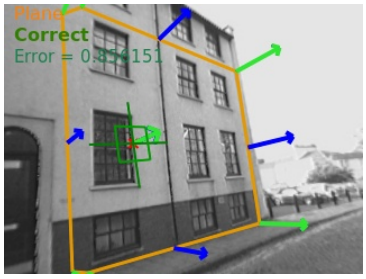


Figure 4.7: *Distribution of errors for testing on independent data.*

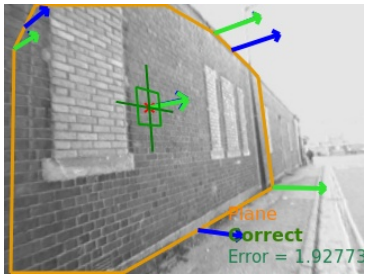
4.3 Independent Results and Examples

While the above experiments were useful, they were not a good test of the method's ability to generalise, since the training and test images, though never actually coinciding, were taken from the same physical locations, and so there would inevitably be a degree of overlap between them.

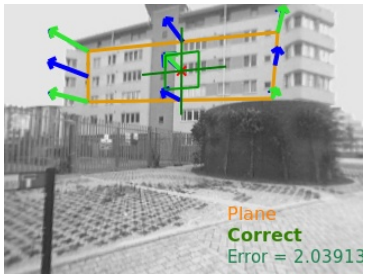
To test the recognition algorithm properly, we used a second dataset of images, gathered from an independent urban location, ensuring the training and test sets were entirely separate. This set consisted of 690 image regions, of an equal number of planes and non-planes (we did not use any reflection or warping on this set), which were marked up with ground truth class and orientation as before. Again, we emphasise these were manually chosen regions of interest, not whole images. Ideally, the intention was to keep this dataset entirely separate from the process of training and tuning the algorithm, and to use it exactly once at the end, for the final validation. Due to the time-consuming nature of acquiring new training data, this was not quite the case, and some of the data would have been seen more than once, as follows. A subset of these data (538 regions) were used to test an earlier version of the algorithm as described in [58] (without colour features and using a different classifier). The dataset was expanded to ensure equal balance between the classes, but the fact remains that most of the data have been used once before. Furthermore, we needed to use this dataset once more to verify a claim made about the vocabulary size: in section 4.1.1 we justified using a larger number of



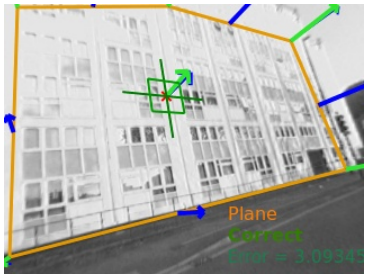
(a) *error = 0.9°*



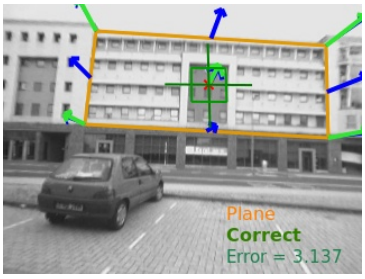
(b) *error = 1.9°*



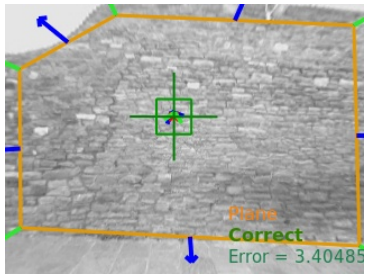
(c) *error = 2.0°*



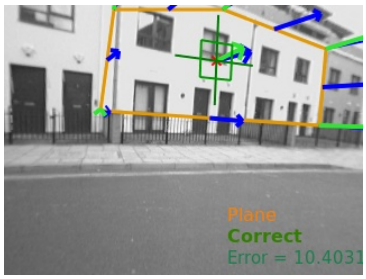
(d) *error = 3.1°*



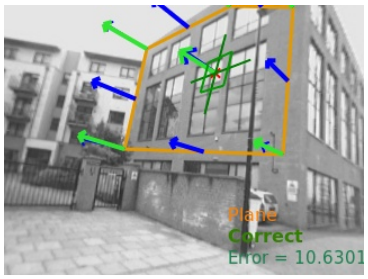
(e) *error = 3.1°*



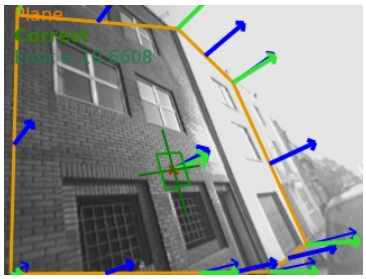
(f) *error = 3.4° **



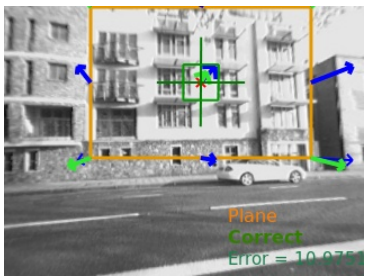
(g) *error = 10.4°*



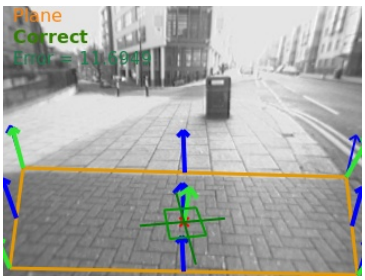
(h) *error = 10.6°*



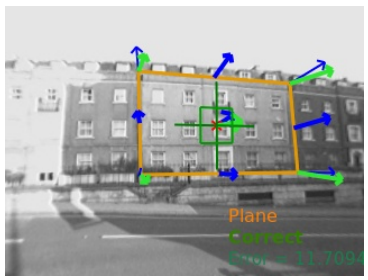
(i) *error = 10.7°*



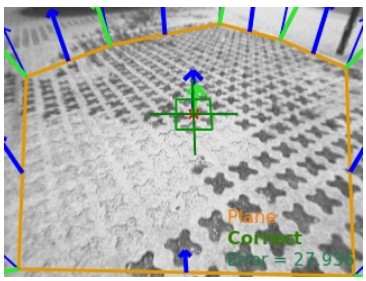
(j) *error = 11.0°*



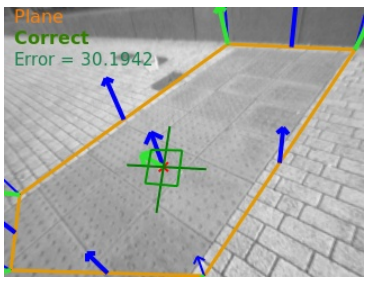
(k) *error = 11.7°*



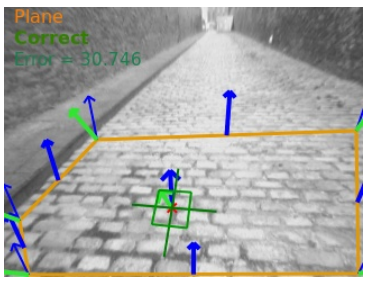
(l) *error = 11.7°*



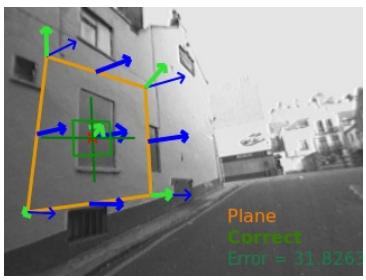
(m) *error = 27.9°*



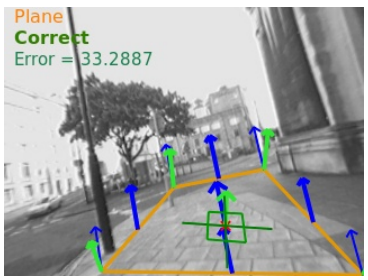
(n) *error = 30.2°*



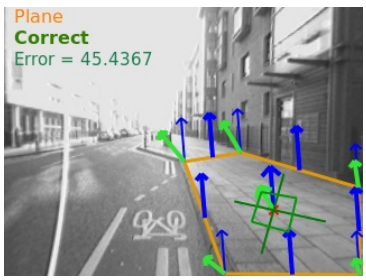
(o) *error = 30.7°*



(p) *error = 31.8°*



(q) *error = 33.2°*



(r) *error = 45.4°*

Figure 4.8: *Example results, selected algorithmically as described in the text, showing typical performance of plane recognition on the independent dataset. The first six (a-f) are selected from the best of the results, the next six (g-l) are from the middle, and the final six (m-r) are from the worst, according to the orientation error. *Note that (f) was picked manually, because it is an important illustrative example.*

words partly by the fact that this should allow the algorithm to generalise better to new data. A test (not described here) was done to see if this was true for our test set (the result showed that using a small vocabulary without topic discovery was indeed detrimental to performance, more so than implied by the proximity of the two curves toward the left of Figure 4.1a). Other than these lapses, the independent dataset was unseen with respect to the process of developing and honing our method.

The results we obtained for plane recognition were a mean classification accuracy of 91.6% and a mean orientation error of 14.5°. We also show in Figure 4.7 a plot of the orientation errors; this is to be compared to the results in Figure 4.6, and shows that here too the spread of orientation errors is good, with the majority of regions being given an accurate normal estimate. This suggests that the algorithm is capable of generalising well to new environments, and supports our principal hypothesis that by learning from a set of training images, it is possible to learn how appearance relates to 3D structure; and that this can be applied to new images with good accuracy. We have not compared this to other methods, due to the lack of appropriately similar work with which to compare (though a comparative evaluation of the full plane detector is presented in Chapter 6) — but we believe that these results represent a good level of accuracy, given the difficulty of the task, and the fact that no geometric information about the orientation is available.

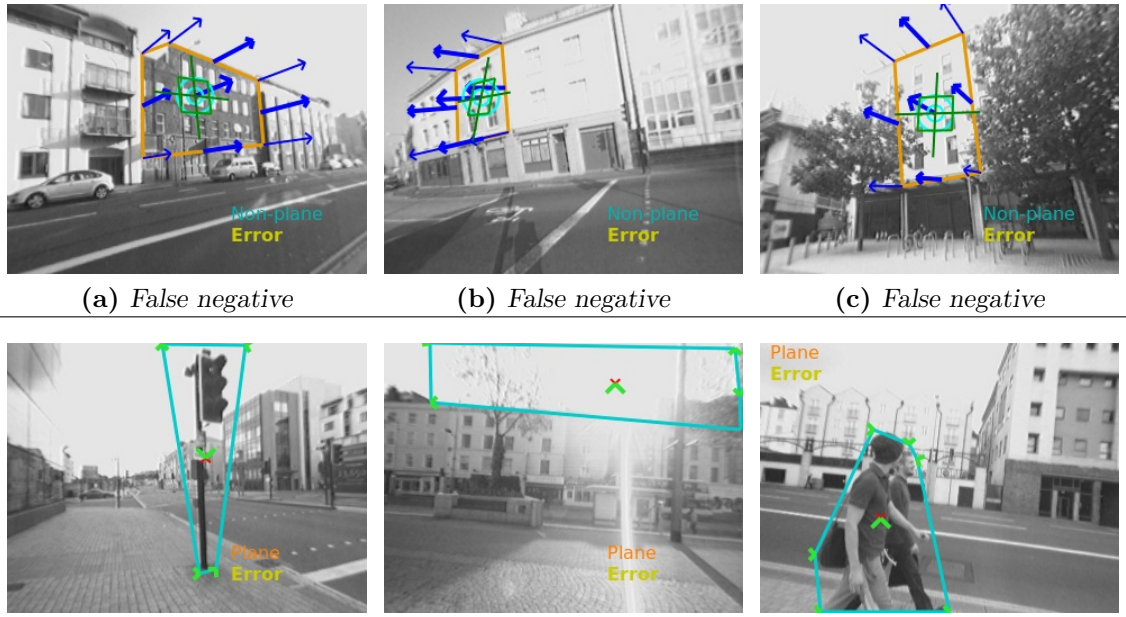
Figures 4.8 to 4.10 show typical example results of the algorithm, on this independent training set. To avoid bias in choosing the results to show, the images were chosen as follows. The correct classifications of planar regions (true positives) were sorted by their orientation error. We took the best ten percent, the ten percent surrounding the median value, and the worst ten percent, then chose randomly from those (six from each set) – these are shown in Figure 4.8. The only exception to this is Figure 4.8f which we picked manually from the best ten percent, because it is a useful illustrative example; all the others were chosen algorithmically. This method was chosen to illustrate good, typical, and failure cases of the algorithm, but does not reflect the actual distribution of errors (c.f. Figure 4.7) which of course has more good than bad results. We then chose randomly from the set of true negatives (i.e. correct identification of nonplanes), false

Figure 4.9: *Correct classification of non-planes, in various situations (selected randomly from the results).*

negatives, and false positives, as shown in the subsequent images, again in order to avoid biasing the results we display.

These results show the algorithm is able to work in a variety of different environments. This includes those with typical Manhattan-like structure, for example 4.8a — but crucially, also those with more irregular textures like Figure 4.8f. While the former may well be assigned good orientation estimates by typical vanishing-point algorithms [73], such techniques will not cope well with the more complicated images.

We also show examples of successful non-plane detection in Figure 4.9. These are mostly composed of foliage and vehicles, but we also observed the algorithm correctly classifiying people and water features.



(a) *False negative* **(b)** *False negative* **(c)** *False negative*

(d) *False positive* **(e)** *False positive* **(f)** *False positive*

Figure 4.10: *Some cases where the algorithm fails, showing false negatives (a-c) and false positives (d-f) (these were chosen randomly from the results).*

It is also interesting to consider cases where the algorithm performs poorly, examples of which are shown in the lower third of Figure 4.8 and in Figure 4.10. The former shows regions correctly classified as planes, but with a large error in the orientation estimate (many of these are of ground planes dissimilar to the data used for training), while the rectangular window on a plain wall in Figure 4.8p may not have sufficiently informative visual information. The first row of Figure 4.10 shows missed planes; we speculate that Figure 4.10c is misclassified due to the overlap of foliage into the region. The second row shows false detections of planes, where Figure 4.10d may be confused by the strong vertical lines, and Figure 4.10e has too little visual information to be much use. These examples are interesting in that they hint at some shortcomings of the algorithm; but we emphasise that such errors are not common, with the majority of regions being classified correctly.

4.4 Comparison to Nearest Neighbour Classification

Relevance vector machines for classification and regression perform well, however it is not straightforward to interpret the reasons why the RVMs behave as they do. That is, we cannot easily learn from them which aspects of the data they are exploiting, or indeed

if they are functioning as we believe they are. We investigated this by using a K-nearest neighbour (KNN) classifier instead. This assigns a class using the modal class of the K nearest neighbours, and the orientation as the mean of its neighbours' orientations. By looking at the regions the KNN deemed similar to each other, we can see why the given classifications and orientations were assigned. Ultimately this should give some insight into the means by which the RVM assigns labels.

The first step was to verify that the KNN and RVM gave similar results, otherwise it would be perverse to claim one gives insight into the other. This was done by running a cross-validation experiment with the KNN on varying amounts of training data. We used the recognition algorithm in the same way as above, except for the final classification step.

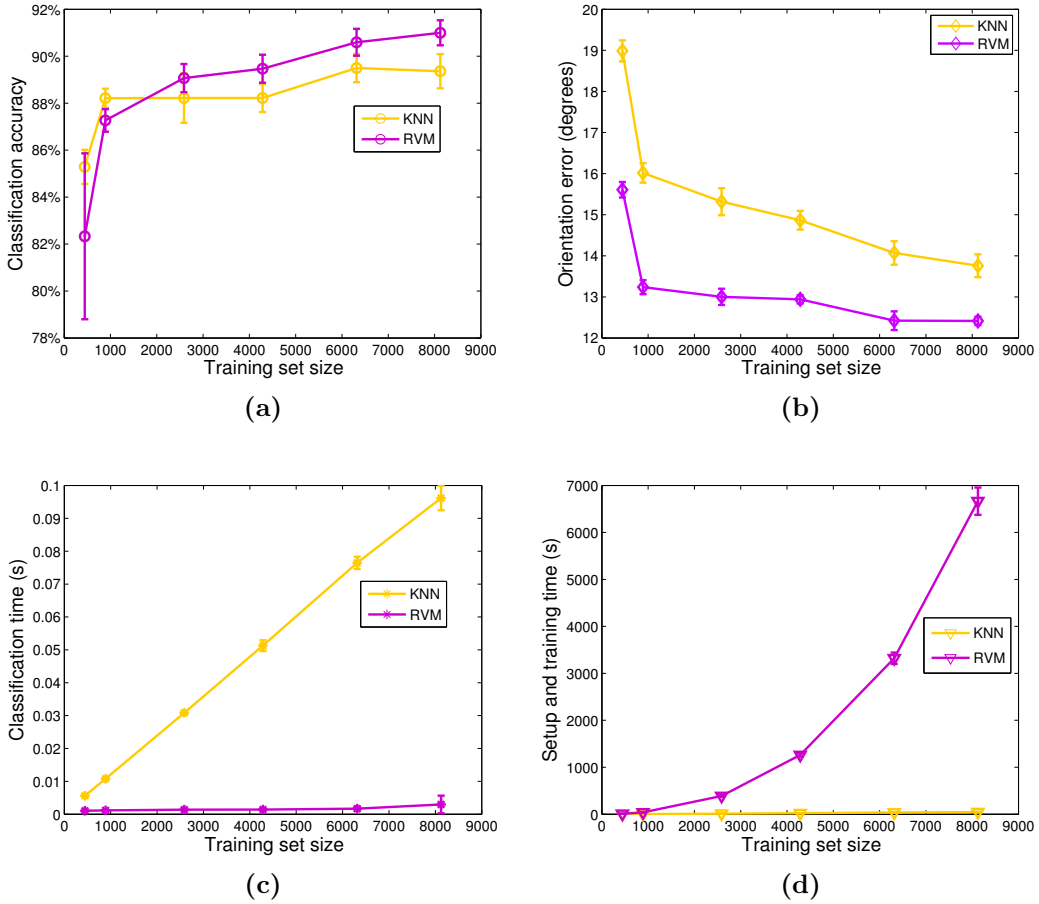


Figure 4.11: *Comparison of RVM and KNN, showing improved accuracy and better scalability to larger training sets (at the expense of a slow training phase for the RVM).*

The results, from ten runs of cross-validation, were a mean classification accuracy of 95.6% ($\sigma = 0.49\%$) and an orientation error of 13.9° ($\sigma = 0.16°$), neither of which were substantially different from results with the RVM. As Figures 4.11a and 4.11b show, performance for both algorithms improved with more training data, and was fairly similar (although the RVM is generally better). Figure 4.11c compares classification time, showing that the KNN was much slower and scaled poorly to larger training sets, justifying our choice of the RVM. The main drawback of the RVM, on the other hand, is its training time (the KNN requires no training). Figure 4.11d compares the setup time (creation of training descriptors and training the classifiers if necessary) for both algorithms, where the time taken for the RVM increased dramatically with training set size.

We also tested the KNN version of the algorithm on our independent test set, and again found similar performance: classification accuracy was 87.8%, while orientation error increased to 18.3°.

4.4.1 Examples

In this section we show example results from the independent test set, with the ground truth and classification overlaid as before, accompanied by the nearest neighbours for each (using $K = 5$ neighbours). Obviously, this set is no longer completely independent or unseen, since it is the same as used above to test the method using the RVM; but no changes were made based on those results before using the KNN.

As above, the examples we show here were not selected manually, but chosen randomly in order to give a fair representation of the algorithm. This was done as above, i.e. taking the best, middle, and worst ten percent of the results for true positive cases (Figure 4.12), and selecting randomly from each set. Examples of true negatives (Figure 4.13), and false positives and negatives (Figure 4.14), are selected randomly.

These images illustrate that classification was achieved by finding other image regions with similar structure. It is interesting to note that the neighbours found were not always perceptually similar, for example Figures 4.12b and 4.12d. This is important, since while an algorithm which matched only to visually similar regions would work to an extent, it would fail when presented with different environments. Figure 4.13c, for example, shows how non-planes can be correctly matched to similar non-planar regions in the training

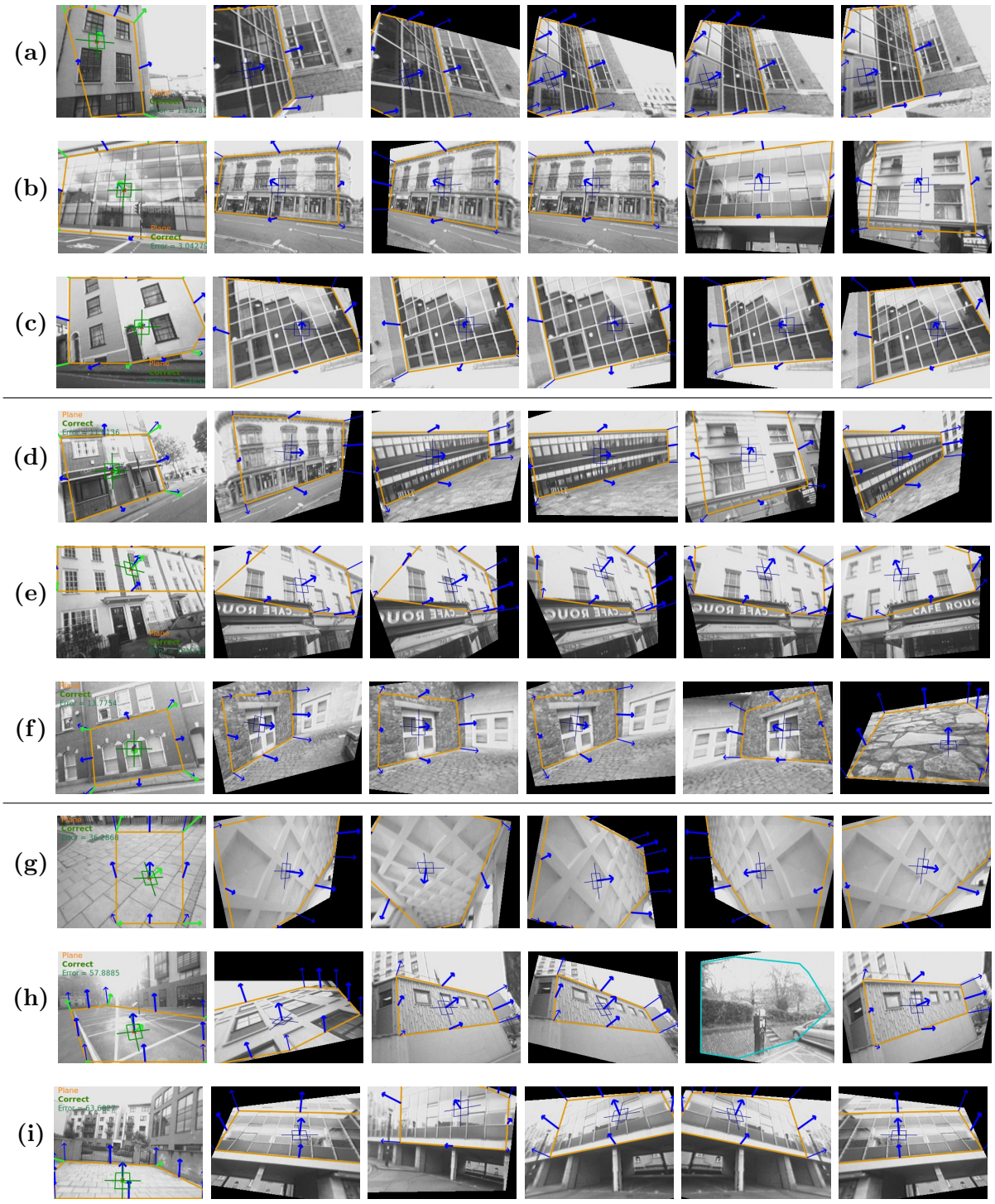


Figure 4.12: *Examples of plane classification and orientation when using a K-nearest neighbour classifier, showing the input image overlaid with the classification and orientation (left), and the five nearest neighbours from the training set. These show triplets of images selected randomly from the best (a-c), middle (d-f), and worst (g-i) examples, for correct plane classification.*

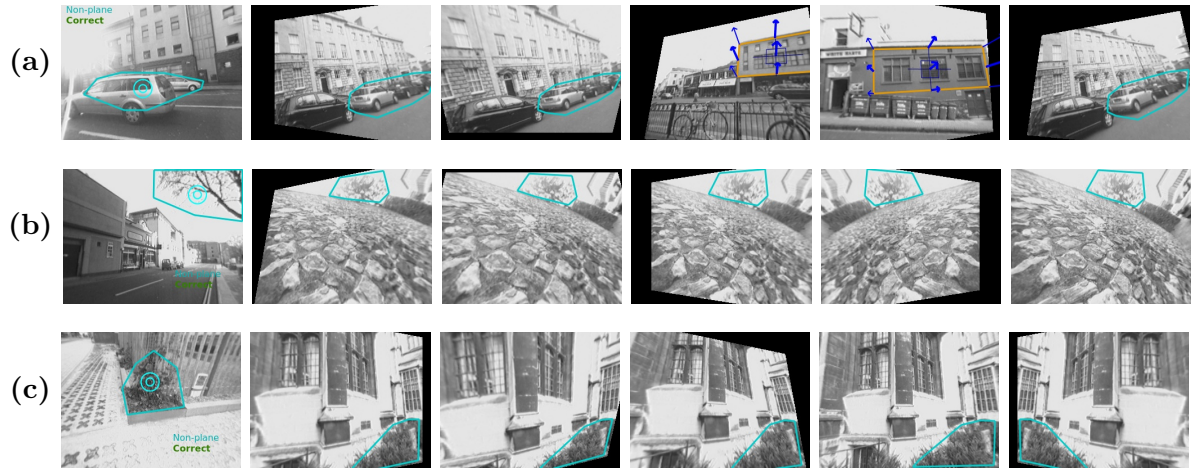


Figure 4.13: *Examples of correct identification of non-planar regions, using a K-nearest neighbour classifier; these examples were chosen randomly from the results.*

set, but Figure 4.13b is also classified correctly, despite being visually different from its neighbours.

It is interesting to note the role that the reflected and warped data play in classification – in many situations several versions of the same original image are found as neighbours (for example Figure 4.12e in particular). This stands to reason as they will be quite close in feature space. On the other hand, the tendency to match to multiple versions of the same image with different orientations can cause large errors, as in Figure 4.12g.

It is also instructive to look at examples where the KNN classifier performed poorly, since now we can attempt to discover why. Figure 4.12i, for example, has a large orientation error. By looking at the matched images, we can see that this is because it has matched to vertical walls which share a similar pattern of lines tending toward a frontal vanishing point, but whose orientation is not actually very similar. Misclassification of a wall occurs in Figure 4.14a where a roughly textured wall was predominantly matched to foliage, resulting in an incorrect non-planar classification (interestingly, there is a wall behind the trees in the neighbouring training regions). Figure 4.14b was also wrongly classified, perhaps due to the similarity between the ventilation grille and a fence. These two examples are interesting since they highlight the fact that what is planar can sometimes be rather ambiguous. Indeed, Figure 4.14c shows the side of a car being classified as planar, which one could argue is actually correct.

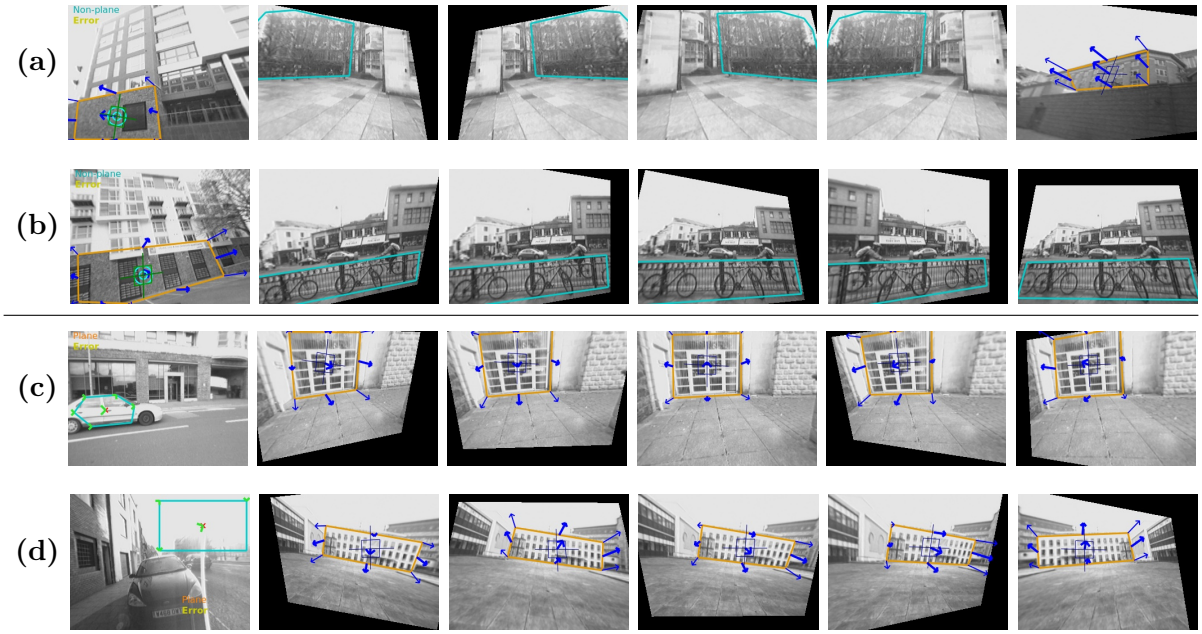


Figure 4.14: *Examples of incorrect classification of regions, showing randomly chosen examples of false negatives (a,b) and false positives (c,d).*

4.4.2 Random Comparison

A further useful property of the KNN was that we could confirm that the low average orientation error we obtained was a true result, not an artefact of the data or test procedure. It is conceivable that the recognition algorithm was simply exploiting some property of the dataset, rather than actually using the features we extracted. For example, if all the orientations were actually very similar, any form of regression would return a low error. We refute this in Figure 4.15b, which shows the spread of orientation errors obtained (in cross-validation) when using randomly chosen neighbours in the KNN, instead of the spatiogram similarity. This means there was no image information being used at all. Compared to results obtained using the KNN classifier (shown in Figure 4.15a), performance was clearly much worse. The histogram of results for the KNN also shows similar performance to the RVM (Figure 4.6 above).

These experiments with the KNN were quite informative, since even an algorithm as simple as the KNN classifier can effectively make use of the visual information available in test data, in an intuitive and comprehensible way, to find structurally similar training examples. The RVM and KNN exhibit broadly similar performance, though they work by different mechanisms. It is reasonable, therefore, to consider the RVM as being a more efficient way of approximating the same goal, that of choosing the regions in feature space most appropriate for a given test datum [11]. The superior performance of the RVM at

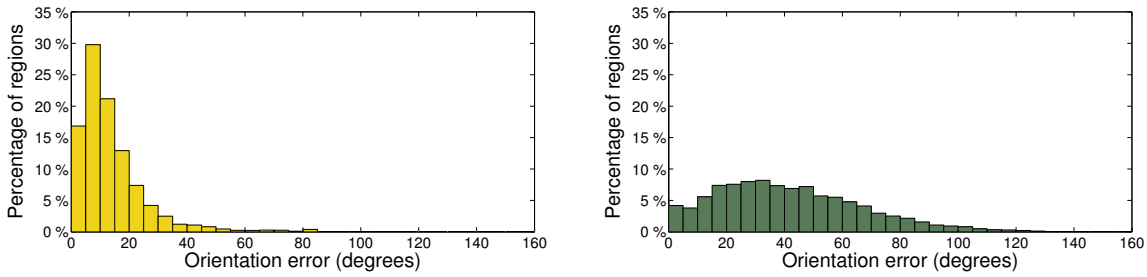


Figure 4.15: *Comparison of the distribution of orientation errors (in cross-validation), for K-nearest neighbour regression (a) and randomly chosen 'neighbours' (b).*

a lower computational cost (during testing), and more efficient handling of large training sets, suggests it is a suitable choice of classifier.

4.5 Summary of Findings

In this chapter, we have shown experimental results for the plane recognition algorithm introduced in Chapter 3. We investigated the effects of various parameters and implementation choices, and showed that the methods we chose to represent the image regions were effective for doing plane classification, and out-performed the simpler alternatives. We also showed that the algorithm generalises well to environments outside the training set, being able to recognise and orient planar regions in a variety of scenes.

4.5.1 Future Work

There are a number of further developments of this algorithm which would be interesting to consider, but fall outside the scope of this thesis. First, while we are using a saliency detector which identifies scale, and using it to select the patch size for descriptor creation, this is not fully exploiting scale information (rather, we are striving to be invariant to it). However, the change in size of similar elements across a surface is an important depth cue [43], and is something we might consider using — perhaps by augmenting the spatiograms to represent this as well as 2D position. On the other hand, whether the notion of image scale detected by the DoG detector has any relation to real-world scale or depth is uncertain. Alternatively, investigation of other types of saliency may

be fruitful.

We could also consider using different or additional feature descriptors. We have already shown how multiple types of descriptor, each with their own vocabulary, can be combined together using topics, and that these descriptors are suited to different tasks. We could further expand this by using entirely different types of feature for the two tasks, for example using sophisticated rotation and scale invariant descriptors for classification, following approaches to object recognition [33, 70], and shape or line based [6] features for orientation estimation.

4.5.2 Limitations

The system as described above has a number of limitations, which we briefly address here. First of all, because it is based on a finite test set, it has a limited ability to deal with new and exceptional images. We have endeavoured to learn generic features from these, to be applicable in unfamiliar scenes, though this may break down for totally new types of environment. Also, while we have avoided relying on any particular visual characteristics, such as isotropic texture, the choices we have made are tailored to outdoor locations, and we doubt whether it would perform well indoors where textured planar surfaces are less prevalent.

The most important and obvious limitation is that it requires a pre-segmented region of interest, both for training and testing, which means it requires human intervention to specify the location. This was sufficient to answer an important question, namely, given a region, can we identify it as a plane or not and estimate its orientation? However, the algorithm would be of limited use in most real-world tasks, and we cannot simply apply it to a whole image, unless we already know a plane is likely to fill the whole image.

The next step, therefore, is to place the plane recognition into a broader framework in which it is useful for both finding and orienting planes. This is the focus of the following chapter, in which we show how it is possible, with a few modifications, to use it as part of a plane detection algorithm, which is able to find planes from anywhere within the image, making it possible to use directly on previously unseen images.

CHAPTER 5

Plane Detection

This chapter introduces our novel plane detection method, which for a single image can detect multiple planes, and predict their individual orientations. It is important to note the difference between this and the plane recognition algorithm presented in Chapter 3, which required the location of potential planar regions to be known already, limiting its applicability in real-world scenarios.

5.1 Introduction

The recognition algorithm forms a core part of our plane detection method, where it is applied at multiple locations, in order to find the most likely locations of planes. This is sufficient to be able to segment planes from non-planes, and to separate planes from each other according to their orientation. This continues with our aim of developing a method to perceive structure in a manner inspired by human vision, since the plane detection method extends the machine learning approach introduced in the recognition method. Again this means we need a labelled training set of examples, though with some key differences.

5.1.1 Objective

First we more rigorously define our objective: in short, what we intend to achieve is the detection of planes in a single image. More precisely, we intend to group the salient points detected in an image into planar and non-planar regions, corresponding to locations of actual planar and non-planar structures in the scene. The planar regions should then be segmented into groups of points, having the same orientation, and corresponding correctly to the planar surfaces in the scene. Each group will have an accurate estimate of the 3D orientation with respect to the camera. This is to be done from a single image of a general outdoor urban scene, without knowledge such as camera pose, physical location or depth information, nor any other specific prior knowledge about the image. This cannot rely upon specific features such as texture distortion or vanishing points. While such methods have been successful in some situations (e.g. [13, 44, 73, 93]), they are not applicable to more general real-world scenes. Instead, by learning the relationship between image appearance and 3D structure, the intention is to roughly emulate how humans perceive the world in terms of learned prior experience (although the means by which we do this is not at all claimed to be biologically plausible). This task as we have described it has not, to the best of our knowledge, been attempted before.

5.1.2 Discussion of Alternatives

Given that we have developed an algorithm capable of recognising planes and estimating their orientation in a given region of an image (Chapter 3), a few possible methods present themselves for using this to detect planes. Briefly, the alternatives are to subdivide the image, perhaps using standard image segmentation algorithms, to extract candidate regions; to find planar regions in agglomerations of super-pixels; and to search for the optimal grouping by growing, splitting and merging segmentations over the salient points.

Amongst the simplest potential approaches is to provide pre-segmented regions on which the plane recogniser can work, using a standard image segmentation algorithm. However, image segmentation is in general a difficult and unsolved problem [36, 130], especially when dealing with more complicated distinctions than merely colour or texture, and it is unlikely that general algorithms would give a segmentation suitable for our purposes. To illustrate the problem, Figure 5.1 shows typical results of applying Felzenszwalb and Huttenlocher's segmentation algorithm [36], with varying settings (controlling roughly

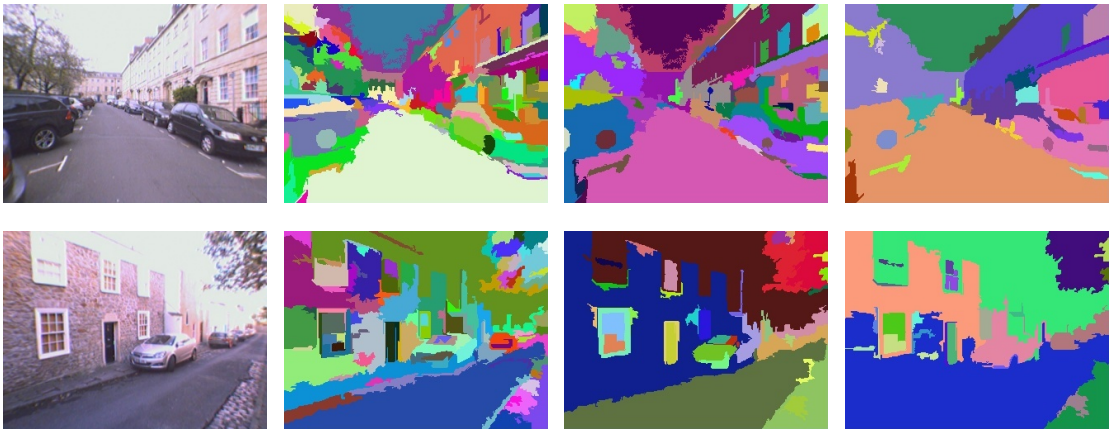


Figure 5.1: *This illustrates the problems with using appearance-based segmentation to find regions to which plane recognition may be applied. Here we have used Felzenszwalb and Huttenlocher's algorithm [36], which uses colour information in a graph-cut framework. While the image is broadly broken down into regions corresponding to real structure, it is very difficult to find a granularity (decreasing left to right) which does not merge spatially separate surfaces, while not over-segmenting fine details.*

the number of segments) to some of our test images. The resulting segments are either too small to be used, or will be a merger of multiple planar regions, whose boundary is effectively invisible (for example the merging of walls and sky caused by an over-exposed image).

An even more basic approach would be simply to tile the image with rectangular blocks, and run plane recognition on each, then join adjacent blocks with the same classification/orientation. This does not depend on any segmentation algorithm to find the boundaries, but will result in a very coarse, blocky segmentation. Choosing the right block size would be problematic, since with blocks too large we gain little information about the location or shape of surfaces, but too small and the recognition will perform too poorly to be of any use (c.f. Chapter 3 and experiment 6.2.1). However, one could allow the blocks to overlap, and while the overlapped sections are potentially ambiguous, this could allow us to use sufficiently large regions while avoiding the blockiness — this is something we come back to later.

Rather than use fixed size blocks, we could deliberately over-segment the image into so-called 'superpixels', where reasonably small homogeneous regions are grouped together. This allows images to be dealt with much more efficiently than using the pixels themselves. Individual superpixels would likely not be large enough to be classifiable on their own, but ideally they would be merged into larger regions which conform to planar or

non-planar surfaces. Indeed, this is rather similar to Hoiem et al. [66], who use segments formed of superpixels to segment the image into geometric classes. They use local features such as filter responses and mean colour to represent individual superpixels, which are very appropriate for grouping small regions (unlike our larger-scale classification). Even so, finding the optimal grouping is prohibitively expensive. In our case we would also have to ensure that there are always enough superpixels in a collection being classified at any time, constraining the algorithm in the alterations it can make to the segmentation.

Another alternative is to initialise a number of non-overlapping regions, formed from adjacent sets of salient points rather than superpixels (an initial segmentation). Plane recognition would be applied to each, followed by iterative update of the regions' boundaries in order to search for the best possible segmentation. The regions could be merged and split as necessary and the optimal configuration found using simulated annealing [34], for example. Alternatively this could be implemented as region growing where a smaller number of regions are initialised centred at salient points, and are grown by adding nearby points if they increase the ability of the region to describe a plane; then split apart if they become too large or fail to be classified confidently. The problem is that while region growing has been successful for image segmentation [129], our situation differs in that the decision as to whether a point should belong to a plane is not local — i.e. there is nothing about the point itself which indicates that it belongs to a nearby region. Rather, it is only after including the point within a region to be classified that anything about its planar or non-plane status can be known. This is different from segmenting according to colour, for instance, which can be measured at individual locations.

These methods would need some rule for evaluating the segmentation, or deciding whether regions should be merged or split, and unfortunately it is not clear how to define which cost function we should be minimising (be it the energy in simulated annealing or the region growing criterion). One candidate is the probability estimate given by the RVM classifier, where at each step, the aim would be to maximise the overall probability of all the classifications, treating the most confident configuration as the best. The problem is that we cannot rely on the probability given by the RVM, partly due to the interesting property of the RVM that the certainty of classification, while generally sensible for validation data, actually tends to increase as the test data move further away from the training data. This is an unfortunate consequence of enforcing sparsity in the model [106], and cannot be easily resolved without harming the efficiency

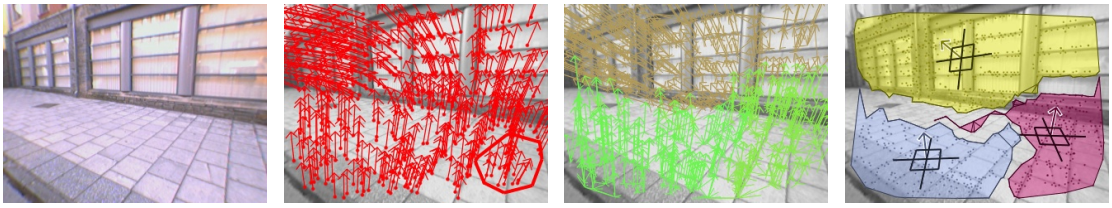
of the algorithm. Indeed, in some early experiments using region growing, we found that classification certainty tended to increase with region size, so that the final result always comprised a single large region no matter what the underlying geometry. This particular problem is specific to the RVM, although similar observations hold for classification algorithms more generally — any machine learning method would be constrained by its training data, and liable to make erroneous predictions outside this scope.

Bearing the above discussion in mind, what we desire is a method that does not rely upon being able to classify or regress small regions (since our plane recogniser cannot do this); avoids the need for any prior segmentation or region extraction (which cannot be done reliably); does not rely on accurate probabilities from a classifier in its error measure (which are hard to guarantee); and will not require exploration or optimisation over a combinatorial search space. In the following, we present such a method, the crucial factor being a step we call region sweeping, and use this to drive a segmentation algorithm at the level of salient points.

5.2 Overview of the Method

This section gives a brief overview of the plane detection method, the details of which are elaborated upon in the following sections, and images representing each of the steps are shown in Figure 5.2. We begin by detecting salient points in the image as before, and assigning each a pair of words based on gradient and colour features. Next we use a process we call region sweeping, in which a region is centred at each salient point in turn, to which we apply the plane recognition algorithm. This gives plane classification and orientation estimation at a number of overlapping locations covering the whole image. We use these to derive the 'local plane estimate' which is an estimate at each individual salient point of its probability of belonging to a plane, and its orientation, as shown in Figure 5.2b. These estimates of planarity and orientation at each point are derived from all of the sweep regions in which the point lies.

While this appears to be a good approximation of the underlying planar structure, it is not yet a plane detection, since we know nothing of individual planes' locations. The next step, therefore, is to segment these points into individual regions, using the local plane estimates to judge which groups of points should belong together, as illustrated in Figure 5.2c. The output of the segmentation is a set of individual planar and non-planar segments, and the final step is to verify these, by applying the plane recognition



(a) (b) (c) (d)

Figure 5.2: *Example steps from plane detection: from the input image (a), we sweep the plane recogniser over the image to obtain a pointwise estimate of plane probability and orientation (b). This is segmented into distinct regions (c), from which the final plane detections are derived (d).*

algorithm once more, on these planar segments. The result, as shown in Figure 5.2d, is a detection of the planar surfaces, comprised of groups of salient points, with an estimate of their orientation, derived from their spatial adjacency and compatibility in terms of planar characteristics.

5.3 Image Representation

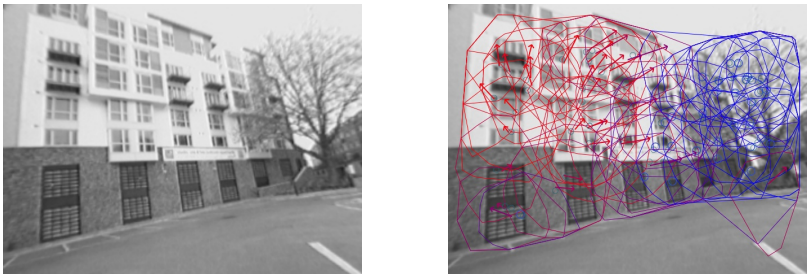
The representation of images will closely follow the description in Section 3.3, except that now we need to consider the entire image, as well as multiple overlapping regions. Because regions overlap, feature vectors are shared between multiple regions.

We begin by detecting a set of salient points $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ over the whole image, using the difference of Gaussians (DoG) detector. For each point, we create a gradient and colour descriptor. Assuming that we have already built bag of words vocabularies, we quantise these to words, so that the image is represented by pairs of words (gradient and colour) assigned to salient points (for more details, refer back to Section 3.3). The further stages of representation – word/topic histograms and spatiograms – depend on having regions, not individual points, and so are not applied yet. Conveniently, this set of salient points and words can be used for any regions occurring in the image.

5.4 Region Sweeping

In order to find the most likely locations of planar structure, we apply a 'region sweeping' stage, using the set V of salient points. Region sweeping creates a set of approximately circular overlapping regions $\mathcal{R}$, by using each salient point $\mathbf{v}_i$ in turn to create a 'sweep region' $R_i \in \mathcal{R}$, using the point as the centroid and including all other points within a fixed radius κ. We define R_i as the set of all salient points within the radius from the centroid: $R_i = \{\mathbf{v}_j | \|\mathbf{v}_j - \mathbf{v}_i\| < \kappa, j = 1, \ldots, n\}$. To speed up the process, we generally use every fourth salient point only, rather than every point, as a centroid. Points are processed in order from the top left corner, to ensure the subset we use is approximately evenly distributed.

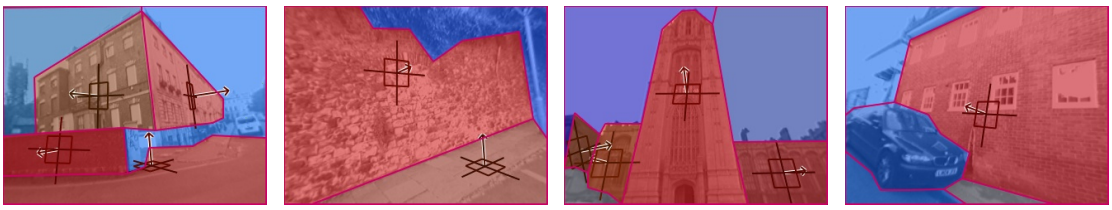
Topic spatiograms are created for each sweep region (see Section 3.3.5). Using these, the plane recognition algorithm can be applied, resulting in an estimate of the probability $p(R_i) \in (0, 1)$ of belonging to the plane class, and an estimate of the orientation $\mathbf{n}(R_i) \in \mathbb{R}^3$ (normal vector), for each sweep region R_i in isolation. The result – before any further processing – can be see in in Figure 5.3, showing multiple overlapping regions $\mathcal{R}$ coloured according to their probability of being planar, with the orientation estimate shown for each planar region. These regions are classified and regressed using RVMs, the training data for which is described in the next section.



(a) **(b)**

Figure 5.3: *Input image (left) and the result of region sweeping (right) — this shows the hull of each region, coloured according to its estimated probability of being a plane (red is planar, blue is non-planar, and shades of purple in between), and the regressed normal for plane regions (only a subset of the regions are shown for clarity).*

Note that the choice of region size is dictated by two competing factors. On one hand, larger regions will give better recognition performance, but at the expense of obtaining coarser-scale information from the image, blurring plane boundaries. Small regions would be able to give precise and localised information, except that accuracy falls as region size



(a) (b) (c) (d)

Figure 5.4: *Examples of manually segmented ground truth data, used for training the classifiers, showing planes (red) with their orientation and non-planes (blue).*

decreases. Fortunately, due to the segmentation method we will introduce soon, this does not mean our algorithm is incapable of resolving details smaller than the region size. We investigate the implications of region size in our experiments presented in Section 6.2.1.

5.5 Ground Truth

Before discussing the next step in the detection algorithm, it is necessary to explain the ground truth data. This is because it is crucial for training the recognition algorithm, as well as validation of the detection algorithm (see Chapter 6).

Unlike in the previous chapters, these ground truth data will contain the location and orientation of *all* planes in the image, not just a region of interest. We begin with a set of images, selected from the training video sequences, and hand segment them into planar and non-planar regions. We mark up the entire image, so that no areas are resigned to being ambiguous. Plane orientations are specified using the interactive vanishing line method as before (refer to Figure 3.1). Examples of such ground truth regions are shown in Figure 5.4.

5.6 Training Data

In Section 3.2, we described how training data were collected by manually selecting and annotating regions from frames from a video sequence. These regions are no longer suitable, because the manually selected region boundaries are not at all like the shapes obtained from region sweeping. As a consequence, classification performance suffers.

Furthermore, we can no longer guarantee that all regions used for recognition will be purely planar or non-planar, since they are not hand-picked but are extracted from images about all salient points. Thus training regions which are cleanly segmented and correspond entirely to one class (or orientation) are not representative of the test data.

5.6.1 Region Sweeping for Training Data

To obtain training data more appropriate to the plane detection task, we gather it using the same method as we extract the sweep regions themselves (see above), but applied instead to ground truth images described in the previous section. When creating regions by grouping all salient points within a radius of the central salient point, we use only a small subset of salient points in each image as centre points, so that we do not get an unmanageably large quantity of data. Since these are ground truth labelled images, we use the ground truth to assign the class and orientation labels to these extracted regions. Inevitably some regions will lie over multiple ground truth segments, as would test data. This is dealt with by assigning to each salient point the class of the ground truth region in which it lies, then the training regions are labelled based on the modal class of their salient points. The same is done for assigning orientation, except for using the geometric median.

We also investigated making use of the estimated probability of each region being a plane, which was calculated as the proportion of planar points in the region. The aim would be to regress such a probability estimate for test regions, but experiments showed no benefit in terms of the resulting accuracy, and so we use the method outlined above. The downside to this approach is that regions whose true class is ambiguous (having for example almost equal numbers of plane and non-plane points) will be forced into one class or the other. However, this will be the same during testing, and so we consider it sensible to leave these in the training set.

This method has the advantage that it allows a very large amount of training data to be extracted with little effort, other than the initial marking up of ground truth. Indeed, this is far more than we can deal with if we do not sparsify the sweeping process considerably. As such it is not necessary to apply warping as well (Section 3.2.2), although we still consider it beneficial to reflect all the ground truth images before extraction begins.

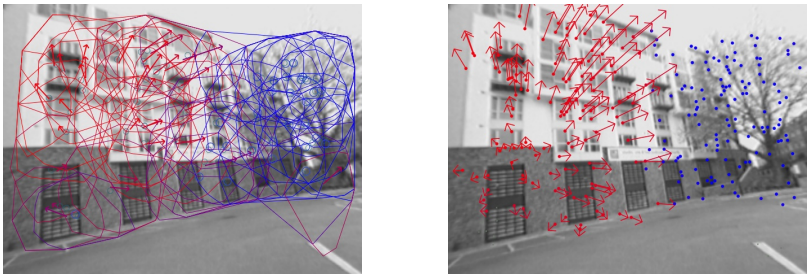
5.6.1.1 Evaluation of Training Data

To verify that gathering an entirely new set of training regions is necessary, we collected a test set of planar and non-planar regions by applying region sweeping to an independent set of ground truth images (those which are later used to evaluate the full algorithm). This produced a new set of 3841 approximately circular regions. When using the plane recognition algorithm on these, using the original hand-segmented training set from Chapter 4, the results were a classification accuracy of only 65.8%, and a mean orientation error of 22°, which would not be good enough for reliable plane segmentation and detection. However, running the same test using the new sweeping-derived training set described here increases classification accuracy to 84.6%, indicating that having an appropriate training set is indeed important. We note with some concern that the mean orientation error decreased only marginally to 21°. Possibly this is because both training and testing data now include regions containing a mixture of different planes, making it more difficult to obtain good accuracy with respect to the ground truth.

5.7 Local Plane Estimate

After running region sweeping, as described in Section 5.4, and classifying these regions with classifiers trained on the data just described, we have a set of overlapping regions $\mathcal{R}$ covering the image. This gives us local estimates of what might be planar, but says nothing about boundaries. Points in the image lying inside multiple regions have ambiguous classification and orientations. We address this by considering the estimate given to each region R_i containing that point as a vote for a particular class and orientation. Intuitively, a point where all the regions in which it lies are planar is very likely to be on a plane. Conversely a point where there is no consensus about its class is uncertain, and may well be on the boundary between regions. This observation is a crucial factor in finding the boundaries between planes and non-planes, and between different planes.

More formally, we use the result of region sweeping to estimate the probability of a salient point belonging to a planar region, by sampling from all possible local regions it could potentially be a part of, before any segmentation has been performed. For points which are likely to be in planar regions, we also estimate their likely orientation, using the normals of all the planar surfaces they could lie on. Each salient point $\mathbf{v}_i$ lies within multiple regions $\mathcal{R}^i \subset \mathcal{R}$, where $\mathcal{R}^i$ is defined as $\mathcal{R}^i = \{R_k | \mathbf{v}_i \in R_k\}$; that is, the subset



(a) (b)

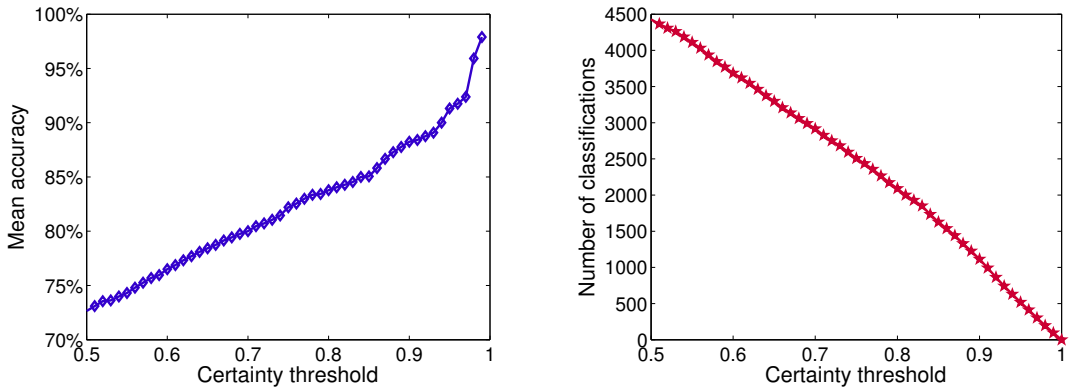
Figure 5.5: *Using the sweep regions (left), we obtain the local plane estimate, which for each point, located at* $\mathbf{v}_i$*, assigns a probability* q_i *of belonging to a plane, and an orientation estimate* $\mathbf{m}_i$ *(right). Probability is coloured from red to blue (for degree of plane to non-plane), and orientation is shown with a normal vector (only a subset of points are shown for clarity).*

of regions R_k in which $\mathbf{v}_i$ appears. Each point $\mathbf{v}_i$ is given an estimate of its probability of being on a plane, denoted q_i, and of its normal vector $\mathbf{m}_i$, calculated as follows:

$$\begin{aligned} q_i &= \zeta(\{p(R_k)|R_k \in \mathcal{R}^i\}) \\ \mathbf{m}_i &= \zeta^G(\{\mathbf{n}(R_k)|R_k \in \mathcal{R}^i, p(R_k) > 0.5\}) \end{aligned} \tag{5.1}$$

where ζ and ζ^G are functions calculating the median and geometric median in $\mathbb{R}^3$ respectively. Note that $\mathbf{m}_i$ is calculated using only regions whose probability of being a plane is higher than for a non-plane. To clarify, equation 5.1 describes how the plane probability and normal estimate for the point i come from the median of the regions in $\mathcal{R}^i$. We use the median rather than the mean since it is a more robust measure of central tendency, to reduce the effect of outliers, which will inevitably occur when using a non-perfect classifier. Figure 5.5 illustrates how the sweep regions lead to a pointwise local plane estimate.

In order to improve the accuracy of the local plane estimate, we discard regions whose classification certainty is below a threshold. The classification certainty is defined as the probability of belonging to the class it has been assigned, and thus is $p(R_i)$ and $1-p(R_i)$, for planar and non-planar regions respectively. This discarding of classified regions is justified by a cross-validation experiment (similar to those in Section 4.1). As shown in Figure 5.6, as the threshold is increased, to omit the less certain classifications, the mean classification accuracy increases. This is at the expense of decreasing the number of regions which can be used. It is this thresholding for which having a confidence



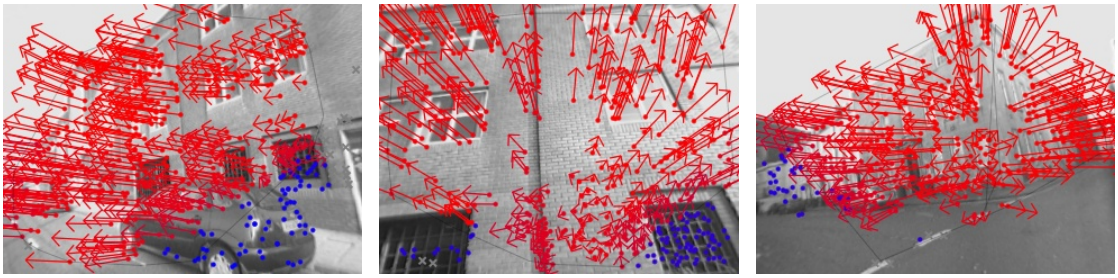
(a) **(b)**

Figure 5.6: *As the threshold on classifier certainty increases, less confident regions are discarded, and so accuracy improves (a); however, the number of regions remaining drops (b).*

value for the classification, provided by the RVM, is very useful, and is an advantage over having a hard classification. The effect of this is to remove the mid-range from the spread of probabilities, after which the median is used to choose which of the sides (closer to 0 or 1) is larger. This is equivalent to a voting scheme based on the most confident classifications. However, our formulation can allow more flexibility if necessary, for example using a different robust estimator.

Since there are usually a large number of overlapping regions available for each point, the removal of a few is generally not a problem. In some cases, of course, points are left without any regions that they lie inside ($\mathcal{R}^i = \emptyset$), and so these points are left out of subsequent calculations. Such points are in regions where the classifier cannot be confident about classification, and so should play no part in segmentation.

The result is an estimate of planarity for each point, which we will refer to as the local plane estimate — examples are shown in Figure 5.7. We have now obtained a representation of the image structure which did not require the imposition of any boundaries, and circumvented the problem of being unable to classify arbitrarily small regions. It is encouraging to note that at this stage, the underlying planar structure of the image is visible, although as expected it is less well defined at plane boundaries, where orientation appears to transition smoothly between surfaces.



(a) (b) (c)

Figure 5.7: *Examples of local plane estimates, for a variety of images; colour, from red to blue, shows the estimated probability of belonging to a plane, and the normal vector is the median of all planar regions in which the point lies.*

5.8 Segmentation

Although the local plane estimate produced above is a convincing representation of the underlying structure of the scene – showing where planes are likely to be, and their approximate orientation – this is not yet an actual plane detection. Firstly, plane boundaries remain unknown, since although we know the estimates at each point, it is not known which points are connected to which other points on a common surface. Secondly, each pointwise local plane estimate is calculated from all possible surfaces on which the point might lie. As such, this is not accurate, which is especially clear in Figure 5.7c where points near plane-plane boundaries take the mean of the two adjoining faces.

This section describes how the local plane estimate is used to segment points from each other to discover the underling planar structure of the scene, in terms of sets of connected, coplanar points. This consists of segmenting planes from non-planes, deciding how many planes there are and how they are oriented, then segmenting planes from each other.

5.8.1 Segmentation Overview

Our segmentation is performed in two separate steps. This is because we found it was not straightforward to select a single criterion (edge weight, energy function, set of clique potentials etc.) to satisfactorily express the desire to separate planes and non-planes, while at the same time separating points according to their orientation. Therefore, we first segment planar from non-planar regions, according to the probability of belonging

to a plane given by the local plane estimate. Next, we consider only the resulting planar regions, and segment them into distinct planes, according to their orientation estimate, for which we need to determine how many planes there are. We do this by finding the modes in the distribution of normals observed in the local plane estimate, which represent the likely underlying planes. This is based on the quite reasonable assumption that there are a finite number of planar surfaces which we need to find.

5.8.2 Graph Segmentation

We formulate the segmentation as finding the best partition of a graph, using a Markov random field (MRF) framework. A MRF is chosen since it can well represent our task, which is for each salient point, given its observation, to find its best 'label' (for plane class and orientation), while taking into account the values of its neighbours. The values of neighbouring points are important, since as with many physical systems we can make the assumption of smoothness. This means we are assuming that points near each other will generally (discontinuities aside) have similar values [78]. Without the smoothness constraint, segmentation would amount simply to assigning each point to its nearest class label, which would be trivial, but would not respect the continuity of surfaces. Fortunately, optimisation of MRFs is a well-studied problem, and a number of efficient algorithms exist.

First we build the graph to represent the 2D configuration of points in the image and their neighbourhoods. We do this with a Delaunay triangulation of the salient points to form the edges of the graph, using the efficient S-Hull implementation[1] [118]. We modify the standard triangulation, first by removing edges whose endpoints never appear in a sweep region together. This is because for two points which are never actually used together during the plane sweep recognition stage, there is no meaningful spatial link between them, so there should be no edge. To obtain a graph with only local connectivity, we impose a threshold on edge length (typically 50 pixels), to remove any undesired long-range effects.

[1] The code is available at www.s-hull.org/

5.8.3 Markov Random Field Overview

A MRF expresses a joint probability distribution on an undirected graph, in which every node is conditionally independent of all other nodes, given its neighbours [78] (the Markov property). This is a useful property, since it means that the global properties of the graph can be specified by using only local interactions, if certain assumptions are made. In general the specification of the joint probability of the field would be intractable, were it not for a theorem due to Hammersley and Clifford [60] which states that a MRF is equivalent to a Gibbs random field, which is a random field whose global properties are described by a Gibbs distribution. This duality enables us to work with the MRF in an efficient manner and to find optimal values, in terms of maximising the probability.

If we define a MRF over a family of variables $F = \{F_1, \ldots, F_N\}$, each taking the value $F_i = f_i$, then $f = \{f_1, \ldots, f_N\}$ denotes a particular realisation of F, where some f describes the labels assigned to each node, and is called a configuration of the field. In this context, the Gibbs distribution is expressed as

$$P(f) = Z^{-1} \times e^{-U(f)} \tag{5.2}$$

where $U(f)$ is the energy function, and for brevity we have omitted the temperature parameter from the exponent. Z is the partition function:

$$Z = \sum_{f \in \mathbb{F}} e^{-U(f)} \tag{5.3}$$

which is required to ensure the distribution normalises to 1. Since this must be evaluated over all possible configurations f (the space $\mathbb{F}$) evaluating (5.2) is generally intractable. However, since Z is the same for all f, it is not needed in order to find the optimal configuration of the field — i.e. it is sufficient that $P(f) \propto e^{-U(f)}$. The optimal configuration is the f which is most likely (given the observations and any priors), so the aim is to find the $f = f^*$ which maximises the probability P.

The energy function $U(f)$ sums contributions from each of the neighbourhood sets of the graph. Since the joint probability is inversely related to the energy function, minimising the energy is equivalent to finding the MAP (maximum a-posteriori) configuration f^* of the field. Thus, finding the solution to a MAP-MRF is reduced to an optimisation problem on $U(f)$. This is generally written as a sum over clique potentials, $U(f) =$

$\sum_{c \in \mathcal{C}} V_c(f)$, where a clique is a set of nodes which are all connected to each other, and V_c is the clique potential for clique c (in the set of all possible cliques $\mathcal{C}$). In our case, deal with up to second order cliques – that is, single nodes and pairs of nodes – so the energy function can be expressed as:

$$U(f) = \sum_{\{i\} \in \mathcal{C}_1} V_1(f_i) + \sum_{\{i,i'\} \in \mathcal{C}_2} V_2(f_i, f_{i'}) \quad = \quad \sum_{i \in \mathcal{S}} V_1(f_i) + \sum_{i \in \mathcal{S}} \sum_{i' \in \mathcal{N}_i} V_2(f_i, f_{i'}) \quad (5.4)$$

where V_1 and V_2 are the first and second order clique potentials respectively. The left hand equation expresses the energy as a sum over the two clique potentials, over the set of all first order $\mathcal{C}_1$ (single nodes) and second order $\mathcal{C}_2$ (pairs connected by an edge) cliques; the right hand side expresses this more naturally as a sum of potentials over nodes, and a sum over all the neighbourhoods for all the nodes; $\mathcal{S}$ is the set of all nodes in the graph and $\mathcal{N}_i \subset \mathcal{S}$ denotes all the neighbours of node i. The two clique potentials take into account respectively the dependence of the label on the observed value, at a single node, and the interaction between pairs of labels at adjacent nodes (this is how smoothness is controlled). In summary, it is this energy $U(f)$, calculated using the variables f_i of the configuration f and the functions $V_{1,2}$, which is to be minimised, in order to find the best configuration f^* (and thus the optimal labels f_i^* for all the nodes).

We use iterative conditional modes (ICM) to optimise the MRF, which is a simple but effective iterative algorithm developed by Besag [8]. The basic principle of ICM is to set every node in turn to its optimal value, given the current value of its neighbours, monotonically decreasing the total energy of the MRF. The updates will in turn alter the optimal value for already visited nodes, so the process is repeated until convergence (convergence to a (local) minimum can be proven, assuming sequential updates). ICM is generally quite fast to converge, and although it may not be the most efficient optimisation algorithm, it is very simple to implement, and was found to be suitable for our task.

5.8.4 Plane/Non-plane Segmentation

The first step is to segment planes from non-planes. We do this using a MRF as follows: let p represent a configuration of the field, where each node $p_i \in \{0, 1\}$ represents the class

of point i (1 and 0 being plane and non-plane, respectively). We then seek the optimal configuration p^*, defined as $p^* = \arg\min_p U(p)$, where $U(p)$ represents the posterior energy of the MRF:

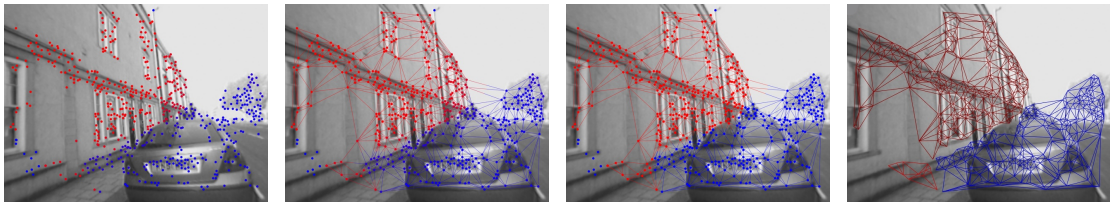
$$U(p) = \sum_{i \in \mathcal{S}} V_1(p_i, q_i) + \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{N}_i} V_2(p_i, p_j) \tag{5.5}$$

Here, q_i is the observation at point i, which is the estimated probability of this point belonging to a plane, obtained as in equation 5.1. The set $\mathcal{S}$ contains all salient points in the image (assuming they have been assigned a probability). The functions V_1 and V_2 are the single site and pair site clique potentials respectively, defined as

$$V_1(p_i, q_i) = (p_i - q_i)^2 \qquad V_2(p_i, p_j) = \delta_{p_i \neq p_j} \tag{5.6}$$

where $\delta_{p_i \neq p_j}$ has value 0 iff p_i and p_j are equal, 1 otherwise. Here we express the function V_1 with two arguments, since it depends not only on the current value of the node but also its oberved value. This function penalise deviation of the assigned value p_i at a point from its observed value q_i, using a squared error, since we want the final configuration to correspond as closely as possible to the local plane estimates. We desire pairs of adjacent nodes to have the same class, to enforce smoothness where possible, so δ in function V_2 returns a higher value (1) to penalise a difference in its arguments.

Each p_i is initialised to the value in $\{0, 1\}$ which is closest to q_i (i.e. we threshold the observations to obtain a plane/non-plane class) and optimise using ICM. We generally find that this converges within a few iterations, since the initialisation is usually quite good. It tends to smooth the edges of irregular regions and removes small isolated segments. After this optimisation, each node is set to its most likely value (plane or not), given the local plane estimate and a smoothness constraint imposed by its neighbours, so that large neighbourhoods in the graph will correspond to the same class. The process is illustrated with examples in Figure 5.8. Finally, segments are extracted by finding the connected components corresponding to planes and non-planes, which now form distinct regions (Figure 5.8d).



(a) (b) (c) (d)

Figure 5.8: *The process of segmenting planes from non-planes. Using the probabilities estimated at each point from region sweeping (a), we initialise a Markov random field (b); this is optimised using iterative conditional modes (c), resulting in a clean segmentation into smooth disjoint regions of planes and non-planes (red and blue respectively) (d).*

5.8.5 Orientation Segmentation

Once planar regions have been separated from non-planar regions using the above MRF, we are left with regions supposedly consisting only of planes. However, except in very simple images, these will be made up of multiple planes, with different orientations. The next step is to separate these planes from each other, using the estimated orientation $\mathbf{m}_i$ at each salient point.

This is done by optimising a second MRF, with the goal of finding the points belonging to the same planar surfaces. This is defined on the subgraph of the above which contains only points in planar segments. This graph may consist of multiple independent connected components, but this makes no difference to the formulation of the MRF.

In contrast to the simplicity of the first MRF, which was a two-class problem, the values we have for plane normals are effectively continuously valued variables in $\mathbb{R}^3$, where we do not know the values of the true planes. Some brief experimentation suggested that attempting to find an energy function which is able to not only segment the points, but also find the correct number of planes, was far from straightforward. Generally, a piecewise constant segmentation approach did enforce regions of constant orientation, but typically far too many, forming banding effects (essentially discretising the normals into too-fine graduations).

We take an alternative approach, by using mean shift to find the modes of the kernel density estimate of the normals in the image. This is justified as follows: we assume that in the image, there are a finite number of planar surfaces, each with a possibly different orientation, to which all of the planar salient points belong. This implies that

close to the observed normals, there are a certain number, as yet unknown, of actual orientations, from which all these observations are derived. The task is to find these underlying orientations, from the observed normals. Once we know the value of these, the problem reduces to a discrete, multi-class MRF optimisation, which is easily solved using ICM. The following sections introduce the necessary theory and explain how we use this to find the plane orientations.

5.8.5.1 Orientation Distribution and the Kernel Density Estimate

Kernel density estimation is a method to recover the density of the distribution of a collection of multivariate data. Conceptually this is similar to creating a histogram, in order to obtain a non-parametric approximation of a distribution. However, histograms are always limited by the quantisation into bins, introducing artefacts. Instead of assigning each datum to a bin, the kernel density estimate (KDE) places a kernel at every datum, and sums the results to obtain an estimate of the density at any point in the space, as we illustrate in Figure 5.9 with the example of a 1D Gaussian kernel.

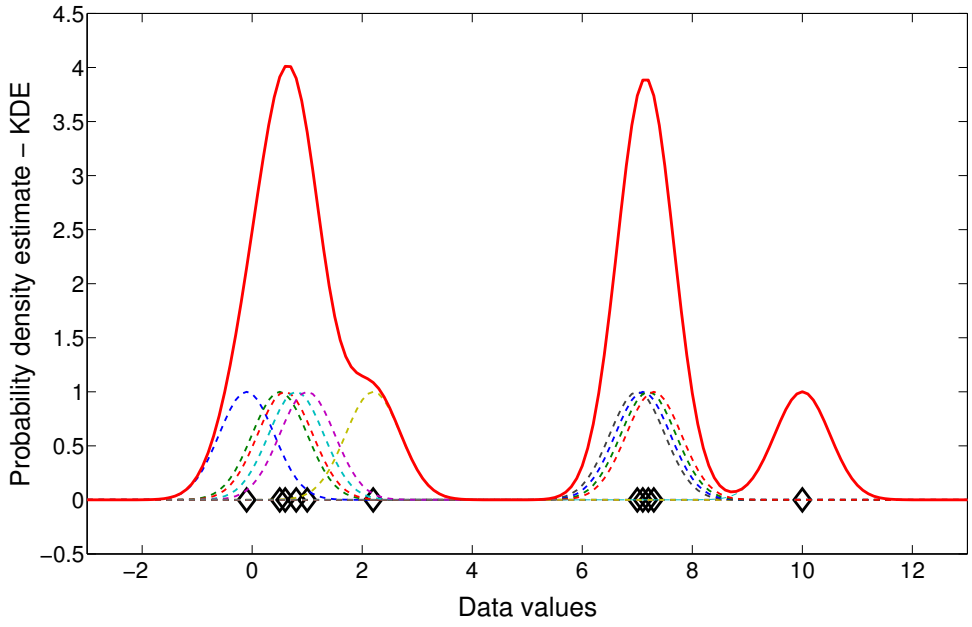


Figure 5.9: *This illustrates the central principle behind the kernel density estimate. For 1D data (the black diamonds), we estimate the probability density by placing a kernel over each point (here it is a Gaussian with $\sigma = 0.5$), drawn with dashed lines. The sum of all these kernels, shown by the thick red line, is the KDE. Clusters of nearby points' kernels sum together to give large peaks in the density; outlying points give low probability maxima.*

In general, the KDE function $\hat{f}$ for multivariate data $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, evaluated at a point $\mathbf{y}$ within the domain of $\mathbf{X}$ is defined as [23]

$$\hat{f}(\mathbf{y}) = \frac{1}{Nh^d} \sum_{i=1}^{N} K\left(\frac{\mathbf{y} - \mathbf{x}_i}{h}\right) \tag{5.7}$$

where K is the kernel function, h is the bandwidth parameter, and d is the number of dimensions.

Following the derivation of Comaniciu and Meer [23], the kernel function is expressed in terms of a profile function k, so that $K(\mathbf{x}) = ck(\|\mathbf{x}\|^2)$, where c is some constant. For a Gaussian kernel, the profile function is

$$k(x) = \exp(-\frac{1}{2}x) \tag{5.8}$$

which leads to an isotropic multivariate Gaussian kernel:

$$K(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right) \tag{5.9}$$

where the constant c became $(2\pi)^{-\frac{d}{2}}$. Substituting the Gaussian kernel function (5.9) into equation 5.7 yields the function for directly calculating the KDE at any point:

$$\hat{f}(\mathbf{y}) = \frac{1}{Nh^d} \sum_{i=1}^{N} \frac{1}{(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{1}{2}\left\|\frac{\mathbf{y} - \mathbf{x}_i}{h}\right\|^2\right) \tag{5.10}$$

which states that the estimate of the density at any point is proportional to the sum of kernels placed at all the data, evaluated at that point, as the illustration above showed.

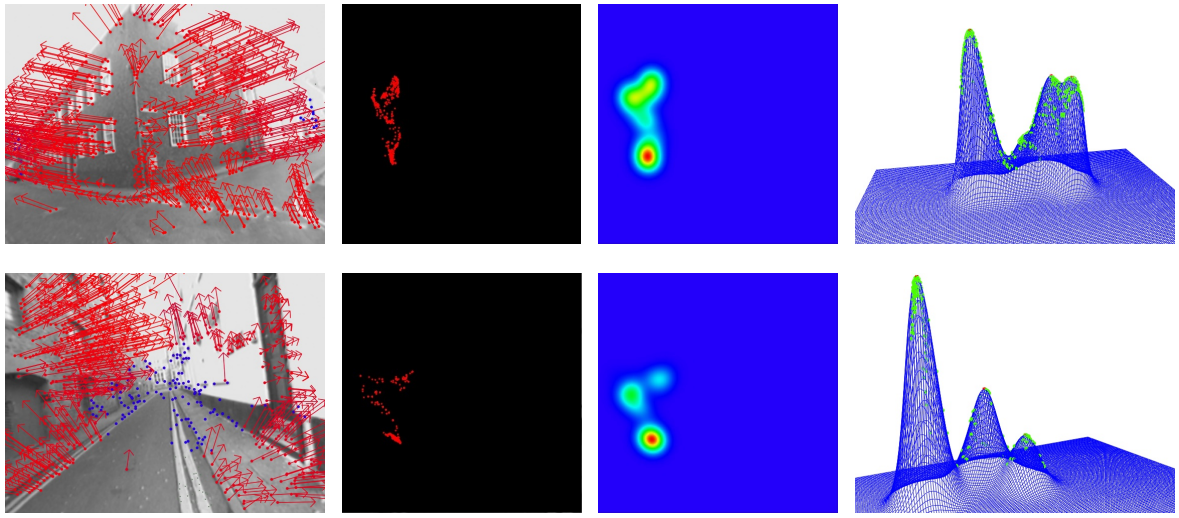
In our case, we use the KDE to recover the distribution of normals observed in the

image, based on the hypothesis that while the normals are smoothly varying due to the region sweeping, they will cluster around the true plane orientations. In a two-plane image, for example, two distinct modes should be apparent in the density estimate. The values of these modes will then correspond to the normals around which the observations are clustered, and which will be used to segment them. These normals become the discrete set of class labels for the MRF. If we assume the data vary smoothly, and are approximately normally distributed about the true normals, this justifies the use of a Gaussian kernel function.

The normal vectors have so far been represented in $\mathbb{R}^3$, but can be more compactly represented with spherical coordinates, using a pair of angles θ and ϕ. An important issue when dealing with angular values is to avoid wrap-around-effects (angles of 350° and -10° are the same, for example). Conveniently, because we only represent angles facing toward the camera (because we cannot see planes facing away), only half of the space of all θ, ϕ is used (to be precise, the hemisphere $\theta \in [\frac{\pi}{2}, \frac{3\pi}{2}]$ and $\phi \in [0, 2\pi]$), and so with careful choice of parameterisation, we can avoid any such wrapping effects, and stay away from the 'edge' of the parameter space (and so our formulae below may appear different from standard spherical coordinates). The transformation of a normal vector $\mathbf{n} = (n_x, n_y, n_z)^{\mathrm{T}}$ to and from angular representation is therefore:

$$\begin{aligned} \theta &= \tan^{-1}\left(\tfrac{n_x}{n_z}\right) & \qquad n_x &= \sin(\theta)\sin(\phi) \\ \phi &= \cos^{-1}\left(\tfrac{n_y}{|\mathbf{n}|}\right) & \qquad n_y &= \cos(\phi) \\ & & \qquad n_z &= \cos(\theta)\sin(\phi) \end{aligned} \tag{5.11}$$

To use the KDE we need to set the value of the bandwidth parameter h. There are various methods outlined in the literature for automatic bandwidth selection [22], or even the use of varying bandwidths according to the data [24]. For convenience, we set our bandwidth by observing the performance on training data, to a value of 0.2 radians (see our experiments in Section 6.2.2) — though we acknowledge that more intelligent selection or adaptation of the bandwidth would be a worthwhile area for exploration.



(a) (b) (c) (d)

Figure 5.10: *Visualisation of the kernel density estimate (KDE) for the distribution of normals in an image. The estimated normals from the local plane estimate (a) are used to calculate a KDE with a Gaussian kernel. Plots (b) show all the normals represented in θ, ϕ space, and in (c) the colour map of the KDE (red denotes higher probability density). (d) shows a 3D representation, making the structure of the density clearer.*

5.8.5.2 KDE Visualisation

Because the kernel density estimates are calculated in the 2D space of angles, the process is easy to understand and visualise. The density can be represented as a 2D image, with the horizontal and vertical axes corresponding to the two angles, where the KDE at each point is represented by the colour of the pixel. Since the 2D heat map is not particularly easy to interpret, we can also show the KDE visualised as a 3D surface, where the axes in the horizontal plane correspond to the angles and the height is the magnitude of the density estimate at the point (the mesh is built simply by connecting points in a 4-way grid).

In Figure 5.10 we show some examples, consisting of the local plane estimate (Section 5.7), showing all the normals in the image, plus the 2D and 3D representations of the KDE. In each example, one can clearly see the peaks in the KDE corresponding to the dominant planes in the scene, whose relative height (density estimate) roughly corresponds to the size of the plane (i.e. the number of observations relating to it). An important point about the KDE is that it requires no *a priori* knowledge of the number of modes (in contrast to K-means, for example), and is entirely driven by the data. This

gives us an elegant way of choosing the number of planes present in the image.

5.8.5.3 Mode Finding and Mean Shift

The above examples suggest that the KDE is a suitable way of describing the underlying plane orientations, but does not yet give an easy way to actually find the values of these modes. Sampling the space, at the resolution displayed above (the data are in a space of 100×200 divisions in θ and ϕ), is rather time consuming since the KDE must be evaluated at each point, and each evaluation of $\hat{f}$ involves the summation of Gaussian kernels centred at each of the N observations.

A better solution is to use mean shift [17], which is a method for finding the modes of a multivariate distribution, and is intimately related to the KDE. The idea in mean shift is to follow the direction of steepest ascent (the normalised gradient) in the KDE until a stationary point is reached, i.e. a local maximum. By starting the search from sufficiently many points in the domain, all modes can be recovered. In order to find the modes, it is not actually necessary to calculate the KDE itself (neither over the whole space as in the visualisations, nor even strictly at the points themselves, unless we wish to recover the probabilities, which we do once after convergence), since the necessary information is encoded implicitly by the gradient function.

The mean shift vector $\mathbf{m}(\mathbf{y})$ at some point $\mathbf{y}$ describes the magnitude and direction in which to move from $\mathbf{y}$ toward the nearest mode. We omit a full derivation of how the mean shift vector is obtained from the partial derivatives of the KDE equations — a thorough exposition can be found in [23]. In general, the mean shift vector for a point $\mathbf{y}$ is defined as:

$$\mathbf{m}(\mathbf{y}) = \frac{\sum_{i=1}^{N} \mathbf{x}_i g(\| \frac{\mathbf{y}-\mathbf{x}_i}{h} \|^2)}{\sum_{i=1}^{N} g(\| \frac{\mathbf{y}-\mathbf{x}_i}{h} \|^2)} - \mathbf{y} \quad = \quad \acute{\mathbf{m}}(\mathbf{y}) - \mathbf{y} \tag{5.12}$$

where $g(x) = -k'(x)$ is the negative derivative of the kernel profile described above, and we have used $\acute{\mathbf{m}}(\mathbf{y})$ to conveniently denote the left hand term in the mean shift expression. Since this is the direction in which the point $\mathbf{y}$ should be moved, the new value for $\mathbf{y}$, denoted $\mathbf{y}'$, is simply $\mathbf{y}' = \mathbf{y} + \acute{\mathbf{m}}(\mathbf{y}) - \mathbf{y} = \acute{\mathbf{m}}(\mathbf{y})$, and so $\acute{\mathbf{m}}(\cdot)$ is a function updating the current point to its next value on its path to the mode. Using the Gaussian

profile function $k(x) = \exp(-\frac{1}{2}x)$, the function $g(x) = -\frac{1}{2}\exp(-\frac{1}{2}x)$, and by substituting this into the above, we obtain

$$\acute{\mathbf{m}}(\mathbf{y}) = \frac{\sum_{i=1}^{N} \mathbf{x}_i \exp(-\frac{1}{2}\|\frac{\mathbf{y}-\mathbf{x}_i}{h}\|^2)}{\sum_{i=1}^{N} \exp(-\frac{1}{2}\|\frac{\mathbf{y}-\mathbf{x}_i}{h}\|^2)} \tag{5.13}$$

To run mean shift we use a set of points $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$, initialised by setting all $\mathbf{y}_i = \mathbf{x}_i$, i.e. a copy of the original data. At each step, we update all the $\mathbf{y}_i$ with $\mathbf{y}_i' = \acute{\mathbf{m}}(\mathbf{y}_i)$, and iterate until convergence. Using separate variables $\mathbf{x}$ and $\mathbf{y}$ highlights the fact that while it is the original data that we update and follow, the data $\mathbf{X}$ used for calculating the vectors remain fixed (otherwise the KDE itself would alter as we attempt to traverse it).

5.8.5.4 Accelerating Mean Shift

The mean shift process itself is time consuming, since it requires iteration for every one of the data points, of which there are several hundred. This amounts to a significant factor in the overall time for plane detection. Fortunately, we can make some alterations to achieve significant speedups. This is because at each iteration of mean shift for a given point, its next value $\mathbf{y}'$ is determined entirely by its current value $\mathbf{y}$ (i.e. location in θ, ϕ space) and the direction and magnitude of the gradient $\mathbf{m}(\mathbf{y})$. Furthermore, there is no distinction between the points, which means once we know the trajectory of a given point toward its mode, then all other points, which fall anywhere on that trajectory, will behave the same. This means we can avoid re-computing trajectories which will eventually converge, allowing us to significantly accelerate mean shift.

Two points' trajectories may not coincide exactly, so we divide the space into a grid of cells, their size being on the order of the kernel bandwidth. As soon as a point enters a cell through which another point has passed (at any time during its motion), we remove it, leaving just one point to be updated for that trajectory. Ideally, this will lead to only one point per mode being updated by the end of the process, which means we are still able to find the same modes, without the wasted computation of following every $\mathbf{y}_i$ which will terminate there.

This algorithm is based on the assumption that two points within the same cell will

converge to the same mode, which is not always true, because points arbitrarily close to a watershed between two modes' basins of attraction will diverge. We found that if the cell size is made smaller than the bandwidth, this does not appear to present a problem. This fairly simple approach is sufficient for our needs, achieving a speedup of around 100× while giving almost identical results to full mean shift. The experiments described in Section 6.2.2 provide evidence for this.

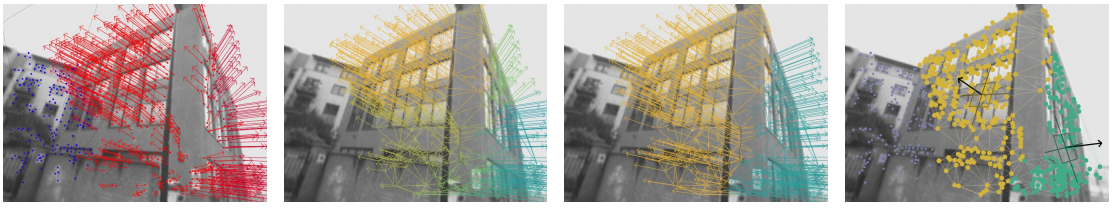
5.8.5.5 Segmentation

Finally, we use the modes given by mean shift as the discrete labels in the MRF (after converting back to normals in $\mathbb{R}^3$). Mean shift can give us the identity of the mode to which each point converges. Equivalently, we can initialise the nodes by setting their initial label, denoted $\mathbf{n}_i$ to be the one closest (measured by angle) to its observed value (where the observed values of the nodes are the $\mathbf{m}_i$, from region sweeping).

The MRF optimisation can now proceed, using only the discrete labels provided by mean shift. This guarantees it will output the number of different planes we have already found to be in the image, and avoids searching over the continuum of normals. This is formulated as optimisation of an energy function $E(n)$, where n is a configuration of normals $n = \{\mathbf{n}_1, \ldots, \mathbf{n}_{|\mathcal{S}'|}\}$ on the nodes in $\mathcal{S}' \subset \mathcal{S}$, the subset of points in the graph which were segmented into planar regions. As before we desire to find the optimal configuration $n^* = \arg\min_n E(n)$, by minimising the energy E(n), expressed as a sum of clique potentials:

$$E(n) = \sum_{i \in \mathcal{S}'} F_1(\mathbf{n}_i, \mathbf{m}_i) + \sum_{i \in \mathcal{S}'} \sum_{j \in \mathcal{N}_i} F_2(\mathbf{n}_i, \mathbf{n}_j) \tag{5.14}$$

where both clique potential functions F_1 and F_2 return the angle between two vectors in $\mathbb{R}^3$, thus penalising deviation of labels from the observations, and neighbours from each other. This is optimised using ICM, which converges to groups of spatially contiguous points corresponding to the same normal. This is illustrated in Figure 5.11.



(a) (b) (c) (d)

Figure 5.11: *Segmentation of planes using their orientation. From the initial sweep estimate of orientation (a), we initialise a second MRF (b), where each normal is coloured according to its orientation (see Figure 5.12), showing smooth changes between points. After optimisation, the normals are now piecewise constant (c), and correspond to the two dominant planes in the scene. After segmentation by finding connected components with the same normal, we obtain an approximate plane detection, where the normals shown are simply the mean of all salient points in the segments (d).*

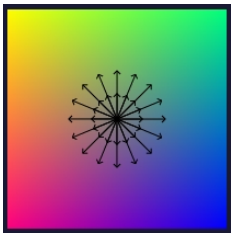


Figure 5.12: *The colours which represent different orientation vectors (the point in this map to which a normal vector from the centre of the image would project gives the colour it is assigned).*

5.8.6 Region Shape Verification

After running the two stages of segmentation, we are left with a set of non-planar segments (from the first stage), and a set of planar segments with orientation estimates, from the second. Before using these to get the final plane detection, we discard inappropriate regions. First, any regions smaller than a certain size are discarded, since they are not likely to contribute meaningfully to the final detection, nor be given reliable orientation estimates in the next step. We apply a threshold to both the number of points in the region, typically 30, and to the pixel area covered by the region, typically 4000 pixels. Second, we also remove excessively elongated regions, a fairly unusual shape for planes, by calculating the Eigenvalues of the 2D points in a region, and discarding those where the smaller value is less than a tenth of the larger.

5.9 Re-classification

Figure 5.8d above shows the result of plane segmentation, a collection of disjoint groups of points with normal estimates. However, these are not themselves the final plane detection, since they have not actually been classified or regressed. These regions simply have an average of the plane class and orientation of the points which lie within them, which in turn are derived from all sweep regions in which they lie. This means the estimates for these regions use data which extend well beyond their extent in the image.

We are now finally in a stage where we can run our original plane recognition algorithm from Chapter 3 on appropriately segmented regions. We re-classify the planar segments, to ensure that they are planar — generally they remain so, though often a few outliers are discarded at this stage. We do not attempt to re-classify the non-planar segments, because we have already removed them from consideration, and they were not part of the orientation segmentation. For those which are classified as planes, we re-estimate their orientation, so that the orientation estimate is derived from only the points inside the region. In general we find that the re-estimated normals are not hugely different from the means of the segments' points. This is encouraging, since it suggests that using the sweep estimates to segment planes from each other is a valid approach.

5.9.1 Training Data for Final Regions

The data we wish to classify here are rather different in shape from the training data. This is because the training data were created by region sweeping, because we wished to learn from regions of similar shape to those used during detection (Section 5.6). However, the segments resulting from the MRF are often much more irregular, as well as being different in shape and size from our original manually segmented regions. To address this we introduce another set of training data, thereby creating a second pair of RVMs, trained for this final task alone.

Appropriate training data are generated by applying the full plane detection algorithm to our ground truth training images (Section 5.5), and using the resulting plane and non-plane detections as training data. These should have similar shapes to detections on test data. Of course, we can use the ground truth labels for these segments, so the classification and orientation accuracy on the final segments is irrelevant. We also enhanced this training set by including all marked-up regions from the ground truth

images, whose shapes correspond to those of true planes and non-planes. We found that replacing the final RVMs with these trained on better data increased the orientation accuracy of the final result by several degrees on average (the segmentation itself, of course, is completely unaffected by this).

We now have an algorithm whose final output is a grouping of the points of the image into planar and non-planar regions, where the planar regions have a good estimate of their orientation, provided by our original plane recognition algorithm trained on example data.

5.10 Summary

This chapter has introduced a new algorithm to detect planes, and estimate their 3D orientation, from a single image. This has two important components: first, a region sweeping stage in which we repeatedly sample regions of the image with the plane recognition algorithm from Chapter 3, in order to find the most likely locations of planes. This allows us to calculate a 'local plane estimate' which gives an approximate plane probability and orientation to all salient points. Second, we use this intermediate result within a two-stage Markov random field framework, to segment into planar and non-planar regions, before running plane recognition again on these to obtain the final classification and orientation.

Unlike existing methods, we do not rely upon rectilinear structure or vanishing points, nor on specific types of texture distortion. This makes the method applicable to a wider range of scenes. Furthermore, since we do not rely on any prior segmentation of the image, the algorithm is not dependant on any underlying patterns or structure being present, for example planar regions being demarcated by strong lines. Most importantly, this algorithm requires only a single image as input, and does not need any cues from stereo or multiple views, in contrast to most methods for plane detection. In the following chapter, we evaluate the performance of the algorithm on various images, and investigate the effects of the design decisions outlined above. We also compare the method to existing work.

CHAPTER 6

Plane Detection Experiments

This chapter presents the results of experimental validation of our plane detection algorithm. We discuss the effect of some of the parameters on plane detection, and describe the experiments with which we chose the best settings, by using our training dataset. We then show the results of our evaluation on an independent dataset. Finally, we describe the comparison of our algorithm to a state of the art method for extracting scene layout, with favourable results.

6.1 Experimental Setup

To evaluate the performance of the plane detection algorithm, we used the manually segmented ground truth data described in Section 5.5. These were whole images hand-segmented into planar and non-planar regions, and the planar regions were labelled with an orientation. This means that every pixel in the image was included in the ground truth, and there was no ambiguity. For regions which were genuinely ambiguous (such as stairs and fences), we tried to give the semantically most sensible labels, and made sure that we were consistent across training and test data.

The key difference compared to the experiments in Chapter 4 is that we now have marked-up detections, rather than class and orientation for individual planes. The difficulty is to evaluate how well one segmentation of the image (our detection, which has sections missing) corresponds to another (the ground truth). Comparing segmentations in general is not easy (especially when there are multiple potentially 'correct' answers [130]). Our approach is based on assessing the classification and orientation accuracies across all the detected segments.

6.1.1 Evaluation Measures

We use two evaluation measures, for the classification accuracy and orientation error. We cannot directly compare ground truth regions and detected regions, since there will not be a direct correspondence. Instead, we perform the comparison via the set of salient points, which is the level at which our detection and grouping actually operate.

We measure classification accuracy as the mean accuracy over all salient points (mean over the image, or mean over all points in all images when describing a full set of tests). That is, we take ground truth class of points as the assigned class of the labelled region in which they fall, and the estimated class as the class of the detected plane of which it forms a part.

We take a slightly different approach to evaluating orientation accuracy. While we could have taken the mean orientation error over salient points, as above, this would not give a true sense of how points are divided into planes, giving undue influence to large planes, which contain more salient points. In keeping with the objective of evaluating the planes themselves, rather than the points, we evaluate orientation accuracy as the orientation error for whole regions. This is calculated by comparing to the true orientation of the region, which is taken to be the mean orientation of its salient points, whose orientation comes from the true region in which they lie. This means that if the detected plane is entirely within one ground truth region, we are comparing to that region's normal; otherwise, we are comparing to the mean of those regions it covers, weighted by the number of points from each region.

6.2 Discussion of Parameters

Each of the components of the algorithm described in Chapter 5 will have some effect on the performance of the plane detection algorithm. In this section, we show the results of experiments conducted to investigate how performance changes for different configurations of some important parameters. The results of these experiments were then used to choose the best parameters to use in subsequent evaluation. These experiments used our training set of ground truth data, comprised of 439 images.

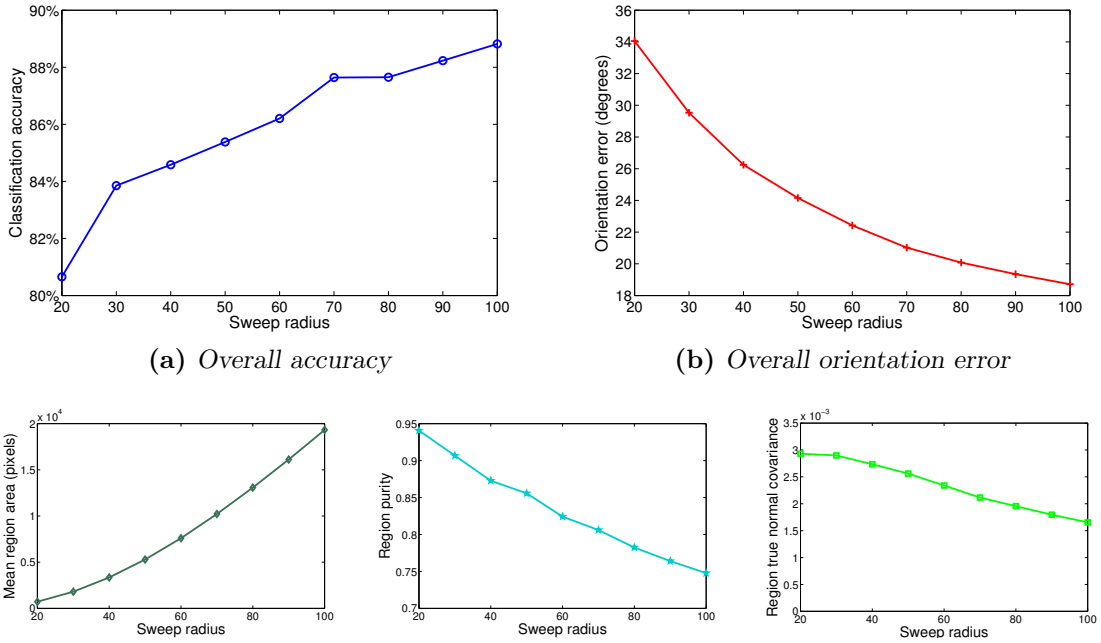
6.2.1 Region Size

The first parameter we needed to set was the region size for sweeping (the process by which we extract multiple overlapping regions from an image, described in Section 5.4). This is both a part of the plane detection algorithm, and used to gather the training data used for the plane recognition algorithm.

It was not immediately obvious what the best size would be as there is a trade-off between the accuracy and the specificity of regions. We would generally expect larger regions to perform better than smaller regions, since there is more visual information available. However, using regions which are as large as possible was not advisable. This is because these regions were not hand segmented, so larger regions would begin to overlap adjacent true regions, which would have a different class or orientation. Thus we needed to compromise between having large regions, and having 'pure' regions, by which we mean those that lie within only one ground truth region.

To investigate region size, we first conducted an experiment where we stepped through different sizes of region when using the plane recognition algorithm. We ran this experiment using plane recognition, rather than plane detection, since it was simpler to set up and gave a direct way of seeing how region size affected classification, without considering other factors such as the segmentation.

Using each of the region sizes, we harvested training regions by sweeping over a large set of fully labelled ground truth images (see Sections 5.4 and 5.5), where region size was specified by the radius around the centre point within which neighbouring salient points were included. We then fully trained the plane recognition algorithm, as described in Chapter 3, and tested it using cross-validation.



(a) *Overall accuracy*

(b) *Overall orientation error*

(c) *Area of regions*

(d) *Purity, defined as the amount by which true regions are mixed*

(e) *Covariance of region orientations*

Figure 6.1: *Using different region sizes (measured in pixels) when using sweeping to get training data. These experiments are for plane recognition.*

Figure 6.1 shows the results. Increasing the region size improved performance; yet contrary to expectation performance continued to rise even for very large regions, which occupied a significant fraction of the whole image, mixing the different classes. This is evident in Figures 6.1a and 6.1b, which show the mean classification accuracy and orientation error respectively. Figure 6.1c confirms that as the radius was increased, the actual area of regions did indeed get larger.

Nevertheless, using arbitrarily large regions is inadvisable. As Figure 6.1d shows, when the regions were bigger, they were on average less 'pure', meaning they tended to overlap multiple ground truth regions, mixing classes and orientations. We define purity as the percentage of points in a sweep region which belong to the largest truth region falling inside that sweep region. Thus, the trade-off is that as we use larger regions, we can be less confident that they accurately represent single planar or non-planar regions, potentially making the training set more difficult to learn from as appearance corresponds less consistently to specific orientations.

We explain the continually increasing performance with Figure 6.1e, where we plot the determinant of the covariance matrix formed from all the regions' true orientations.

As the regions got bigger, the covariance got smaller, which means the normals were becoming more similar to each other. This suggests that as regions get larger, they will tend to cover more nearby planes, averaging out the difference between them. As planar regions approach the size of the image, the orientation is simply the mean orientation over all points in the ground truth. For an image with multiple planes, these will likely average out to an approximately frontal orientation. Thus we would expect to see less overall variation in the normals, as indeed we did. The lower amount of variability in turn makes these orientations easier to regress.

This experiment confirmed our suspicion that very small regions were not suitable for classification, which supports the case for using our sweeping-based method rather than over-segmentation (c.f. Section 5.1.2). The experiment also suggested that overly large regions were not suitable either. Unfortunately since the performance measures kept on rising, this experiment could not be used to determine which region size should be used, and it was necessary to consider what effect the region size had on segmentation. As such the following experiment was conducted to further investigate this issue.

6.2.1.1 Testing Region Size using Plane Detection

Since testing the recognition algorithm on its own was not a good way to find the optimum region size for plane detection, we instead ran a test on the whole plane detection algorithm, in cross-validation, for different region sizes. From this we could directly observe any effect of changing the region size on the results of the plane detection, rather than relying on a proxy for how well it would perform. This experiment involved carrying out all of the steps described in the previous chapter, repeatedly, for multiple sets of truth regions with different sweep settings.

The experimental procedure consisted of k-fold cross-validation with n repeats, described as follows. We looped over a set of region sizes, from a radius r of 20 pixels up to 100 in increments of 10. For each, we ran k-fold cross-validation, where we divided the data into k equal sets, and for each fold, used all but one to train the algorithm. This was done by detecting salient points, creating and quantising features and so on, then running region sweeping to extract training regions, with which we trained a pair of RVMs. It was necessary to also gather regions for the final re-classification step, so we ran the plane detection on these test images and retained the resulting detected regions (plus the ground truth regions themselves), and trained a second pair of RVMs. Finally, we

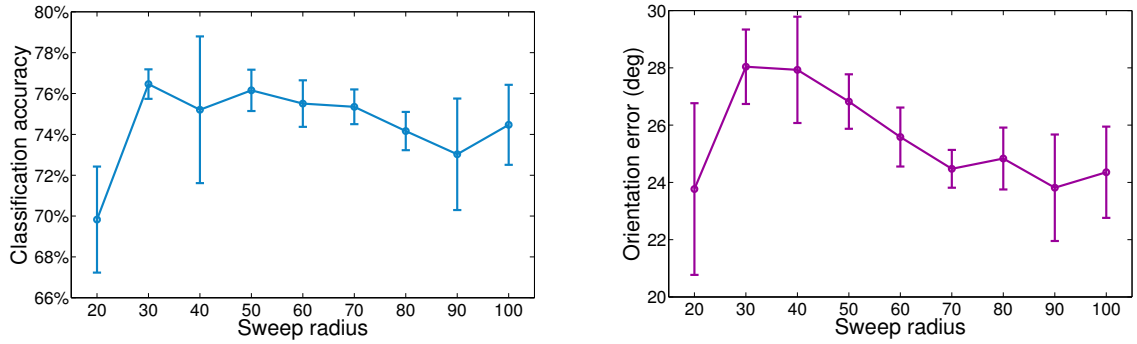


Figure 6.2: *Changing the size of sweep regions (measured in pixels) for training data used for plane detection. The error bars show one standard deviation either side of the mean, calculated over twelve repeat runs.*

used this plane detector on the images held out as test data for this fold, using a sweep radius r, and stored the mean accuracy and orientation error compared to its ground truth. We repeated all of this n times, and calculated the mean and standard deviation of the results for each radius.

The results are shown in Figure 6.2, which graphs the classification accuracy and orientation error, against sweep region radius. The graphs show the results from twelve runs of five-fold cross-validation for each sweep size – the points are at the mean, and the error bars show one standard deviation either side of the mean, over the twelve runs. Even with this many runs, the results were somewhat erratic. This may have been due to using only a quarter of the set of truth images we had available (115 regions were used), and limiting the training stage to use a maximum of 1000 sweep regions for each run, due to the large amount of time this took. As such the overall accuracy might have been diminished, and the uncertainty shown by the error bars is so large that the best parameters are hard to determine.

The results hint that the optimal sweep size for classification and regression may be a little different, with the best classification accuracy occurring at small radii, while orientation estimation seemed to improve up to a radius of around 90 pixels. If this behaviour is real, we could in principle use two separate region sizes for the two tasks, which would entail having two entirely different sets of training regions, not just different descriptors, c.f. Section 3.3.4.5. However, since the results of this experiment are not conclusive, we chose the simpler option of using the same radius for both; this experiment does not dictate a clear value to use, so a radius of 70 pixels was chosen as a compromise.

This value was used for the rest of the experiments in this chapter, unless otherwise stated.

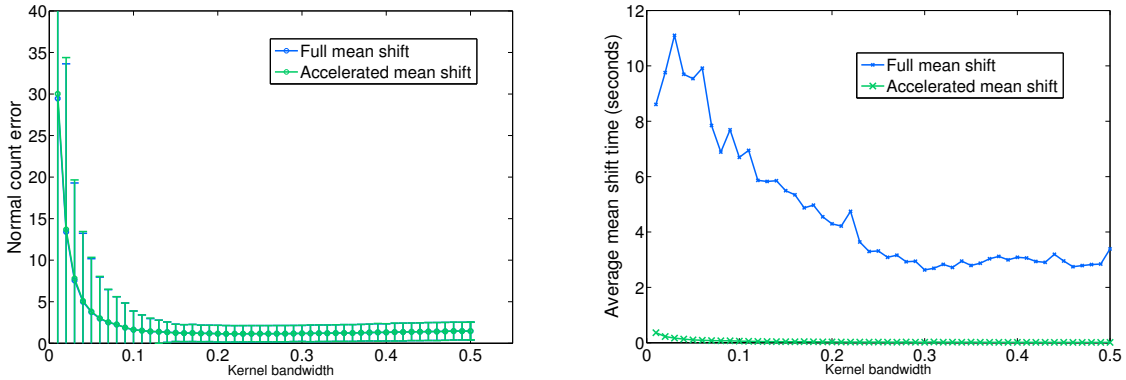
6.2.2 Kernel Bandwidth

We performed an experiment to determine the optimal kernel bandwidth for mean shift (to find the modes of the kernel density estimate of all orientations in an image, as described in Section 5.8.5.1). It would be possible to evaluate this as above, by running the whole plane detection algorithm for a range of different bandwidth parameters. However, this would give an indirect measurement of the quality of the segmentation, since it would measure the accuracy after having re-classified the detected regions, the result of which would depend on all the stages of plane detection. To compare kernel bandwidths, we needed only to determine the best granularity for splitting planes from each other, given the local plane estimate. This was done by using just the local plane estimate, calculated not from classification but from ground truth data, which was sufficient since for ground truth images we know the true number of planes we should find.

The experiment was set up as follows: we calculated the local plane estimate, using region sweeping, but rather than run full plane recognition we simply used the ground truth data for each region. We processed overlapping regions as before, taking the median and geometric median of classifications and orientations respectively. This gave us a local plane estimate which was as good as possible, independent of the features or classifiers.

After segmenting the planes from non-planes (using the first MRF), we took all the estimated normals from points in planar regions and ran mean shift to find the modes. We were able to evaluate these modes without running the subsequent segmentation, because ideally each true plane would correspond to one mode, and so we measured the difference between the number of modes and the number of true planes (we counted parallel planes as one, since their orientation does not distinguish them). This evaluation did not consider whether the orientations for the modes returned by mean shift were actually correct, but given that we were testing on a ground truth local plane estimate, it should not be an issue.

Figure 6.3 shows the results. Clearly, very small bandwidths (corresponding to a very rough density estimate) were poor, as they led to far too many plane orientations. Accuracy improved until around 0.2, after which there was a very slow deterioration (pre-



(a) *The mean error in number of modes compared to true planes. Error bars correspond to one standard deviation, calculated over all test images*

(b) *Mean time taken for mean shift, comparing the full algorithm and our accelerated version*

Figure 6.3: *Results for evaluating bandwidth for mean shift, when finding the modes within the local plane estimate.*

sumably arbitrarily high bandwidths would increase the error as there would be always one mode), and so we chose this as our bandwidth value for further experiments.

We also compared the full mean shift algorithm, in which all the data were iterated until they reached their mode, with our accelerated version (Section 5.8.5.4). The results confirm that this approximation does not increase the error. Figure 6.3a shows the results of the two overlaid, with virtually no visible difference in either mean or standard deviation. Figure 6.3b plots the average time per image for both methods. Clearly, our accelerated version offered a huge improvement in speed, being on average 100 times faster.

6.3 Evaluation on Independent Data

We proceeded to evaluate our algorithm on an independent dataset. For training, we used the set of 439 ground-truth images from above, which were first reflected about the vertical axis, to double the number of images from which training data was gathered. From these we extracted around 10000 regions by sweeping, which were used to train the full plane detector. Our independent dataset was taken from a different area of the city, to ensure we had a proper test of the generalisation ability of the algorithm. This consisted of 138 images, which were also given ground truth segmentations, and plane

class and orientation labels. These exhibited of a variety of types of structure including roads, buildings, vehicles and foliage. As in Chapter 4 the aim was for these to be totally unseen image regions, though we must specify a few caveats. First, these data are harvested from the same video sequences (but not the same frames) as the independent dataset of Chapter 4, so the location is not wholly new (again, this location was never used in training data). A subset of the data were used to evaluate the algorithm as described in [57], so some of the images will have been seen before during testing (this is the subset used in Section 6.4 below). We also used some of these images (because of the ground truth labelling being available) in section 5.6.1.1, to show that extracting training regions by sweeping was necessary when the test regions were similarly extracted. Other than these exceptions, the training data described here are new and unseen during the process of developing the algorithm.

6.3.1 Results and Examples

When running the evaluation on our test data, we obtained a mean classification accuracy of 81%, which was calculated over all points, except those which were not used in any regions. We obtained a mean orientation error of 17.4° (standard deviation 14.7°). Note that standard deviation was calculated over all the test images, not over multiple runs. This is larger than the error of 14.5° obtained on basic plane recognition (Section 4.3), but the task was very different, in that it first needed to find and segment appropriate regions.

These results are a little worse (though within a few percent for both measures) than we originally reported in [57]. However, those were obtained using a smaller set of test images, which may have been less challenging.

To clarify what these results mean, we show a histogram of the orientation errors in Figure 6.4. Comparing this to Figure 4.7 in Chapter 4 shows it to be only a little worse, even though this is a much more difficult task, with a large majority of the orientation errors remaining below 20°. We believe that these results are very reasonable given the difficulty of the task, in that these planes were not specified *a priori* but were segmented automatically from whole images, without recourse to geometric information.

We now show example results from this experiment. First, in Figure 6.5, we show some manually selected example results, to showcase some of the more interesting aspects of

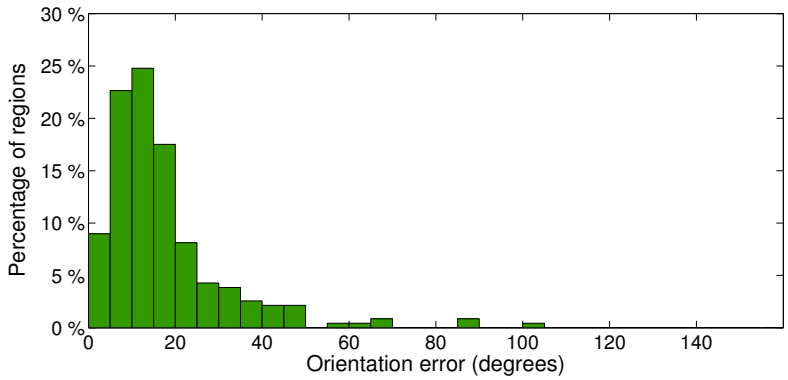


Figure 6.4: *Distribution of orientation errors for regions detected in independent set of test images.*

the method. Then in order to show a fair an unbiased sample from our results, we choose the example images to display in the following way. We sort all the results by the mean orientation error on the detected planar surfaces (this is one of two possibilities, as we could also use the classification accuracy over the salient points – the former was chosen since we believe orientation accuracy is the more interesting criterion here). We then take the best ten percent, the worst ten percent, and the ten percent surrounding the median error; then chose six random images from these sets. This is in order to give a fair sampling of the best cases, some typical results in the middle, and situations where the algorithm performs poorly. The best, medium, and worst examples are shown in Figures 6.6, 6.7 and 6.8 respectively.

Our algorithm is able to extract planar structures, and estimate their orientation, when dominant, orthogonal structures with converging lines are apparent. Figures 6.6c and 6.7a, for example, show it can deal with situations typical for vanishing line based algorithms. Crucially, we also show that the algorithm can find planes even when such structure is not available, and where the image consists of rough textures, such as Figure 6.6e. This would be challenging for conventional methods, since there are not many intersections between planes, and the tops of the walls are not actually horizontal. Another example is shown in Figure 6.5c, where the wall and floor have been separated from each other.

Our method is also able to reliably distinguish planes from non-planes, one of the most important features of the algorithm. For example, Figures 6.5a and 6.5d show a car and a tree, respectively, are correctly separated from the surrounding planes before attempting to estimate their orientations. This sets our approach apart from typical shape from texture methods, which tends to assume that the input is a plane-like surface which

Figure 6.5: *Examples of plane detection on our independent test set. These images were hand-picked to show some interesting baheviour, rather than being a representative sample from the results (see the next images). Columns are: input image, ground truth, local plane estimate, plane detection result.*

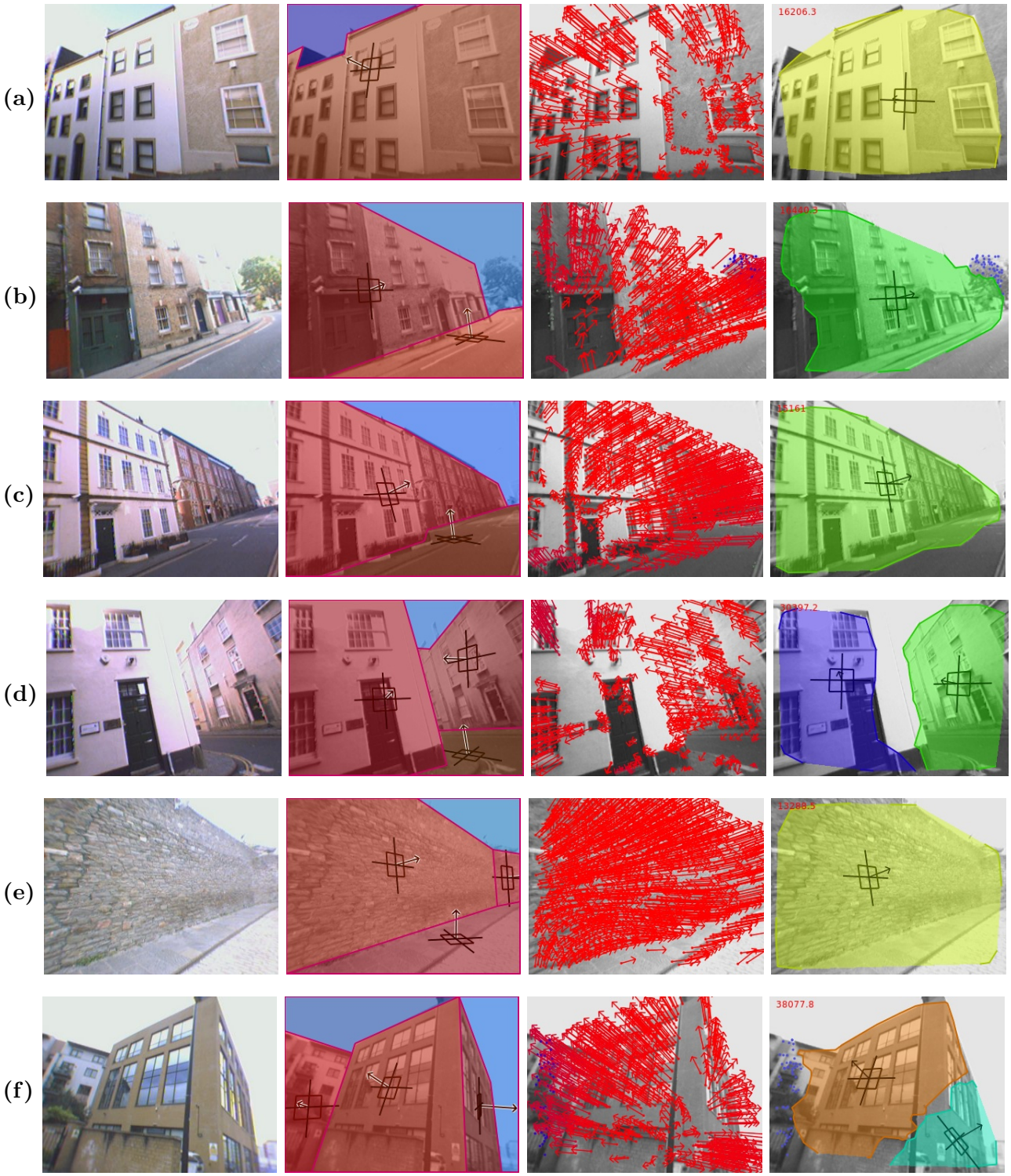


Figure 6.6: *Examples of plane detection on our independent test set. These are randomly chosen from the best 10% of the examples, sorted by orientation error. Columns are: input image, ground truth, local plane estimate, plane detection result.*

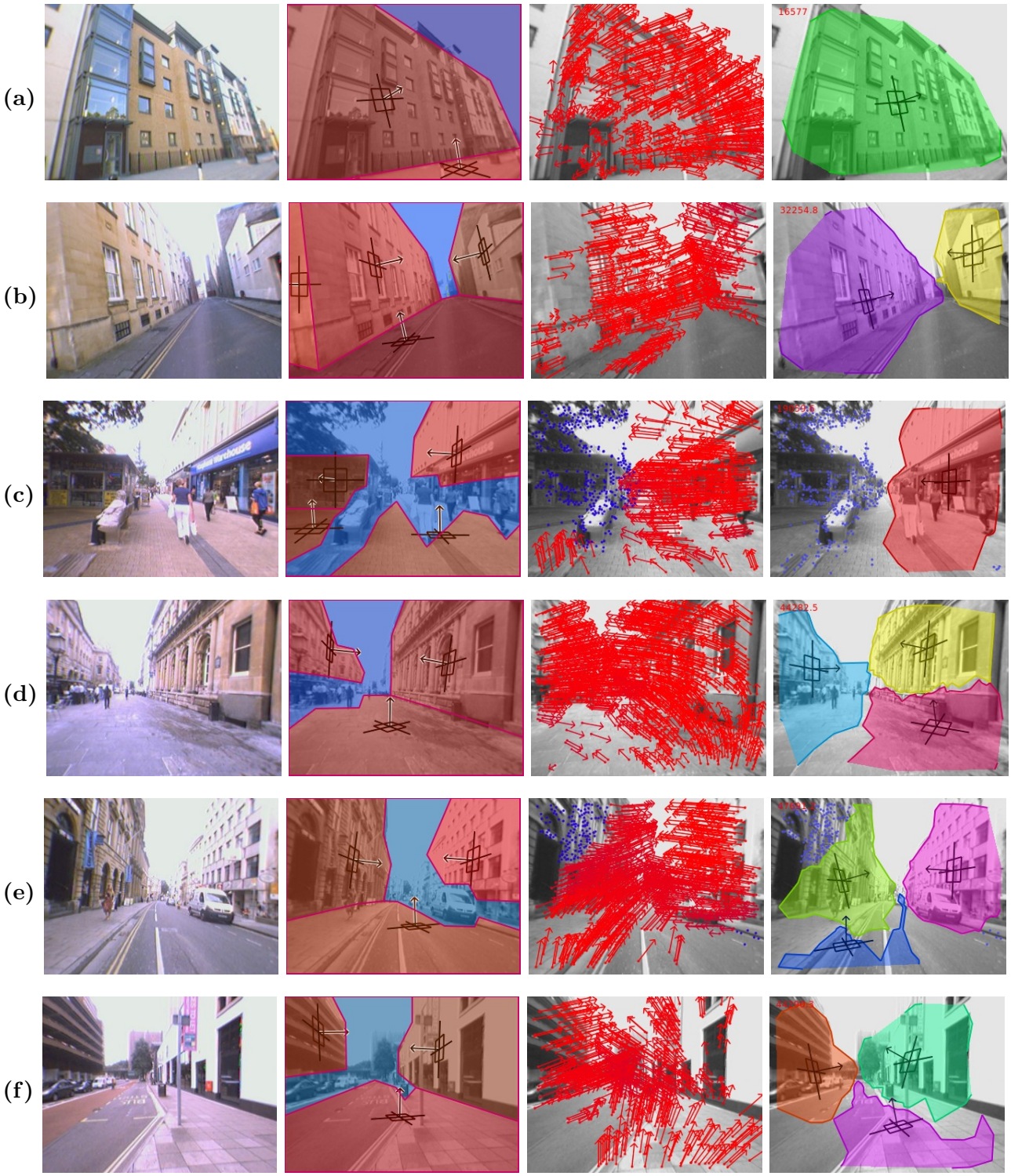


Figure 6.7: *Examples of plane detection on our independent test set. These have been selected randomly from the 10% of examples around the median, sorted by orientation error. Columns are: input image, ground truth, local plane estimate, plane detection result.*

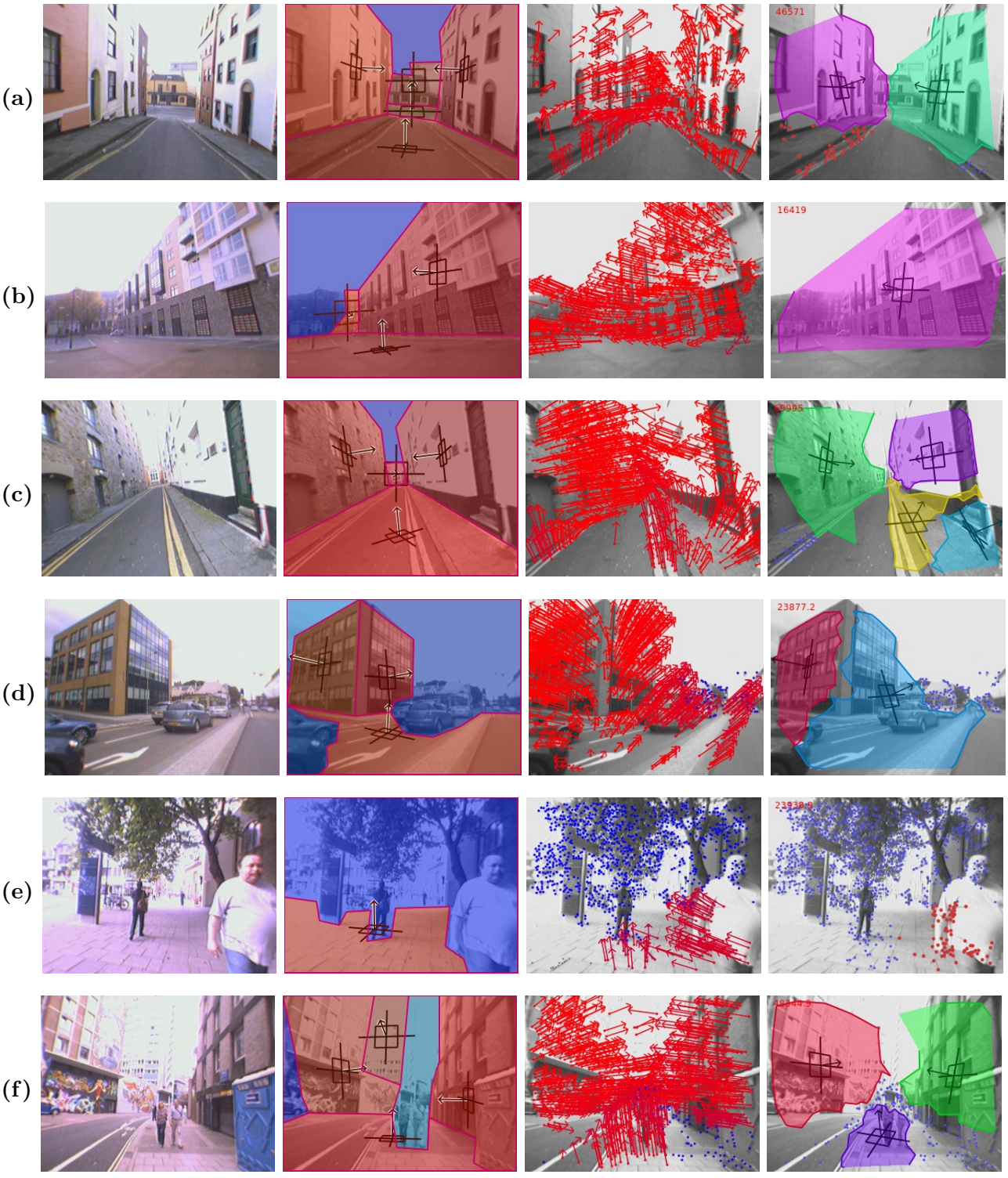


Figure 6.8: *Examples of plane detection on our independent test set. These have been selected randomly from the worst 10% of examples, sorted by orientation error. Columns are: input image, ground truth, local plane estimate, plane detection result.*

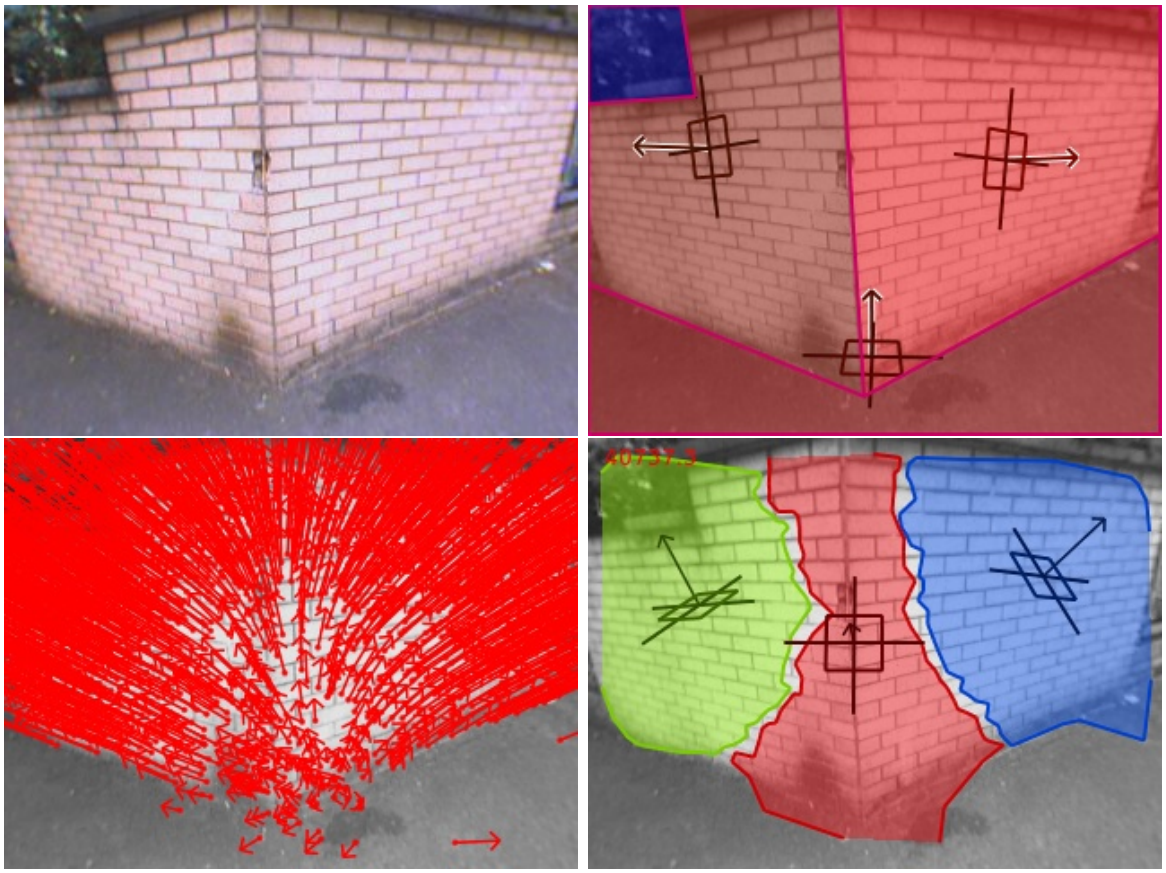


Figure 6.9: *An examples of plane detection on an image from our independent test set, where two planes have been split into three, due to the algorithm being unable to perceive the boundary between them. From top left: input image, ground truth, local plane estimate, plane detection result.*

needs its orientation estimated, without being able to report that it is not in fact planar.

These results also suggest that our algorithm is able to generalise quite well to new environments. It does this by virtue of the way we have represented the training data using quite general feature descriptors, which should encode the underlying relationships between gradient or colour and structure, as opposed to using the appearance directly. This is supported by Figure 6.7f for example, where the car park structure on the left is correctly classified even though such structures are not represented in the training set.

However, we observe that in these images there are some missing planes. It is common for our method to miss ground planes due to the lack of texture. This is because if there are insufficient salient points, there will be no classification nor orientation assigned, so major planar structures may be missed – for example in Figure 6.5e where there are insufficient salient points on the ground to support a plane, despite the grid-like

appearance; and in Figure 6.7b where the road is entirely featureless and thus no planes (or indeed non-planes) can be found. On the other hand, in some cases, such as Figure 6.5b, the ground plane can be detected and given a plausible orientation, at least in the region in which salient points exist.

6.3.2 Discussion of Failures

Our method fails in some situations, examples of which are shown in Figure 6.8, which are sampled randomly from the worst 10% of the results by orientation error. Misclassification during the sweeping stage causes problems, for example Figure 6.8b where the tree was classified as being planar, which led to the plane region over-extending the wall. Another example of a plane leaking beyond its true boundary is shown in Figure 6.8d, where the orientation estimated for the ground and the vertical wall are unfortunately quite similar, leading them to be grouped into the same segment. Misclassification also causees problems in Figure 6.8e where a pedestrian has been partly classified as planar (and the ground missed), though the region in question is rejected by the final classification step.

A common problem is the inability to deal with small regions. This is due to the region-based classifier, and how fine detail is obscured by the nature of our region-sweeping stage. This is shown in Figure 6.5f for example, where rather complex configurations of planes partially occluded by other planes are not perceived correctly. We observed a tendency to over-segment, such as Figure 6.8c, where the ground plane has been unnecessarily divided in two, due to a failure to merge the varying normals into one segment. Conversely, the algorithm may fail to divide regions when it should, as we pointed out above in Figures 6.8b and 6.8d, and also in Figure 6.5f where the whole scene has been merged into one large plane. A related problem is the over-extension or 'leaking' of planes, as evinced by Figure 6.7b where the plane envelopes the pavement as well as the wall. This is an example where the algorithm has failed to respect true scene and image boundaries.

Finally we consider the interesting case of Figure 6.9 (enlarged to more clearly show the variations in the local plane estimate, bottom left). As one might expect, the local plane estimate shows how the normals at the points vary smoothly around the sharp corner. Unfortunately, the MRF has segmented this into three rather than two segments, effectively seeing the middle of the transition as a plane in its own right. Currently there

is nothing in our algorithm to prevent this, if those normals are sufficiently numerous to be another mode in the kernel density estimate. This is also an example where the final classifier has assigned incorrect orientations to the three planes, perhaps caused by the segmentation being incorrect.

These failures are the most common types of error we observe (though we emphasise that, in accordance with Figure 6.4, the majority of orientations are good), and can generally be explained given the way the algorithm works. They do, however, suggest definite ways in which it could be improved, and hint at directions for future work.

6.4 Comparative Evaluation

As we discussed in Chapter 2, our algorithm is quite different from most existing methods. For example, algorithms that use information such as vanishing points to directly estimate the scene geometry should perform better than ours when such features are available, but are not applicable to the more general types of scene we encountered. As such we omit a direct comparison with such methods, though we acknowledge that when obvious vanishing point structure is visible, they will almost certainly perform better. Shape from texture methods, on the other hand, may perform well in more general environments, but impose constraints of their own. As we mentioned in the background chapter, these are not usually able to find the planes, and assume orientation is to be estimated for the whole image, so a comparison is not well defined.

A good example of a single-image interpretation algorithm which can deal with similar types of image to our work is the scene layout estimation of Hoiem et al. [66] (which we will henceforth refer to as HSL). This uses a machine learning algorithm to segment the image into geometric classes representing vertical, support surfaces, and sky, including a discrete estimate of surface orientation, and has been used to create simple 3D reconstructions, and as a prior for object recognition [65]. In the following, we explain the relevant details of this algorithm and how it relates to our own, before showing the results of using it for plane detection on our dataset.

6.4.1 Description of HSL

We have already described the HSL algorithm in our background chapter — refer to Section 2.3.2. Here we recap some of the more important aspects, as relevant to the comparison. First, note that it uses superpixels, obtained by over-segmentation (based on intensity and colour) as its atomic representation, rather than working with salient points. While these superpixels give some sense of where image boundaries are, these boundaries are only as accurate as the initial image segmentation.

Superpixels are grouped together by a multiple segmentation process. This uses classifiers to decide whether two superpixels should be together; whether a segment is sufficiently homogeneous in terms of its labelling; and to estimate the likelihood of a class label per segment. By using cues extracted from the segments, using these to form segments, and extracting larger-scale features from these, the algorithm can build up structure from the level of superpixels to the level of segments. A variety of features such as colour, texture, shape, line length, and vanishing point information are used for this.

The result is a set of image segments, each labelled with a geometric class, which represent geometric properties of image elements, as opposed to their identity or material. The three main classes are ground, sky, and vertical surfaces, which should be able to represent the majority of image segments. The vertical class is divided further into left, right, and forward facing planes; and porous and solid non-planes. This allows the underlying scene layout to be perceived, but offers no finer resolution on orientation than these labels.

6.4.2 Repurposing for Plane Detection

Although HSL was developed for coarse scene layout estimation, and not for plane detection, there are some important similarities. By separating the sky from the other main classes, and subdividing the vertical class into the planar and non-planar (porous and solid) subclasses, this is effectively classifying regions into planar or non-planar classes. Thus, HSL can be used as a form of plane detection, and so it was our intention to evaluate how well it could do this — specifically, how well it could perform at our stated task of grouping points into planar regions and estimating their orientation.

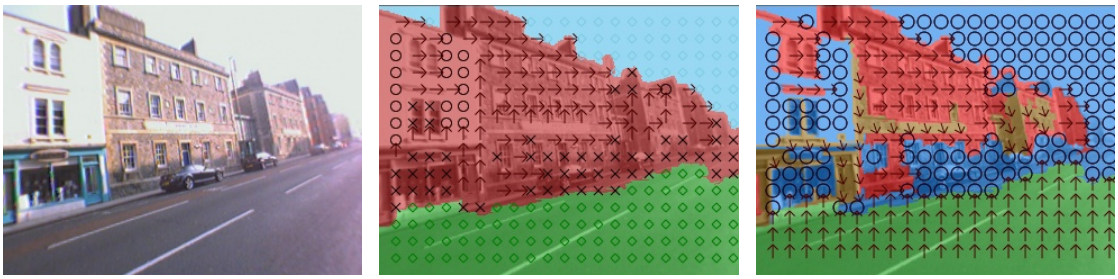
We ran the unmodified HSL algorithm on our dataset, then processed the output so that

it represents plane detection. We considered the ground class, and the left, right, and forward facing subclasses of the vertical class to be planar, and the rest (sky, porous, and solid) to be non-planar. This corresponds to our separation of plane from non-plane. Plane orientations come from the planar subclasses of the vertical segments, and the support class. Re-drawing the output to show this is very interesting, as it shows some shortcomings of the algorithm that are not obvious in the usual way the output is drawn (e.g. in [64, 66]). As Figure 6.10 shows, regions which are correctly deemed to be vertical, and usually drawn all in red, may contain a mixture of conflicting orientations, and include non-planar sections.

For these experiments, we used code provided by the authors[1]. We made no changes to adapt it to our dataset, and the fact that we did not retrain it using the same dataset used for our detection algorithm (due to the difficulty of marking up the data in the required manner) may cause some bias in our results. However the training data used to train the provided classifiers (described in [66]) should be suitable, since the range of image sizes covers the size we use, and the type of image are similar. The results we obtained appear reasonable compared to published examples, suggesting the method is able to deal sufficiently well with the data we collected.

To compare with our algorithm, we looked at the difference in classification and orientation at each salient point in the image. This is because there was no easy way to

[1] Available at www.cs.uiuc.edu/homes/dhoiem/



(a) *Input*

(b) *Output drawn similar to the original*

(c) *Drawn to show plane detection*

Figure 6.10: *Re-purposing HSL[66] for plane detection: the original way to draw the output (b) indicates the majority of classifications are correct (indeed, the whole surface is 'vertical'); however when we draw to highlight planar and non-planar regions and to distinguish orientations (c) (colours as in Figure 6.11) errors become apparent, including regions of non-planar classification falling within walls.*

correspond the regions in HSL to either ours or the ground truth; and because comparing at every pixel does not make sense for our method, since unlike HSL we do not segment using every pixel in the image. Salient points are not used as part of HSL, but we assume that they are a sufficiently good sampling to represent the segmentation. This comparison will thus focus on textured regions, where salient points lie, so if HSL performs better in texture-less regions we cannot measure this; and we acknowledge that comparing the algorithms only at the locations where ours outputs a value, rather than all locations, is not necessarily a fair comparison.

For this comparison, we continued to use the percentage of points assigned to the correct class as the measure of classification accuracy (as above), since this is well defined for both methods. However, we needed to compare orientation differently, since our algorithm assigns orientations to each plane as a normal vector in $\mathbb{R}^3$, whereas HSL can only give coarser division into orientation classes. To do this we sacrificed the specificity of our method, and quantised our orientation estimates into one of the four orientation classes. This was done by finding which of the four canonical vectors, representing left, right, upwards and forwards, were closest in angle to a given normal vector. Using this quantisation, for both the ground truth and detected planes, orientation error was measured as a classification accuracy. This may introduce quantisation artefacts (arbitrarily similar orientations near a quantisation boundary will be treated as different), but gave us a fair comparison.

6.4.3 Results

This experiment was performed using a subset of 63 of our labelled ground truth data, using plane detection trained on a subset of the training data described above (the initial datasets for which we reported results in [57]). The detector was trained by the same procedure as before, except that we used regions of radius 50 pixels. We do not imagine this has a major impact on our conclusions, since the difference in performance between the two radii according to Figure 6.2 was relatively minor.

The results are shown in Table 6.1. Our algorithm gave better results than HSL— which stands to reason since our method is geared specifically toward plane detection, and uses a training set gathered solely for this purpose.

Example results are shown in Figure 6.11 (see caption on facing page for explanation

	Ours	HSL
Classification accuracy	84%	71%
Orientation accuracy	73%	68%

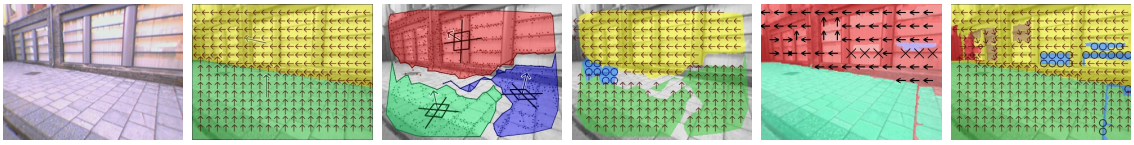
Table 6.1: *Comparison of plane and quantised orientation classification accuracy between our method and Hoiem et al. [66] (HSL) for plane detection.*

of colours). Note that these are illustrative examples hand-picked from the results, in order to show the similarities and differences between the algorithms, as opposed to a fully representative sample. The fifth column shows the typical output of HSL; as we mentioned above, these segmentations appear accurate, but do not illustrate performance of plane detection. The final column shows the same result when drawn to show plane detection, which no longer appear so cleanly segmented, and numerous errors in plane extent and surface orientation are visible. We draw the result of our method and the ground truth in the same manner for comparison, as well as the input image and standard output of our algorithm for reference.

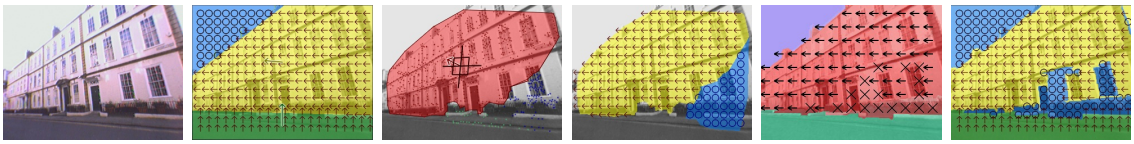
6.4.4 Discussion

These images show some interesting similarities and differences between the two algorithms. In many situations, they performed similarly, such as Figures 6.12a and 6.12b, where in both images the main plane(s) were found and assigned a correct orientation class. It is worth noting that in Figure 6.12b our method did not detect the ground plane, due to the lack of texture and salient points, but this posed no problem for HSL. This is partly down to the different features used (such as shape and colour saturation), but also because image position is an explicit feature in HSL, meaning that pixels near

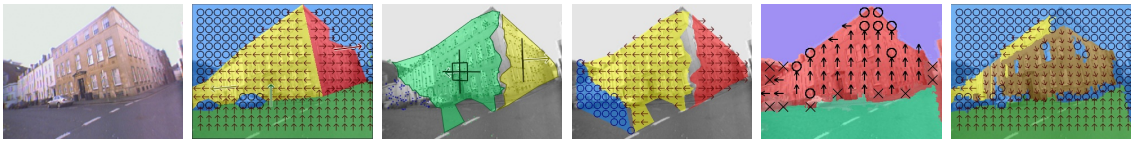
Figure 6.11: *Illustrative examples, comparing our method to the surface layout method of Hoiem et al. [66] (HSL). The columns are the input image, ground truth, our method, our method quantised, original output of HSL, and HSL drawn to show orientation classes. The original HSL outputs are drawn here as in their own work, where green, red, and blue denote ground, vertical, and sky classes respectively, and the symbols denote the vertical subclasses. The second, fourth and sixth columns are drawn so that non-planar regions are shown in blue, and left, right, frontal, and horizontal surfaces are drawn in yellow, red, brown and green respectively, overlaid with appropriate arrows — thus illustrating the orientation classes. Captions show classification accuracy for both methods, displayed as plane/non-plane classification and orientation classification accuracy, respectively.*



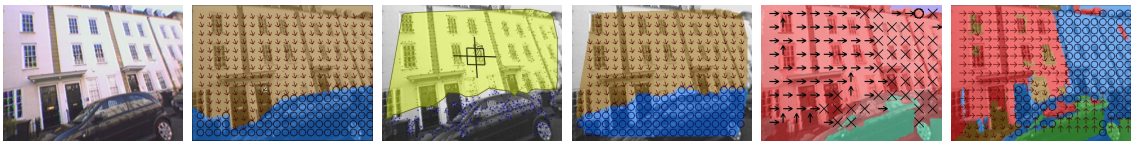
(a) *Ours: (99% , 91%) HSL: (95% , 92%)*



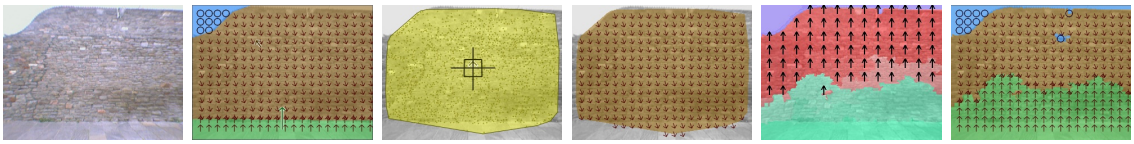
(b) *Ours: (84% , 96%) HSL: (81% , 100%)*



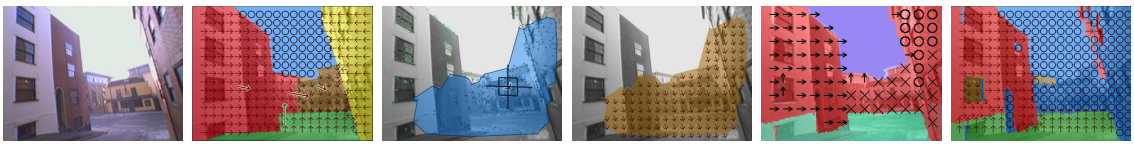
(c) *Ours: (85% , 93%) HSL: (75% , 20%)*



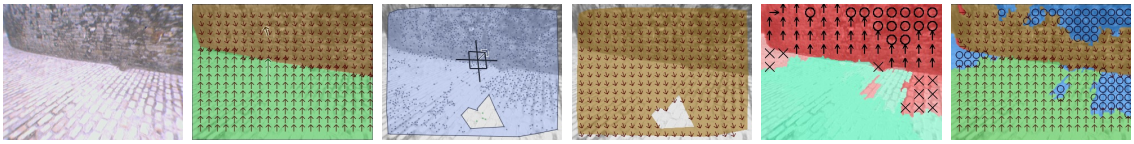
(d) *Ours: (94% , 100%) HSL: (53% , 28%)*



(e) *Ours: (98% , 97%) HSL: (99% , 68%)*



(f) *Ours: (98% , 29%) HSL: (41% , 71%)*



(g) *Ours: (100% , 59%) HSL: (73% , 99%)*

the bottom of the image are quite likely to be classified as ground. This can itself lead to problems, as in Figure 6.12e where a slight change in intensity mid-way up a stone wall misled HSL into extending the ground plane too far.

In other cases, our algorithm performed better, such as being able to disambiguate the two planes in Figure 6.12c, by assigning them different orientations, whereas HSL merged them together as a forward-facing plane. This failure of geometric classification to disambiguate surfaces suggests that being able to estimate actual orientations is beneficial. Also, in Figure 6.12d our algorithm found the whole plane, and gave an orientation class matching the ground truth; HSL missed half of the wall and assigned the 'wrong' orientation. It could be argued that the true orientation for Figure 6.12d should not be frontal (brown) but right (red). This ambiguity in orientation class caused by arbitrarily angled planes is exactly the reason we require fine-grained plane orientation, rather than geometric classification.

On the other hand, HSL clearly out-performed our method in Figure 6.12f by finding some of the planes, whereas our method was confused by the multiple small surfaces. The use of superpixels in HSL allows it to perceive smaller details than our sweeping approach, as well as showing better adherence to strong edges, by cleanly segmenting the edges of buildings (even if it does misclassify some). Figure 6.12g is another example where directly seeing edges may help, since our algorithm failed to distinguish two orthogonal planes, giving a nonsensical orientation for the whole image.

Despite the differences between the two algorithms – and the fact that HSL is not designed specifically to detect planes – they both gave quite similar performance when presented with the same data. Given that our algorithm was capable of producing better results on our test data, and was superior in a number of cases, this suggests that there is benefit in using our method, rather than simply re-purposing HSL for the task. Furthermore, as the results have shown, there is a good reason for estimating continuous orientation as opposed to discrete classes, since the latter can fail to disambiguate non-coplanar structure. The ability to more accurately distinguish orientations could also be useful in various applications, as we begin to investigate in the next chapter.

6.5 Conclusion

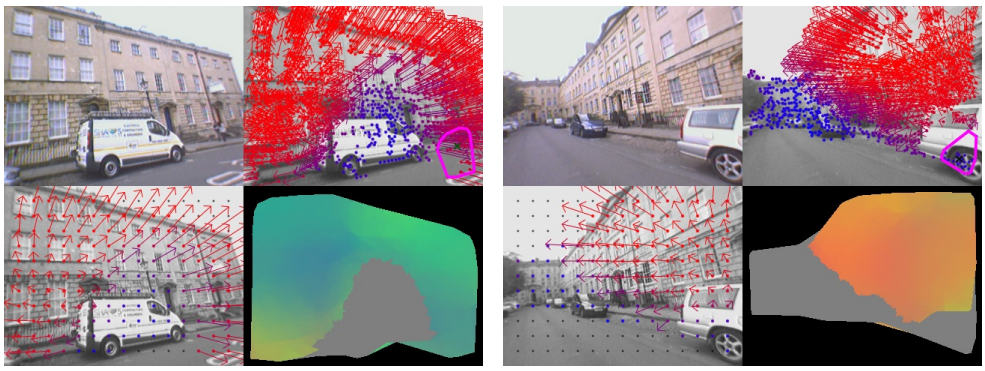
In this chapter we have thoroughly evaluated our plane detector. We began by showing, through cross-validation on training data, how its performance changed as various parameters were altered. These experiments allowed us to select the best parameters empirically, before applying it to real test data. We then demonstrated the algorithm working on an independent and previously unseen dataset, captured in a different area of the city (albeit with some fairly similar structures). The performance on these data shows that our algorithm generalises well to new environments.

More generally, these results show that our initial objective, of developing a method using machine learning to perceive structure in a single image, is achievable. We emphasise what is being accomplished: the location of planar structures, with estimates of their 3D orientation with respect to the camera, are being found from only a single image, using neither depth nor multi-view information, and without using geometric features such as vanishing points or texture distortion, as in previous methods. While the results we show here exhibit room for improvement, we believe they conclusively show that such a method has promise, and that exploiting learned prior knowledge – inspired by, but not necessarily emulating, human vision – is a worthwhile approach.

Despite the algorithm's success, the experiments have exposed a few key limitations, which we discuss in more detail here. First, due to the way we obtain the initial local plane estimates via region sweeping, our method is not able to perceive very small regions. While the MRF segmentation does in principle allow it to extract small segments (certainly smaller than the 70 pixel radius segments used for the first stage), very small planes are generally not detected because of the smoothly varying local plane estimate, which comes from sampling the class and orientation estimates from overlapping regions. This also means the algorithm is not perceptive of boundaries between regions, except when there is a noticeable change in the orientation estimates. Of course, as we discussed in Chapter 3, we would not want to rely on edge or boundary information, since this may not always be present or reliable, but some awareness of it could be advantageous.

6.5.1 Saliency

The comparison with HSL has also highlighted another limitation, albeit one added to the algorithm deliberately. Because we use only salient points, our plane detection



(a) (b)

Figure 6.12: *It is possible to create a denser local plane estimate, by using a different set of points than the salient points used to create descriptors. In these two examples, for the input image (top left) we create the local plane estimate (top right) at the salient points as normal; we can also do this at a regular grid (bottom left), or even at every pixel (bottom right), where the colours represent orientation, as described in Figure 6.13. Grey means non-plane and black is outside the swept regions. The two images show very different orientations, and hence their colours are in different parts of the colour map. Note that no segmentation has been performed yet.*

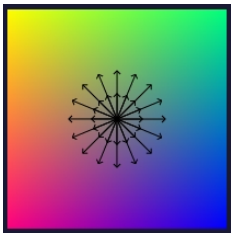


Figure 6.13: *The colours which represent different orientation vectors (the point in this map to which a normal vector from the centre of the image would project gives the colour it is assigned).*

method does not deal with any regions in which there is no texture. This was done in order to focus on 'interesting' regions, and to avoid wasting computational effort. However, this means that comparatively blank parts of the image, which may still be important structures, such as roads, are omitted.

We could possibly simply increase the density of the points, either by lowering the saliency threshold, using a different measure of saliency, or even using a regular grid. Such points may not be ideal for creating descriptors, but we can decouple the set of points used for image representation and those used to build the local plane estimate. Presently the two sets coincide primarily for convenience. It is possible to use one set of salient points to build the word histograms and spatiogram descriptors, to describe the sweeping regions, while sampling at another set of points to create the local plane estimate (a point will lie inside multiple sweep regions, and can be assigned a class probability and approximate normal, independently of whether it has associated feature

Figure 6.14: *When the image of the plane on the right is moved to the left of the image, it appears to have a different orientation, due to the perspective at which it is apparently being viewed.*

vectors).

We experimented with this technique, to create dense local plane estimates over an entire image (using every pixel, except for those outside any sweep region) as shown in Figure 6.12. This gives us a pixel-wise map of estimated orientation across a surface (although as with the local plane estimate, it does not show where the actual planes are). In principle, such a dense map would allow us to extend detection to non textured regions. However, in these regions, there will have been insufficient sweep regions (because they are still centred on salient points) to give a reliable and robust estimate, so accuracy may suffer. The best combination of salient points and local plane estimate density would require further work.

6.5.2 Translation Invariance

In Section 3.3.5 we described how we shift the points before creating spatiogram descriptors, such that they have zero mean, giving us a translation invariant descriptor. While this seems like a desirable characteristic, it has important implications, since in effect we are saying that the position of a plane in the image is not relevant to its orientation. However, on further consideration this does not seem to be true. If a plane is visible in one part of an image, then the exact same pixels in another part of the image would imply a different orientation — see for example Figure 6.14, where we have copied the right half of the image onto the left half. Even though the two sides are identical, to an observer they appear to have different orientations, with the right half appearing to be more slanted away from the viewer. This is because, as a parallel plane moves across one's field of view, it should usually appear more or less foreshortened.

This has an important implication for our plane detector. Since this effect has not been

accounted for, we have been implicitly assuming that the planes are sufficiently far from the observer for this to not be relevant. Fortunately, since the planes are not generally very close to the viewer (where such parallax effects are strongest), we do not envisage it would cause a drastic difference. Indeed, in the image shown, despite moving the plane to the other side of the image, the orientation change is fairly small. Nevertheless, we also investigated how much of a difference a translation invariant representation has on our results.

First, we compared the performance of the plane recognition algorithm, when using two types of spatiogram. These were the translation invariant version as before ('zero-mean'), and an alternative where we use the original image point coordinates ('absolute'). In the latter, similar regions in different locations are described differently. This could in fact be an advantage, as it implicitly uses image location itself as a feature, which Hoiem et al. [66] found to be most effective. The experiment was carried out using a large set of image regions, harvested from the ground truth images as described in the previous chapter, and by running five runs of five-fold cross-validation. The results are shown in Table 6.2.

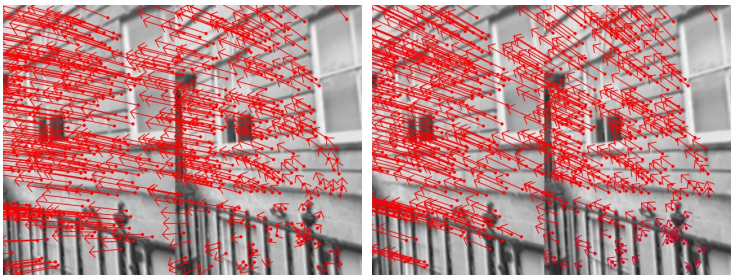
	Zero-mean	Absolute position
Classification accuracy	83% (0.3%)	79% (1.0%)
Orientation error	23.0° (0.1°)	23.1° (0.3°)

Table 6.2: *The difference between using zero-mean (translation invariant) and absolute position within spatiograms, when running plane recognition.*

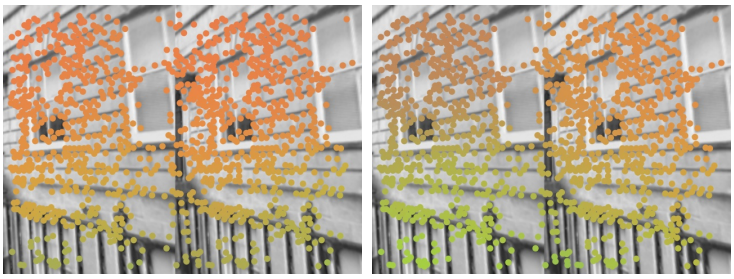
Interestingly, the results using the absolute image coordinates were slightly worse, although there was no drastic change. We speculate that competing factors were at work. Non-invariant descriptors may give a better representation of image regions according to their orientation, but at the cost of making similar appearance in different places seem more different than it should be. We hypothesise that this would mean that more training data are required to achieve the same goal, since there will be fewer potential matches for a region at a given image location, though further experiments would be needed to confirm this. This is more likely to be an issue for plane classification, since unlike orientation estimation, image location should be irrelevant, which partly explains the larger observed difference for this measure.

Despite our concerns, the experiment appears to suggests there is no benefit in adding spatial location information (at least in the way we have done so), so the model of plane recognition we have used is not invalidated. Nevertheless, so far we have considered only plane recognition performance, which was sufficient to determine whether the two means

of description behave similarly, but may miss some important effects. To address this, we conducted a further experiment to visualise what the change in representation means for plane detection. This was done by training a new plane detector, using the same images as before but using the absolute position spatiograms, and applying both this and the original to the artificial split plane from Figure 6.14.



(a) *Local plane estimate (subset shown for clarity)*



(b) *Local plane estimate, orientation illustrated by colour*

(c) *Plane detection result*

Figure 6.15: *An example of using the zero-mean (left) and absolute position (right) spatiograms for plane detection. The first row shows the local plane estimate, which is more clearly seen when represented as colours (b). The estimated orientations are different when using absolute position, for identical image content, reflecting how it is perceived. Given the right mean shift bandwidth, this means the two planes can actually be separated, when using MRF segmentation, unlike when using our original translation invariant representation (c).*

The results are shown in Figure 6.15, for the original zero-mean representation on the left and the absolute position version on the right. The first row shows the local plane estimates (LPE). Since this is difficult to interpret with so small an orientation change,

we show the LPEs using coloured points in the second row, where the colours correspond to orientation as before (Figure 6.13). This indicates that the orientations for the two halves, in the zero-mean parameterisation, are basically identical. This is as expected, since there is nothing to distinguish between the two halves of the image. When using the absolute positions, on the other hand, the colours are different for the two halves of the image. The effect is rather subtle, and may not be different enough to give 'correct' orientation for either side, but is sufficient to show that by incorporating image location into the representation we can get different orientation values depending on a plane's location within the image. In fact, as the bottom row shows, this is enough of a difference to be able to split the surface into two planes during MRF segmentation, whereas the original zero-mean representation does not have enough difference between halves. To do this it was necessary to lower the mean shift bandwidth (to 0.05, for both versions), but crucially the halves cannot be split with any bandwidth using the original parameterisation.

To conclude, while the zero-mean spatiogram representation may not be able to faithfully represent differences in appearance caused by changes in orientation as we intended, the alternative parametrisation using absolute position does not perform better. However, since we have shown that an awareness of image location is able to tell the difference between exactly the same image patterns when presented in different places – taking advantage of location context – this would be another useful avenue of future work, potentially able to increase the algorithm's ability to discriminate between planes and predict more accurately their orientation.

6.5.3 Future Work

We defer an in depth discussion of future work and applications, involving more drastic changes to our method, to Chapter 8, but briefly mention some improvements that could be made to the algorithm as it is.

There are two main reasons why our detector performs poorly, being either due to errors in the local plane estimate (LPE), which in turn must cause the segmentation to fail (see for example Figure 6.5f); or for the MRF segmentation to be unable to correctly extract planes from a (potentially complicated) LPE. We consider these two issues in turn in suggesting future work.

An inferior LPE is primarily caused by incorrect classification or regression. This means that any method to accumulate the information at salient points will have difficulty. We have gone some way to deal with erroneous classification when forming the LPE by using a robust estimator (the median) to calculate plane probability and orientation at salient points. We could further develop this by using more sophisticated robust statistics, such as the Tukey biweight or other M-estimators [90]. Even assuming all plane recognitions are correct, the problem remains that the LPE will be very smooth, making small regions and fast changes invisible. One possible way forward would be to derive separate estimates of the reliability of regions, before incorporating them into the LPE. We might for example be able to classify whether a region is likely to belong entirely to a single region, or be straddled across a surface boundary, and use this to pre-filter what goes into building the LPE.

Next we consider methods to improve the segmentation of a given LPE, by using a more sophisticated method of segmenting the MRF. So far we have only used the iterative conditional modes algorithm to optimise the configuration over the field, chosen for its computational simplicity, though other more complex methods may give us results closer to what we desire. It would also be worthwhile to investigate the use of other energy functions in the MRF, either by weighting the contribution differently for the single and pair site clique potentials (with values learned from training data), or by including higher-order cliques to model long-range dependencies.

Additionally, motivated by the under-segmentation and leaking of planes seen above, we could attempt to incorporate edge information into our segmentation. As it stands, segmentation in the MRF uses only the local plane estimates at each point, making boundaries between planes hard to perceive, whereas if we can incorporate information about edges it may improve plane segmentation. This could either be edge information from the image itself, using an edge detector or gradient discontinuity information; or derived from further classification, in which we attempt to classify whether pairs of points or regions belong to the same or different planes. This could potentially be incorporated into the same probabilistic framework, by using a MRF to deal with nodes representing both the points and boundaries between regions [78]. Similar use of edge information was shown to be beneficial by [67].

Given the strengths and weaknesses of our plane detector as compared to the work of Hoiem et al. [67] (HSL), as illustrated by our results, it would be worthwhile to consider combining them to create a hybrid system. It is unclear how our sweeping could be used with their multiple-segmentation approach. On the other hand, it may be useful to use

their algorithm to find coarse structure such as the ground plane – at which it excels, especially in texture-free regions – and use this to guide our segmentation. One might even consider using our plane recognition algorithm on groups of superpixels, once HSL has joined them into putative clusters, rather than relying on geometric classification for the final scene layout.

CHAPTER 7

Application to Visual Odometry

In this chapter, we demonstrate the use of our plane detection algorithm in a real-world application, showing that it can be of practical use. We focus on the task of real-time visual mapping, where we integrate the plane detector into an existing visual odometry system which uses planes in order to quickly recover the structure of a scene. This work was originally published in collaboration with José Martínez-Carranza [59].

7.1 Introduction

In vision based mapping, whether for visual odometry (VO) or simultaneous localisation and mapping (SLAM), early and fast instantiation of 3D features improves performance, by increasing robustness and stability of pose tracking [19]. For example, careful selection of feature combinations for initialisation can yield faster convergence of 3D estimates and hence better mapping and localisation, as described in [61].

A powerful way of improving the initialisation of features is to make use of higher-level structures, such as lines and planes, which has shown promise in terms of aiding intelligent feature initialisation and measurement, thus increasing accuracy [47]. An

important benefit of structure-based priors is that they can be used to quickly build a more comprehensive map representation, which can be much more useful for human understanding, interaction and augmentation [16]. This is in contrast to point clouds, which are difficult to interpret without further post-processing.

This chapter introduces a new approach to speeding up map building, motivated by the observation that if knowledge of objects or structural primitives is available, and a means of detecting their presence from a single video frame, then it provides a quick way of deriving strong priors for constructing the relevant portions of the map. At one extreme this could involve instantaneous insertion of known 3D objects, derived, for example, from scene specific CAD models [71, 104], or of navigation with a rough prior map [102]. However, these limit mapping to previously-known scenes, and require the effort of creating the models or maps. Rather, our interest is in the more general middle ground, to consider whether knowledge of the appearance and geometry of generic primitive classes can allow fast derivation of strong priors for directing feature initialisation.

We investigate this by using the algorithm we developed in the previous chapters, to focus specifically on map building with planar structure. This is an area which has received considerable attention due to the ubiquity of planes in urban and indoor environments. Previous approaches to exploiting planar structure in maps have included measuring locally planar patches [95], fitting planes to point clouds [47], and growing planes alongside points [88, 89]. These methods are generally handicapped by having to allow sufficient parallax (and hence time) for detecting planes in 3D, either by waiting for sufficient 3D information to become available before fitting models, or by simultaneously estimating planar structure while building the map. This is due to the fact that such methods are not able to observe higher-level structures directly, and rely on being able to infer them from measurements of the geometry of simpler (i.e. point) features.

This suggests that planar mapping would benefit from a method of obtaining structure information more directly, and independently of the 3D mapping component. Such information would be able to inform the map building and act as a prior on the location and orientation of planes, and as a guide for the initialisation of planar structure even before it becomes fully observable in a geometric sense. Indeed, this was nicely demonstrated in the work of Castle et al. [14], in which specific planar objects with known geometry were detected and inserted in the map, to quickly build a rich map representation; and also by Flint et al. [40], who use the regular Manhattan-like structure of indoor scenes to quickly build 3D models.

Our work further develops such ideas, in order to derive strong priors for the location of planar structure in general, without reference to specific planes, or being overly restricted by the type of environment. This is achieved by combining our plane detection algorithm with an extended Kalman filter (EKF) visual odometry (VO) system, by modifying the plane growing method developed by Martínez-Carranza and Calway [89]. Monocular visual odometry, and the closely related problem of simultaneous localisation and mapping (SLAM), are interesting areas on which to focus since we believe they have the most to gain from single-image perception, since depth, an important property of any visual feature, is not directly observable from a monocular camera. A single image, as we discussed in Chapter 1, is rich enough for a human to immediately get a sense of scene structure; and yet currently much of this information is discarded, to focus only on point features.

In the next section we discuss related work in the field of plane-based SLAM and VO, followed by an overview of our hybrid detection-based method. This is followed by a description of the baseline VO system we use in Section 7.3, and a more in-depth discussion of how the two methods are brought together to form our combined plane detection–visual odometry (PDVO) system in Section 7.4. Our results are presented in Section 7.5, which show that the approach is capable of incorporating larger planar structures into the map and at a faster rate than previously reported in [89] – averaging around 60 fps – while still giving good pose trajectory estimates. This demonstrates the potential of the approach both for the specific case of planar mapping, and more generally the plausibility of using single image perception to introduce priors for map building. Section 7.6 concludes, with a summary and some ideas for future work, including a discussion of how our PDVO system might be extended to allow the detector to be informed by the 3D map, with the potential of ultimately learning about structure from the environment directly.

7.1.1 Related Work

The use of planes in monocular SLAM/VO has a long history, motivated by the ubiquity of planar structure in human-made scenes. Planes are useful for mapping in a variety of ways, from being a convenient assumption during measurement, to being an integral part of an efficient state parameterisation.

Amongst the earliest to use planar features in visual SLAM were Molton et al. [95], who

use locally planar patches rather than points as the basic feature representation, within an EKF framework. Salient points can usually be considered locally planar, with some orientation. After estimating this orientation, the image patch is warped in order to account for distortion due to change in view, to allow better matching and tracking of features by predicting their appearance. This results in maps consisting of many plane-patch features, which as well as improving the localisation ability of the mobile camera, give an improved interpretability of the resulting 3D map. However, the planar patches remain independent, and are not joined together into higher-level, continuous surfaces.

A similar approach was developed by Pietzsch [103], in which the parameters of the planar patches are included into the SLAM state (again using an EKF), rather than being estimated separately. This work shows that even a single planar feature is sufficient to accurately localise a moving camera, something which would require very many point correspondences. Planes are measured using image alignment, which allows accurate measurements to be made. However, including all pixels in the SLAM state incurs a significant penalty in computational complexity, since updating the EKF is quadratic in the number of features and cubic in the number of measurements (due to inversion of the innovation matrix). Furthermore, no mechanism is described for the detection of such planar features, relying instead upon manual initialisation.

More extensive use of image alignment for planar surfaces is made by Silveira et al. [117], in which the whole SLAM problem is treated as optimisation, not only over camera and scene parameters but also surface properties and illumination. The assumption that a scene can be well approximated by a collection of planar surfaces can even apply to large scale outdoor scenes, to the extent that this method is capable of localising a camera while mapping a large, complex outdoor scene. Since the trajectories shown close no loops it is difficult to evaluate the global accuracy of the method.

The above methods effectively use planar structures to either improve the appearance of a map or to make better use of visual features; but planes can also help reduce the complexity of the map representation. If many points lie on the same plane, they can all be represented with a more compact representation (essentially by exploiting the correlations between their states). A good example is by Gee et al. [47] who use planes to collapse the state space in EKF SLAM. Planes are detected from a 3D map built using regular point-based mapping, by applying RANSAC to the point cloud in order to find coplanar collections in a manner similar to Bartoli [5]. Once such planes have been found, they are inserted into the EKF to replace the point features, so that whole sections of the map are represented by their relationship to the plane. This effectively achieves

a reduction in state size, while maintaining a full map, at the same time as introducing higher-level structures which may be used for augmentation [16]. Efficiency in terms of the state size is important in EKF SLAM since the filter updates are quadratic in the size of the state, therefore a reduction in state size has a big impact on increasing computational efficiency, or allows larger environments to be mapped.

The disadvantage of [47] is that it requires a 3D point cloud in order to find the planes, which must already have converged sufficiently (i.e. 3D points are well localised). This means initialisation of planar surfaces can take some time, especially in larger environments. Thus while the detection of planar surfaces can be most useful once they have been detected, the initial mapping itself derives no benefit from the planarity of the scene. Furthermore, since the primary aim is to reduce the state size, it does not necessarily follow that detected planes will correspond to true planes in the world. It is possible for coplanar configurations of points within the cloud to be mistaken for planes, especially in complex and cluttered scenes. This makes no difference to the state reduction ability, but means interpreting the features as belonging to true planes in the world is problematic.

An alternative method of finding planes while performing visual SLAM, without relying on converged coplanar points, was developed by Martínez-Carranza and Calway [86], based on the appearance of images in regions hypothesised to belong to planes. The basis of the method is to use triplets of 3D points visible in the current image (obtained by a Delaunay triangulation of the visible points) and test whether they might form a plane, by determining if pixels inside the triangle obey a planar constraint across multiple views. Adherence to this constraint disambiguates planar surfaces from other triplets of points. Crucially, the method takes into account the uncertainty estimate maintained by the EKF of the location of the camera and the 3D points, using a χ^2 test to determine whether there is sufficient evidence to reject the planar hypothesis; this is effectively a variable threshold on the sensitivity of the planar constraint according to the filter uncertainty. The method was shown to be successful in single-camera real-time SLAM, and was used for making adaptive measurements of points on known planes to cope with occlusion and enable tracking without increasing the state size [87].

Building on the above ideas is the inverse depth planar parameterisation (IDPP) [88], which adapts the inverse depth representation [19] to planar features. Planes are detected as the map is built, allowing planes to be initialised and represented compactly within the filter, while their parameters (represented as an inverse depth and orientation with respect to a reference camera) are optimised, based on a number of points estimated to

belong to the plane. This was shown to be successful in a variety of settings, and easily adapted for visual odometry [89]. We use this method for our plane detection enhanced visual odometry application, and give further details in Section 7.3.

The above are good examples of how, by exploiting the existence of planar structures, SLAM and VO can be enhanced. These concepts are taken further by Castle et al. [14], who use the learned appearance of individual planar objects within an EKF SLAM framework. This involves the recognition, in real-time, of a set of planar objects with known appearance, thereby incorporating some very specific knowledge. These are inserted into the map once they are detected, using SIFT [81] descriptors to match to the object prototype, and localised, by using corner points of the planar surfaces as standard EKF map features. Placing the recognised objects into the map immediately gives a more interpretable structure. More importantly, by using the known dimensions of the learned objects, the absolute metric scale of the mapped scene can be recovered, something which cannot be done with pure monocular SLAM. Other than the ease of tracking objects in the map, it does not depend upon them being planar, and has been extended to non-planar objects [21].

The main disadvantage to this is that it requires a pre-built database of the objects of interest. Learning consists of recording SIFT features relating to each image, along with its geometry and the image itself, in a database, which precludes automatic online learning, and constrains the method to operate only within previously explored locations, rather than being able to learn planar features in general or extracting them from new environments. Nevertheless, the idea of recognising planes visually and inserting them into the map is most appealing, as we discuss below.

An interesting application of higher level structures is demonstrated by Wangsiripitak and Murray [132]. They reason about the visibility of point features, within a SLAM system based on the parallel tracking and mapping (PTAM) of [72], and use this information to more judiciously decide which points to measure. Usually a point-based map is implicitly assumed to be transparent, so in principle points can be observed from any pose, but in reality they will often be occluded by objects. To address this they use a modified version of the plane recognition enhanced SLAM of [14] to recognise 3D objects in the map, as well as automatically detecting planar structures from the point cloud. These structures are used to determine the visibility of points, which is useful because if a point is behind an occluding surface with respect to the camera, it should not be measured. This increases the duration of successful tracking, by avoiding the risk of erroneous matches and focusing on those points which are currently visible.

Finally, the work of Flint et al. [40] is interesting in that it explores the use of semantic geometric labels of planar surfaces within a SLAM system, in order to create more comprehensible maps, based on the fact that many indoor environments obey the Manhattan world assumption. It also uses PTAM [72], where only a sparse point cloud is initially available, but this is supplemented by using a vanishing-line method similar to Košecká and Zhang [73], taking the lines detected in a scene and grouping them according to which of the three vanishing directions they belong, if any. Because the vanishing directions (in a world coordinate frame) remain fixed, they may be estimated from multiple images, giving a much more robust and stable calculation. Once these are known, surfaces are labelled according their orientation, introducing semantic labels based on context (i.e. floor, wall, ceiling). The result is that quite accurate schematic representations of 3D scenes, comprising the main environmental features, can be built automatically, without relying on fitting planes directly to the point cloud. While the dependence on multiple frames for the optimisation means this is fundamentally quite different from our work, it is interesting to note how this assumption of orthogonal planes is very powerful in creating clean and semantically meaningful descriptions, for simple indoor scenes; though this is of course limited to places where at least two sets of surfaces are mutually orthogonal.

7.2 Overview

We now proceed to give an overview of the plane detection–visual odometry (PDVO) system we developed. This uses our detector to find planes and estimate their orientation, given only a single image from a moving monocular camera navigating an outdoor environment. The mapping algorithm, which attempts to recover the camera trajectory in real time by tracking points and planes, is based on the IDPP VO system [88]. Ordinarily this maps planes in the scene by growing them from initial seed points, but because the location of planes is not known *a priori*, it is forced to attempt plane initialisation at many image locations, and to grow them slowly and conservatively.

The novelty comes from the fact that using single image plane detection helps the VO system to initialise planes quickly and reliably. By introducing planes using only one frame, with an initial estimate of their image extent and 3D orientation, we can make plane growing faster and more reliable. The result is that fewer planes are created, covering more of the scene, while being restricted only to those areas in which planes

are likely to occur, rather than attempted at every possible location. This means that, unlike previous approaches, we need not wait for 3D data or multi-view constraints to be available, but can directly use the information available in a single frame to guide the initialisation and optimisation of scene features. Therefore plane detection is driving the mapping, rather than feeding off its results. This process allows planes to be added quickly, but later updated as multi-view information becomes available, so that 3D mapping can benefit from prior information but is not corrupted by incorrect estimates.

We also show that when these planes are added to the map, we can quickly build a rough plane-based model of the scene, by leaving the detected and updated planes – which tend to cover a reasonable about of the scene – in the map once they are no longer in view. This results in a more visually appealing reconstruction, more easily interpretable by humans.

In the next sections, we explain how the baseline VO system works, how we adapted our plane detector to use with it, and how they are combined. The results we show in Section 7.5 indicate that this is a promising approach, and while our evaluation is not sufficient to show that it is necessarily more accurate than existing methods, our aim is to demonstrate that using learned generic prior information is a valid and useful way of building fast structure-driven maps in a real-time setting.

7.3 Visual Odometry System

This section describes the visual odometry system we used, which is based on the inverse depth plane parameterisation (IDPP) [88]. This uses an extended Kalman filter (EKF) SLAM engine ultimately based on [19, 29]. The distinction which makes this visual odometry (VO), rather than full SLAM, is that features are removed from the state once they are no longer observed, which means the estimation can progress indefinitely as new environments are explored, at the cost of losing global consistency.

7.3.1 Unified Parameterisation

An important aspect of IDPP is that two different types of feature are used, namely points and planes. All features are mapped using an inverse depth formulation [19], in

which depth is represented by its inverse, allowing a well behaved distribution over depths even for points effectively at infinity. This representation is extended to planar features, in a unified framework, which allows both points and planes to be encoded in the same efficient way. A general planar feature is described by its state vector $\mathbf{m}_i = [\mathbf{r}_i, \boldsymbol{\omega}_i, \mathbf{n}_i, \rho_i]$, which represents the position and orientation (using exponential map representation) of the camera, normal vector of the plane, and inverse depth respectively, giving a total of 10 dimensions. To represent points, the normal $\mathbf{n}_i$ is omitted, leaving a 7D parameterisation. This is further reduced to simply a 3D point for features which have converged to a good estimate of depth.

Planes are defined as collections of observed point features which satisfy a planarity constraint, and are measured and updated by projecting these points into the current image, using the recovered orientation estimate. Measurements are made using normalised cross correlation of image patches, and like Molton et al. [95] patches can be warped according to their planar orientation in order to improve matching. Using patch correlation is generally faster than using descriptor based methods [15]. Moreover, while such descriptors are powerful due to being scale, rotation and view invariant, this means they will match at a greater range of possible orientations. Cross correlation, on the other hand, will only find a match with the correct warping. This means patches not actually on the plane will fail to be matched, and thus discarded.

Further efficiency gains are achieved by sharing reference cameras between features (be they points or planes) initialised in the same frame. This means that even though the features have more dimensions than the 6D features of inverse depth points, the overall state size is reduced both by sharing of reference cameras, and because coplanar points are represented as part of the same planar feature. This leads to a large reduction in state size when the scene is dominated by planes, and hence more efficient mapping, or the ability to maintain a map of larger areas.

7.3.2 Keyframes

An important feature of IDPP is the use of keyframes. These are images retained when new features are initialised, relating the initial camera view to the feature state, and to which new observations must be matched for measurement. The pose of the camera when the keyframe is taken is stored as a reference camera, associated with the image; this is very convenient for plane detection, as will become apparent. Note that this use

of keyframes is very different from keyframe-based bundle adjustment methods (such as [72, 119]), where points are tracked between frames to localise the camera but only measurements on keyframes themselves are used to update the map; whereas since IDPP is based on a filter, all measurements in all frames are used to update the state recursively.

7.3.3 Undelayed Initialisation

An important difference compared to planar SLAM methods such as [47] is the undelayed initialisation of planar features. As soon as a point is observed, being a candidate for a new plane, it is added to the state and estimation of its location proceeds immediately (this can happen even though the initial depth is unknown due to the inverse depth representation). Immediately after initialisation, this initial 'seed' point can be grown into a plane, by finding nearby salient points and testing whether they belong to a plane with the same possible orientation. By this method, the algorithm simultaneously estimates which points are part of the plane, while updating its orientation.

In more detail, plane growing starts with one initial seed point on the keyframe, and assuming this is successfully matched, new points in the keyframe, within some maximum distance of the original and chosen randomly, are added to the plane. If these are also matched, a plane normal can be calculated from them, along with an estimate of its uncertainty (done as part of the filter update, since the normal is part of the state). The process continues, adding more points in the vicinity of the previous points, expanding in a tree structure — but only from those points whose measurements show they are indeed part of the same plane. Points which are not compatible are discarded, and it is this which allows plane growing to fill out areas of the image whose geometry implies a plane is present. Using such a geometric constraint does mean that planes will also be detected from coincidentally coplanar points (assuming the patches can still be matched), or where the points are so far away that no parallax is observed. Nevertheless, while these are not actually planar, they are still beneficial in making measurements and collapsing the state space, until an increase in error forces points to be discarded.

It should be emphasised that while planar feature initialisation is undelayed in terms of the filtering, it is not the entire planar structure which is instantaneously available. Rather, 'undelayed' refers to the fact that seed points can be updated and grown as soon as they are added to the map. Planes take time to grow, and must be augmented cautiously lest incorrect points be added, leading to an incorrect estimate of orientation,

which can be hard to recover from. Such errors are liable to occur, and it is a delicate matter to allow planes to grow sufficiently fast to be useful, and while they are still visible, without introducing erroneous measurements into the filter.

7.3.4 Robust Estimation

Careful feature parameterisation and view-dependent warping, as discussed above, both help to improve the efficiency and accuracy of the algorithm, but it would quickly become over-confident and inconsistent if not for some robust outlier rejection. One way this is done is by using the one-point RANSAC algorithm, first described in [114] and implemented for visual odometry by Civera et al. [20]. RANSAC (RAndom SAmple Consensus) [39] is a hypothesise-and-verify framework for robust model fitting and outlier rejection, where a random, minimal subset of measurements is taken, and used to hypothesise a possible model for all the data points (for example, four point correspondences in a pair of images to generate a homography). This is scored by the number of inlier measurements, i.e. how consistent all the data are with this model. The most consistent model is chosen, and only the inliers are used to calculate the final model.

The number of samples required for RANSAC to reliably find the correct model is a function of the number of expected inliers and the number of measurements required to form a minimal hypothesis. The key to one-point RANSAC is to reduce the number of measurements needed as far as possible, to using only one, and to use prior information about the scenario to complete the model. Therefore while a single measurement cannot generate an instance of the model, it is sufficient to choose from a one-parameter family.

In IDPP, one-point RANSAC is used to test possible camera poses, to ensure that no outlier measurements corrupt the current estimate. Instead of using the five point correspondences typically needed to generate a camera pose, one-point RANSAC allows the current state and its covariance to limit the range of most likely poses, meaning that only a single point measurement is sufficient to hypothesise a new camera. Because the minimal set is so much smaller, in order to get a high probability of finding the correct pose only seven measurements are sufficient on average, making this significantly faster than full RANSAC.

One-point RANSAC proceeds by selecting each point in turn, from the randomly chosen set, and using it to perform a partial update on the EKF (to update the state but not the

full covariance). Then the difference between the updated points and their measurements (innovation) is used to find which are the inlier points, since after the update those which have a high innovation are not consistent, and so are likely to be outliers. Once the hypothesis with the highest number of low-innovation inlier measurements is found, these are used to perform a full state update. However, it is possible that some of the points rejected as outliers are actually correct (such as those recently initialised and not converged, for example), so the final step is to rescue high-innovation liners, by finding those points which still are consistent with the model, even if they are less certain.

However, as powerful as it is, one-point RANSAC is not sufficient to make the algorithm fully robust. A further level of checking is added, in the form of a 3D consistency test (for full details see [85]), allowing the extra information available from the 3D map to be used. This is useful because often 2D points will seem to be consistent with their individual covariance bounds, whereas they do not lie on the correct plane in 3D. The covariance of the planar features in 3D is propagated to individual planar points, meaning points not actually conforming to the plane can be removed. The 3D consistency check is more selective, but more time consuming, than the 2D consistency check, so the 2D check is run first for all points and the 3D check run afterwards to remove any remaining outliers.

7.3.5 Parameters

Amongst the parameters which control the operation of the plane growing algorithm is the minimum distance between planar points, which determines from how far away new points are added to the plane, measured in the keyframe (set to a value of 12 pixels). A related parameter is the number of new points which are added to a plane at each step (set to 3). Together these two parameters control the speed at which planes grow. Finally, there is the maximum number of measurements (of either feature type) which can be made in one frame, which increases map density to the detriment of frame rate (set to 200).

7.3.6 Characteristics and Behaviour of IDPP

This plane parameterisaton was shown to be successful in detecting planes in cluttered environments, such as an office. Moreover, due to the ability to quickly add relevant measurements to a plane as they become available, it is able to deal smoothly with

occlusion, so that while parts of a plane are occluded, other parts are measured. The method was even shown to work in reasonably featureless indoor environments, by virtue of picking up many more features than is possible with point-based SLAM.

The IDPP system, when being used as visual odometry, performs well in outdoor environments, and achieves comparable performance to [20] on the Rawseeds dataset[1], compared to ground truth GPS data, while running at a significantly faster frame-rate [89]. Visual odometry discards the built map, so it can run indefinitely with constant processing time; but even so, its accuracy is sufficient to almost close large loops (although some scale drift is inevitable).

However, despite the benefits of the algorithm outlined above, there are a number of disadvantages which must be considered. First, although initialisation is undelayed and planar structures can be mapped immediately, it still takes time to build these up, and gather sufficient measurements to determine that a plane does indeed exist. Often collections of points can be measured as planes for some time before realising this is not actually the case. At worst this risks introducing erroneous measurements, and at best slows down execution as many false planes are attempted and discarded.

Successful plane growing also relies on having sufficient parallax, so if the camera is stationary or performing pure rotational motion, there will be no way to recover planes given any number of frames. Consequently, the accuracy of the resulting planes will be a function of the distance the camera has translated and the distance to the plane. As well as potentially introducing false planes, this could lead to incorrect orientations being assigned, meaning the map does not correspond well to reality, but also making it difficult for the planes to be corrected when reliable measurements are made, or to continue growing.

One of the main implications of not knowing where planes are beforehand is that many potential locations must be investigated. A large number of planar features must be initialised and grown, which is rather computationally expensive. While this blind probing is ultimately effective, it risks many planes being grown over non-planar structures, or for planes to overlap and compete for measurements, further slowing the process of finding true planar structure. An ideal solution to this and the other problems, of course, would be some way of knowing *a priori* which image regions correspond to planes.

[1] See www.rawseeds.org

7.4 Plane Detection for Visual Odometry

Now that we have outlined the important details of the IDPP VO system we use as a basis, and that the details of the plane detection algorithm are thoroughly explained in Chapters 3 and 5, we proceed to describe how the two are combined, to produce a plane-based visual odometry system that runs in real time.

7.4.1 Structural Priors

As we stated earlier, the function of our plane detection algorithm here is to provide a prior location, and orientation, of planar structures in the image. That is, we do not rely exclusively upon our detection algorithm, since it is prone to error, nor do we repeatedly use it to refine the estimates of the planes as they are observed. This cleanly separates the domain of the two algorithms: plane detection finds the location of the planes in the image, and passes this information to the VO system, which then initialises planes and continues to measure and map them as appropriate. Plane detection runs as described previously, using only the information in a single colour image as input, independently of any estimates present in the map.

One issue to consider is how the data are passed from the plane detector to IDPP. Plane detection operates at the level of salient points (Section 3.3.1), but this is only a limited sampling of the image; and while the VO also uses a set of salient point to track and localise features, these are not necessarily the same points (in fact, we use the difference of Gaussians detector, which picks blob-like salient regions at multiple scales, while IDPP uses the corner-like features provided by FAST [110]). We bridge this gap by modifying our algorithm to produce a mask as output, indicating which pixels belong to which planes. This is by nature an approximation since we do not have pixel-level information, but it is valid if we assume that planes are continuous between the salient points assigned to them, and do not extend significantly beyond their planar points. We create the mask using the Delaunay triangulation, which is already available having created the graph for the Markov random field (Section 5.8.2), and assign pixels which fall inside each triangle to have the same class (and orientation) as the three vertices, for those triplets which are in the same segment. Pixels inside triangles whose vertices belong to different segments, or those outside any triangles, are not assigned to a planar region, and assumed non-planar (the non-planar regions are not important here)

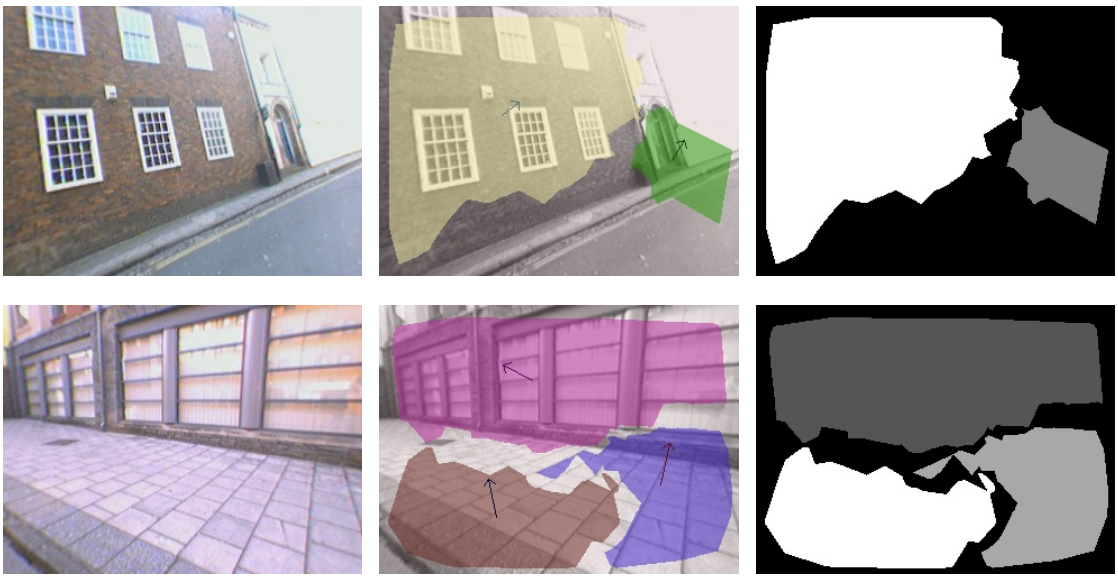


Figure 7.1: *Examples of the masks we use to inform the visual odometry system of plane detections. From the input image (left) we use a Delaunay triangulation to create a mask, showing which pixels belong to planes, each with a grey level mapping to an ID with which orientation is looked up (right). These triangulations are similar to how the plane detection is usually displayed (centre).*

The masks that result from this are vectorised to integer arrays, where all non-planar pixels are set to 0, and pixels in a plane have a value corresponding to the plane's ID number, which is used to recover the plane normal from a list. We can also show these masks as grey scale images, examples of which are in Figure 7.1, where for display purposes IDs are mapped to visible grey levels. The approximated pixel-wise segmentation is actually the same as the way we have illustrated the extent of planar regions in previous chapters.

7.4.2 Plane Initialisation

When a keyframe is created by IDPP, instead of attempting to initialise planes at any and all locations, the keyframe image is passed to the plane detector, and upon receipt of the plane mask image, planar features are created. We initialise one IDPP planar feature per detected plane, using the centroid of the region (specifically, the nearest FAST corner to the centre of mass of the mask pixels) as the location of the seed point. To avoid overwriting existing planes, we ensure that each centroid is above a minimum distance (set to 30 pixels) of any planar point in the keyframe, and discard the detected plane otherwise. Where no planes are present (black areas of the mask), point features are initialised as

normal, if necessary, to ensure there is sufficient coverage of the environment.

The estimated normal vector from the plane detector is used to initialise the normal in the feature state. The aim is that this will be close to the true value, much more so than the front-on orientation which must be assumed in the absence of any knowledge. This allows faster convergence of the normal estimate, and avoids falling into local minima. However, this must be initialised with sufficiently large uncertainty as while an initial estimate is useful for initialisation, it cannot be relied upon to be accurate, and so it must be able to be updated as more measurements are made without the filter becoming inconsistent. Using these initial normals also helps ensure non-planar points are not used for measurements, since as described above, their warped image will be less similar if the correct normal is used.

7.4.3 Guided Plane Growing

It is important to note that we do not initialise the whole extent of the plane immediately, spreading points over the mask region. This would risk introducing incorrect estimates into the filter, caused by not having enough baseline to correct the normal estimate from all the measurements. Such errors would be difficult to recover from and may corrupt the map.

Instead, we use the regular plane growing algorithm to probe the extent of the plane, but allow the growing to proceed faster than normal, by adding more new points per frame. The number of new planar points added to a plane in each frame is increased to 10 (from 3 in IDPP), meaning we can exploit the prior knowledge of the plane location and orientation to map planes quickly, but retain the ability to avoid regions which are not actually coplanar. Furthermore, any points which do not conform to the planar estimate are automatically pruned by the algorithm's 3D consistency test (see Section 7.3.4 above), so minor errors in the plane detection stage (planes leaking into adjoining areas, for example) do not cause problems in the map estimation.

The planes are not permitted to grow outside the bounds set by the plane detector, which means the growing is automatically halted once the edge of the detected plane has been reached (they cannot inadvertently envelop the whole image). No other planar features are allowed than those detected. The result is a much smaller but more precise set of planes, corresponding better to where they should be; and no wasted time in

attempting to grow planes in regions which are not appropriate. This, combined with the computational savings achieved by initialising with a good normal estimate, allows for a substantial increase in frame rate.

What we are striving for, in effect, is a system whereby the strengths of both methods are combined, in order to correct each others' errors. The inability of IDPP to quickly map out new planes is mitigated by using the prior information, and the potential for detected planes to have inaccurate orientations is ameliorated by the iterative update over subsequent frames.

7.4.4 Time and Threading

While our original implementation of the plane detector was reasonably fast, running in one to two seconds per image, this was using un-optimised code. For application in a real-time system, the fastest execution possible is desired. As such we parallelise portions of our algorithm to decrease its execution time. The region sweeping stage is ideal for splitting between simultaneous processes or threads, since each sweep region is independent (indeed, it is basically a separate invocation of the plane recognition algorithm). This means the creation of descriptors and classification of the regions can be done separately for sets of regions, then combined together when creating the local plane estimate.

While this is still not sufficient to run in real-time (the camera images are received at a rate of 30Hz) it is not strictly necessary for our detector to run this fast (i.e. with an execution time below 33ms, which may be possible but difficult to obtain). This is because we can run the detector in the background, in a separate processor thread, and return the result for use by the VO system when it has completed.

Since the plane detector is running in a separate thread, we take full advantage of this by running it continuously. As soon as the plane detector has finished one frame, it will move on to the next current image from the camera. This means the VO thread has a constant stream of plane detections, as soon as they are available.

This is where the keyframes of the VO system are particularly convenient, since these are retained even after the camera has moved on, and their associated camera poses are maintained in the state. This means once the plane detector has finished, planes can be related back to the keyframe on which they were detected, in order to grow them

and update the reference camera. Updates to one keyframe, even in the 'past', can be used to update the current map. The disadvantage is that updates resulting from measuring planes arrive several frames late, although this will not be a problem under the assumption that there are no drastic changes in camera pose in the interim, and so long as the planes are still in view. The delay is more than compensated for by the increased speed at which planes can grow and converge.

While one of the primary reasons to use only a single frame to detect planar structure was to do this quickly, it appears we must still wait several frames for detections to become available. One might argue that rather than wait for the single-frame estimate, it would be simpler to take all the frames from a corresponding time period and apply standard stereo or multi-view algorithms to recover planar structure. This is not an appropriate alternative, and we refute it thus:

The problem with simply using all the images in a 20-30 frame window to recover 3D information is that accuracy is very sensitive to the baseline. The camera would need to move far enough, over a period of around one second, to perceive large differences in depth, which is rather unlikely when the camera is moving at moderate speeds in an urban scene. Indeed, this is the limitation which stops IDPP growing planes instantaneously. While one could, in principle, make use of information from all these frames, it would not necessarily be of any benefit; indeed, the original IDPP algorithm, which uses information from every frame, filtered by an EKF, would itself be one of the best ways of doing this. In contrast, even if our plane detection allows 30 frames to pass by unseen while detecting planes in just one of them, it does not matter even if all those frames are the same.

A second reason is that the plane detection algorithm uses an entirely different kind of visual cue. Even if some superior multi-view geometry algorithm could be used to extract accurate planar orientation from such a narrow time window, this still uses only the geometric information apparent from depth and parallax cues. On the other hand, by exploiting the appearance information of the image, our algorithm is attempting to interpret structure directly, based on learned prior knowledge. This is independent of 3D information, and so is complementary to the kind of measurements made by IDPP. We draw a parallel with the work of Saxena et al. [111], who show that estimating a depth map from a single image helped to improve the results from a stereo camera system, by combining the two complementary sources of information.

7.4.5 Persistent Plane Map

One objective of our PDVO is to create a plane-based map of the scene. In general, the accuracy of visual odometry is not good enough to create a persistent map, and the quality and extent of planes created by IDPP is not sufficient for a 3D map display. However, by using our plane detection, the planes tend to be larger, more accurate, and better represent the true 3D scene structure, and we can be more confident about the planes' authenticity than with IDPP alone. This suggests an easy way to create maps quickly, by simply leaving the mapped planes in the world, even after they are no longer in view. Given that this is visual odometry, they are removed from the state, maintaining constant-time operation, and so will not be re-estimated when they are again in view. We find that the accuracy of their pose is sufficient to build such a 3D map, to immediately give a good sense of the 3D structure of the world; note that this work is at a preliminary stage, leading to quite rough 3D models.

7.5 Results

A number of experiments were carried out using videos of outdoor urban scenes. These were recorded using a hand-held webcam running at 30Hz, of size 320×240 pixels and corrected for distortion caused by a wide-angle lens. Our intention was to investigate what is possible when using learned planar priors, rather than to exhaustively evaluate the difference between the two methods. As such, we tuned both methods to work as well as possible by altering the number of new planar points that can be initialised at each frame. For IDPP, this was set to 3, for conservative plane growing, while for PDVO we used a value of 10, allowing planes to more rapidly fill the detected region, permissible since it is less sensitive to how planes are grown into unknown image regions.

First we consider the implications of the delay in initialisation while waiting for the plane detector to run, compared to the undelayed initialisation of (seeds of) planes by IDPP. In Figure 7.2 we show the development of a keyframe over several frames after initialisation in both methods. The first row shows the initial input image, and the result of plane detection used to initialise planes in the PDVO system. Following this are images showing the progression of plane estimation. It is clear that IDPP, in the left column, quickly initialised many planes, at many image locations (some of which were not at all planar), but these took some time to grow, and competed for measurements.

Figure 7.2: *Comparison of the initialisation of plane features using the original IDPP method (left) and when augmenting it with plane detection (PDVO, right). Images of the camera view after 2 (initialisation), 14 (detection ends), and 46 frames have elapsed are shown, demonstrating that although there is a delay of many frames while the plane detector runs, the good initial estimate makes up for this in terms of the number and quality of the resulting planes. The bottom row shows planes in 3D at frame 46.*

When using plane detection (right column), a single plane was initialised at the centroid of the detected region, and grew quickly. Even with a delay of around 14 frames before the detector finished, the plane expanded rapidly, overtaking those initialised by IDPP in number of measurements and image coverage. The bottom row shows 3D visualisations of the planes, corresponding to the last camera frame shown; the many planes created by IDPP had not yet attained good poses, while the plane initialised in PDVO already shows appropriate orientation. Again, this difference was partly because only the one plane was initialised, and because the plane prior allowed us to choose a less conservative rate for plane growing, highlighting that fact that the two methods operate in very different ways.

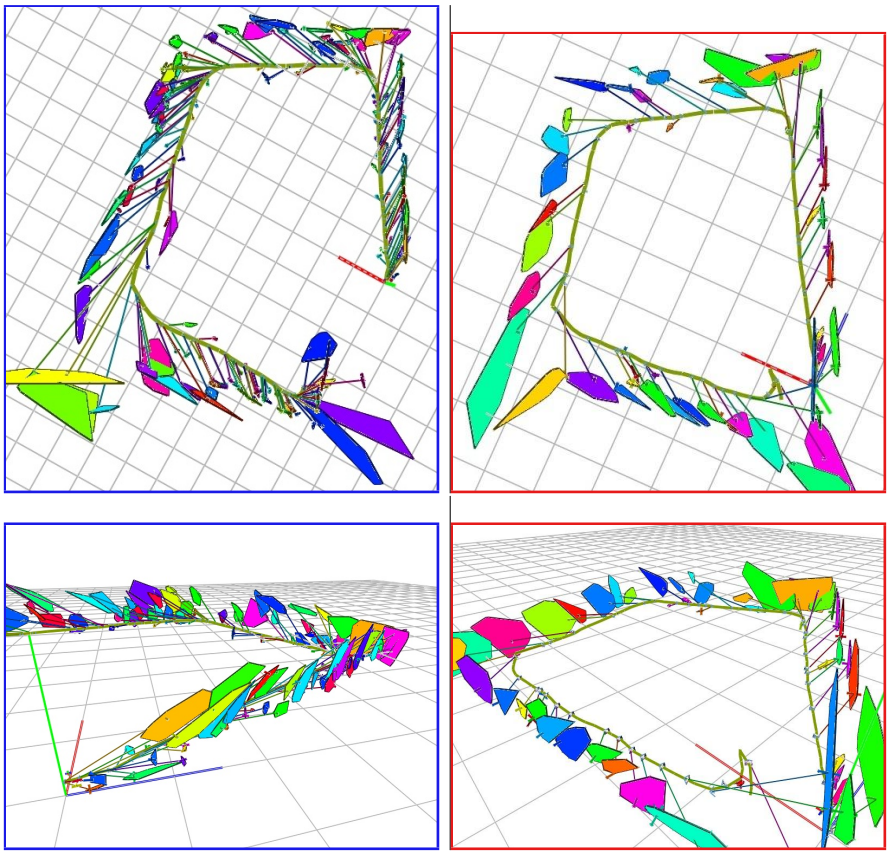


Figure 7.3: *Some views of the Berkeley Square sequence, showing the original IDPP (left) and our improved method (right). The top images show a top-down view of the whole path, while the lower images show oblique views, illustrating that the PDVO method produces less clutter and larger planes.*

Next, we compared the two methods on a long video sequence, as the camera traversed a large loop of approximately 300 metres — this was a square surrounded by houses, with trees on the inside (the Berkeley Square sequence). 3D views resulting from the two methods are compared in Figure 7.3; while both recovered an approximately correct

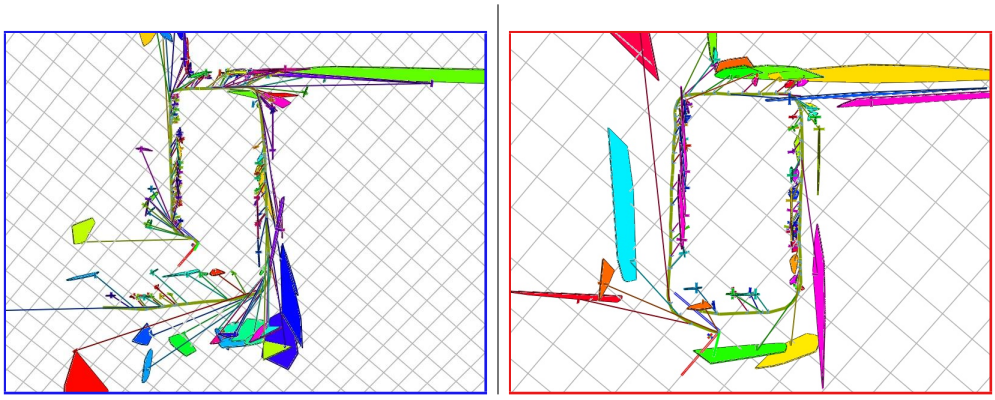


Figure 7.4: *Comparison on the Denmark Street sequence: IDPP (left) again has more numerous and smaller planes than PDVO (right) (note that the grid spacing is arbitrary and does not reflect actual scale).*

trajectory (the true path was not actually square, but the ends should meet) and placed planes parallel to the route along its length, it is clear in the PDVO method (right) that there are fewer planes, which tend to be larger and less cluttered, giving a clearer representation of the 3D environment. The oblique views underneath show this clearly, where compared to PDVO, the planes mapped by IDPP are smaller, more irregular, and with more varying orientations.

We also show results for another video sequence, taken in a residential area, surrounded by planes on all sides (the Denmark Street sequence), shown in Figure 7.4. Again, the map visualisation created using our method is more complete and clear than that with the original IDPP, with fewer and larger planes. Examples of planes as seen from the camera are shown in Figure 7.5, and further examples of plane detections acquired during mapping are shown in Figure 7.6, showing our detection algorithm is quite capable of operating in such an environment.

Method	Total planes	Points per plane	Average area (pixels)
IDPP	205	17.9	521.0
PDVO	52	28.9	1254.4

Table 7.1: *Comparison of summary statistics for the IDPP and PDVO methods, on the Berkeley Square sequence.*

Table 7.1 compares statistics calculated from mapping the Berkeley Square sequence, in order to quantify the apparent reduction in clutter. These confirm our intuition that when using plane priors, fewer planes will be initialised, by avoiding non-planar regions. Furthermore, the planes resulting from PDVO are measurably larger, both in terms of the average number of point measurements, and number of pixels covered.

(a) (b) (c) (d)

Figure 7.5: *Visual odometry as seen from the camera. For IDPP, many planes are initialised on one surface (a), or on non-planar regions (b); whereas PDVO has fewer, larger planes, being initialised only on regions classified as planes (c,d).*

Figure 7.6: *Examples of plane detection from the Berkeley Square sequence, showing the area deemed to be planar and its orientation. Note the crucial absence of detections on non-planar areas; and that multiple planes are detected, being separated according to their orientation.*

As we emphasised earlier, our intention is to show the potential for using the plane detection method for fast map building, and not necessarily to produce a more accurate visual odometry. However, it is interesting to analyse the accuracy of PDVO compared to IDPP against the areas' actual geography. Ground truth was not available, but the trajectories can be manually aligned with a map, as shown in Figures 7.7 and 7.8 for the Berkeley Square and Denmark Street sequences respectively. The latter is a compelling example, suggesting that, under certain conditions, our method helps to ameliorate the problem of scale drift (a well known problem for monocular visual odometry [119]); of course, many more repeated runs would be needed to quantify this, but we consider these initial tests to be good grounds for further investigation.

One of our main hypotheses was that by using strong structural priors, we can make mapping faster by more carefully selecting where to initialise planes. Our experiments confirmed this, according to Figure 7.9 where we compare the computation time (measured in frames per second) for both methods, on the Berkeley Square sequence. As previously reported in [89], the IDPP system achieves a frame rate of between 18 and 23 fps (itself an improvement on similar methods running at 1 fps [20]), which is confirmed

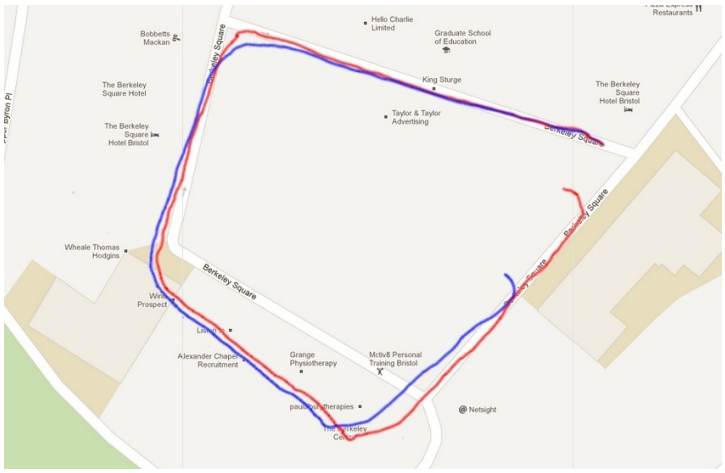


Figure 7.7: *In lieu of ground truth data, the trajectories were manually overlaid on a map for comparison, for the Berkeley Square sequence. The true trajectory closes the loop, and while both methods show noticeable drift, the error for our PDVO method (red) was an improvement on that of IDPP (blue).*

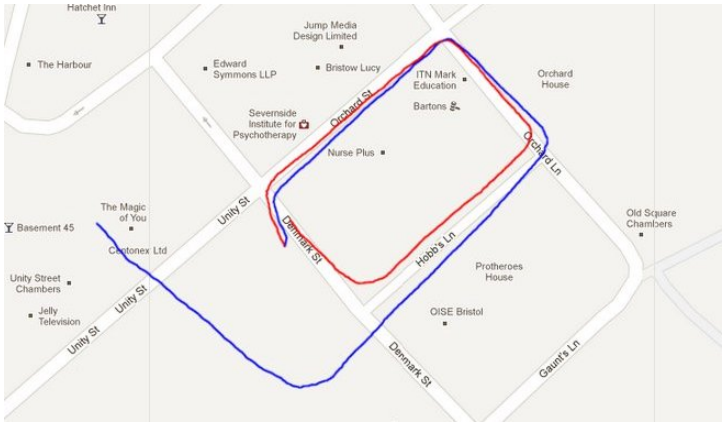


Figure 7.8: *A comparison of mapping ability of the two methods on the Denmark Street sequence, compared to a map. Again, both methods exhibit gradual drift, but this is reduced by our PDVO method (red) compared to IDPP (blue).*

by this experiment (blue curve). Our method clearly out-performed this, achieving both a substantially higher average frame rate of 60 fps and being consistently faster throughout the sequence (this times the VO threads, so does not include the time taken to run the plane detector). We are not aware of existing visual odometry systems running at such high frame rates for a similar level of accuracy, suggesting that our use of learned structural knowledge is a definite advantage. Running at such high speeds is beneficial since it means more measurements can be made for the same computational load, which tends to increase accuracy [120], or frees computation time for global map correction methods [119].

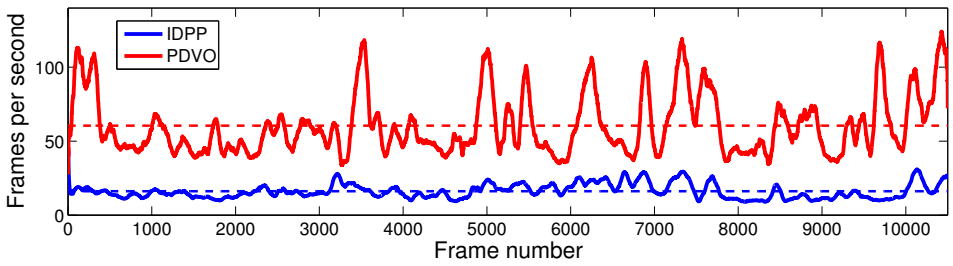


Figure 7.9: *Time (frames per second) for each of the methods (smoothed with a width of 100 frames for clarity). The mean is also shown for both.*

7.6 Conclusion

This chapter has described how we can use our plane detection algorithm as part of a visual odometry system, being a good example of a real-world application. We achieved this by modifying an existing plane-based visual odometry system to take planes from our detector and use them to quickly initialise planar features in appropriate image locations.

Part of the success of this approach was due to careful choice of the baseline VO system we used. The IDPP visual odometry system is based on taking measurements from one keyframe image, and growing planes from seed points, and these qualities make it ideal for incorporating planar priors, by using them to specify where on the key frame a plane should be initialised, and by having the confidence to grow these planes much faster. Furthermore, this VO system can make use of the single image estimate of the plane normal in a principled way, by using it to initialise the plane feature directly in the filter state. This means that our estimated value can help with faster initialisation, without having to wait for image measurements; but on the other hand reasonable errors in this value will not cause problems since it will be updated as more multi-view information becomes available.

However, there is no reason why this would be the only type of SLAM or VO system able to benefit from having plane priors, and we could consider the use of methods based on fitting planes to points [47] or based on bundle adjustment [132]. For example, we might use RANSAC to find collections of coplanar points in a point cloud, and filter out false planes using our plane detection algorithm, to detect planes both geometrically and with some semantic guidance.

A key contribution of this chapter was to show that by exploiting general prior knowledge

about the real world – encoded via training data – we can derive strong structural priors, which are useful for fast initialisation of map features. Direct use of such general prior information has not been done in this way before — the closest equivalents are Flint et al. [40] who use assumptions on the orthogonality of indoor scenes to semantically label planar surfaces, and Castle et al. [14] who use knowledge of specific planar objects to enhance the map and recover absolute scale.

7.6.1 Fast Map Building

We also demonstrated that by detecting planes almost immediately from a single camera frame, they can be inserted directly into the map, to quickly give a concise and meaningful representation of the 3D structure, again due to having good priors. While planes are added to the map as they are built in the regular VO system, of course, the difference is that we have good reason to believe these planes will better reflect the actual scene structure, as opposed to being planes grown from coincidental coplanar structure. This was supported by our results, showing the detected planes to be fewer in number, larger in size, and seeming to align better with the known scene layout.

It would be interesting to develop this further, toward producing fast and accurate plane-based 3D models of outdoor environments as they are traversed. This could be used, for example, to create quick visualisations of a scene, with textures on the planes taken from the camera images. By giving a better sense of the scene structure such maps would also be useful for robot navigation and path planning, being better able to avoid vertical walls or traverse ground planes; or for human-robot interaction, making it easier to communicate locations and instructions in terms of a common 3D map.

7.7 Future Work

This section discusses potential developments to this PDVO system, for further evaluation and use as part of an online learning-system; discussion of future work in other applications is deferred to Chapter 8.

7.7.1 Comparison to Point-Based Mapping

One important area which requires further investigation is exactly why we see the performance gains we do, in terms of frame-rate and accuracy. Our algorithm is able to initialise planes in a more intelligent way, by only creating seed points in regions deemed planar by our detector, which avoids the potential problems, and computational burden, of growing planes in inappropriate places. However, it could be that some of the benefits come simply from having fewer planes in total, for example if reducing the number of planes, irrespective of where they are created, is beneficial. If, hypothetically, introducing planar structures inevitably leads to errors, then using fewer of them would improve performance, calling into question to benefit of using single image plane detection. This could be investigated by comparing the plane detection enhanced version, and the standard plane-growing version of IDPP, to a purely point-based system, to evaluate the differences between them (note that the point features used in IDPP are parameterised using the efficient, unified parameterisation, and so have advantages over standard point-based systems).

We could also compare PDVO and a similar version which initialises the same number of planes, but in random locations. We would expect, if the conclusions of this chapter hold, that using the detected locations would be significantly better. Furthermore, the IDPP method was itself originally compared to point-only mapping, both in simulation and on real data, and found to be superior [85, 89]. This suggest it is likely that using detected planes as opposed to a random subset would be beneficial, so we maintain our assumption that plane detection provides benefit over only points.

7.7.2 Learning from Planes

Presently, the training data for PDVO comes from manually labelled training data, the creation of which is a time consuming process. An alternative would be to use the IDPP visual odometry system itself to detect planes, and use these as training data. Once IDPP has been used to map an environment, the result is a map with planar structure, with relates planes in the world to the keyframes from which they were observed. This means the information available in the keyframes is fairly similar to the type of ground truth data we have manually labelled, and so we could process these as described in Section 5.4, to extract training regions by sweeping over the whole image. Not only would this avoid manually annotating images, but would allow us to easily tailor the

detector to new environments, by obtaining a training set more similar to the type of scenery encountered.

There are a few issues to consider in pursuing this idea. Firstly, while keyframes are ideal for recovering the identified planar structures, it is not clear at which point during the planes' evolution they should be used as training data. It would be prudent to wait for planes to expand and to converge to a stable orientation, although planar points may be removed as they go out of view or are occluded, so waiting too long would lead to fewer or smaller planes. Points will also be removed from planes if they are later found not to be co-planar, so taking the largest coverage of planes would also be inadvisable. In addition to this, it would not be possible to know that regions without planes are indeed non-planar, since the absence of planes might simply be due to having sufficiently many measurements without initialising additional planes on the keyframe.

A further issue in using planes detected by IDPP – or indeed any geometric method – is that these cannot be used to determine what a human would consider planar, but only what the mapping algorithm considers to be planar. This may be useful, in terms of giving a prior on the locations of the kind of plane that will be detected by the mapping system; but it would no longer be encapsulating any human prior knowledge, or any planar characteristics complementary to what geometric methods can see. This is unfortunate, since it seems that a key benefit of PDVO is the ability to avoid planes in inappropriate places, and to predict their extent in the image, something which is difficult for IDPP.

The above idea can be extended further, by combining both training and detection (which would both be autonomous) into an online system. The ideal would be a combined plane-detection/plane-mapping visual odometry system that starts with no knowledge of planes or the environment. As it maps planes using multi-view information, it would gradually learn about their appearance. From this it would detect new planes from single images, increasing the efficiency of detection as it learns more about its surroundings. In order to achieve this it would be necessary to make some changes to the plane detection method, primarily to make training feasible in an online system. Training the RVM would no longer be practical, since this takes a large amount of time, and would require retention of the whole training set (since the relevance vectors are liable to change as more data become available). An alternative classifier would be necessary, for example random forests [12] which would be easier to incrementally train as data become available.

This would be a very interesting way to develop the PDVO algorithm, since it would be

a step toward creating a self-contained perceptual system, which explores its environment, learns from it, and uses this learned knowledge to aid further exploration. Again, we make an analogy with the way humans perceive their environment, as discussed in Chapter 1. Biological systems are capable of such learning, ultimately starting with no prior information, which implies it may be possible to design a vision system to achieve similar goals.

However, many challenges remain before developing such a system using the methods described above. Because training data would come from plane growing, this could lead to undesirable drift in what is considered a plane, if false planes are detected and used as training examples. Alternatively, if few planes are detected, there would be insufficient data to learn from, making the plane detector unable to help initialise new features. As such, building an online learning system with the methods we have discussed here remains a distant prospect requiring considerable further work.

CHAPTER 8

Conclusion

This thesis has investigated methods for finding structure in single images. This was inspired by the process of human vision, specifically the way that humans are thought to learn how to interpret complex scenes by virtue of their prior knowledge. As we discussed in Chapter 1, learning from experience appears to play an important part in how humans see the world — evinced be phenomena such as optical illusions. Since humans appear capable of perceiving structure from both reality and in pictures, without necessarily using stereo or parallax cues, this can provide useful insights into how computer vision algorithms might approach such tasks.

This motivated us to take a machine learning approach to tackle the problem, where rather than explicitly specify the model underlying single image perception, we learn from training data. This is similar in spirit to a number of approaches to tasks such as object recognition [37, 69], face verification [7], robotics [1], and so on. We are also motivated by other recent work [66, 113], who have also used machine learning methods for perceiving structure in single images; and driven by the range of possible applications single-image perception would have.

Amongst the myriad possibilities for attempting single-image perception (such as recovering depth or estimating object shapes), we have begun by focusing on the task of

plane detection. This was chosen because planes are amongst the simplest of geometric objects, making them easy to incorporate into models of a scene, and can be described with a small number of parameters. Furthermore, planes are ubiquitous in human-made environments, so can be used to compactly represent many different indoor or urban environments. The importance of this task is underlined by many recent works on plane detection generally [5, 42], as well as attempts to extract planar structure in single images [73, 93]. However, as we described in Chapter 2, existing methods for single image plane detection suffer shortcomings such as a dependence on certain types of feature, or an inability to accurately predict orientation, so we believe our new method satisfies an important need.

In order to begin the interpretation of planar scenes, we developed a method for recognising planes in single images and estimating their orientation, which uses basic image descriptors in a bag of words framework, enhanced with spatial information (Chapter 3). We use this representation to train classifiers which can then predict planarity and orientation for new, previously unseen image regions. We believe this is a good approach to the problem, since it avoids the use of extracting potentially difficult structures such as vanishing points, which may not be appropriate in many situations.

Our experiments in Chapter 4 confirm the validity of such a learning based approach, showing that it can deal with a variety of situations, including both regular Manhattan-like scenes and more irregular collections of surfaces. We acknowledge that our method may give orientation accuracies inferior to direct methods (using vanishing points, for example) in the more regular scenes. However, we are not bound by their constraints, and can predict orientation in the absence of any such regular structure. This chapter also explored a number of design choices involved in creating the algorithm. This is important, since it gives some insight into why the method achieves its results, and its potential limitations.

This work was not in itself complete, since it required the correct part of the image to be marked up. We addressed this in Chapter 5, where we demonstrated that this plane recognition algorithm can be incorporated into a full plane detection system, based on applying it multiple times over the image, in order to sample all possible locations of planes. This allows us to estimate planarity at each point (the 'local plane estimate'), which gives sufficient cues to be able to segment planes from non-planes, and from each other, implemented with a Markov random field (MRF).

We showed in Chapter 6 that this detection method is indeed capable of detecting planar

structure in various situations, including street scenes with orthogonal or vanishing-point structures, but also more general locations, without the kind of obvious planar structure usually required for such tasks. We emphasise that our algorithm also deals with non-planar regions, and can determine if there are not in fact any planes in the image. These experiments confirm our initial hypothesis, that by learning about the relationship between appearance and structure in single images, we can begin to perceive the structure in previously unseen images, without needing multi-view cues or depth information. Nevertheless, the task of perceiving structure generally is far from complete, in that we have looked only at planar structures so far. Moving on to more complicated types of scene would be an interesting avenue to explore in future, although potentially much more challenging.

Chapter 7 showed how our plane detector can be applied in a real application. Here we investigated its use for visual odometry (VO), a task where planes have been useful for efficient state representation and recovering higher-level maps [46, 87]. We experimented by modifying an existing visual odometry system [89] that can simultaneously grow planes and estimate their orientation in the map, while using them to localise the camera. We chose this system since it was clear that it would benefit from knowing the likely locations of planes. We used our plane detector to find planes in a set of keyframes, and from there initialise planar features in the map, using our estimated orientation as a prior. This allowed planes to be initialised only in locations where they should be, to grow quickly into their detected region and not exceed their bounds, and to be initialised with an approximate orientation, which while not perfect was better than assuming the planes face toward the camera. This increased the accuracy of the resulting maps, and drastically increased the frame-rate over the baseline VO system, while also allowing fast construction of plane-based maps, by retaining detected planes even after they had been removed from the state.

8.1 Contributions

Here we briefly summarise the key contributions of this thesis:

- We described a method of compactly representing image regions, using a variety of basic features in a bag of words framework, enhanced with spatial distribution information.

- Using this representation, we trained a classifier and regressor to predict the planarity and orientation of new candidate regions, and showed that this is accurate and performs well with a variety of image types.
- We developed this for use as part of a plane detection algorithm, which is able to recover the location and extent of planar structure in one image, and give reliable estimates of their 3D orientation.
- This method is novel, in that it is able to both detect planes, using general image information – without relying either on depth information or specific geometric cues – and is able to estimate continuous orientation (normal vector as opposed to orientation class).
- We also compared our method to a state of the art method for extracting geometric structure from a single image, and found it to compare well, with superior performance on average on our test set according to the evaluation measures we used.
- Finally, we demonstrated that single-image plane detection is useful in the context of monocular visual odometry, by giving reliable priors on both the location and orientation of planar structures, enabling faster and more accurate maps to be constructed.

8.2 Discussion

Having outlined the primary contributions and achievements of this work, we now discuss some of the more problematic areas. One implicit limitation in our current plane recognition algorithm is that it depends upon having a camera of known and constant calibration. Images from different cameras with different parameters may look considerably different, due to the effects of lens distortion or picture quality for example. This would impact upon classification accuracy, but could be solved by increasing the quantity of training data. However, while the calibration parameters do not appear in any of our equations or image representations, they are required in order to relate the four marked corners of the image to the ground truth normal (Section 3.2). This implies that if an image exists with an identical quadrilateral shape, that comes from a camera with different parameters, the true normal would actually be different, causing our algorithm to give erroneous results. This limits our algorithm to images and videos taken with the

same known camera (the results in this thesis were all obtained thus); and yet it would clearly be beneficial to attempt to generalise our method, firstly to work with any given camera matrix, and ultimately to be independent of calibration, to use images from a more diverse range of sources. We envisage a system able to freely harvest images from the internet, and intriguingly perhaps even from Google Street View[1], which already contains some orientation information.

As we discussed in Section 6.5.2, the model we have assumed in order to relate appearance to structure might cause problems. Specifically, by using a translation invariant descriptor (the spatiograms with zero-mean position), we are assuming that location in the image is irrelevant for either plane classification or orientation. Not only does this miss out on a potentially useful source of information, but is inaccurate, since experience shows that an identical planar appearance seen in different image locations may imply a different orientation. The brief experiments we conducted to investigate this fortunately show that it is not a pressing issue. The difference in orientation as the plane moves across an image is slight, and the results using both types of description showed broadly similar performance. Nevertheless, it would be desirable to ensure the way we represent appearance and orientation allows their true relationship to be expressed, and may lead to better results.

We described in Chapter 5 how we use a two-step process for segmenting planes according to class then orientation. We found this worked well in practice, but are aware that the use of two stages, plus mean shift to discretise the orientations, might be an over-complication. In principle it should be possible to perform the entire segmentation in one step, simultaneously estimating class and orientation for plane segments, while also finding the best set of orientations. This would perhaps be more efficient, or at least more computationally elegant. Such a one-step segmentation could potentially be achieved by treating it as a hierarchical segmentation problem on a MRF [78]; or by using a more sophisticated model such as a conditional random field, where the best parameters to use would be learned from labelled training data [105].

One significant problem which we observed (see for example the images in Chapter 6) is the plane detector's inability to adhere to the actual edges of planar surfaces. Frequently, planes will not reach the edges of the regions, since they only exist where salient points occur; but we also observe many cases where planes overlap the true boundaries and leak into other areas. This is an important problem since for any kind of 3D reconstruction,

[1] www.maps.google.com

or when using the planes to augment the image, this will lead to errors, making the actual structure more difficult to interpret. As we have discussed, making use of edge information, or explicitly classifying the boundaries, would be possible ways to mitigate this. As it stands, the fact that planes can be detected in the right location with broadly correct extent is encouraging, but expanding our algorithm to respect plane boundaries would not only significantly improve the presentation of the results, but bolster the case that learning from images can be more powerful than traditional techniques such as explicit rectangle detection.

We also note that one of the criticisms we levelled at the work of Hoiem et al. [66] is its dependence on scenes structured in a certain way — for example having a horizontal ground plane and a visible horizon. Our method is not constrained thus, since it does not explicitly require any such characteristics of its test images. Nevertheless, a useful avenue of future research would be to more thoroughly investigate this, by evaluating the performance of the algorithm on particularly unusual viewpoints (pointing upward to a ceiling, for example), or of images arbitrarily rotated. In theory our method would be able to cope with such situations (depending on the training data), and quantifying this would further support the idea of extrapolating from training data to unexpected scenarios.

We attempted to show in Chapter 7 how using the plane detector can improve monocular visual odometry. Our results are suggestive that this is possible, but we acknowledge that the experiments shown here are not rigorous enough to be certain of any increased accuracy. Ideally, we should have run the point-based, IDPP, and enhanced PDVO versions of the algorithms in simulation, to determine error bounds and verify consistency under carefully controlled conditions, before attempting to evaluate in a real-world scenario. More rigorous outdoor testing is also required to validate our apparent increase in trajectory accuracy over IDPP, using ground truth data if possible (using GPS, for example).

8.3 Future Directions

The plane detection algorithm we developed was shown to work in various situations, and to be useful in a real-world application. However, there are many avenues left unexplored, both in terms of improvements to the algorithm itself, and further development of the ideas to new situations. We have already described some modifications and extensions

in earlier chapters; in this section we briefly outline some further interesting directions this work could lead to.

8.3.1 Depth Estimation

To investigate the potential of using single images for structure perception, we have focused on plane detection, but it would be interesting to investigate other tasks using a similar approach. As we discussed in the introduction, another useful ability would be depth perception — something which has important implications in many situations, as evinced by the recent popularity of the Kinect sensor [96], which is able to sense depths in indoor locations. We have so far ignored the implications of depth for our plane detection, and assume either that it is not relevant (Chapter 5) or can be detected by other means (Chapter 7)

Estimating accurate depth maps for the whole image is challenging, and has been admirably tackled by Saxena et al. [112]. This is rather different from the task we considered, having focused so far on region-based description, whereas depth maps require finer grained perception. However, if we consider depth estimation for planes themselves, we could use the results of our plane detection algorithm as a starting point for perceiving depth for large portions of the scene. Thus, we would not need to estimate depth at individual pixels or superpixels, but by estimating a mean depth for a plane, could approximately position it in 3D space; along with other such planes, in relation to each other and to the camera, this would provide a rough but useful representation of the scene, perhaps even suitable for simple 3D visualisation [64].

8.3.2 Boundaries

We discussed at the end of Chapter 6 ideas for using edge information to enhance plane detection, in order to better perceive the boundaries between planar or non-planar regions. One other way in which we might explore this is to use other cues as well, such as segmentation information. Hoiem et al. [66] and Saxena et al. [113] used an over-segmentation of the image to extract initial regions, for scene layout and depth map estimation respectively; however, as we stated previously, we do not believe this would be appropriate for creating regions for plane detection. On the other hand, we could use the boundaries between segments as evidence for discontinuities between surfaces,

essentially treating it as a kind of edge detection, where the edges have more global significance and consistency. For example, Felzenszwalb and Huttenlocher's segmentation algorithm [36] tries to avoid splitting regions of homogeneous texture, while simply using a Canny edge detector could fill such regions with edges.

8.3.3 Enhanced Visual Mapping

As described in Chapter 7, using plane detection can enhance visual odometry, by showing it where planes should be initialised. Our implementation does not apply any consistency between plane detections, however, and assumes each frame is independent. This is a reasonable assumption when there is up to one second between frames, as the camera is likely to have moved on in between; but as available computing power increases (or we further optimise the detection algorithm), it may eventually be possible to apply plane detection to every frame in real time.

In this situation it would be desirable to impose some kind of temporal consistency, to use information across multiple images. This would not necessarily be stereo or multi-view information of course — as we have emphasised before, the camera may well be stationary or purely rotating between adjacent frames. We could, for example, use the previous frame to guide detection of planes in the current, to ensure the layout is similar. Without this, we would likely observe fairly large changes between frames, in orientation and even in number and size of planes, since the MRF segmentation is begun anew each time (plus the locations of salient points may shift dramatically). A principled way of achieving this would be to extend the MRF to have temporal as well as spatial links, so that points in two (or more) images are linked together in the graph, and segmentation takes both into account. This could be applied to video sequences using a sliding window approach, taking multiple frames at a time to build a graph across space and time.

If we can develop such an algorithm, to run in real time and give temporally consistent results, this would have many benefits for our visual odometry application. Rather than the plane detector running on whichever frame is next available – which might not be the best frames to use – we could send requests to the plane detector at times deemed necessary by the mapping system, in the knowledge that the result would be available fast enough to make a decision regarding initialisation. Planes would either be initialised immediately, or the plane detector be called again in a subsequent frame. Alternatively, the decision to initialise planes could come from the detector. If it is executed for every

incoming frame, we can decide only to initialise planar features in frames where they are particularly large or have good shape characteristics, and keep detecting in every frame to find the best opportunities.

8.3.4 Structure Mapping

Finally, we mention another potential approach to building maps using our plane detector, without being reliant on a pre-existing VO/SLAM system. If we can reliably detect planes and their orientation in one image, then move the camera and detect planes again, we could begin to attempt associating structures between frames. If the frames are close together in time, it is likely we would be viewing the same set of planes. This could be used to gauge depth (up to the unknown scale of the camera motion, as in monocular SLAM), and by aligning the orientation of the separate planes, recover a rough estimate of camera motion. We envisage a probabilistic approach where information from multiple frames is combined to try to accurately infer structure from unreliable plane measurements, while tracking the camera motion. This could be an interesting new approach to plane-based mapping, since the planes are detected first then used to build the map, the reverse of the standard approach.

We could also consider a more topological approach, in which the relationships between planes (identity across frames, and locations within frames) is maintained in a graph structure to build a semantic representation of the scene, without ever estimating locations in a global coordinate frame; this would be quite a different result from the structure-enhanced maps discussed above, more like the relational maps produced by FAB-MAP [27] and non-Euclidean relative bundle adjustment [91], and would be a very interesting way to develop our single-image algorithm into a way of gaining a larger scale semantic understanding of scenes as a whole.

8.4 Final Summary

To conclude, we briefly summarise what we have achieved in this thesis. We have investigated methods for single image perception, inspired by the learning-based process of human vision, and focused on plane detection in order to develop our ideas. Our hypothesis that a learning-based approach to perception can be successful was supported

in that we developed a single-region recognition algorithm for planes, followed by a full plane detection algorithm. We showed that this can be done with a significant relaxation of the assumptions previously relied upon; we retain only the requirement that the test images have appearance and structure reasonably similar to the training set. Therefore we conclude that a method based on learning from training examples is a valid and successful approach to understanding the content of images. Furthermore, we have shown that such an algorithm can be useful in the context of exploring and mapping an unknown outdoor environment, and able to extract a map of large-scale structures in real time.

We believe our specific approach has the potential to be further developed, to perform better using more visual information, and be adapted to cope with new tasks. There is also much interesting work left to do in terms of using machine learning techniques to perceive structure from single images, to move on from only finding and orienting planes and to understand their 3D relationships or gauge their depth, for example. Ultimately the goal would be to go beyond using only planar structures, and recover a more complex understanding of the scene, pushing ever further the extent to which human-inspired models of visual perception can be used to make sense of the 3D world.

References

[1] P. Abbeel, A. Coates, and A. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *Int. Journal of Robotics Research*, 29(13):1608–1639, 2010.

[2] S. Albrecht, T. Wiemann, M. Günther, and J. Hertzberg. Matching CAD object models in semantic mapping. In *Proc. IEEE Int. Conf. Robotics and Automation, Workshop*, 2011.

[3] L. Antanas, M. van Otterlo, O. Mogrovejo, J. Antonio, T. Tuytelaars, and L. De Raedt. A relational distance-based framework for hierarchical image understanding. In *Proc. Int. Conf. Pattern Recognition Applications and Methods*, 2012.

[4] O. Barinova, V. Konushin, A. Yakubenko, K. Lee, H. Lim, and A. Konushin. Fast automatic single-view 3-d reconstruction of urban scenes. In *Proc. European Conf. Computer Vision*, 2008.

[5] A. Bartoli. A random sampling strategy for piecewise planar scene segmentation. *Computer Vision and Image Understanding*, 105(1):42–59, 2007.

[6] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *Proc. Conf. Advances in Neural Information Processing Systems*, 2000.

[7] T. Berg and P. Belhumeur. Tom-vs-Pete classifiers and identity-preserving alignment for face verification. In *Proc. British Machine Vision Conf.*, 2012.

[8] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society B*, 48(3):259–302, 1986.

[9] N. Bhatti and A. Hanbury. Co-occurrence bag of words for object recognition. In *Proc. Computer Vision Winter Workshop, Czech Pattern Recognition Society*, 2010.

[10] S. Birchfield and S. Rangarajan. Spatiograms versus histograms for region-based tracking. In *Proc. IEEE Conf. Computer Vision and Pattern Recognitionn*, 2005.

[11] C. Bishop. *Pattern Recognition and Machine Learning*. Springer New York, 2006.

[12] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[13] L. Brown and H. Shvaytser. Surface orientation from projective foreshortening of isotropic texture autocorrelation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(6):584–588, 1990.

[14] R. Castle, G. Klein, and D. Murray. Combining monoSLAM with object recognition for scene augmentation using a wearable camera. *Image and Vision Computing*, 28(11):1548–1556, 2010.

[15] D. Chekhlov, M. Pupilli, W. Mayol-Cuevas, and A. Calway. Real-time and robust monocular SLAM using predictive multi-resolution descriptors. In *Proc. Int. Symposium on Visual Computing*, 2006.

[16] D. Chekhlov, A. Gee, A. Calway, and W. Mayol-Cuevas. Ninja on a plane: Automatic discovery of physical planes for augmented reality using visual SLAM. In *Proc. Int. Symposium on Mixed and Augmented Reality*, 2007.

[17] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[18] S. Choi. Algorithms for orthogonal nonnegative matrix factorization. In *Proc. Int. Joint Conf. Neural Networks*, 2008.

[19] J. Civera, A. Davison, and J. Montiel. Inverse depth parametrization for monocular SLAM. *IEEE Trans. Robotics*, 24(5), October 2008.

[20] J. Civera, O. Grasa, A. Davison, and J. Montiel. 1-point RANSAC for EKF-based structure from motion. In *Proc. Int Conf. Intelligent Robots and Systems*, 2009.

[21] J. Civera, D. Gálvez-López, L. Riazuelo, J. Tardós, and J. Montiel. Towards semantic SLAM using a monocular camera. In *Proc. Int. Conf. Intelligent Robots and Systems*, 2011.

[22] D. Comaniciu. An algorithm for data-driven bandwidth selection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 25(2):281–288, 2003.

[23] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[24] D. Comaniciu, V. Ramesh, and P. Meer. The variable bandwidth mean shift and data-driven scale selection. In *Proc. Int. Conf. Computer Vision*, 2001.

[25] O. Cooper and N. Campbell. Augmentation of sparsely populated point clouds using planar intersection. In *Proc. Int. Conf. Visualization, Imaging and Image Processing*, 2004.

[26] A. Criminisi, I. Reid, and A. Zisserman. Single view metrology. *Int. Journal of Computer Vision*, 40(2):123–148, 2000.

[27] M. Cummins and P. Newman. Highly scalable appearance-only SLAM FAB-MAP 2.0. In *Proc. Robotics Science and Systems*, 2009.

[28] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2005.

[29] A. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29(6): 1052–1067, 2007.

[30] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[31] C. Ding, T. Li, and W. Peng. On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing. *Computational Statistics & Data Analysis*, 52(8):3913–3927, 2008.

[32] P. Dorninger and C. Nothegger. 3D segmentation of unstructured point clouds for building modelling. *Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 35(3/W49A):191–196, 2007.

[33] S. Ekvall and D. Kragic. Receptive field cooccurrence histograms for object detection. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2005.

[34] Z. Fan, J. Zhou, and Y. Wu. Multibody motion segmentation based on simulated annealing. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2004.

[35] P. Favaro and S. Soatto. A geometric approach to shape from defocus. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(3):406–417, 2005.

[36] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *Int. Journal of Computer Vision*, 59(2):167–181, 2004.

[37] R. Fergus, M. Weber, and P. Perona. Efficient methods for object recognition using the constellation model. Technical report, California Institute of Technology, 2001.

[38] R. Fergus, P. Perona, and A. Zisserman. A sparse object category model for efficient learning and exhaustive recognition. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2005.

[39] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM Archive*, 24:381–395, 1981.

[40] A. Flint, C. Mei, D. W. Murray, and I. D. Reid. Growing semantically meaningful models for visual SLAM. In *Proc. IEEE Int. Computer Vision and Pattern Recognition*, 2010.

[41] D. Forsyth. Shape from texture and integrability. In *Proc. Int. Conf. Computer Vision*, 2001.

[42] D. F. Fouhey, D. Scharstein, and A. J. Briggs. Multiple plane detection in image pairs using j-linkage. In *Proc. ICPR*, 2010.

[43] J. Gårding. Shape from texture for smooth curved surfaces in perspective projection. *Journal of Mathematical Imaging and Vision*, 2:329–352, 1992.

[44] J. Gårding. Direct estimation of shape from texture. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(11):1202–1208, 1993.

[45] E. Gaussier and C. Goutte. Relation between PLSA and NMF and implications. In *Proc. Int. Conf. Research and Development in Information Retrieval*, 2005.

[46] A. Gee, D. Chekhlov, W. Mayol, and A. Calway. Discovering planes and collapsing the state space in visual SLAM. In *Proc. British Machine Vision Conf.*, 2007.

[47] A. Gee, D. Chekhlov, A. Calway, and W. Mayol-Cuevas. Discovering higher level structure in visual SLAM. *IEEE Trans. on Robotics*, 24:980–990, 2008.

[48] J. Gibson. *The Perception of the Visual World*. Houghton Mifflin, 1950.

[49] J. Gibson. The ecological approach to the visual perception of pictures. *Leonardo*, 11:227–235, 1978.

[50] J. J. Gibson. The information available in pictures. *Leonardo*, 4:27–35, 1971.

[51] E. H. Gombrich. *Interpretation: Theory and Practice*, chapter The Evidence of Images: 1 The Variability of Vision, pages 35–68. 1969.

[52] L. Gong, T. Wang, F. Liu, and G. Chen. A lie group based spatiogram similarity measure. In *Proc. IEEE Int. Conf. Multimedia and Expo*, 2009.

[53] R. Gregory. Perceptions as hypotheses. *Philosophical Trans. Rotal Society*, B 290: 181–197, 1980.

[54] R. Gregory. Knowledge in perception and illusion. *Philosophical Trans. Royal Society of London. Series B: Biological Sciences*, 352(1358):1121–1127, 1997.

[55] R. Gregory and P. Heard. Border locking and the café wall illusion. *Perception*, 8: 365–380, 1979.

[56] A. Gupta and M. Efros, A. Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *Proc. European Conf. Computer Vision*, 2010.

[57] O. Haines and A. Calway. Detecting planes and estimating their orientation from a single image. In *Proc. British Machine Vision Conf.*, 2012.

[58] O. Haines and A. Calway. Estimating planar structure in single images by learning from examples. In *Proc. Int. Conf. Pattern Recognition Applications and Methods*, 2012.

[59] O. Haines, J. Martínez-Carranza, and A. Calway. Visual mapping using learned structural priors. In *Proc. IEEE Int. Conf. Robotics and Automation*, 2013.

[60] J. M. Hammersley and P. Clifford. Markov fields on finite graphs and lattices.

[61] A. Handa, M. Chli, H. Strasdat, and A. Davison. Scalable active matching. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2010.

[62] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[63] T. Hofmann. Probabilistic latent semantic analysis. In *Proc. of Uncertainty in Artificial Intelligence*, 1999.

[64] D. Hoiem, A. Efros, and M. Hebert. Automatic photo pop-up. *ACM Trans. Graphics*, 24(3):577–584, 2005.

[65] D. Hoiem, A. Efros, and M. Hebert. Putting objects in perspective. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2006.

[66] D. Hoiem, A. Efros, and M. Hebert. Recovering surface layout from an image. *Int. Journal of Computer Vision*, 75(1):151–172, 2007.

[67] D. Hoiem, A. Stein, A. Efros, and M. Hebert. Recovering occlusion boundaries from a single image. In *Proc. Int. Conf. Computer Vision*, 2007.

[68] J. Hou, J. Kang, and N. Qi. On vocabulary size in bag-of-visual-words representation. *Advances in Multimedia Information Processing*, pages 414–424, 2010.

[69] M. Jones and P. Viola. Robust real-time object detection. In *Proc. Workshop on Statistical and Computational Theories of Vision*, 2001.

[70] S. Kim, K. Yoon, and I. Kweon. Object recognition using a generalized robust invariant feature and Gestalt's law of proximity and similarity. *Pattern Recognition*, 41(2):726–741, 2008.

[71] G. Klein and D. Murray. Full-3D edge tracking with a particle filter. 2006.

[72] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Int. Symposium on Mixed and Augmented Reality*, 2007.

[73] J. Košecká and W. Zhang. Extraction, matching, and pose recovery based on dominant rectangular structures. *Computer Vision and Image Understanding*, 100 (3):274–293, 2005.

[74] P. Koutsourakis, L. Simon, O. Teboul, G. Tziritas, and N. Paragios. Single view reconstruction using shape grammars for urban environments. In *Proc. Int. Conf. Computer Vision*, 2009.

[75] J. Lavest, G. Rives, and M. Dhome. Three-dimensional reconstruction by zooming. *IEEE Trans. Robotics and Automation*, 9(2):196–207, 1993.

[76] D. Lee and H. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[77] D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *Proc. European Conf. Machine Learning*, 1998.

[78] S. Li. *Markov Random Field Modeling in Image Analysis*. Springer-Verlag New York Inc, 2009.

[79] T. Lindeberg. Scale-space. *Encyclopedia of Computer Science and Engineering*, 4: 2495–2504, 2009.

[80] T. Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of applied statistics*, 21(1-2):225–270, 1994.

[81] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2):91–110, 2004.

[82] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. Int. Conf. Computer Vision*, 1999.

[83] D. Lyons. Sharing and fusing landmark information in a team of autonomous robots. In *Proc. Society of Photo-Optical Instrumentation Engineers Conf.*, 2009.

[84] C. D. Manning and H. Raghavan, P. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[85] J. Martínez-Carranza. *Efficient Monocular SLAM by Using a Structure Driven Mapping*. PhD thesis, University of Bristol, 2012.

[86] J. Martínez-Carranza and A. Calway. Appearance based extraction of planar structure in monocular SLAM. In *Proc. Scandinavian Conf. Image Analysis*, 2009.

[87] J. Martínez-Carranza and A. Calway. Efficiently increasing map density in visual SLAM using planar features with adaptive measurement. In *Proc. British Machine Vision Conf.*, 2009.

[88] J. Martínez-Carranza and A. Calway. Unifying planar and point mapping in monocular SLAM. In *Proc. British Machine Vision Conf.*, 2010.

[89] J. Martínez-Carranza and A. Calway. Efficient visual odometry using a structure-driven temporal map. 2012.

[90] P. Meer, D. Mintz, A. Rosenfeld, and D. Kim. Robust regression methods for computer vision: A review. *Int. Journal of Computer Vision*, 6(1):59–70, 1991.

[91] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid. RSLAM: A system for large-scale mapping in constant-time using stereo. *Int. Journal of Computer Vision*, 2010.

[92] J. Michels, A. Saxena, and A. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proc. Int. Conf. Machine learning*, 2005.

[93] B. Mičušík, H. Wildenauer, and J. Košecká. Detection and matching of rectilinear structures. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2008.

[94] B. Micušık, H. Wildenauer, and M. Vincze. Towards detection of orthogonal planes in monocular images of indoor environments. In *Proc. IEEE Int. Conf. Robotics and Automation*, 2008.

[95] N. Molton, A. Davison, and I. Reid. Locally planar patch features for real-time structure from motion. In *Proc. British Machine Vision Conf.*, 2004.

[96] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Proc. Int. Symposium on Mixed and Augmented Reality*, 2011.

[97] C. Ò Conaire, N. O'Connor, and A. Smeaton. An improved spatiogram similarity measure for robust object localisation. In *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2007.

[98] J. Oliensis. Uniqueness in shape from shading. *Int. Journal of Computer Vision*, 6(2):75–104, 1991.

[99] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *Int. Journal of Computer Vision*, 42(3):145–175, 2001.

[100] A. Oliva and A. Torralba. Scene-centered description from spatial envelope properties. In *Proc. Biologically motivated computer vision*, 2002.

[101] G. Orbán, J. Fiser, R. Aslin, and M. Lengyel. Bayesian model learning in human visual perception. *Advances in neural information processing systems*, 18:1043, 2006.

[102] M. Parsley and S. Julier. SLAM with a heterogeneous prior map. In *Proc. SEAS-DTC Conf.*, 2009.

[103] T. Pietzsch. Planar features for visual SLAM. *Advances in Artificial Intelligence*, pages 119–126, 2008.

[104] M. Pupilli and A. Calway. Real-time camera tracking using known 3D models and a particle filter. In *Proc. Int. Conf. Pattern Recognition*, 2006.

[105] A. Quattoni, S. Wang, L. Morency, M. Collins, and T. Darrell. Hidden conditional random fields. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29(10): 1848–1852, 2007.

[106] C. Rasmussen and J. Quinonero-Candela. Healing the relevance vector machine through augmentation. In *Proc. Int. Conf. Machine learning*, 2005.

[107] E. Ribeiro and E. Hancock. Estimating the 3D orientation of texture planes using local spectral analysis. *Image and Vision Computing*, 18(8):619–631, 2000.

[108] E. Ribeiro and E. Hancock. Estimating the perspective pose of texture planes using spectral analysis on the unit sphere. *Pattern recognition*, 35(10):2141–2163, 2002.

[109] L. Roberts. Machine perception of three-dimensional solids. Technical report, DTIC Document, 1963.

[110] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. *Lecture Notes in Computer Science*, 3951:430–443, 2006.

[111] A. Saxena, J. Schulte, and A. Ng. Depth estimation using monocular and stereo cues. In *Proc. Int. Joint Conf. Artificial Intelligence*, 2007.

[112] A. Saxena, S. Chung, and A. Ng. 3-d depth reconstruction from a single still image. *Int. Journal of Computer Vision*, 76(1):53–69, 2008.

[113] A. Saxena, M. Sun, and A. Ng. Make3D: learning 3D scene structure from a single still image. *IEEE Trans. Pattern Analysis and Machine Intelligence*, pages 824–840, 2008.

[114] D. Scaramuzza, F. Fraundorfer, and R. Siegwart. Real-time monocular visual odometry for on-road vehicles with 1-point RANSAC. In *Proc. IEEE Int. Conf. Robotics and Automation*, 2009.

[115] S. Scott and S. Matwin. Feature engineering for text classification. In *Proc. Machine Learning Int. Workshop*, 1999.

[116] H. Shimodaira. A shape-from-shading method of polyhedral objects using prior information. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(4):612–624, 2006.

[117] G. Silveira, E. Malis, and P. Rives. An efficient direct approach to visual SLAM. *IEEE Trans. on Robotics*, 24(5):969–979, 2008.

[118] D. A. Sinclair. S-Hull: a fast radial sweep-hull routine for Delaunay triangulation. Technical report, S-Hull, Cambridge UK, 2010.

[119] H. Strasdat, J. Montiel, and A. Davison. Scale drift-aware large scale monocular SLAM. In *Proc. Robotics Science and Systems*, 2010.

[120] H. Strasdat, J. Montiel, and A. Davison. Real-time monocular SLAM: Why filter? In *Proc. IEEE Int. Conf. Robotics and Automation*, 2010.

[121] E. Sudderth, A. Torralba, W. Freeman, and A. Willsky. Depth from familiar objects: A hierarchical model for 3D scenes. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2006.

[122] B. Super and A. Bovik. Planar surface orientation from texture spatial frequencies. *Pattern Recognition*, 28(5):729–743, 1995.

[123] A. Thayananthan, R. Navaratnam, B. Stenger, P. Torr, and R. Cipolla. Multivariate relevance vector machines for tracking. In *Proc. European Conf. Computer Vision*, 2006.

[124] S. Thorpe, D. Fize, and C. Marlot. Speed of processing in the human visual system. *Nature*, 381(6582):520–522, 1996.

[125] M. Tipping. Sparse Bayesian learning and the relevance vector machine. *The Journal of Machine Learning Research*, 1, 2001.

[126] M. Tipping and A. Faul. Fast marginal likelihood maximisation for sparse Bayesian models. In *Proc. Int. Workshop on Artificial Intelligence and Statistics*, 2003.

[127] A. Torralba and A. Oliva. Depth estimation from image structure. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(9):1226–1238, 2002.

[128] A. Torralba and A. Oliva. Semantic organization of scenes using discriminant structural templates. In *Proc. Int. Conf. Computer Vision*, 1999.

[129] A. Tremeau and N. Borel. A region growing and merging algorithm to color segmentation. *Pattern Recognition*, 30(7):1191–1203, 1997.

[130] R. Unnikrishnan, C. Pantofaru, and M. Hebert. Toward objective evaluation of image segmentation algorithms. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29(6):929–944, 2007.

[131] V. Viitaniemi and J. Laaksonen. Spatial extensions to bag of visual words. In *Proc. ACM Int. Conf. Image and Video Retrieval*, 2009.

[132] S. Wangsiripitak and D. Murray. Reducing mismatching under time-pressure by reasoning about visibility and occlusion. *Journal of Computer Vision*, 60(2):91–110, 2004.

[133] A. Witkin. Recovering surface shape and orientation from texture. *Artificial Intelligence*, 17(1-3):17–45, 1981.

[134] J. Yang, Y. Jiang, A. Hauptmann, and C. Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Proc. Int. Workshop on Multimedia Information Retrieval*, 2007.

[135] J. Yoo and S. Choi. Orthogonal nonnegative matrix factorization: Multiplicative updates on Stiefel manifolds. *Intelligent Data Engineering and Automated Learning*, pages 140–147, 2008.

[136] M. Zucchelli, J. Santos-Victor, and H. Christensen. Multiple plane segmentation using optical flow. In *Proc. British Machine Vision Conf.*, pages 313–322, 2002.