Visual Navigation for Mobile Robots using the Bag-of-Words Algorithm

Tom Botterill

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

University of Canterbury, Christchurch, New Zealand

2010

Abstract

Robust long-term positioning for autonomous mobile robots is essential for many applications. In many environments this task is challenging, as errors accumulate in the robot's position estimate over time. The robot must also build a map so that these errors can be corrected when mapped regions are re-visited; this is known as Simultaneous Localisation and Mapping, or SLAM.

Successful SLAM schemes have been demonstrated which accurately map tracks of tens of kilometres, however these schemes rely on expensive sensors such as laser scanners and inertial measurement units. A more attractive, low-cost sensor is a digital camera, which captures images that can be used to recognise where the robot is, and to incrementally position the robot as it moves. SLAM using a single camera is challenging however, and many contemporary schemes suffer complete failure in dynamic or featureless environments, or during erratic camera motion. An additional problem, known as scale drift, is that cameras do not directly measure the scale of the environment, and errors in relative scale accumulate over time, introducing errors into the robot's speed and position estimates.

Key to a successful visual SLAM system is the ability to continue operation despite these difficulties, and to recover from positioning failure when it occurs. This thesis describes the development of such a scheme, which is known as BoWSLAM. BoWSLAM enables a robot to reliably navigate and map previously unknown environments, in real-time, using only a single camera.

In order to position a camera in visually challenging environments, BoWSLAM combines contemporary visual SLAM techniques with four new components. Firstly, a new Bag-of-Words (BoW) scheme is developed, which allows a robot to recognise places it has visited previously, without any prior knowledge of its environment. This BoW scheme is also used to select the best set of frames to reconstruct positions from, and to find efficient wide-baseline correspondences between many pairs of frames. Secondly, BaySAC, a new outlier-robust relative pose estimation scheme based on the popular RANSAC framework, is developed. BaySAC allows the efficient computation of multiple position hypotheses for each frame. Thirdly, a graph-based representation of these position hypotheses is proposed, which enables the selection of only reliable position estimates in the presence of gross outliers. Fourthly, as the robot explores, objects in the world are recognised and measured. These measurements enable scale drift to be corrected. BoWSLAM is demonstrated mapping a 25 minute 2.5km trajectory through a challenging and dynamic outdoor environment in real-time, and without any other sensor input; considerably further than previous single camera SLAM schemes.

Acknowledgements

I would like to thank the following people who have helped so much with my research. Firstly, my supervisors, Steven Mills and Richard Green for all the invaluable advice they have provided over the last three years, and for all their help in proof-reading this thesis. I'm especially grateful to Steve for his continued involvement in my work. Secondly, I'd like to thank all the people I've worked with at the Geospatial Research Centre, in particular Chris Hide, for our productive collaboration. Thirdly, I'd like to thank everyone who has provided feedback and suggestions on my work and on this thesis. Finally, and most importantly, I'd like to thank my partner Hilary McMillan, who has provided a massive amount of help and advice over the last three years, particularly in proof-reading this thesis.

Contents

Publications

BoWSLAM, and many of its component parts, have previously been described in several refereed publications, which are listed below.
Firstly, the first version of our BoW scheme for fast robot localisation in previously unknown environments is described in the following paper:

> T. Botterill, S. Mills, and R. Green. Speeded-up Bag-of-Words algorithm for robot localisation through scene recognition. In *Proceedings of Image and Vision Computing New Zealand*, Lincoln, NZ, November 2009.

Secondly, BaySAC, our improved version of the RANSAC robust estimation framework which allows camera positions to be computed quickly and reliably from unreliable feature matches, is described in the following paper:

> T. Botterill, S. Mills, and R. Green. New conditional sampling strategies for speeded-up RANSAC. In *Proceedings of the British Machine Vision Conference*, London, September 2009.

Thirdly, the BoWSLAM scheme for robust single camera SLAM, is described in the following paper:

> T. Botterill, S. Mills, and R. Green. Bag-of-words-driven Single Camera SLAM. *To appear in the Journal of Field Robotics*, 2010.

Fourthly, initial work on the SCORE scheme to learn and recognise classes of objects in a robot's environment, and to use measurements of these objects to correct scale drift, is described in the following paper:

> T. Botterill, R. Green, and S. Mills. A Bag-of-Words Speedometer for Single Camera SLAM. In *Proceedings of Image and Vision Computing New Zealand*, Wellington, NZ, November 2009.

In addition I have co-authored several publications with Chris Hide et al., which integrate several components of BoWSLAM into an inertial measurement unit (IMU)-based indoor pedestrian navigation sensor. Firstly, a navigation sensor using BoW location recognition and an IMU to localise a pedestrian in a previously-mapped environment is described in the following paper:

> C. Hide, T. Botterill, and M. Andreotti. An integrated IMU, GNSS and image recognition sensor for pedestrian navigation. In *Proceedings of the Institute of Navigation GNSS Conference*, Fort Worth, Texas, 2009.

Secondly, details of how to compute camera orientations relative to a prior map using BaySAC, and how to integrate these measurements into a pedestrian navigation system, are described in the following paper:

> C. Hide and T. Botterill. Development of an integrated IMU, image recognition and orientation sensor for pedestrian navigation. In *Proceedings of the International Technical Meeting of the Institute of Navigation*, San Diego, CA, 2010.

Thirdly, a scheme to use a hand-held camera as part of a pedestrian navigation scheme, by computing a sparse optical flow field from the ground plane using BaySAC, then integrating these measurements of orientation and velocity with measurements from an IMU, are described in the following paper:

> C. Hide, T. Botterill, and M. Andreotti. Vision-aided IMU for pedestrian navigation. In *Proceedings of the Institute of Navigation GNSS Conference*, Fort Worth, Texas, 2010.

Chapter 1

Introduction

Reliable long term positioning is an essential enabling technology for autonomous mobile robots (a 'core competency' according to Thrun et al., 2005), and many solutions have been proposed. However, in many real-world situations reliable positioning is still not possible, for example positioning indoors with low-cost sensors. In many environments cameras appear ideal sensors for long term positioning; they collect rich information about the environment that may be used both to recognise when places are re-visited, even when the robot is lost, and to estimate the incremental motion of the robot. Previous solutions for positioning using vision can accurately position a camera for long periods over a limited area, but on a larger scale they often suffer from several major problems. These problems include including complete failure when measurements are interpreted incorrectly, difficulties negotiating visually challenging environments, and high algorithmic complexity which limits large scale operation. As a result, they have only been demonstrated on relatively small-scales, of a few hundred metres at most, and through mostly static environments.

In order to address the limitations of previous schemes, and to demonstrate that a single camera is a suitable sensor for robust long-term positioning, I have developed a new scheme, BoWSLAM, which enables a mobile robot equipped with a single camera to position itself in real-time as it explores a previously unknown environment. BoWSLAM combines the latest contemporary robot navigation techniques with new algorithms designed to enable the successful navigation of dynamic environments despite erratic, unpredictable camera motion, corrupted position estimates, total disorientation, and large accumulated errors. A map of the environment that has been explored is built and this map is refined and improved when places are re-visited, hence enabling accurate long-term positioning.

The novel algorithms which enable BoWSLAM to operate robustly in these circumstances include firstly, a Bag-of-Words (BoW) scheme allowing a robot to recognise places that have been visited before, without requiring any prior knowledge of its environment. Secondly, a new robust estimation framework, BaySAC, is developed, which enables reliable camera positioning in visually challenging dynamic environments. Thirdly, a scheme to efficiently calculate many position hypotheses from the BoW image representation is proposed, and a graph-based map of these multiple position hypotheses is developed, allowing robust mapping based on only the most reliable position estimates. Fourthly, the SCORE2 (Scale Correction by Object Recognition) scheme is developed. SCORE2 recognises and measure objects which are observed, then to uses these measurements to improves the accuracy of maps. BoWSLAM is demonstrated mapping a 2.5km trajectory in real-time through a dynamic environment, with no other sensor input, motion assumptions or odometry input. Together with results from several other large scale indoor and outdoor datasets, this demonstrates that a single camera is a feasible sensor for long-term navigation. BoWSLAM is designed to enable the positioning of low-cost robots with poor odometry or unpredictable motion, for example walking robots or quadrotors, or to aid pedestrian navigation.

Although designed to enable reliable navigation using a single camera, these new algorithms have many additional applications. The BoW scheme has been integrated into other navigation schemes where the ability to recognise a camera's location is useful (Hide et al., 2009; Hide and Botterill, 2010), and could be incorporated into schemes providing location-based services. The BaySAC framework can be applied to a

wide range of computer vision problems where outliers are common, and for robust estimation problems in other fields. BoWSLAM's robust mapping framework, where only the most reliable of multiple position hypotheses are used, is applicable to any robot navigation scheme where gross errors can occur. Finally, the combination of algorithms proposed has applications to other computer vision problems, such as real-time mapping.
This chapter is arranged as follows: the following section describes the potential applications of autonomous mobile robots, Section 1.2 outlines the problem of long-term positioning for these mobile robots, Section 1.3 gives an overview of the proposed new solution for long-term robot positioning, together with an outline of this thesis, and Section 1.4 summarises the key contributions of this thesis.

1.1 Motivation

Mobile robots that can position themselves autonomously over long periods are enabling the automation of tasks previously requiring human operators. Examples include robotic vehicles equipped with absolute position sensors such as Global Positioning System (GPS) receivers which plough fields (Tillett, 1991), search for meteors in the Antarctic (Apostolopoulos et al., 2000), or drive autonomously through a busy urban environment in order to compete in the DARPA urban challenge (Leonard et al., 2008). Other prototype robots that can localise themselves within previously mapped and tagged environments include automated forklift trucks working in factories (Tamba et al., 2009), and driverless taxis (Firmin, 2006). However in many environments Global Navigation Satellite System (GNSS) signals are unavailable or unreliable, and prior mapping, or the addition of localisation infrastructure, is impractical or prohibitively expensive. In these environments successful applications of autonomous mobile robots are rare. Mobile robots' inability to position themselves reliably for long periods is limiting their potential applications.
These environments where low-cost absolute positioning is not feasible include the places many people work, such as homes, offices and supermarkets. Mobile robots working in these environments today are limited to the simplest tasks where positioning is not required (such as floor-cleaning by following a somewhat inefficient random walk; Prassler et al., 2000), but robots that could position themselves autonomously could one day automate many additional tasks such as providing an indoor courier service (Engelberger, 1993) or carrying out environmental monitoring missions (Elfes et al., 2008). Other environments where autonomous mobile robots could usefully work include places where it is dangerous or expensive to send human workers, such as in contaminated sites, underwater, or in dense forests. Autonomous mobile robots also have the potential to work on other planets or deep underground; places out-of-range of human operators.
Applications of robot positioning schemes are not limited to autonomous mobile robots; they include providing real-time positions and maps to operators of remotely controlled vehicles, semi-autonomous wheelchair control (Park et al., 2010), and the construction of detailed digital models (Newcombe and Davison, 2010) or maps (Hygounenc et al., 2004; Johnson-Roberson et al., 2010). A closely related problem is pedestrian navigation, with applications including 'pedestrian satnav' systems (Hide et al., 2009), providing location-based services to users of mobile devices (Raper et al., 2007) or wearable computers (Wangsiripitak and Murray, 2009), and navigation aids for blind or partially sighted people (Coughlan and Manduchi, 2009).

1.2 Mobile robots and robot positioning

A vast range of mobile robot designs have been proposed or constructed. These robots, the environments in which they operate, and the sensors with which they are equipped are described in detail in the following chapter. These robots including wheeled, walking, flying and swimming robots, which are equipped with a wide range of sensors which may be used for positioning, including Inertial Measurement Units (IMUs), laser and sonar scanners, digital cameras, and magnetometers. Information about the robot's control inputs is also useful for positioning, in conjunction with odometry measurements (for example wheel movement measured using a wheel encoder). These sensor and odometry measurements can be used for positioning in two different ways: firstly, sensors which measure a robot's environment may be used to localise the robot,

by matching features of the environment to the robot's internal map. Secondly, the robot can be positioned incrementally: its position relative to a previous pose can be computed from a combination of its control inputs or odometry, its observations of the environment which change as it moves, and knowledge about its dynamics and kinematics. These small relative position estimates can be accumulated to reconstruct the robot's path as it moves through its environment; this is known as dead-reckoning.

All of these sensor measurements contain uncertainty however, and position estimates obtained by dead-reckoning will accumulate errors over time, eventually rendering them useless. For long-term positioning the environment being explored must also be mapped, so that when the robot re-visits mapped areas it can correct errors in both its position estimate and in the map. This combination of positioning and mapping is known as Simultaneous Localisation and Mapping, or SLAM (Dissanayake et al., 2001), and numerous SLAM schemes have been proposed; these are reviewed in detail in Chapter 3.

SLAM schemes have been proposed using many different combinations of sensors, and some recent schemes allow certain robots to position themselves over tracks of many kilometres, and to generate accurate maps of everywhere that has been explored (Nieto et al., 2003; Newman et al., 2009; Bosse and Zlot, 2008; Sibley et al., 2010). These schemes still have many limitations however, including complete failure when measurements containing gross errors are accidentally used (Bailey and Durrant-Whyte, 2006; Thrun, 2002), and high computational cost and complexity limiting their long term use. These limitations must be overcome if SLAM is to be used for real-world applications.

A further limitation of these SLAM schemes are the relatively expensive, heavy and power-hungry sensors used, including laser scanners and IMUs. For positioning of lightweight, low-cost robots, for example robots that will work in homes or carry out environmental monitoring, the use of lower-cost sensors is desirable. In addition, some of these robots, such as walking robots or small flying robots, have poor odometry which is also subject to gross errors. SLAM schemes suitable for these platforms, or for human navigation aids, must be able to navigate without reliable odometry inputs.

One low-cost, reasonably low-power and passive sensor which is particularly suitable for robust robot positioning is a digital camera. Cameras can operate in a wide range of environments and capture large amounts of detailed information about the world. This information may be used for positioning in three ways: firstly for localisation, by recognising places which have been visited previously; secondly, for incremental positioning, by observing the positions of objects relative to the camera as the camera moves; and thirdly for mapping the environment. Unlike many other sensors, cameras collect enough distinctive information about the world to recognise where they are even after becoming completely lost (Cummins and Newman, 2009; Konolige et al., 2009). While a stereo camera provides depth information useful for positioning, this is not necessary; a single camera can infer the depth of observed objects either by observing them from different positions, or by recognising objects of known size.

While a single camera appears to be an ideal sensor for SLAM, existing schemes for navigating using monocular vision are prone to failure due to gross errors or disorientation in difficult environments (Chekhlov et al., 2006), as both relative position estimates and absolute localisation estimates from computer vision are subject to gross errors. The causes of these errors include incorrect matching of similar-looking objects between pairs of frames (Eade and Drummond, 2008), and moving objects (Cummins and Newman, 2008a; Williams et al., 2008). In addition, high computational costs caused by high algorithmic complexity eventually limit many single camera SLAM schemes' real-time operation.

Given these limitations, I propose that a SLAM scheme allowing robust positioning using a single camera should be designed to meet the following five requirements:

- The SLAM scheme must actively detect when known locations are re-visited, in order to recover from gross errors in position, and to maintain a consistent and accurate map for extended periods.
- The scheme must have the ability to compute relative positions from frames captured from significantly different viewpoints, so that positioning does not fail when sequences of frames are unusable due to occlusion, motion blur, or erratic camera motion.
- Position estimates must be robust in the presence of moving objects and in self-similar environments.

- Despite robust position estimation, errors will inevitably still occur. A framework with the ability to reject these erroneous position estimates before they corrupt a global map is essential for long-term navigation.
- The computational cost and complexity of SLAM must be low enough that a usefully large environment can be explored, and must grow only as a function of the area which is explored, hence enabling positioning to continue indefinitely in a bounded environment.

The remainder of this thesis describes BoWSLAM, a single camera SLAM scheme designed to meet these requirements. BoWSLAM combines a range of efficient contemporary techniques with new robust algorithms, in order to meet these criteria. These new algorithms include a new BoW scheme, which enables a robot to recognise where it is, even when completely lost, and also enables features to be matched efficiently between frames. A selection of the latest contemporary algorithms for relative pose computation, together with a new and efficient estimation framework, BaySAC, allow multiple position estimates for the robot to be computed as it explores. These multiple position estimates are represented in a novel graph-based map. From this map, a subgraph consisting of only the most reliable position estimates is selected and optimised using the TORO optimisation framework (Grisetti et al., 2007b), producing consistent global maps. This combination of techniques is validated by generating accurate global maps from long video sequences in real-time, despite erratic camera motion, moving objects, and complete disorientation.
While for accurate and robust robot positioning a combination of sensors is arguably the best approach (Section 2.4), additional sensors will add to the cost, weight and power requirements of a system, and hence it is worthwhile to investigate how successful navigation with a single camera can be. BoWSLAM demonstrates the strength of cameras alone as positioning sensors, however the robust techniques developed to enable navigation using a single camera are also applicable to systems integrating vision and other sensors, or navigation systems that do not use vision.

1.3 Thesis structure

This thesis describes the development of BoWSLAM. To achieve this, components must be designed or selected to perform several important roles; these components are described in chapters 4 through to 7, then Chapter 8 describes BoWSLAM itself, which combines these components with other novel techniques. Chapters 9 and 10 describe an extension to BoWSLAM (using object recognition to correct the problem of scale drift), and an alternative use of this combination of components (video mosaicing) respectively. More details on each chapter are now given.
Firstly, a wide range of robot designs have been proposed, and many different schemes have been proposed to enable these robots to navigate autonomously. Chapter 2 describes the range of robot platforms available and the sensors with which they are equipped, and Chapter 3 analyses the leading techniques for positioning these robots, including the optimisation algorithms proposed to solve the SLAM problem, and the applications of these algorithms to visual SLAM.
Secondly, in order to position and localise a camera using vision, images must be described in a way that allows them to be compared efficiently. The many different ways images can be described are reviewed in Chapter 4.
A proposed requirement of a Single Camera SLAM system was to enable a robot to localise itself in a map, even following large errors in positioning, or when the robot has become completely lost and disoriented. The leading schemes for active localisation using computer vision are based on the Bag-of-Words (BoW) algorithm, a system for efficiently encoding and comparing sets of descriptors which describe images. In Chapter 5 a BoW implementation is developed which supports real-time indexing of frames, and which can learn how to describe novel environments as they are encountered. This scheme is validated on a large standard dataset (Nistér and Stewénius, 2006), and in an indoor pedestrian navigation experiment.
A second proposed requirement for reliable Single Camera SLAM is that the cameras' relative pose can be estimated reliably between pairs of frames, even in the case of substantial camera motion, dynamic objects and similar-looking features. In Chapter 6, a selection of the latest contemporary algorithms for two-view

relative pose estimation are reviewed, and an appropriate combination of algorithms is selected. Computing relative poses with this combination of algorithms is expensive however, and this cost is dominated by the time needed to remove outliers using the popular RANSAC framework. Therefore, in Chapter 7, BaySAC, a new method for robust model fitting is described. BaySAC uses the prior probabilities of feature point matches to speed-up the identification of inliers in the presence of high rates of outliers, and is shown to outperform previous RANSAC sampling strategies on simulated data, and for relative pose estimation with real data.

Chapter 8 describes BoWSLAM, a scheme which combines BoW localisation and BaySAC robust relative positioning in a novel SLAM framework, in order to allow a mobile robot to navigate dynamic environments where erratic motion, motion blur, and self-similar features make the use of computer vision challenging. Two new uses for the Bag-of-Words image representation allow multiple position hypotheses to be computed for each frame in real-time: this is used firstly, to select the best set of frames to reconstruct positions from; and secondly, to give efficient wide-baseline correspondences between pairs of frames. A novel graph-based representation of these position hypotheses enables the modeling and correction of errors in scale in a dual graph, and the selection of only reliable position estimates in the presence of gross outliers. In this chapter the abilities of BoWSLAM are demonstrated on several datasets, including a 25 minute 2.5km trajectory through a challenging and dynamic outdoor environment without any other sensor input; considerably further than previous single camera SLAM schemes.

One significant limitation of positioning using a single camera is scale drift—the size of the world and speed of the camera can only be estimated up to an unknown scale factor, and as the robot explores, small errors in estimating the relative scale of the features it observes will accumulate, eventually corrupting its position estimate. To enable BoWSLAM to negotiate the most challenging environments, the scale of the world is assumed to follow a particular distribution. This assumption is not true in general however; a robot can travel between different environments where objects occur at a range of different scales. In this circumstance the invalid scale assumption causes distortions in global maps. In Chapter 9 a novel solution to the problem of scale drift based on recognising objects of known scale is described. Our BoW scheme is modified to learn object classes, to recognise object instances, and to use measurements of these objects to correct scale drift. These object observations reduce scale drift by 75% while navigating an indoor environment.

A closely related problem to single camera SLAM is image mosaicing; this is the process of joining overlapping images together to form a larger image, hence enlarging a cameras' field-of-view. One application for image mosaicing is to increase the field-of-view of a camera streaming images back from an aeroplane, to aid an operator by keeping features of interest in view for longer. Chapter 10.1 describes such a mosaicing scheme, which operates in a similar way to BoWSLAM: the BoW scheme enables overlapping images to be selected, BaySAC enables the transformations between pairs of images to be computed efficiently, and an optimised renderer allows a seamless video-mosaic to be displayed in real-time.

The final chapter, Chapter 11, summarises the results from previous chapters, and describes how future developments will further improve these results and enable BoWSLAM to be deployed on autonomous mobile robots.

BoWSLAM incorporates and builds on results from across the computer vision and robotics literature, including, most importantly, the large amount of prior research into SLAM and positioning with a single camera. This literature is analysed in Chapter 3, then additional literature relevant to each of the different components of BoWSLAM is introduced as they are developed as follows: the range of potential platforms and the sensors with which they are equipped are reviewed in the following chapter. Relevant literature relating to image feature detection and description is introduced in Chapter 4. Contemporary techniques for robot localisation, including the Bag-of-Words algorithm, are reviewed in Chapter 5. Algorithms for estimating relative camera poses from image pairs are described in Chapter 6. The RANSAC framework, and many of the variations to it which have been proposed, are analysed in Chapter 7. A summary of those object recognition techniques which are suitable for real-time use on robotic platforms is given in Chapter 9. Finally, a review of contemporary approaches to real-time incremental image mosaicing is given in Chapter 10.1.

Chapters have abstracts, in order to provide a concise overview of their aims and conclusions (although

many have not been published as papers). A glossary of abbreviations and technical terms is provided at the end of the thesis, and mathematical notation used in each chapter is summarised in Tables 5.1, 7.1, 8.1 and 9.1, and introduced in the text as required. Natural logarithms are used unless otherwise stated, and maps typically have arbitrary distance units, due to an unknown global scale, and local distortions. An overview of the Singular Value Decomposition, and the Levenberg-Marquardt algorithm, which are used frequently throughout this thesis for linear and nonlinear least-squares optimisation respectively, is given in Appendix A.

Some terminology used throughout the thesis includes **localisation**, which refers to positioning with respect to a map; **pose**, which refers to the combination of position and orientation, and **gross errors**, which refers to measurement errors many times larger than would ever be observed if measurements were approximately normally distributed. Gross errors are predominately caused by incorrectly matching features between images, or by assuming that moving objects are static.

All execution times given are calculated on a computer with 4GB of RAM and a 3GHz Intel Core 2 Duo processor, running C++ code compiled using gcc.

The source code for BoWSLAM and all of its component parts is available online at `http://www.hilandtom.com/tombotteril`
The same website features videos of BoWSLAM, and links to publications related to this thesis.

1.4 Contributions of this thesis

This section lists the important contributions of this thesis. The main contribution is to design and develop BoWSLAM, a robust single camera SLAM scheme which is capable of positioning a camera for extended periods in previously unknown environments, in real-time. BoWSLAM includes a combination of new algorithms enabling multiple relative pose estimates for each camera position to be computed efficiently. A novel graph-based map of these multiple relative pose estimates enables an outlier-free subset of pose estimates to be selected and optimised.

Several additional contributions are also made: in Chapter 5 a simple scheme for generating effective hierarchical BoW dictionaries dynamically, using an approximate clustering scheme, is proposed. Unlike contemporary BoW schemes which learn dictionaries dynamically, this scheme is proved to achieve the best possible complexity for such a scheme.

In Chapter 4, the localisation accuracy and repeatability of leading contemporary feature detectors is compared. A theoretical limit on the accuracy of these detectors is derived, and several detectors are shown to be close to achieving this limit. This localisation accuracy is shown to be important for improving the performance of feature descriptors, and in Chapter 6, I show that localisation accuracy is important for minimising the time required to compute relative camera poses from correspondences contaminated with many gross outliers.

In Chapter 7, two improved sampling strategies for RANSAC are developed, BaySAC and SimSAC. These new strategies both combine prior estimates of the probabilities of data points being inliers, with information gained by trying to fit models to these data points and failing. Both methods outperform previous sampling strategies, and are shown to both make effective use of low-quality correspondences between sets of similar-looking feature matches, and to reduce the time required to compute sequences of relative camera poses.

In Chapter 9, a novel solution to the problem of scale drift in single camera SLAM is proposed: classes of objects, and their size distributions, are learnt as the robot explores. Subsequent measurements of the size of these objects allow the robot's speed to be estimated accurately.

Finally, in Chapter 10.1, the combination of robust techniques developed to enable BoWSLAM to compute real-time position estimates in visually challenging environments are applied to the related problem of image mosaicing. These techniques, together with a novel motion model used to estimate the probability of wide-baseline correspondences being inliers, given previous camera motion, enable high resolution frames to be registered and rendered into a seamless video mosaic in real-time.

Chapter 2

Sensors for mobile robot navigation

Abstract

This chapter provides an overview of potentially-autonomous mobile robot platforms, and the sensors with which they are equipped. Understanding the characteristics and limitations of these sensors is necessary for the design of SLAM systems. As BoWSLAM uses a single digital camera as a navigation sensor, this sensor is discussed in detail.

2.1 Introduction

A wide range of robot platforms have been proposed or constructed, and these robots have been equipped with many different sensors. Understanding these robots' motion, their sensors, and the environments in which they operate is necessary to design robust SLAM systems, or to adapt existing positioning techniques to different sensors.
This chapter gives an overview of these robots and the sensors with which they are equipped, and is organised as follows. Firstly, the following section describes a range of robot platforms on which SLAM would be useful. Secondly, Section 2.3 describes the sensors with which these robots are equipped, and their characteristics and limitations. BoWSLAM uses only a single camera, so the properties of digital cameras are discussed in detail.

2.2 Mobile robots

Mobile robots come in a huge variety of forms depending on their intended environment and application. While most of these robots do not have the ability to navigate autonomously, improvements to the reliability of SLAM schemes will enable their successors to autonomously operate in *a priori* unknown environments. This section describes some of the platforms where robust positioning and autonomous operation will enable useful applications.
For indoors applications, many ground-based robots have been built, including many 'humanoid' robots which stand upright with a small footprint in order to access the same places as humans; these may be tracked, wheeled or walking. Indoor flying robots have also been built, including quadrotors (Figure 2.1; Hoffman et al., 2007) and 'Micro Air Vehicles' (MAVs). MAVs weigh as little as eight grams (Klaptocz and Nicoud, 2009) and are often inspired by flying insects. Characteristics of these indoor flying robots include their small payloads and complex, often unstable dynamics (Andert and Adolf, 2009). Other biologically-inspired compact mobile robots have designs inspired by snakes and spiders (Figure 2.2); these will potentially enable robots to access environments out of reach of humans.
Robots are are employed in a wide range of outdoor environments, these include many land-based robots, often based on all-terrain wheeled or tracked vehicles, but also including tree-climbing robots for environmental

monitoring (Elfes et al., 2008), and jumping robots, which can jump obstacles in order to traverse rugged terrain (Kovac et al., 2008). At sea, many different autonomous underwater vehicles (AUVs; Johnson-Roberson et al., 2010; Barkby et al., 2009) have been deployed; these are often submarine-like, but which include more exotic designs such as an octopus-based robot (Laschi et al., 2009). Many unmanned aerial vehicles (UAVs; Figure 2.1) have also been flown, including airships (Hygounenc et al., 2004; Elfes et al., 2008) and unmanned planes (Bryson and Sukkarieh, 2007). While most of these robots are still in the proof-of-concept stage, some have operated for longer periods, including the highly successful Mars rovers (Maimone et al., 2007), wheeled remote-control robots which have operated for several years with only occasional directions from earth; and the Nomad robot which autonomously searched Antarctic ice for meteors for 16 hours (Apostolopoulos et al., 2000).

Figure 2.1: UAVs under development at the Geospatial Research Centre, University of Canterbury, for which positioning is useful include this quadrotor (photo by John Stowers), and this remote-controlled fixed-wing UAV designed for the Antarctic, images from which are used in Chapter 10.1 (from Barnsdale, 2010).

When positioning a robot, its environment, dynamics and kinetics can provide many useful constraints on its possible motion. The simplest case of such a constraint applies to wheeled or walking robots operating on a locally planar surface, such as indoors, or on roads for example. These robots can often be assumed to be upright and operating on a plane (Bosse and Zlot, 2008). This combination of 2D position and 1D orientation (heading direction) gives simple three degrees-of-freedom (DOF) motion, whereas for a robot which can move in 3D, and which can roll, pitch and yaw, such as an AUV, the robot's pose has 6 DOF; a 3D position and a 3D orientation. Other useful constraints come from wheeled robots' maximum speed and turning characteristics (for example their minimum turning circle), or from constraints on the accelerations needed to maintain certain orientations (for example a bicycle can only lean sideways when cornering; Scaramuzza et al., 2009). Very often a robot's motion can be modelled, so that its future position can be estimated from its previous motion. "Constant velocity" motion models are often applied, where the velocity at a later time is assumed to be distributed about the current velocity (Davison, 2003). A constant velocity motion model might assume that the robot's acceleration between two time steps is normally distributed with mean zero, a good assumption when momentum dominates other forces. For very erratic motion, constant position motion models are occasionally used (Chekhlov et al., 2006); in this case only the current position (rather than the current velocity) is used to predict the distribution of future positions. For situations where the robot is constrained to a small environment (such as a room) a decaying-velocity model might be appropriate (Klein and Murray, 2007). Constant or zero angular velocity can also be modeled.
While the smallest MAVs may only have the very limited amount of processing power needed to stay airborne (Klaptocz and Nicoud, 2009), most other robots are capable of carrying portable computers such as modern mobile phones or tablet PCs, which are capable of real-time image processing (Wagner et al., 2008). Larger

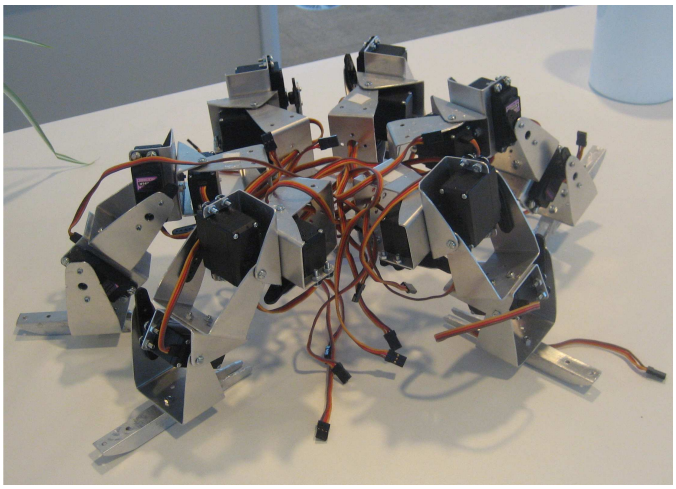


Figure 2.2: This walking robot inspired by a spider is under development by Qing Ou at the University of Canterbury.

robots can power and carry powerful computers such as the blade servers carried by DARPA-Challenge autonomous vehicles (Leonard et al., 2008).

2.3 Robot sensors

To carry out many useful tasks, these robots need to be able to position themselves. Positioning is essential for robots working out of range of human operators, such as AUVs, so that the robot does not get lost forever. In this section we review the range of different sensors with which robots are equipped, the measurements these sensors can make, and how these measurements allow positioning or localisation.
These measurements can be divided into the following three categories. Firstly, IMUs, and the robot's control input and odometry, measure the robot's motion without making any measurements of its environment. These sensors are described in Section 2.3.1. Secondly, global positioning sensors such as GPS provide absolute measurements of a robot's position, these are described in Section 2.3.2. Thirdly, sensors such as cameras and laser scanners make measurements of the robot's environment which may be used for incremental positioning, for mapping, and for localising the robot in its map. These are described in Section 2.3.3.

2.3.1 Measurements of robot motion

Odometry and control inputs

To some extent, most robots can position themselves over short distances by accumulating the expected effects of their control inputs, for example a wheeled robot can estimate how far it will move for a given control input, or can use a wheel encoder to measure how far its wheels have rolled. Control inputs are not enough on their own, however predictable the robot's motion is, as eventually small errors will accumulate and render these position estimates useless. In practice, control inputs and odometry are often unreliable and are prone to gross errors. For example, UAVs will move in the wind or in draughts, and AUVs are subject to movement from waves and currents (Siegwart and Nourbakhsh, 2004). Indoor robots may be picked up and moved, or may fall over, or may experience unobservable motion (for example from riding in a lift; Mei et al., 2009). Walking robots' actual motion can vary massively with the terrain being traversed (Raibert et al., 2008) and wheeled robots often suffer from skidding and wheel-slip (Maimone et al., 2007). Any navigation scheme using control inputs or odometry measurements for navigation must be able to cope with these gross errors.

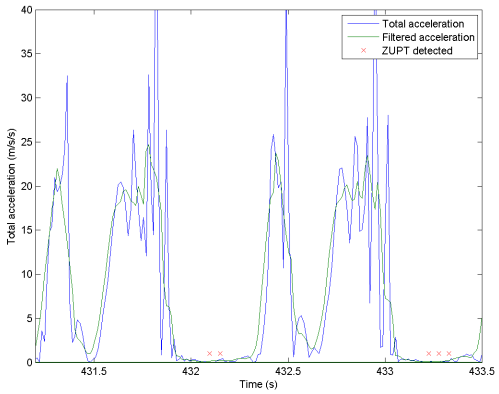


Figure 2.3: The acceleration of a foot-mounted IMU is briefly zero when the foot is on the ground between steps. This event is easily detectable, allowing a zero-velocity update to be used to correct the IMU's drift (from Hide et al., 2009).

Inertial measurement units

Inertial measurement units, or IMUs, measure a robot's acceleration. Acceleration is usually measured in 3D with three accelerometers, and changes in orientation are also measured (often by measuring the difference in accelerations measured by two parallel accelerometers). Measured accelerations are integrated twice to give firstly the robot's velocity, and secondly the robot's position, giving measurements of full 6 DOF motion. As IMUs are passive and make no measurements of their environment, they can be used in any environment, and the errors in positioning can usually be assumed to be free of gross outliers.
A significant problem with positioning from IMU measurements however is that a single small error in a measurement from an accelerometer will introduce errors to all subsequent velocity estimates when integrated; hence many small errors in acceleration integrated twice give errors in position that grow quadratically with time. Even though errors in acceleration may be very small, these errors in pose will still accumulate. The accumulated errors are known as "drift", typical low-cost IMUs can only be used unaided for a few seconds without accumulating substantial drift, and "navigation grade" IMUs costing thousands of dollars will still drift within minutes or hours (Hide, 2003). For this reason IMUs must normally be used in conjunction with other sensors, or odometry input, so that this drift can be corrected.
While IMUs do not measure their environment, the accelerations they measure can be used to infer additional information by using knowledge of the motion of the platform on which they are mounted. One example is a foot-mounted IMU (Foxlin, 2005, Figures 2.3 and 2.4): the acceleration followed by deceleration of a stride is easily detected, and the IMU can be assumed to be stationary between strides, allowing a 'zero-velocity update' to be used to correct accumulated drift. An IMU can even be used to provide location cues, for example by detecting the characteristic motions associated with getting into a car or unlocking a door (Peddemors et al., 2009). As IMUs are sensitive to the direction of gravity, this measurement can be used to correct certain errors in orientation from other sensors. One case when IMU-only navigation is possible is when the assumption that the paths most likely to be visited are those that were visited previously can be used (Robertson et al., 2007).

2.3.2 Absolute position sensors

Ideally a sensor will give an absolute position anywhere in the robot's environment, as this will allow maps to be built which are free of gross errors or large scale distortions. Global Navigation Satellite Systems (GNSS), such as GPS, provide absolute position estimates in many outdoor environments. GPS is unavailable or

unreliable in many environments where robots may work however, including indoor environments, in 'urban canyons', in mountainous or densely forested areas, near the poles, underwater, underground and on other planets. Even when GPS is available, measurements may only be accurate to a few tens of metres (as seen in Figure 8.24). Often mapping and localisation are still useful to robots which use GPS, in order to improve the quality of maps (Barkby et al., 2009), or to obtain more accurate position estimates (Apostolopoulos et al., 2000; Tillett, 1991).
Another absolute position sensor that can provide orientation information is a compass. Compasses (magnetometers) give noisy and biased measurements however (Barkby et al., 2009), and are subject to local magnetic field variations (including those caused by the robot's electronics). This is particularly a problem indoors, where large magnetic field variations arise from electrical appliances and steel objects. This variable field is not unusable as a positioning sensor however; Haverinen and Kemppainen (2009) measure the 3D magnetic field variation around a building, then use the map of variations to localise a person equipped with a magnetometer.

2.3.3 Environmental sensors

Sonar scanners, laser scanners, cameras, and radio receivers are commonly mounted on robots to measure properties of the robot's environment, enabling many mobile robot functions such as local path planning and obstacle avoidance, and the identification of objects with which to interact (for example people or other robots). These sensors can be used for positioning by measuring the change in the positions of features in the world, relative to the robot, when the robot moves. The performance of these sensors varies depending on environmental conditions, and robust methods must normally be used that can cope with the objects that are being sensed moving or changing with time.
Laser scanners are popular sensors for navigation of wheeled robots in planar environments (Apostolopoulos et al., 2000; Bosse and Zlot, 2008); a single 1D scan gives the depth of the first solid object encountered in each direction along a range of angles. As wheeled indoor robots move on a plane, and many indoor features have vertical edges (e.g. walls, doors, table legs) then a pair of consecutive scans will show many of the same points. These scans can be used to build a 2D map of local features to which future scans can be aligned, hence giving the position of the robot. For outdoor applications, robust scan-alignment, which can cope with the robot moving on an uneven surface, can still provide reliable position estimates (Bosse and Roberts, 2007).
By continually moving (spinning or sweeping) a laser scanner, 2D scans can be obtained; these can then be used for 3D mapping (Albrecht et al., 2006; Cole and Newman, 2006). Laser scanners have several problems however, including noisy and unusable results in environments where glass objects are common, and the compromise between choosing a laser that is eye-safe (i.e. is safe to use around humans), but is powerful enough to measure ranges to more distant features. Some laser scanners can also be confused by sunlight (Folkesson and Christensen, 2007), and by lasers on other robots, and are often heavy and expensive (Kwok and Dissanayake, 2003); a widely-used "light" and "low power" SICK laser scanner has typical power requirements of 8W and weighs 1kg (SICK LD-MRS-400001 laser scanner technical data, 2010); so is not a practical sensor for many small robots.
Sonar scanners provide a lower-cost alternative to laser scanners, but typically have lower angular resolution. They are a popular sensor for slow-moving underwater robots, having a longer range underwater than, for example, cameras. Barkby et al. (2009) match sonar scans from an AUV to a coarse, two-metre resolution depth map, allowing both the robot's odometry estimate and the depth map to be to be improved.
Sonar and laser scanners both work by emitting signals and detecting their reflection. For applications where the robot does not want to be easily detectable, or where these signals could interfere with other equipment, a passive approach is desirable. One passive option is to detect electromagnetic 'signals of opportunity' (Fletcher, 2009), such as the radiation emitted by electronic appliances, or to use radio signals intended for other communication applications. Evennou and Marx (2006) use varying Wi-Fi signal strengths to help position a robot indoors. A similar passive positioning approach which can provide absolute position estimates uses 'time difference of arrival' techniques normally used for GPS positioning to compute positions from time-synchronised communications signals, for example DAB digital radio signals (Palmer et al., 2009).

Figure 2.4: One potential use for SLAM is as a pedestrian navigation aid. This vision-aided pedestrian navigation system combines computer vision with measurements from a IMU, and from GPS when available. A Sony Handycam DCR-SR47 is used to collect images, and a low-cost Microstrain 3DM-GX1 IMU is mounted to the pedestrian's foot (from Hide et al., 2009).

The use of such measurements is still restricted to small scale proof-of-concept demonstrations such as these. In addition, electromagnetic radiation sensors are limited to environments where signals of the correct frequency are present, and systems using these sensors must be robust to electromagnetic noise from other sources, and changing signals as equipment is switched on or off.

Other properties of the environment that can be measured by robots include sound and temperature, and these measurements could potentially be used to aid robot positioning. These measurements might be used for example to infer that wheel-slip is more likely when the temperature is below freezing, and that position estimates computed in noisy environments containing dynamic objects such as people and vehicles are less reliable than position estimates computed when everything is stationary and quiet.

One sensor that is considerably more powerful as a localisation sensor is a digital camera. Cameras are passive, relatively low-cost and low-power sensors which capture rich, distinctive information about the robot's environment. There is sufficient distinctive information in a photograph to regularly and reliably localise a camera on a car travelling for hundreds of kilometres of roads (Cummins and Newman, 2008a), and often images contain enough information to identify where in the world a photograph was taken (Hays and Efros, 2008).

Images can also provide information about the 3D structure of the environment being viewed, and hence can be used to reconstruct the path a camera has taken. The location of a feature in an image gives its bearing; by moving the camera and measuring its bearing from a different location the position of the corresponding object in the world may be triangulated. Alternatively, a second camera may be attached to provide stereo vision, allowing the structure of the scene to be reconstructed even when the camera does not move. Stereo cameras also make 3D reconstruction simpler and more reliable, as the location of each feature in one image constrains its location in the other image. As either a single camera, or a stereo camera moves thorough the world, the change in the positions of observed objects relative to the camera gives the camera's trajectory through the world.

Images from a single camera alone cannot measure the scale of the world: for example the images do not indicate whether the camera is moving slowly past nearby objects, or rapidly past far-away objects, and hence the absolute speed of a single camera cannot be determined without additional information. This additional information may be knowledge of the size of particular objects in the environment (Davison, 2003), or input from additional sensors. Without such additional information the scale of the world, or speed of the camera, can only be determined up to an unknown scale factor. In addition, errors in computing the relative scale of objects in the environment as the camera moves through it will accumulate over time, a problem known as 'scale drift'.

Scale drift is not usually a problem when stereo cameras are used, as the known baseline (the distance

Figure 2.5: Contour HD wearable camera. This rugged helmet-mounted camera with a 135 degree FOV weighs 116g, currently costs around US$250, and has a battery/recording life of about 2 hours. Images from this camera are used to test BoWSLAM, as described in Chapter 8.

between the cameras) allows absolute distances to objects viewed to be measured. Stereo cameras are not suitable for all platforms however; they can only provide useful depth information if their baseline length is a sufficient fraction of the distance to points in the world. In addition there are additional cost, space, weight and power requirements in carrying an additional camera.

To use a camera as a navigation sensor there must be sufficient light (which may come from an attached light source), the scenes viewed must contain sufficient visual information (images of completely blank walls, or showing entirely sky, provide limited information), and there must be overlap between pairs of frames. To increase the chances of viewing the more detailed parts of a scene, and to increase the amount of overlap between multiple images of a location, a wide field-of-view (FOV) is often desirable, although at the cost of either decreased resolution or increased image size. The widest FOV images come from omnidirectional or panoramic cameras, which use lenses, mirrors, or a combination of several cameras to view a large area of their surroundings (Bur et al., 2006; Lemaire and Lacroix, 2007). These cameras can add to the cost and processing complexity of a visual navigation sensor however. An alternative way to increase the proportion of consecutive frames that will overlap is to capture images at a high framerate, however this reduces the amount of processing time available per-frame.

Compact machine vision cameras are available which are small enough and have low enough power consumption for most robot platforms, for example the Point Grey Chameleon camera (Point Grey Chamelion Datasheet, 2010), weighing 37g and requiring less than 2W of power. Images from the low-cost wearable camera shown in Figure 2.5 are used in Chapter 8 to test BoWSLAM.

Cameras are also versatile sensors, which can be used in a wide range of situations including many indoor (Bur et al., 2006), outdoor (Nistér et al., 2006) and underwater (Johnson-Roberson et al., 2010) environments, however challenges exist where effects such as dust or rain degrade images (Peynot et al., 2009).

The potential of computer vision to be used for navigation is illustrated by the way animals have evolved to use vision. Many animals use vision for positioning and navigation (in conjunction with many other senses); for humans this is stereo colour vision (Ramachandran, 2002c), however many animals, such as horses (Equine vision; Wikipedia, 2010), and many fish, have eyes positioned on opposite sides of the head and their field-of-view has little or no overlap, leaving them with essentially panoramic monocular vision. Many animals have poor colour vision, and humans who are colour-blind, or blind in one eye can still negotiate the world with similar ability to fully sighted people (Ramachandran, 2002a). In addition to stereo vision, the human brain uses optical flow, and moving occlusion boundaries, to infer the 3D structure of the world from the motion of the head (Ramachandran, 2002b).

Some animals have evolved to use colour bands outside the range visible to humans (bees can sense ultraviolet light, and some snakes can use infra-red radiation to sense prey). These bands could potentially be useful

for navigation, especially in the dark (Fehlman and Hinders, 2010, use a robot-mounted infra-red camera to classify obstacles at night), however many human environments are illuminated with artificial lights designed to match the human visible range, and many parts of our environment (signs, handles, road markings) are designed to be visible in this range.
Time-of-flight cameras are a recent development allowing the depth of each pixel to be measured. A source of light or infra-red radiation is used to illuminate the scene, then the phase of the reflected light is used to infer the depth. Unlike stereo cameras, depths can be measured across featureless areas of images, providing a complete depth map without interpolation. Time-of-flight cameras do not always work in sunlight however, and exhibit large errors in depths with "complex" characteristics, but even so have successfully been used as a robot positioning sensor (May et al., 2009).

2.4 Summary of robots and sensors

In summary, there are a vast range of different potential robotic platforms, with a range of different dynamics and kinetics, and which operate in a wide range of environments. For some of these platforms, for example wheeled robots, information about the robot's control inputs and dynamics will provide useful information about the robot's motion, however even in this case these measurements are liable to contain gross errors (from wheel-slip or unobservable motion). Many platforms, such as robots which fly or jump, or wearable computers (Figure 2.5), can move unpredictably or erratically; this limits the assumptions which can be made about their motion.
Of the many different sensors attached to these robots, cameras are possibly the most versatile, as they can be used for both incremental positioning and for large scale localisation, and hence we believe camera is the only low-cost sensor that has the potential to be used on its own for long-term robot positioning in a wide range of environments.
In practice a combination of sensors, with complementary strengths, would be a more reliable approach to robot positioning. Many positioning algorithms (described in the following chapter) are designed for multiple sensor inputs, and when multiple measurements are available, outliers are considerably easier to identify, as these will normally be incompatible with other measurements. However, additional sensors add to the cost, weight, and processing and power requirements of the system, and for a few applications (lightweight MAVs, jumping robots with high accelerations; Kovac et al., 2008), a camera may be the only feasible option. In this thesis I develop BoWSLAM, a system allowing positioning using a single camera alone, demonstrating the strength of a camera as a navigation sensor, however techniques developed to enable navigation using a single camera are also applicable to systems integrating cameras and other sensor inputs, or to robust navigation systems that do not use cameras.

Chapter 3

Positioning and SLAM using a single camera

Abstract

This chapter describes the many solutions which have been proposed for the SLAM problem, and identifies those solutions which are appropriate for large scale navigation using a single camera. Contemporary SLAM algorithms based on Kalman filters and particle filters are described and analysed, then more recent algorithms based on the optimisation of pose graphs are reviewed. All of these algorithms have been applied to Visual SLAM: these schemes are described and evaluated. The most successful scalable stereo camera SLAM schemes all use some form of incremental bundle adjustment, or multi-view pose refinement, on a subset of recent frames, then compute globally accurate maps by optimising the graph of transformations between local coordinate frames. This approach is also likely to be successful for large scale single camera SLAM.

3.1 Introduction

The ability for a mobile robot to position itself in a previously unknown environment for extended periods is considered essential for it to operate autonomously (a 'core competency' according to Thrun et al., 2005), however position estimates obtained by accumulating many small relative position measurements (dead-reckoning) from sensors such as vision, wheel odometry, or laser scanners will accumulate errors (drift) over time, eventually rendering them useless. For long term positioning using these sensors, the environment being explored must also be mapped, so that when the robot re-visits mapped areas it can correct errors in both its position estimate and in the map. This combination of positioning and mapping is known as Simultaneous Localisation and Mapping, or SLAM (Dissanayake et al., 2001). The map produced is not only useful for positioning, but may also be useful for path planning, accomplishing objectives, and mapping may even be the robot's ultimate task.

SLAM is a challenging problem to solve however, as errors in a robot's position will introduce errors to the map it is making, and similarly, errors in a robot's map will lead to errors in its position estimate (Smith et al., 1990). Modeling the correlation between these errors is key to a successful SLAM algorithm, and numerous solutions have been proposed (several of the most popular are reviewed by Bailey and Durrant-Whyte, 2006; Durrant-Whyte and Bailey, 2006). This chapter presents an analysis of the SLAM problem, and a review of different approaches to solving it, and is organised as follows: the following section analyses the SLAM problem and describes some leading contemporary systems. In Section 3.3, a selection of the most successful schemes for visual SLAM and visual odometry are described and analysed. Finally, Section 3.4 discusses these proposed solutions and identifies the techniques which are most appropriate for large scale robust single camera SLAM.

3.2 The SLAM problem and contemporary solutions

As a mobile robot moves through the world its sensors collect information about its environment ('measurements'). In addition, control inputs or odometry may also be available, together with knowledge of the robot's dynamics, for example the maximum speed and minimum turning circle for a wheeled robot. Prior knowledge of the robot's environment may also be useful, for example knowledge that the robot is exploring a building with horizontal floors and vertical walls and furniture. In addition, absolute position estimates may be available, for example from GPS.
These measurements, odometry inputs, and absolute position estimates all have associated uncertainties, which will introduce uncertainty into any map of the environment that is constructed from them. The goal of SLAM is to build a map from these measurements that is good enough to be useful despite the errors, and preferably errors in the map should be reduced as more measurements become available. The map generated will be a stochastic map (Smith et al., 1990), where each position has an associated uncertainty.
Most existing SLAM algorithms assume that measurement and odometry errors are approximately normally distributed, and hence that gross errors either do not occur, or can be filtered out before being used. These algorithms aim to find a map consisting of the positions of landmarks in the world, and the position, or pose history, of the robot with respect to these landmarks (Figure 3.1). Ideally the map estimated would be a maximum likelihood estimate (MLE) of the landmark positions, given the measurements. This estimation is challenging however; finding the exact ML solution involves a nonlinear optimisation, due to the nonlinear relationship between errors in the robot's orientation. Typical gradient-descent algorithms which converge to the exact solution, such as Levenberg Marquardt (Appendix A.2), have complexity cubic in the size of the mapped area, and are too expensive to use to estimate large maps (Olson et al., 2006). Instead, SLAM algorithms aim to compute approximate solutions, which can be updated incrementally as the robot explores.

In addition to computational complexity, two particular challenges for SLAM are *data association* and *loop closure*. Data association is the matching of measurements of a landmark to subsequent measurements of the same landmark: as landmarks may be matched incorrectly, measurements of a landmark may include gross outliers. Most SLAM algorithms assume Gaussian error distributions, with no gross outliers, and implementations go to considerable lengths to detect and remove outlier measurements. A second challenge for SLAM is loop closure (discussed in detail in Chapter 5); this involves detecting when a previously-visited location is re-visited. Detecting loop closure events allows accumulated errors in position and map estimates to be corrected, hence enabling accurate long-term positioning.
There are two basic formulations to the SLAM problem. Firstly, the "Classic SLAM" problem, where an MLE for the map and current robot pose are estimated. This problem is often solved using an Extended Kalman Filter (EKF; reviewed in Section 3.2.2). The second formulation is based on the observation that estimated positions of landmarks are independent, given the robot's trajectory, and hence only the most likely robot trajectory given the measurements must be measured. Estimated landmark positions are then given relative to this trajectory. This second formulation is used by the particle-filter based FastSLAM algorithm (reviewed in Section 3.2.3), and pose graph optimisation-based approaches (reviewed in Section 3.2.5).
The following sections describe the variety of optimisation algorithms which have been used to solve the SLAM problem, and their strengths and limitations. The algorithms, their properties, and some of the more successful applications are described first. The adaptations of these algorithms to vision-only SLAM are reviewed in Section 3.3.

3.2.1 Occupancy grids

One of the earliest models of a robot's environment involves dividing the world into a regular grid of cells (Moravec, 1979). As a robot explores, the contents of the cell it occupies, and the contents of surrounding cells are estimated. Usually the cells are categorised as containing obstacles or not containing obstacles, or the probability that the cell contains an obstacle is estimated. A major limitation of this approach however is the difficulty of correcting accumulated errors when subsequent visits are made to a location (Thrun et al., 2005).

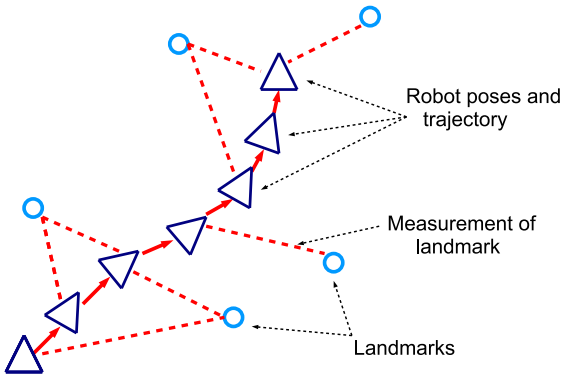


Figure 3.1: A robot moves through a previously unknown environment while observing landmarks (marked with circles). The sequence of robot poses is marked with triangles. Constraints between robot poses (from odometry), and between robot poses and landmark positions (from measurements) are shown by arrows and dashed lines respectively. SLAM consists of an optimisation to estimate the positions of landmarks, and the robot's position or trajectory, which is most likely given these constraints. Optimising this network by (for example) Levenberg-Marquardt is costly; many of the more-efficient, and often incremental solutions which have been proposed are described in this chapter.

One occupancy grid-based SLAM system is RatSLAM (Milford et al., 2004), where a 3D array of cells represent possible robot positions in 2D space, together with the robot's orientation. RatSLAM is inspired in part by the way a rat's brain is believed to model its environment, with individual cells corresponding to different locations and poses. This representation allows a robot equipped with multiple sensors including a camera and laser scanner, to be positioned in a small artificially-constructed environment, however the map generated is not consistent, with single locations mapped multiple times in different cells (Milford et al., 2006).
Robertson et al. (2007) describe a occupancy grid-based SLAM scheme for pedestrian navigation using an IMU. In order to ensure a consistent map, multiple maps are maintained, with new maps generated when the pose of the pedestrian could fall into one of multiple cells. A good global map estimate is given by the most likely of these maps, while the least likely maps are discarded. This approach has high memory requirements and complexity in the size of the mapped environment however, as the number of potential maps can double every time there is uncertainty in which of two cells a robot occupies, As a result, real-time operation is limited to very small environments, such as a single room.
Despite these problems with occupancy grids, they do have the potential to be used locally as part of larger-scale SLAM schemes (e.g. Thrun and Bucken, 1996), and are also useful for local path-planning and obstacle-avoidance (Andert and Adolf, 2009). A grid-based approach is also suitable when a globally accurate prior map is available, and SLAM is being used for local refinements, and to fill-in unmapped regions (Barkby et al., 2009), as global errors should not accumulate in this situation, however in this scheme the resolution of the grid limits the accuracy to which the environment can be mapped.

3.2.2 Extended Kalman-filter SLAM

One of the most popular contemporary solutions to the SLAM problem is to use an EKF (Dissanayake et al., 2000) to estimate the pose of the robot together with the positions of landmarks in a map (Figure 3.2(a)). An EKF combines multiple measurements with a model of the robot's dynamics, to incrementally update the robot's estimate of both its position and map (Figure 3.3). Measurement errors are assumed to be Gaussian, and the robot's motion, and response to control inputs, are assumed to be locally linear with Gaussian

measurement noise; if these measurements and motion were truly linear then the pose and map estimate obtained would be an MLE, although errors in orientation will always introduce nonlinear dependencies between measurements.

The EKF maintains a state-vector, which includes its best estimate of the current robot pose and motion, and (usually) the estimated positions of landmarks which have been observed. A matrix of covariances between the estimates in this vector is also maintained. These encode the correlation between the current pose and landmark estimates. The estimated variance of landmark position estimates falls monotonically as more measurements are made of landmarks, and Dissanayake et al. (2001) proved that landmark position estimates also move monotonically towards their true position as more observations are made (assuming the linearisation assumption holds, and that data association is always successful). This simple model of uncertainty, and the ease with which multiple sensor measurements and motion models can be incorporated, make EKFs a popular choice for SLAM, however the drawback is that EKF updates have complexity quadratic in the number of landmarks mapped, which soon limits the size of environments which can be mapped in real-time.

Dissanayake et al. (2001) demonstrate EKF-SLAM with a car equipped with an accurate millimetre-wave radar. The radar measures the range and bearing of nine radar reflectors and a few natural landmarks. As the car drives in loops, these landmarks, and the car's position and orientation, are estimated in an EKF. The estimated errors in these landmark positions fall as they are observed multiple times, and the positions converge towards their true position. A similar scheme by Nieto et al. (2006) positions a car equipped with odometry and a laser scanner as it drives loops around a carpark; an innovation introduced in this scheme is that point-landmarks are not necessary, instead the positions of scans form landmarks, with additional measurements of these landmarks calculated by aligning later overlapping scans.

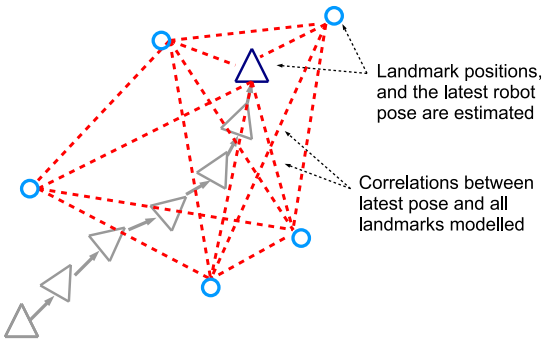
In practice, some sensors have significant latency, and simultaneous measurements of a robot's environment from different sensors arrive at different times. In this case 'delayed state' Kalman filters can be used (Bailey and Durrant-Whyte, 2006). Delayed state Kalman filters also allow uncertain measurements to be kept out of the Kalman filter the first few times they are observed. This allows more measurements to be made, ensuring that observations (such as landmark positions reconstructed from multiple bearing measurements) have approximately Gaussian uncertainty by the time they are incorporated into the Kalman filter, and also allows potential outliers (for example moving landmarks) to be discarded before they can contaminate the map. A delayed state Kalman filter is used to incorporate measurements from a laser scanner making sweeping movements by Cole and Newman (2006), as measurements are only available once the laser scanner has completed its sweep. A detailed and accurate map from a single loop around a building is produced.

Delayed-state EKFs are also useful for incorporating measurements from computer vision. Bryson and Sukkarieh (2007) combine reliable and approximately-Gaussian measurements from an IMU and GPS, with the occasional landmark observation from a camera, in order to position a UAV. Each of the landmarks can be reconstructed more accurately as more observations are made over a sequence of frames. The position of each landmark is incorporated into the EKF only when sufficient measurements have been made to ensure that the errors in its position are approximately Gaussian; subsequent observations can be used to further refine its position.

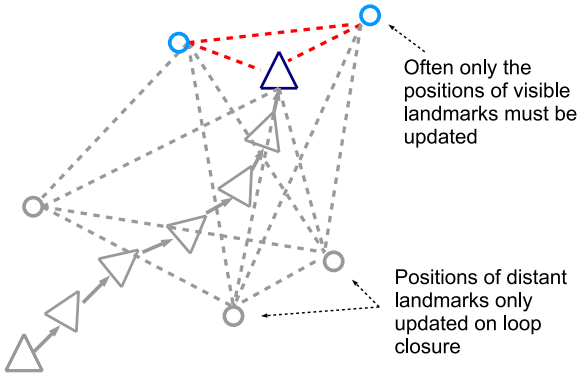
A closely-related estimation algorithm, which is often computationally less expensive, is the Sparse Extended Information Filter (SEIF; Thrun et al., 2004). A SEIF maintains the information matrix (the inverse of the covariance matrix) rather than the covariance matrix, and this can be maintained in constant time when not re-visiting environments. Complexity is still quadratic when re-visiting regions however.

An alternative optimisation to speed up EKF-SLAM is the Compressed EKF (Guivant, 2002, Figure 3.2(b)). This limits the complexity of EKF updates by only updating the positions of landmarks which are spatially nearby on each time step, rather than the positions of all landmarks. Updates to the positions of remaining landmarks are stored and applied later. This later update, which is essential for incorporating loop closure information, has quadratic complexity however. Guivant (2002) demonstrates Compressed EKF SLAM by accurately positioning a vehicle from data collected when it was travelling on a 3km looping path amongst trees, however the cost of full updates are sufficient that mapping is not real-time (Nieto et al., 2003).

In summary, EKF-SLAM works well on a reasonably small scale, however it has two major limitations when



(a) Classic EKF-SLAM updates all landmark positions and current pose.



(b) Various optimisations are possible where only the positions of nearby landmarks are updated most of the time.

Figure 3.2: (a) The classic EKF-SLAM approach refines the current robot pose, and landmark positions simultaneously. As consecutive poses are highly correlated, eliminating previous poses leaves correlations between all landmark positions and the current pose (marked with dashed lines). These correlations are represented by the EKF's covariance matrix. (b) In practice, most updates affect only the positions of nearby landmarks. For this situation many more efficient SLAM algorithms have been proposed (Guivant, 2002), which simplify EKF-SLAM by updating only the poses of nearby landmarks, however on loop closure many more landmark positions must be updated.

At each time step:
Predict robot position and position uncertainty using motion model.
For each measurement:
Update estimated map, and estimated position relative to map.

Figure 3.3: Classic incremental solution to the SLAM problem, as solved by EKF-SLAM.

applied on larger scales. Firstly, any gross error in the input data, for example incorrectly matching features, can corrupt the map and cause positioning to fail (Thrun, 2002; Chekhlov et al., 2006; Bailey and Durrant-Whyte, 2006; Folkesson and Christensen, 2007), and secondly, that the complexity of updates is at best quadratic in the number of landmarks observed, limiting the size of environments which can be explored. The first problem can be partly addressed by additional filtering of input data, for example using a delayed state EKF, however this will not always be sufficient, particularly when measuring landmarks which are liable to move. The computational costs can be reduced by various optimisations, for example by reducing the number of landmark positions which are estimated by combining landmarks (for example into planar features, Martnez-Carranza and Calway, 2009), although costs will still grow quadratically. An additional more minor problem in applying EKF-SLAM to large environments is that artefacts caused by linearising errors in orientation can distort global maps (Olson et al., 2006; Folkesson and Christensen, 2007). For these reasons, EKF-SLAM is best used on a small scale in circumstances where measurements are approximately Gaussian (for example when using an IMU), or where any outliers can be reliably detected. For larger-scale navigation the methods described in Section 3.2.5 can be used to fuse many small maps into a larger map.

3.2.3 Particle-filter SLAM

An alternative optimisation algorithm suitable for SLAM is a particle filter. Particle filters have long been used for the problem of robot localisation in a known map ('Monte-Carlo localization'; Dellaert et al., 1999). A set of particles is generated, each representing a possible robot position. Measurements are made of local landmarks, then each particle is scored according to its compatibility with these measurements (for example the likelihood of the measurements if the particle is correct), then the top scoring particle(s) are chosen as the best estimate of the robot's position. A new set of particles is generated by applying the robot motion model to these particles, then sampling from the weighted distribution of possible robot positions. Unlike localisation with Kalman filters, multimodal distributions can be represented, for example when a robot may have entered one of two neighbouring rooms.
A simple application of a particle filter to the SLAM problem would be to use a set of particles, each representing a possible state vector (or occupancy grid). This is computationally infeasible however, due to the number of particles which would be required to represent the state-space (Durrant-Whyte and Bailey, 2006). Instead, Montemerlo et al. (2002) propose FastSLAM (Figure 3.4), a particle filter-framework for SLAM based on the observation that given the robot pose, landmarks observations are independent. In FastSLAM, each of M particles represent a particular trajectory, together with the positions of all of the K landmarks observed relative to this trajectory. The position of each of these landmarks is represented by a 2D (or 3D) Kalman filter, each of these can be updated cheaply every time the landmark is observed. On each iteration, the motion model is applied to each particle, and each particle is scored based on its likelihood given the observations, then particles are replaced with new particles distributed about the most likely particles ('resampling'). Each particle is resampled efficiently by updating only the position of the robot relative to observed landmarks; with landmarks stored in a tree this has complexity $O(\log K)$, giving each iteration complexity $O(M \log K)$. Remarkably, few particles are needed in many situations, in particularly when measurement error is low and data association reliable. Like EKF-SLAM, FastSLAM can be proved to converge to the correct map (even with a single particle) if data associations are known (Montemerlo et al., 2003).
Two successful implementations of FastSLAM are by Nieto et al. (2003) and Stachniss et al. (2005). Nieto et al. accurately position a laser scanner-equipped car, driving for over 3km amongst trees (which provide

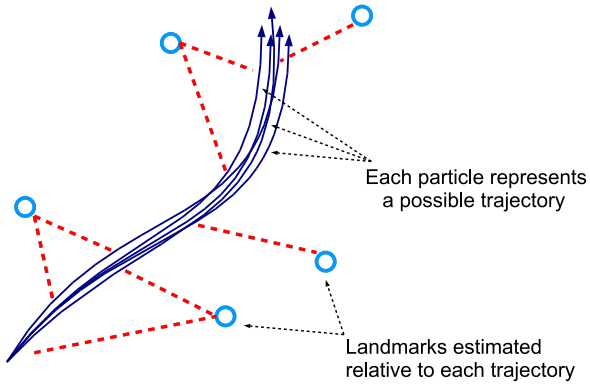


Figure 3.4: In FastSLAM each particle represents a possible robot trajectory. The set of particles represents the distribution of possible trajectories which could have lead to these observations. Landmark positions are estimated relative to each trajectory. When measurement errors are low, accurate long-term positioning is possible (Nieto et al., 2003), however when errors are larger (for example when using computer vision; Eade and Drummond, 2006) ensuring that the particles reflect a reasonable distribution of trajectories is challenging.

point-landmarks in 2D). The map generated closely matches ground-truth, and is generated considerably faster than real-time. FastSLAM is also modified so that when a match between trees is uncertain, both can be represented with different particles, with the incorrect association being removed later when more measurements have been made. Stachniss et al. use FastSLAM to accurately position a laser-scanner-equipped robot indoors, but find that large numbers of particles are needed to avoid particle-depletion (particle-depletion occurs when too few particles are used to represent the distribution of possible trajectories). They propose a scheme to actively instruct the robot to close many small loops to solve this problem, however this indicates that FastSLAM would struggle to maintain a good distribution of trajectories in a larger-scale environment containing larger loops, or with less accurate sensors.

While FastSLAM uses many 2D or 3D Kalman filters to estimate landmark pose relative to one of the trajectories, an alternative is to use many particle filters, one for each landmark viewed from each trajectory, instead (Yuen and MacDonald, 2004). Yang et al. (2008) compare this system to FastSLAM in a small artificial setup, and it is shown to be slightly less accurate, however its total computational cost is lower, and the particle filters used to model each landmark's position could potentially be used to model ambiguous data associations.

In summary, the leading particle-filter-based SLAM framework, FastSLAM, performs well at mapping environments where accurate sensor measurements are available, and can resolve uncertain data associations. The effects of a gross errors in either odometry or data-association could contaminate all particles however. Although FastSLAM has lower complexity than EKF-SLAM, computational costs will still grow as the area explored increases.

3.2.4 Submaps

One solution to allow these methods to scale to larger environments is to use one of these techniques locally, then to join the local maps together into a larger map. This section describes some of these submap techniques, and their successful applications. Many of these techniques formulate the global SLAM problem as a network of transformations between submaps, then optimise this network to compute the relative positions of all submaps. More applications of submaps are described when visual SLAM is discussed later in this chapter.

A popular and scalable graph-based formulation is given by the Atlas framework (Bosse et al., 2004). In this hybrid approach, each node represents a small group of landmarks in their own local coordinate frame (a 'submap') where a conventional SLAM algorithm is used, and each edge represents a transformation between these coordinate frames. Accurate positions of a node relative to a root node can be reconstructed efficiently by finding the 'shortest' path through the graph, where edges are weighted by a measure of their transformation's accuracy.
One of the most successful SLAM schemes to-date (Bosse and Zlot, 2008) maps a trajectory of over 17km through a dynamic urban environment, using only the circle of measurements from a 360 degree laser scanner. In each submap, a sequence of recent poses is estimated by an EKF, using odometry measurements from aligning each scan with recent scans. These local frames are connected via the ATLAS framework. Each scan is encoded as an orientation-histogram, as described by (Bosse and Roberts, 2007). When loop closure is likely, this orientation-histogram representation of the scan data allows the map to be searched for a location matching the current scan. Sufficient loop closure events are detected for a global accurate map to be generated, however in some regions loop closure events are missed due to the low distinctiveness of laser scans in some environments. Dynamic objects filling over 50% of the field-of-view would cause the method to fail, although the global laser-scan loop closure detection scheme proposed by the same authors would solve this problem (Bosse and Roberts, 2007). A locally-accurate map is built in real-time; this is optimised to generate a globally-accurate map offline when required.
A similar approach, where a global map is always maintained, is described by Estrada et al. (2005). On a small scale a conventional SLAM algorithm (EKF-SLAM) is used. When the number of landmarks grows beyond some limit a new map is started with a new coordinate frame, and the transformation from the previous map is taken to be the position of the robot in the previous map, with associated covariance. Importantly, mapping starts afresh, without using landmarks from the previous map, every time a new submap is started. This ensures that relative positions within maps are independent of relative positions with earlier maps. When loops are closed, the relative positions of submaps are refined, subject to the constraint that transformations compose to the identity around cycles. This constrained least-squares optimisation is solved via a potentially costly quadratic programming algorithm, however in practice this is not too expensive. A building is mapped by a wheeled robot with a laser scanner travelling on a 735m trajectory, including three large loops. An accurate global map, free of significant linearisation errors is generated; at worst the update when a loop is closed takes around one second. False loop closure events are detected from their likelihood given the estimated relative poses of the ends of the loop.

3.2.5 Pose graph relaxation

Several recent SLAM schemes avoid some of the difficulties in scaling to large environments with entirely graph-based representations of the map. The SLAM problem consists of constraints (measurements) between robot poses and landmarks. These constraints are represented by the edges in a graph. A global map can then be estimated by optimising this graph to find a trajectory and landmark positions which minimise the total error in constraints (Thrun and Montemerlo, 2006).
In GraphSLAM (Thrun and Montemerlo, 2006) a graph is built where nodes represent robot and landmark poses, and edges represent odometry measurements and landmark observations. Each edge corresponds to a constraint on the relative pose/position of two nodes. The landmarks are now eliminated from this graph, introducing relative pose constraints between the robot positions from which they were observed (Figure 3.5). The GraphSLAM framework naturally incorporates loop closure: when a loop is closed, landmarks observed on the two visits introduce constraints between the corresponding poses. The result of this variable elimination is a graph of robot poses with many relative pose constraints between pairs of poses. Edges are weighted by their likelihood, and the constraints they represent are linearised. The graph can now be optimised to find the set of poses maximising the likelihood of the observations. This linearisation/optimisation is repeated several times to reduce errors from linearisation; an initial solution close to the ML solution aids convergence, but even so, the optimisation proposed is costly, with map generation being a slow offline process.
A closely related approach to SLAM is to construct the pose graph directly: when landmarks are observed which were observed from previous frames, relative pose constraints between these poses are introduced

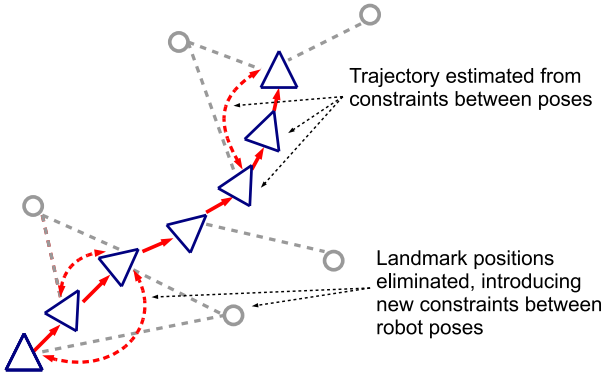


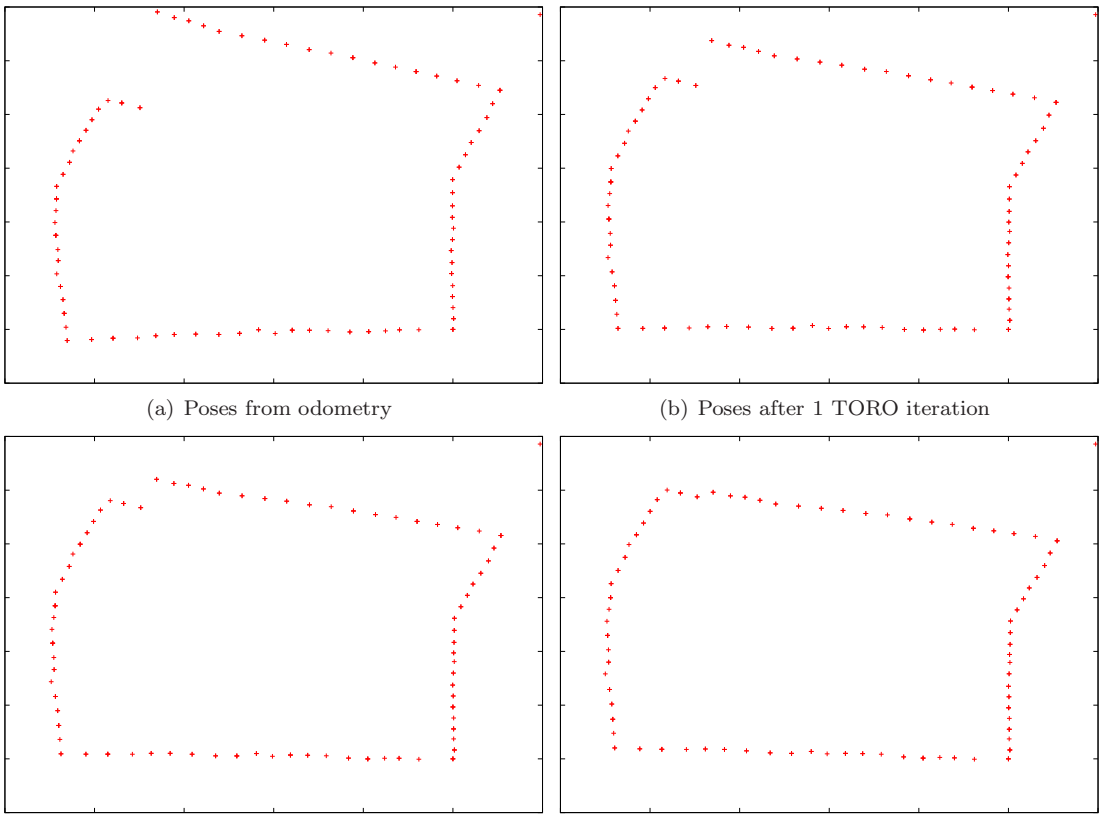
Figure 3.5: GraphSLAM (Thrun and Montemerlo, 2006), simplifies the full global optimisation over landmarks and poses by eliminating the landmark observations; introducing constraints between poses in the process. An offline optimisation over poses then generates the ML global map; alternatively a more efficient pose graph optimisation could be used.

(Figure 3.6). As with GraphSLAM, when a loop is closed the observations of previously-observed landmarks introduce constraints between the latest pose and poses from the previous visit. The global map is then estimated by finding the robot poses minimising the total residual in relative positions. Solving this problem via global methods such as Levenberg-Marquardt nonlinear optimisation is not computationally feasible, and linearising the problem and solving via a Kalman filter can lead to significant artefacts from linearisation (Olson et al., 2006), however several efficient approaches have been proposed.

The most efficient algorithms to perform the global optimisation needed to generate world maps from these graph-based representations iteratively update positions to reduce local errors, either via a relaxation procedure (Frese et al., 2005), or by stochastic gradient descent (SGD; Olson et al., 2006).

To optimise these maps by relaxation, the constraints are linearised, and a quadratic error function on the robot pose history is set up, which could be optimised using a linear least-squares approach (with complexity quadratic in the number of poses at best). As the previous solution is known however, a relaxation procedure can be used. Each relaxation procedure involves iterating over all robot pose estimates and minimising the error from each estimate in turn; in a sparse map this has linear complexity in the number of poses. This is iterated until convergence, which often requires very few iterations. When loops are closed however, the number of iterations required is also linear in the number of poses, giving quadratic complexity. Frese et al. (2005) propose to speed this up using a method based on multi-level relaxation, a technique often used to solve partial differential equations. A sequence of coarser pose graphs are constructed by subsampling from the original graph, for example every second pose, every fourth pose, etc. The relaxation procedure is applied to the sequence of coarser levels in order of increasing coarseness, an exact solution is computed at the coarsest level (with low cost as this level contains few poses), then the relaxation is applied at each level from coarse to fine. This ensures the number of iterations is low even when closing loops; Frese et al. (2005) argue that in real-world environments, where looping is limited, the number of iterations needed is bounded by a constant, giving the algorithm linear complexity in the number of poses.

Olson et al. (2006) optimise pose graphs using a variation on SGD; like multi-level relaxation this technique was originally used for solving partial differential equations. Each SGD iteration finds an increasingly accurate pose graph relative to the nonlinear relative pose constraints. This method starts from an approximate initial solution, selects one constraint (relative pose estimate), calculates the adjustment to the robot poses needed to reduce the error in this constraint, then applies this adjustment to each robot pose. Instead of choosing constraints at random, as in the original SGD algorithm, each is considered in turn. This, together with the structure of the problem (updates usually only affect temporally close/subsequent positions) means



(a) Poses from odometry

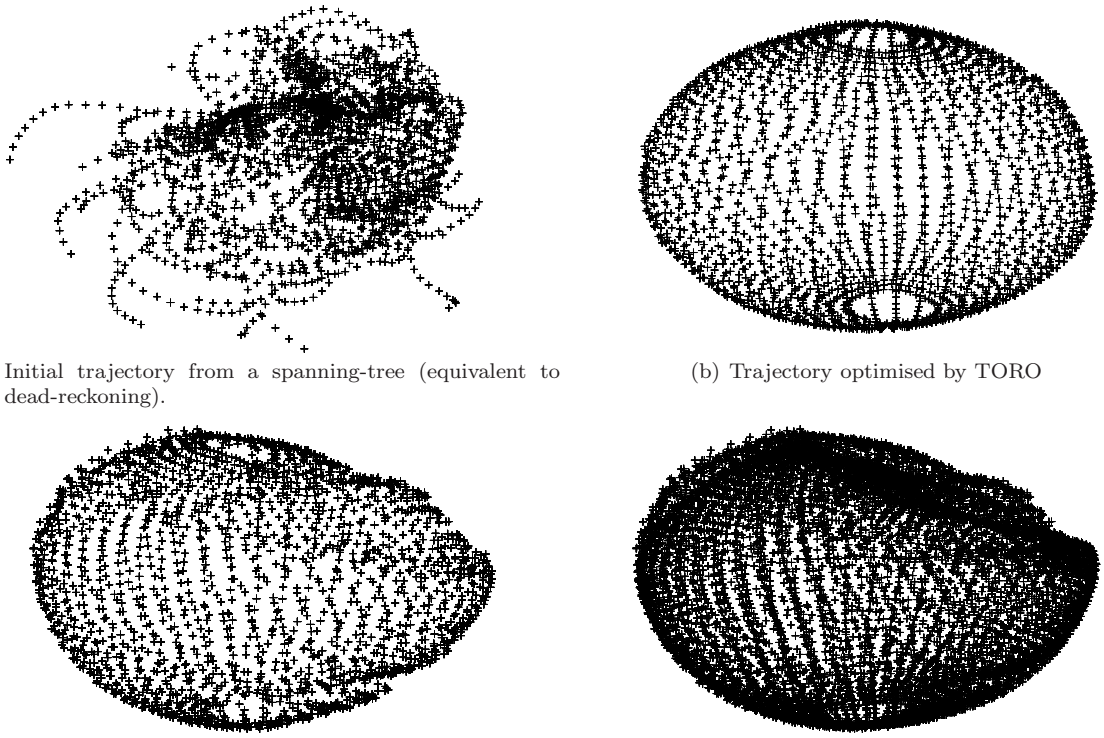
(b) Poses after 1 TORO iteration

(c) Poses after 2 TORO iterations

(d) Poses after 10 TORO iterations. Gap is closed.

Figure 3.6: Estimated positions of a robot as it travels around a rectangular path. Errors accumulate, so the robot's odometry does not show it returning to exactly the same place, however the loop closure event is detected and an edge is added to the pose graph. Each iteration of the pose graph optimiser (TORO) reduces the size of the gap in the top-left corner.

that updates can be applied to a tree of stored updates, so that sequences of successive nodes can be updated efficiently later without having to iterate over them each time. This update is repeated for each constraint on each iteration. Updates are increasingly damped on each successive iteration, and iterations continue until the total residual is low. The effect of each iteration is to partly close gaps in loops, distributing the error around the loop. The optimisation converges rapidly to a globally accurate map, and this map is free of the obvious linearisation artefacts seen when optimised using an EKF, or a simple relaxation procedure.
The TORO framework (Grisetti et al., 2007b) improves on the method by Olson et al. (2006) by building an update tree based on spatial rather than temporal proximity of nodes. This reduces the number of nodes that must be updated in most environments, particularly where a robot repeatedly visits the same area, or where the density of constraints is higher. Accurate maps are generated from initial maps containing large accumulated errors, as shown in Figures 3.7(a) and 3.7(b).
These approaches work in 2D, however they do not easily generalise to 3D, as 3D rotations are not commutative. Instead, for the 3D version of TORO (Grisetti et al., 2007a), the translational and rotational components of motion are separated and updated alternately. Errors in rotation are distributed over sequences of relative poses by spherical linear interpolation. This separation means covariances between the orientation and translation of each relative position estimate are ignored, however this approximation does not appear to degrade optimised maps.
A successful application of graph-based SLAM is described by Folkesson and Christensen (2007). This scheme



(a) Initial trajectory from a spanning-tree (equivalent to dead-reckoning).

(b) Trajectory optimised by TORO

(c) Trajectory optimised by TORO when one erroneous relative pose measurement is introduced

(d) Trajectory optimised by TORO when one incorrect loop closure is introduced, with certainty much lower than other constraints.

Figure 3.7: The TORO framework Grisetti et al. (2007a) transforms a graph of relative poses into an optimised map. This data simulates a robot moving on a sphere, with regular loop closure events. TORO, like EKF-SLAM, finds an approximation to the most likely map, assuming errors are Gaussian. This assumption makes these approaches sensitive to outliers, as illustrated by (c) and (d) where one outlier is introduced in each case. 8650 constraints between 2200 vertices are optimised in about 10 seconds. This raw data was from Stachniss (2008) and was used in the paper by Grisetti et al. (2007a).

uses a map of robot poses and landmarks based on GraphSLAM, which is generated from laser scanner and IMU measurements from a wheeled robot. A consistent map of a university campus is produced from a track of about 1km in length. In addition to GraphSLAM-like landmark elimination, sequences of pose estimates are merged into local submaps, speeding up later optimisation. Regularly applying multi-level relaxation generates the global map.

3.2.6 Summary of SLAM algorithms

In summary, many SLAM systems have been proposed which work well when either the environment is small, or measurement errors are low and free of gross errors. These methods have two major limitations however: firstly they usually assume measurement errors are Gaussian, however this is not the case with computer vision where gross errors often occur; and secondly, many of these methods are computationally expensive, and have high complexity in the size of the mapped areas, limiting their large scale application.
The popular EKF-SLAM approach can generate accurate and consistent maps on a small scale, when measurement errors are relatively low and approximately Gaussian. For larger-scale navigation, many small maps generated by EKF-SLAM may be added together, for example using the ATLAS framework (Bosse and Zlot, 2008), however robust data association is essential to ensure no gross errors in position are incorporated into the EKF.

EKF-SLAM schemes have high (quadratic) complexity because a matrix of covariances between landmark positions and the current robot pose is maintained. If the robot trajectory is estimated however, the covariances between landmark positions do not need to be estimated, as landmark observations are conditionally independent given the robot's pose. Particle filters such as FastSLAM (Montemerlo et al., 2002) can efficiently represent the distribution of possible trajectories, possibly with a constant number of particles when trajectories contain many small loops, and sensor and odometry measurements are reliable (Stachniss et al., 2005). A particle filter also has the potential to model uncertain data associations (or equivalently uncertain loop closure events), however these must be resolved soon afterwards (Nieto et al., 2003). There is also a risk of impoverishment, i.e. the set of particles not being representative of the robot's trajectory.
An alternative graphical model of SLAM, where nodes represent the robot's pose and edges represent relative pose estimates, allows accurate maps of large areas to be generated, despite substantial accumulated errors. These methods also achieve lower complexity, and have the advantage that gross errors (for example incorrect loop closures) could potentially be removed. A pose graph free of gross errors is needed initially however, as shown by Figure 3.7. These methods also clearly illustrate the importance of loop closure for accurate mapping.

3.3 Visual Simultaneous Localisation and Mapping

Most of the SLAM algorithms described in the previous section have been applied to visual SLAM, however not always with as much success as with other sensors or sensor combinations. Some existing real-time systems for visual SLAM suffer from large accumulated errors, and often encounter gross errors or fail entirely in visually difficult environments (for example dynamic environments).
This section describes how SLAM algorithms have been adapted to cope with the difficulties of visual SLAM. As described earlier, these difficulties include data association difficulties (incorrectly matching features between frames; Eade and Drummond, 2008), large non-Gaussian error distributions in reconstructed point depths (Montiel et al., 2006), moving objects incorrectly identified as being static (Williams et al., 2008), and poorly-conditioned geometry. An additional problem when positioning using only a single camera is a global scale ambiguity: the actual scale of the world and speed of the camera can only be estimated by identifying something of known size, and small errors in scale can accumulate over time. However a camera has a key advantage over other sensors: rich information about the environment is collected that may be used to recognise when places are re-visited, even when the robot is lost. This ability to detect loop closure allows accumulated errors to be corrected, enabling accurate long-term positioning in bounded environments.
Recently several visual SLAM systems have been developed addressing some of these problems (Clemente et al., 2007; Eade and Drummond, 2006, 2008; Cummins and Newman, 2008b; Sibley et al., 2010; Strasdat et al., 2010). These all work by identifying a stationary 3D world and estimating the positions of both landmarks and the camera within it. These schemes, and other significant advances in visual navigation, are described in this section. More details of the BoW systems used by some of these schemes to actively detect when loops are closed are described in Chapter 5, and more details of the RANSAC framework for outlier rejection are given in Chapter 6. Similarly, the different feature matching and tracking approaches are analysed in more detail in Chapter 4.

3.3.1 EKF-based and particle-filter-based single camera SLAM

One of the first successful schemes for single camera SLAM was MonoSLAM (Davison, 2003), an EKF-based SLAM scheme. MonoSLAM integrates observations of point-landmarks with a constant velocity motion model. When a landmark is first observed, its location is represented by a set of particles sampled from the ray on which it lies. For subsequent frames, these particles are projected back into the image, and the region around each is searched for the same landmark. The particles are then re-scored depending on where the landmark is found to lie, and after a few observations the distribution of landmark depths is approximately Gaussian, and the landmark can be incorporated into the EKF as a 3D point. In subsequent frames, the EKF's prediction of the landmark location and camera pose is used to identify a region where it may lie.

This region is searched to locate the landmark, and this measurement provides an EKF update, which refines the estimate of both the camera's and landmark's positions. The camera can be positioned accurately for extended periods, provided that sufficient landmarks are tracked, however positioning is limited to small-scale environments, with around 100 landmarks in total.

Numerous extensions to MonoSLAM have been proposed, including the Inverse Depth representation (Montiel et al., 2006), which allows immediate initialisation of landmarks when they are first observed, by estimating the ray on which they lie, and the inverse of their depth. While landmark depths recovered from a small number of observations are non-Gaussian, the inverse of these depths is approximately Gaussian. Once points are reconstructed more accurately, their 3D position is used instead. This innovation allows immediate use to be made of new landmarks, and allows more distant points to be used.

Clemente et al. (2007) extend MonoSLAM to larger environments, by using the submap framework by Estrada et al. (2005), which was described in Section 3.2.4. A 250m loop is closed, giving an accurate global map. Validating the mutual consistency of tracked features before incorporating them into the EKF ensures that positioning does not fail, despite a dynamic environment including pedestrians.

Another extension, by Williams et al. (2007), allows recovery from disorientation and positioning failure (from featureless environments or where motion blur occurs). This scheme detects when tracking fails, and switches to a relocalisation mode instead. The tracking stage of MonoSLAM is modified so that correspondences are sought between each corner point in the current frame, and corner points in previous frames which fall in the region where this corner point is predicted to lie (the FAST corner detector, described in Section 4.3.2 is used). This increases robustness to tracking failure, as larger areas can be searched efficiently. A randomised forest classifier (described in Section 4.5.1) is trained and used to match patches despite changes in scale and orientation. When lost, the system enters a relocalisation mode, where landmarks which are likely to be nearby, i.e. those within a sphere with radius increasing with time, are matched to currently-observed features using RANSAC (described in Chapter 6), until a match is found. The EKF then estimates only the camera pose for a few frames, to avoid making measurements which could corrupt the map while its position is still uncertain. The system is demonstrated mapping a small environment (consisting of 70 features) for over an hour. Tracking frequently fails (when the camera is directed at a featureless region), but recovers when pointed back at a mapped location, however the system fails when objects in the environment are moved. The authors suggest using FastSLAM to enable the system to scale to larger environments, although in later work (Williams et al., 2008) the submap scheme of Clemente et al. (2007) is adopted, and demonstrated mapping and detecting loop closure in a 70m loop.

MonoSLAM's motion model limits the cameras' motion, so that the distance that features can move between frames must be small so that tracking does not fail (or become too costly if a larger region is searched for each feature). In order to operate with more erratic camera motion, Gemeiner et al. (2008) describe an optimised version of MonoSLAM capable of operating at 200Hz, however 'real-time' performance is limited to just a few seconds, as the size of the map, and associated computational costs, grow.

Chekhlov et al. (2006) implement a similar monocular SLAM scheme, using an Unscented Kalman Filter (UKF) instead of an EKF. The UKF is an extension of the EKF to highly nonlinear motion and measurement models; internally the state vector is still represented as a multivariate Gaussian random variable, but the update to this state is a highly nonlinear function. Instead of locally linearising this function, a number of samples are drawn from the distribution of the state vector, these are propagated through the function, then the distribution of states obtained is again approximated by a Gaussian distribution. This is used to model the camera orientation, reducing the effects of linearisation during more erratic motion. A small-scale Augmented Reality application is demonstrated; this is an application where erratic camera motion often occurs, and small errors in orientation are undesirable (as they cause inserted objects to appear to float around).

A similar approach by Lemaire and Lacroix (2007) combines bearings to landmarks with a prior depth distribution in an EKF, which refines these landmark position estimates as more observations are made. Panoramic images give a wide range of bearing measurements to each landmark, and provide the same field of view on repeated visits to a location, enabling loop closure to be detected reliably. Navigation of a 200m trajectory is demonstrated, however this is unlikely to scale much further due to the EKF's high

complexity. Panoramic cameras combined with robot odometry are also used by Dellaert and Kaess (2006), with measurements integrated in a 'square root filter'; this has lower complexity than an EKF (the linear system to be solved is kept sparse); a 200m indoor trajectory is mapped in 11 minutes, although no loops are closed.
In summary, EKFs have been used extensively for single camera SLAM. Difficulties including non-Gaussian error distributions, tracking failure, and ensuring tracked points are reliable in dynamic environments, are overcome with a variety of methods. In addition, submap schemes allow these schemes to scale to large environments, however there is always a risk of complete failure if any outlier measurements are incorporated, and regions mapped are still orders of magnitude smaller than regions mapped using other sensors (consisting of just a single loop of a few hundred metres in most cases).
The FastSLAM particle-filter (Montemerlo et al., 2003) has also been adapted for single camera SLAM by Eade and Drummond (2006). Measurements of landmarks consist of feature-points tracked between frames. Real-time operation is limited to a few hundred landmarks however, of which 20 to 30 are observed per frame, and loop closure is demonstrated only on small scales. On a larger scale, the authors had difficulties ensuring that the set of particles adequately reflected the possible distribution of trajectories, and described the system as 'fragile' (Eade, 2008). Subsequently, the authors adopted an alternative optimisation framework (Eade and Drummond, 2007), which is described later in this section.

3.3.2 Submaps, structure-from-motion, and visual odometry

An alternative to conventional SLAM algorithms is to use structure-from-motion (SFM) techniques to estimate the camera's trajectory and the positions of the features it observes. SFM techniques are normally used for generating optimal 3D models from a set of images, or for terrain modeling from aerial images; the popular 'bundle adjustment' approach (Triggs et al., 1999) estimates the structure via a large nonlinear optimisation of structure and camera positions, optimised using Levenberg-Marquardt or a similar algorithm. Levenberg-Marquardt optimisation has complexity quadratic (or often cubic) in the number of landmarks and poses however, and a traditional implementation would be far too costly to run on a large map in real-time. Note that one useful property of SFM is that a high frame-rate is not necessarily required for accurate reconstruction (Nistér, 2001), provided sufficient (and preferably wide-baseline) observations are made of each feature.
The first systems to apply SFM to navigation were visual odometry schemes. In visual odometry, the camera pose is computed incrementally from observations, but no map is made. In order to use SFM techniques for visual odometry, the size of the optimisation must be limited. In the case of visual odometry only the positions of the last few camera poses, and the positions of features currently visible, are optimised. This frees computational resources for the computation of accurate pose estimates, however without the loop closure detection of SLAM, small errors in position will accumulate over time and grow without bound, until eventually the true pose is unknown.
Nistér et al. (2006) use structure-from-motion techniques to position a single camera over a 600m trajectory through a static outdoor environment. Approximate relative poses and feature locations are estimated from a triplet of frames. This structure, and the camera poses, are then refined iteratively to improve the estimate while removing outliers (Nistér, 2003). Several additional frames can subsequently be positioned with respect to this local structure. Feature matches between frames are established by tracking features over consecutive frames. Accumulated errors are as low as 2% of the distance travelled after two thousand frames.
More recently Mouragnon et al. (2009) position a single camera in a similar way; by tracking features between frames, computing the transformation between frames by RANSAC, then optimising recent poses and structure. A trajectory of 500m in a dynamic environment containing pedestrians is reconstructed, and other tracks are reconstructed with errors considerably less than 1% of the distance travelled. With a stereo camera processing costs per frame increase substantially, however positioning is demonstrated on a more complex, looping track.
Visual odometry is also possible using an EKF-based framework. Civera et al. (2009b) position a camera along a trajectory of 650m through a static outdoor environment, which includes some tight corners, from a dataset which includes images, IMU data and wheel odometry. The trajectory is recreated to within

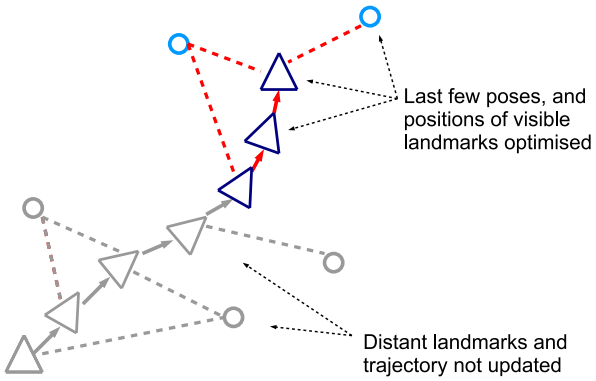


Figure 3.8: Full bundle-adjustment is costly, but in practice optimising only a few recent camera poses, and the features visible from these positions, give accurate trajectories efficiently (sometimes in constant time, until loops are optimised).

1% of ground truth, and hundreds of features are matched robustly between frames using a RANSAC-like framework (described in Section 7.2). Parts of the system run in real-time, although overall the system runs at 1Hz.

SFM techniques have also become popular for solving SLAM. Again, the complexity is limited by optimising the positions of only the last few camera poses, and the features currently visible (Figure 3.8). Klein and Murray (2007) describe a bundle-adjustment based scheme for SLAM, where tracking is used to match features between frames, then occasional keyframes are selected for refinement. A robust cost function together with the large number of points matched between keyframes reduces the effect of outliers, and the system uses the computed geometry to actively search for additional feature matches. In addition, the keyframe selection enables navigation to continue while the camera is stationary; this is a situation which can cause EKF-SLAM, and other systems (Eade, 2008) to fail. A single-room scale real-time Augmented Reality application is demonstrated, and maps considerably more accurate than those estimated with an EKF are generated, although, as with an EKF, the system can fail completely with motion blur or incorrectly tracked features.

A limitation of previous single camera SLAM schemes is that they operate in 3D Euclidean space, where positions in the map are assumed to be normally distributed about some absolute values. This is not an appropriate representation for single camera SLAM, where the global scale ambiguity leads to distances, as perceived by the robot, varying as the scale drifts. This limitation was addressed by Strasdat et al. (2010), who extend the scheme by Klein and Murray (2007) to a larger environment, where frames are connected by transformations incorporating changes in scale. Loop closure detection is also added, by searching the map for landmarks which resemble features which are visible. A 220m loop is mapped in near real-time, then the transformations between keyframes are optimised using Levenberg-Marquardt. As these transformations include the change in scale between pairs of frames, scale drift can be corrected both around the loops and within the loop. Optimising the scale in addition to the transformations is essential to recover an undistorted map.

Global Euclidean representations of space also introduce difficulties when navigating using other sensors; Sibley et al. (2009) argue that it is this representation of space which makes loop closure costly, as many poses must be adjusted in order to incorporate loop closure information. Sibley et al. describe a stereo camera SLAM scheme similar to the scheme by Klein and Murray (2007), with only local poses and landmarks optimised each time a keyframe is added. A sequence of local coordinate frames are maintained, so that no global update is needed on loop closure. The global map consists of a pose graph of these local frames, with odometry and covariances between frames provided by local bundle adjustment. Loops are detected using

BoW (described in Chapter 5). Positioning over a complex track of over a kilometre is demonstrated, with an offline relaxation approach giving an accurate global map. With the addition of active loop closure detection, the scheme is demonstrated generating a topologically correct map of a 121km trajectory (although no map is shown) with position estimates from computer vision used 90% of the time, and a motion model and IMU used to fill in the gaps.

An alternative to this continually-changing co-ordinate frame approach is described by Eade and Drummond (2007). Local maps are built using a bundle-adjustment-like local optimisation. These local maps are fused via an Atlas-like submap framework. The transformations and estimated covariances between coordinate frames are refined by gradient-descent subject to transformations composing to the identity transformation around loops. Large loops can be detected using BoW and closed rapidly by adding edges to the graph (Eade and Drummond, 2008), although optimisation becomes expensive when maps are large (Eade, 2008). Limitations of this scheme include deciding when to start new submaps, and coping with a stationary camera; in these circumstances the system can fail entirely.

A pose graph-based approach using a stereo camera is described by Konolige et al. (2009). A sequence of relative poses between frames are estimated by tracking features, together with relative poses from global loop closure detection, which allow long trajectories to be mapped consistently, and recovery from gross errors, such as becoming lost. Navigation is demonstrated over a 10km outdoor track, although relative positions from an IMU are needed for about 2% of frames where positioning fails, sometimes due to camera occlusion. The authors claim their scheme is applicable to single camera SLAM, although this is not demonstrated, and no framework to model scale drift is proposed. A TORO-like optimisation framework generates global maps, with accuracy indoors comparable to a SLAM scheme using a laser scanner.

3.4 Discussion

In summary, numerous schemes have been demonstrated which enable robots to position themselves using a single camera alone; these schemes often work in real-time and over tracks of hundreds of metres, while often producing accurate maps within a few percent of ground truth. There are two major limitations of these approaches however: firstly the environments explored are largely static, and rapid cornering (during which single camera geometry is poorly conditioned) is largely avoided, however typical indoor or urban environments where robots will one day operate contain moving people and vehicles; a robot moving through these environments must cope not only with these dynamic objects, but also with erratic motion including rapid cornering. To overcome these limitations a successful SLAM scheme must function despite these challenges, and critically, must be able to recover from gross errors in positioning.

The second limitation of many of these schemes (Lemaire and Lacroix, 2007; Dellaert and Kaess, 2006; Eade, 2008; Strasdat et al., 2010) is their high algorithmic complexity—the SLAM algorithms become too costly for real-time operation as the mapped area grows. SLAM methods with lower cost and complexity should be used for longer term navigation.

A further limitation of many of these schemes, particular those based on EKF-SLAM or particle filters, is the potential for irreversible failure if any gross outlier is incorporated into the filter, or occasionally if tracking fails. While numerous schemes work around particular cases where gross errors could cause SLAM to fail, for example by using geometric compatibility tests to validate matches (Clemente et al., 2007; Williams et al., 2007), or by actively detecting tracking failure (Williams et al., 2007), there is always a chance that an outlier will be missed. This is also a potential problem with schemes based on bundle adjustment, however robust cost functions eliminate much of the risk, provided sufficient features are tracked.

The one class of SLAM algorithms which can recover from gross failure are those based on pose graph optimisation, as erroneous relative poses can always be deleted later. A pose graph which is largely outlier-free is needed to generate accurate global maps however. In order to reduce the chance of gross outliers in relative pose estimates, multi-view robust bundle-adjustment based methods, or two-view RANSAC-based methods, appear to be the best approaches, as poses can be computed from large numbers of point matches while outliers are eliminated. High frame-rates are not required for these multi-view approaches, and this provides a possible solution to the problem of tracking failure: by selecting an appropriate sparse sequence

of frames, more expensive but potentially more robust matching approaches can be used to compute relative poses, despite intervening frames containing potential sources of failure such as motion blur, featureless regions, or occlusion by moving objects. This type of approach, where visual odometry is used to estimate incremental poses, which are combined in pose graphs with the occasional loop closure event, is the defining characteristic of the most successful stereo SLAM schemes to-date, including Konolige et al. (2009); Sibley et al. (2010); Newman et al. (2009).

The pose graph optimisation schemes described operate in global Euclidean space, which would make adaptation to model scale drift challenging, however scale drift could potentially be corrected before the pose graph is optimised, or a more expensive optimisation on a pose graph consisting of many local frames, such as the approaches of Strasdat et al. (2010); Estrada et al. (2005); Sibley et al. (2010) may be feasible.

In conclusion, the most suitable contemporary SLAM algorithms for large scale and robust single camera SLAM are those based on pose graph optimisation. These algorithms scale to large environments, can tolerate substantial accumulated errors, and can potentially recover from complete failure caused by gross outliers. A set of initial relative pose estimates which are outlier-free is required, so relative pose estimation should be robust so that few outliers occur, and any remaining outliers should be detected and removed prior to pose graph optimisation.

Chapter 4

Image description for robot positioning

Abstract

This chapter reviews the different ways of detecting features in images, and the different ways these features can be described, in order to choose a suitable detector-descriptor combination for robot positioning. Following this review, several of the most suitable real-time detector-descriptor combinations are compared experimentally. The FAST corner detector almost matches the performance of contemporary corner detectors, while being considerably cheaper to extract. Larger numbers of features provide similar repeatability to only selecting the best features, and all feature detectors are shown to have sufficient localisation accuracy for subsequent pose computation. Considerably higher localisation accuracy is obtained in larger images however. For typical robot motion, cheap image-patch descriptors around FAST corners outperform expensive SURF invariant descriptors, however invariant descriptors would be required if large changes in scale were observed on multiple visits to a location.

BoWSLAM uses images captured from a digital camera to position a robot in two ways: firstly, to recognise when two images show the same place, so that the robot can recognise when it visits somewhere it has been earlier (using BoW; Chapter 5); and secondly, to compute the robot's motion by estimating the position of the camera relative to the observed world as the camera moves (Chapter 6). These processes both require images to be described in a way that allows them to be compared efficiently. In this chapter we examine different ways of describing images, and establish which of these many options are suitable for real-time robot localisation and positioning.

4.1 Image description for location recognition

For robust location recognition it is necessary to detect when two images show the same place regardless of the pose of the robot when they are captured. In the case of an upright robot moving on a plane, for example a wheeled indoor robot or a vehicle following a road, two images showing the same place will show many of the same objects, with the same orientation and vertical configuration, but not necessarily in the same horizontal configuration or at the same scale. For a camera moving in three dimensions (for example attached to a quadrotor or a human), objects can appear at a wide range of orientations and configurations. An additional problem is that two images taken from the same place will not necessarily overlap substantially. Using wider-angle images reduces this risk, and in some circumstances panoramic or omnidirectional images can be used, largely eliminating this problem.
To detect when two images are likely to show the same place they must first be described in a way that allows them to be compared efficiently. For this description either some characteristic of the entire image,

or a set of descriptors each describing a different part of the image can be used.
In some circumstances, for example when there are a large number of training images (many photos of a location from different viewpoints) direct comparisons between colour images downsampled to 32×32 pixels can be used to identify matching images (Torralba et al., 2007). Direct comparisons between pairs of images does not work in general however due to parallax effects and perspective changes, and brute-force searches for transforms which will align images is only computationally feasible in the simplest cases (such as panoramic images captured from nearby locations, where a transform with one degree of freedom will align two images Nielsen et al., 2008). A more suitable option is to use a characteristic of the entire image which does not depend on the image's geometry, such as the distribution of colours it contains. Two images showing the same place are likely to have the same distribution of colours. Histograms are commonly used to describe an image's colours: the colour space (often RGB) is divided into bins, the number of pixels with colours belonging in each bin is counted. A multi-dimensional histogram can be used if multiple colour channels are present. The similarity of pairs of images can then be measured by comparing the similarity of their histograms. The problem with this approach is that two images with the same distribution of colours do not necessarily show the same place; in other words this description is not sufficiently distinctive. However it could potentially be used as a fast first-pass measure of image similarity.
Histograms are used for robot localisation by Werner et al. (2008): 3D 64-bin RGB colour histograms from omnidirectional images are used to partition a building into five zones, each characterised by a different distribution of colours. As a robot moves through the building its position estimate from odometry is improved by matching the histograms it observes to these zones, however the histograms alone do not provide enough distinctiveness for localisation without a prior position estimate. A further limitation of RGB histograms is that changes in illumination will change the distributions of colours observed; a more robust approach is to use a colour space with illumination-invariant channels, for example the hue and saturation channels in the hue-saturation-value (HSV) colour space are approximately illumination-invariant, therefore hue- and saturation-histograms do not change substantially with changing illumination (Filliat, 2007). The value (intensity) channel can also be normalised (adjusted to have a certain mean and variance) to give histograms with reasonable illumination-invariance (Filliat, 2007). 3D histograms are used in combination with other descriptions by Hays and Efros (2008) for a related image-search task: a 3D histogram with 14 bins for each of the illumination-invariant a* and b* channels of the perceptually-uniform La*b* colour space, and a coarser 4-bin representation of the L (luminosity) channel. These histograms are shown to be complementary to descriptions of shape and texture.
An approach to location recognition that provides greater distinctiveness is to describe many different parts of each image (features) with different descriptors. Pairs of descriptors can be compared to measure how similar they appear, so that the two descriptors of an object visible in two images should appear similar. This allows image similarity to be measured: an image with many similar-looking descriptors to another image is likely to contain the same objects, and hence is likely to show the same place. Describing images as sets of descriptors is also useful for computing robot positions incrementally, and is described in the following section.

4.2 Image description for robot positioning

In BoWSLAM, images are also used for incremental positioning, i.e. computing the new position of a robot relative to its previous position, so that its trajectory can be reconstructed. As a robot moves, it captures images of its environment. To be useful for incremental positioning these images must be captured regularly enough that there is substantial overlap between sequences of images, as objects must be observed multiple times in order to estimate the camera's motion relative to them. These images must be described in a way that allows the positions of these objects to be measured in a sequence of images. As with location recognition, one way of doing this is to describe many different parts of each image with a different descriptor, then identify matches between these descriptors.
For a slow-moving robot with a large FOV camera, capturing one or two images every second may be sufficient for positioning. For faster robots moving erratically and cornering quickly, a higher frame-rate

may be needed, for example 10 frames-per-second (10Hz). For real-time operation, descriptions of these images must be computable at comparable speeds (on the kinds of processors with which mobile robots are equipped; Chapter 2), and ideally at higher speeds so that resources may be used for other tasks. In this thesis, datasets captured at between 2Hz (for a wheeled robot indoors) and 10Hz (for a 6 DOF dataset including rapid cornering, erratic motion, and speeds of up to 30 km/h) are used, although lower framerates may also be practical.

Many contemporary camera positioning systems use a feature-tracking approach, rather than matching descriptors. To track features, the locations of features in previous frames are used to predict the region in a subsequent frame where each feature is likely to lie. Higher framerates are required for tracking, so that features do not accelerate substantially between frames; many positioning schemes using tracking require framerates of around 20Hz (e.g. Davison, 2003; Eade and Drummond, 2008; Strasdat et al., 2010), or higher (e.g. Taylor and Drummond, 2009; Gemeiner et al., 2008). When sequences of frames are occluded by moving objects or corrupted by motion blur however, feature tracking will fail and a matching approach is needed to resume navigation.

In summary, when either a low framerate is used, or a sequence of frames is unusable, the problem of positioning a robot first requires the locations of matching objects in two images to be identified, even though the frames may be captured from substantially different viewpoints. Describing an image for this task is closely related to the problem of describing an image for robot localisation, where we want to identify whether two images contain the same objects but are not necessarily interested in matching these objects between images. In the remainder of this chapter we consider contemporary approaches to image description which may be suitable for either of these tasks, ideally however a description scheme suitable for both tasks will be identified.

Describing images as sets of descriptors consists of two steps: firstly the selection of features that will be described, and secondly describing these features in a way that allows the similarity of the objects they describe to be measured. These two steps are described in the following two sections.

4.3 Image feature detection

There are numerous different methods for selecting a set of image features. Most aim to have the following two properties. Firstly, the features should be repeatable: ideally an object corresponding to a detected feature in one image should also lead to a feature in another image showing the same object, so that it is possible to identify that the object exists in both frames when descriptors are compared. Secondly, features should be descriptive: features should selected that are distributed across the most informative parts of scenes so that as much of the information contained in the image is used as possible, maximising the chance that images of the same place can be matched, while distinguishing between different places. A further requirement for the applications described in this thesis is that features should be extracted fast enough for real-time processing of frames. In these applications feature detection is often the only stage where entire images are processed, i.e. where complexity increases with image size.

To ensure that features are repeatable for a particular class of images, some combination of the following properties are required:

Rotation invariance The same features should be detected regardless of the orientation of the object.

Scale invariance The same features should be detected regardless of the distance to the object, i.e. regardless of the apparent size of the object in the image.

Perspective/affine invariance The same features should be detected regardless of the pose of the object. The image of a planar object changes substantially as the camera moves, but these images are related by a perspective transformation (Clarke, 2009). Typically many features are locally planar, and on a small scale the perspective transforms are very well approximated by simpler affine transforms. Perspective/affine-invariant feature detectors aim to detect the same features regardless of the pose of the object, although this is usually only possible up to a certain point due to large changes in local

appearance. Perspective or affine invariance is not possible in general for 3D scenes, as different parts of the same object may be visible in two images.

Illumination invariance Changes in the illumination level should not affect the features that are extracted. This is important for scene recognition, as illumination is likely to change between visits (depending on the time of day, or whether lights are on), but is less important for sequences of frames.

Robustness Motion blur and sensor noise should not affect the features that are extracted. This is usually achieved partly through applying a smoothing function (Gaussian blur) to the image before extracting corners.

Global effects The same features should be detected on an object regardless of the appearance of the rest of the scene. This is impossible to achieve exactly when the number of features extracted from each image is constant (unless features describe the entire image uniformly); often an object comes into view on which many strong features are detected, and other features that were detected in previous images are discarded as being too weak.

For these properties to be useful, the corresponding feature descriptor must also have the equivalent property, for example if a particular object is detected at a range of scales, the descriptors of that object must indicate that the objects appear similar despite the change in scale.
A further important requirement for applications where image geometry is used is that features can be localised accurately within the image. This means that the location of a feature in the image should correspond closely to the projection of that feature's corresponding object in the world into the image. If features are not accurately localisable then errors in locating them in the image will be propagated into estimates of scene and camera geometry derived from these features. Feature localisation accuracy is analysed in detail in Section 4.4.1.
Several classes of image features have been used in various applications, including edges, corner points, blobs, and segments. The following sections detail some of these different approaches.

4.3.1 Edge features

Edge features consist of the lines in an image corresponding to straight edges in the world, for example straight lines on the outline of 3D objects, or the line where an object changes texture or colour. Line features can be detected in intensity (single-channel) images by identifying points where intensity or texture change rapidly in one direction. To find these points we consider the image I to be a function $I(x, y)$ mapping pixel locations to intensity. Two 'derivative images' can then be computed with values $\frac{\partial I(x,y)}{\partial x}$ and $\frac{\partial I(x,y)}{\partial y}$ (these derivatives are approximated efficiently by a filter equivalent to smoothing the image, to reduce the effects of noise and orientation, and taking differences between adjacent pixels). From these derivative images edge pixels can be identified efficiently as the points with high gradient in one direction, as in the Canny edge detector (Canny, 1986) for example.
Alternatively, edges on the boundaries of different textures can be found using a similar approach: firstly, a new "standard deviation image" is computed, where each pixel value is given by the standard deviation of pixel values in a patch around the corresponding point in the original image. The standard deviation is approximately constant within textures, but varies between textures, and on the boundary between dissimilar untextured areas. Extrema in the derivative images of this standard deviation image correspond to edge pixels (Hidayat and Green, 2009). Both of these algorithms give 'edge images' of pixels that are likely to fall on edges.
Straight edges can be detected efficiently from these edge images via a Hough transform (Duda and Hart, 1972), a brute-force but efficient approach where bins representing every possible line in an image are set-up, and each edge pixel votes for the bins corresponding to lines on which it may fall. Bins with the most votes correspond to straight lines in the image. Alternatively the RANSAC robust model-fitting framework, described in detail in Chapter 6, could be used.

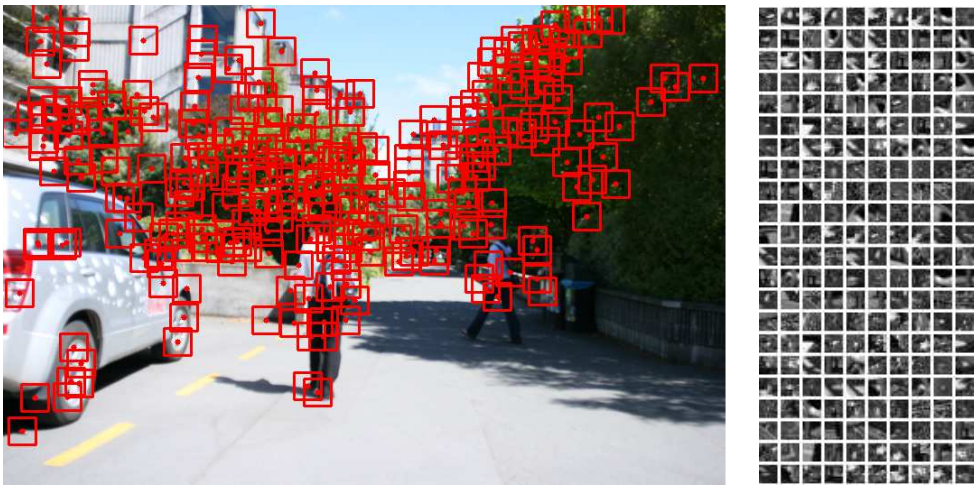


Figure 4.1: Interest points found by the FAST corner detector. A minimum separation constraint is imposed here. Image patches centred on each corner are used as descriptors.

Edge features are widespread in indoor and urban environments, for example around doors, windows, buildings and walls, and in aerial images, for example along roads and boundaries, although straight edges are less common in natural environments. Difficulties in using edge features include describing them reliably so that edges may be matched between images, and in developing algorithms for multi-view geometry based on edge features. While these problems may be overcome in the future, to date the most successful applications of edge features involve tracking edges between video frames, for example Drummond and Cipolla (1999); Rosten and Drummond (2005); Mills and Pridmore (2003), as tracking an edge in a video frame can be achieved using an efficient one-dimensional search around where the edge is predicted to lie.

4.3.2 Corner features

A popular class of features consist of the points in an image corresponding to corners in the world, for example corners on the outline of 3D objects, or corners on the boundary of two regions of different texture or colour. Corners can be detected in intensity images by identifying points where intensity changes rapidly in two directions. One of the most popular corner detectors is the Harris corner detector (Harris and Stephens, 1988), which finds corners efficiently from derivative images as points where the image has significant gradient in the direction perpendicular to its strongest gradient. Harris and Stephens (1988) use an approximation to the gradient strength; however for a small additional cost these gradients can be computed exactly, improving the performance slightly (Shi and Tomasi, 1994). The smallest of the two gradients provides an effective score of the corner's quality.

These detectors only locate corners to the nearest pixel, however potentially more accurate 'subpixel' localisation accuracy can be obtained by examining the corner's surroundings. The kernels used to evaluate image derivatives can be applied to patches at subpixel locations by interpolating between pixel values. A corner's location can then be refined by iteratively moving these kernels until the derivatives are maximised (Wang and Brady, 1992), as implemented in the OpenCV Computer Vision Library (n.d.). Alternatively, corner locations can be improved by fitting a paraboloid to local gradients, then taking the location of its maximum (Zhu et al., 2007).

An alternative way to characterise corners is as local extrema, that is as points with intensity significantly greater (or smaller) than many surrounding points. This approach is taken by the FAST corner detector (Rosten and Drummond, 2006), which finds points with values significantly greater/smaller than 9 (or up to 12) consecutive pixels in a 16-pixel ring around the point. This approach is very fast, as most points can be eliminated quickly. The difference between the centre and the ring pixels provides a measure of the corner's

Figure 4.2: Top 380 interest points found by the FAST (left) and SURF's DoH (right) feature detectors. Features are closely grouped around the most textured regions, potentially leading to poorly-conditioned geometry. Imposing a minimum seperation constraint degrades repeatability (Section 4.4.1).

quality. An example of features found by the FAST detector are shown in Figure 4.1.
The corner detectors described can detect thousands of candidate points across an image, and often these points are tightly grouped together around highly textured areas (Figure 4.2). To ensure that a high proportion of the image is described with a minimal number of corner features, only the highest scoring corners, subject to a constraint that they are further than a certain distance from other corners, can be chosen. This procedure is implemented efficiently by sorting corner locations by their score, then selecting the strongest corners in order of strength, with a minimum separation constraint imposed by binning corners by location, then comparing each candidate with corners in nearby bins. A good spread of feature points across the image may also help to ensure subsequent camera position computation is well-conditioned (Mei et al., 2009).
A big advantage of features corresponding to point locations in the image (i.e. corners rather than edges) is that many problems in multi-view geometry (camera pose and scene structure analysis) can be solved given matching point locations between images (Hartley and Zisserman, 2003). A disadvantage of corner features is that corners often arise in images due to the edges of nearby objects intersecting edges that fall behind them. While these corners are potentially useful for scene recognition (the co-occurrence and proximity of these objects help characterise a particular location) they do not correspond to fixed 3D point locations in the world, and hence usually need to be eliminated before scene or camera pose computation can occur.
If a camera views an object from different orientations, the same corners are usually detected (these corner detectors are rotation-invariant, and are weakly perspective-invariant). However, often when there is a substantial change in scale between images, different sets of corners are detected, due to the fixed size of the detection filters, and the unsophisticated minimum separation constraints. This is only an important consideration when descriptors around these corners are scale-invariant. Corner detectors will detect large numbers of candidate points (before the weaker points are culled) in most indoor, urban or natural environments, and sufficient points for many applications are found even in images appearing feature-poor, for example the Antarctic ice photographs in Figure 7.6.

4.3.3 Blob features

A related class of features are blob features, which consists of small image patches that are brighter or darker than their surroundings. These patches can have a range of sizes (and often shapes), so offer better scale invariance (and some perspective invariance), although blob features are often more costly to extract than corner features. Both a point-location (e.g. a centroid of the patch) and the patch's scale are usually recovered.
One suitable definition of a blob is a point where the Laplacian of the image (the sum of the second derivatives

of the image in two directions, $\frac{\partial^2 I(x,y)}{\partial y^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$) is locally high, indicating a local minimum, maximum or inflection point. The Laplacian operator can be computed at any one scale by convolution with an appropriately-sized kernel, however to detect blobs on a range of scales the Difference-of-Gaussian, or DoG, detector provides a close approximation to the Laplacian (Lowe, 1999). The image is blurred by convolution with Gaussian kernels at a range of different scales, then pairs of these blurred images are subtracted. Local extrema in the difference images correspond to blobs that are brighter or darker than their surroundings.
An alternative measure of local curvature is the Hessian matrix of second derivatives. Curvature is highest when the determinant of the Hessian (DoH) is locally high, corresponding to a blob. Bay et al. (2008) find these blobs by approximating the kernels used to compute local derivatives with Haar wavelets, again at a range of scales. As Haar wavelet responses can be computed efficiently from integral images (images of 2D cumulative sums of pixel values), DoH blobs are considerably faster to extract than DoG, although real-time extraction would still be challenging. DoH features extracted from an image are shown in Figure 4.2.
An alternative blob detector by Matas et al. (2002) finds Maximally Stable Extremal Regions, or MSERs, which are connected components (patches) of images that are brighter (or darker) than their surroundings by some threshold, and which have size that would change only slowly with changes in this threshold. As MSERs are detected independently of the patch's scale or shape these descriptors provide good perspective-invariance. The implementation by Nistér and Stewénius (2008) runs in real-time, although even this implementation is considerably slower than the corner detectors described. A further limitation of MSERs is finding a suitably perspective-invariant descriptor to make use of their invariance. Mikolajczyk et al. (2005) evaluated several highly-invariant blob detection algorithms on a small number of images. MSERs were found to outperform several other detectors at repeatability, although this improvement was marginal, and they note that the parametrisation of detectors had significant effects on the results. Gil et al. (2009) compared MSERs with DoG and DoH blobs and found no significant difference in repeatability, when using high-resolution images captured from a moving camera.

4.3.4 High-level features

An alternative class of features which we are interested in investigating further are higher level objects detected in scenes, for example people detected using the HOG (Histogram of Oriented Gradient) detector (Dalal and Triggs, 2005), text in scenes (signs) read using Optical Character Recognition (Vincent and Ulges, 2008), or corporate or clothing logos (Jenkins, 2008). Extracting these higher level features is generally more costly than low level edge or corner detection, but these observations can potentially tell us much about the content, location and 3D reconstruction of an image (Hoiem et al., 2008).

4.3.5 Random features

Points chosen randomly (Nowak et al., 2006) or systematically (Filliat, 2007) from an image can also be used to describe it, giving good description across the entire image, although often large areas of images contain little information (for example sky or road) so a high descriptor density here is not useful. A disadvantage of choosing features systematically is that lines visible in two images may coincide differently with the grid of features, leading to substantially different descriptions; random features avoid these effects at the expense of slightly worse coverage.
Random or systematically sampled features can be a cheap way to describe images for scene or object recognition (Nowak et al., 2006; Uijlings et al., 2009; Hays and Efros, 2008), but have very poor repeatability and localisation accuracy compared with features found by feature detectors (Rosten et al., 2010). For the related problem of video-search, dense systematically-sampled features appear to outperform detected blobs, and can aid the efficient computation of descriptors (Uijlings et al., 2009).

4.4 Choosing a corner detector

In summary, corner and edge features are the classes of feature most feasible for real-time applications, although blob-features could also be considered if better scale and perspective invariance were needed. The difficulties in describing, matching, and making use of measurements of edges makes corner points more attractive for image recognition applications, and applications where image geometry is used. Many different corner detectors have been proposed which could potentially be suitable for BoWSLAM however, and different detectors perform well on different criteria. In this section an experiment is designed in order to choose an appropriate corner detector based on our criteria.

4.4.1 Design of an experiment to compare corner detectors

While many publications have evaluated different corner and blob detectors (Rosten et al., 2010; Mikolajczyk et al., 2005; Klippenstein and Zhang, 2007; Gil et al., 2009), each evaluates only a few detectors, on a limited number of criteria, and on a particular class of images. These studies are of limited use in choosing a feature detector for real-time robot navigation; three major limitations of these studies are that firstly that the numbers of features detected are usually large (e.g. several thousand; Mikolajczyk et al., 2005), whereas visual navigation schemes often describe images with between as few as ten (e.g. Davison, 2003; Chekhlov et al., 2006), a few tens (e.g. Eade and Drummond, 2007), or a few hundred features (e.g. Mouragnon et al., 2009; Sibley et al., 2009; Handa et al., 2010); and an upper-bound on the number of features is necessary for real-time operation. Secondly, the distribution and localisation accuracy of features are not considered; these are both important when reconstructing camera motion from images. Thirdly the parameters chosen for detectors are rarely justified, and may be sub-optimal. The methods used, and repeatability results obtained, are useful in designing this experiment however.
Schmid et al. (2000) evaluate several corner detectors on images of an artificial scene, where images are captured from measured camera locations. Shi-Tomasi corners ("improved Harris corners") outperform other detectors at repeatability. Rosten and Drummond (2006) also use low-resolution images of an artificial Augmented Reality environment, which are used to compare their proposed FAST corner detector to several other corner detectors. They find FAST corners to be more repeatable than Shi-Tomasi corners, which in turn are more repeatable than several other detectors. Gil et al. (2009) compared the repeatability of various detectors on high-resolution images captured from a moving camera. These images have small to moderate changes in scale and viewpoint. Harris corners outperformed other corner detectors, although Shi-Tomasi or FAST corners were not included.
The performance of several affine-invariant blob detectors, not including DoH or DoG blobs, were evaluated by Mikolajczyk et al. (2005). MSERs outperformed or matched other detectors at both speed and repeatability, however none were extractable in real-time. The evaluation by Gil et al. (2009) included DoH, DoG and MSERs; these were found to have similar repeatability, but were outperformed by Harris corners.
For robot positioning and localisation, the following properties are desirable:

Feasible for real-time applications The faster that features can be extracted, the greater the number of images that can be described in real-time. Ideally feature extraction will leave sufficient resources for other processes. In addition, for real-time applications the number of features detected must also be bounded.

Repeatable As many detected features should be repeated in the other image as possible, so that sufficient matches can be found.

Accurately localisable Necessary to enable scene geometry and camera trajectories to be recovered accurately, and to validate that efficient use is made of large images.

Well-distributed throughout the image Necessary to ensure geometry is well conditioned (Mei et al., 2009), and desirable to allow much of the image's information content to be used for location recognition.

Figure 4.3: Six images used to test feature extraction and description; chosen to be representative of locations where visual navigation would be useful. Different images are used from the training to reduce potential effects of overparametrisation.

Figure 4.4: Images are transformed by adding Gaussian noise, changing lighting levels and applying a perspective warp. Noise with a standard deviation of 5 grey levels is added, together with a change in illumination of up-to 15%. The warp is chosen randomly to scale each axis by up-to 20%, changes in scale of about 10% are typical.

While many corner detection schemes exist (and in addition detectors can always be combined, potentially improving their performance; Mikolajczyk et al., 2005), previous studies allow us to identify the set of detectors which are most likely to be suitable for real-time robot navigation: firstly Harris and Shi-Tomasi corners clearly outperform other corner detectors, and secondly, the recently-proposed FAST detector outperform Shi-Tomasi corners in one study. In addition DoH blobs potentially outperform corners when more invariance is necessary, so a comparison with these is also worthwhile when investigating the limits of corner detectors. As localisation accuracy is potentially important, subpixel refinement of corners is also worth considering.
The simplest way to evaluate these criteria for our selected corner detectors is on images with known ground-truth. A set of real photographs are chosen that are representative of different environments in which the robot will operate (Figure 4.3). For each image, a ground-truth image pair is generated by applying a perspective warp to the first image, then adding simulated sensor noise and a simulated change in illumination. The warp is chosen with rotation, scale, and perspective changes representative of typical robot motion between two views of a scene (Figure 4.4).
These images are not exactly like real images however; the simulated sensor noise and illumination changes are not necessarily realistic, and every feature is a planar feature. However images of 3D scenes would

Figure 4.5: Six images used to train feature extraction and description.

require a model of the world, which would contain errors. These simulated images contain no errors from an incorrect 3D structure, no features from occlusion, no errors from the estimated camera model and are cheap to generate. These properties allow measurements made in these images to be comparable with the theoretical best case. A similar framework to this (including the same sensor noise model) is used in the visual SLAM test image simulator designed by Funke and Pietzsch (2009): synthetic images are rendered by warping real images onto the planar walls of a 3D model of a building, then using a ray tracer to project these images into a synthetic image (these images are used later in this thesis; Figure 8.14).

This experimental setup allows repeatability, localisation accuracy and feature distribution to be measured: repeatability is measured by detecting corners in one image, applying the perspective transform to feature locations to predict where they lie in the second image, detecting corners in the second image, then searching to see if a corner is detected in the same location (within a small radius, e.g. within three pixels of the predicted location). A measure of localisation accuracy is given by the distance of each of these repeated corners from its predicted location. A simple heuristic measure of the distribution of corners is given by dividing the image into a 10×10 grid, then counting the number of grid cells containing at least one of the successfully repeated corners. These performance measures are summarised in Figure 4.6.

A minor issue with this approach is that features may coincidentally fall within three pixels of predicted feature locations, even though they arise as a result of different objects. In large images this is fairly unlikely as the total area within three pixels of a detected feature is small as a fraction of the whole image area. In smaller images this could affect results, particularly as features are often closely grouped around textured regions, however the effect is still small, and should not increase repeatability by more than a few percent, and there is no reason to believe any one detector will be affected more than others.

The second challenge when evaluating corner detectors is to ensure that each one has an appropriate parametrisation for the application. This was found by a brute-force search over the parameter space. The following detectors' parameters were tuned on a separate image set (Figure 4.5) to maximise an arbitrary performance score, the sum of repeatability and distribution, expressed as percentages, for each detector. For the Harris corner detector the parameter k controlling the approximation to the second-strongest eigenvalue is tuned, and is set to a value 0.0035. For both the Shi-Tomasi and the Harris detector the Gaussian kernel size is tuned; a kernel with diameter 3 is used. For all detectors the minimum separation is tuned; the minimum value of 2 is clearly best for every detector. Increasing separation only increases the distribution of repeated features by a small amount, as it causes a large drop in repeatability. While all the corner detectors have minimum quality parameters, these parameters rarely make any difference in the corners chosen, because we only choose a fixed number of the strongest corners from each image. For the DoH detector, default parameters are used, as the parameters all directly affect either extraction time or the level of scale invariance.

Repeatability (%) Proportion of features where a feature is also detected in the correct location in the other image.

Distribution (%) The number of cells in a 100-cell grid containing a repeated feature.

Localisation accuracy (pixels) The distance of repeated features from their predicted location.

Efficiency (%) Proportion of detected features where the closest descriptor in the other image is also the correct match.

Match distribution (%) The number of cells in a 100-cell grid containing a feature where the closest descriptor in the other image is also the correct match.

Figure 4.6: Definitions of detector and descriptor performance measures

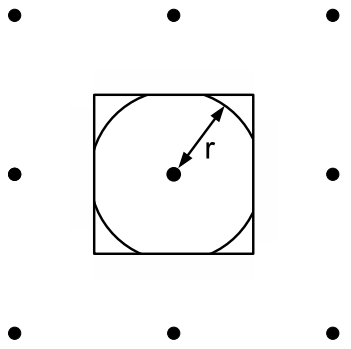


Figure 4.7: A perfect corner detector will always find the pixel centre closest to the projection of the actual 3D corner into the image. The projected point will be distributed uniformly within a square about this closest pixel centre (pixel centres are marked with dots). The cumulative distribution function (c.d.f.) of the absolute error in the corner detector's position for the point is then given by considering the area within this square that also falls within a circle of a particular radius. The p.d.f. is then given by observing that the derivative of the c.d.f. is the length of this circle's perimeter falling within the square, giving Equation 4.4.2.

The experiment was also used to quantify the effects of varying image size, and the effects of increasing the number of features. The experiment is repeated with 250 and 400 features detected per image, and with images sized 968×644 and 484×322.

Measuring image feature localisation accuracy

In this experiment, a perfect corner detector could achieve 100% repeatability and distribution. Any detector with lower performance could potentially be improved. This is not always the case with localisation accuracy however, so in this section we determine how accurate a detector could possibly be.

Many algorithms for structure and motion computation from feature point matches assume that errors in corner localisation are two-dimensional i.i.d. (independently and identically-distributed) normal random variables (Hartley and Zisserman, 2003; Triggs et al., 1999), and hence errors are isotropic. This is a reasonable approximation (when the camera is accurately calibrated), which allows standard least-squares techniques to be applied easily. To aid the analysis of localisation errors, a convenient one-dimensional measure of error is the absolute distance between the projection of the point into the image, and the detected corner location. If the two components of the localisation error are i.i.d. and normal with mean zero and

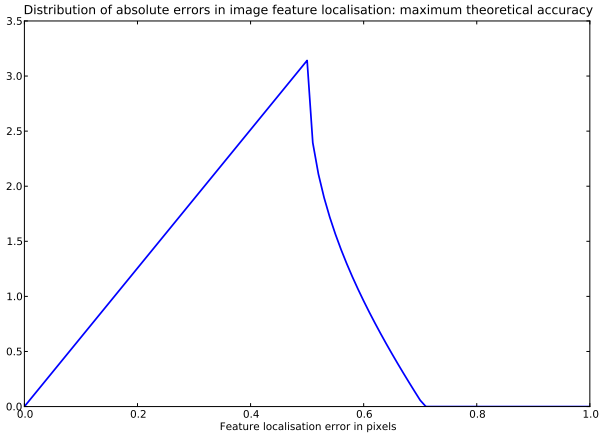


Figure 4.8: The best possible corner detector (without subpixel precision) is one that always finds the pixel centre closest to the projection of the feature into the image. This plot shows the probability distribution of the distances from the projected feature to the closest pixel centre (Equation 4.4.2).

standard deviation σ, this distance r has a Rayleigh distribution with parameter σ (Rayleigh distribution; Wikipedia, 2010), which has p.d.f.:

$$f(r) \quad = \quad \frac{r}{\sigma^2} e^{-r^2/2\sigma^2} \tag{4.4.1}$$

A perfect subpixel-resolution corner detector would localise every feature exactly, however many corner detectors only detect features to the nearest pixel (Figure 4.7). If we consider the best possible pixel-resolution corner detector, which we assume will always locate the pixel centre closest to this projected point, then from Figure 4.7 we can derive the following p.d.f. for the distribution of the feature localisation errors that would be observed:

$$f\left(r\right) = \begin{cases} 2\pi r & 0 < r < \frac{1}{2} \\ 2\pi r - 8\arccos\left(\frac{1}{2r}\right) & \frac{1}{2} < r < \frac{1}{\sqrt{2}} \\ 0 & \text{otherwise.} \end{cases} \tag{4.4.2}$$

This distribution has expectation 0.38260 pixels, and is show in Figure 4.8. Unfortunately the localisation error defined here would be hard to measure directly for a real corner detector, as it requires estimating the exact point in the image that a 3D point should project to. Instead, for the experiment we have devised, where we measure how closely a point detected in one image corresponds to the transformation of that point into in the other image, we have the situation shown in Figure 4.9.

From Figure 4.9, if a subpixel-resolution corner detector localises points with i.i.d. normally distributed errors in each direction about the points' true projection into the image, with standard deviation σ, then the observed localisation errors, $\|\mathbf{E}\|_2$, have a Rayleigh distribution again with parameter $\sigma_R = \mathbb{E}(\|\mathbf{E}\|_2)/\sqrt{\pi}$, and $\sigma = \mathbb{E}(\|\mathbf{E}\|_2)/2$. A perfect subpixel-resolution detector would have $\|\mathbf{E}\|_2 = 0$.

For the detectors which only localise features to the nearest pixel, the best-case theoretical distribution for the localisation error, $\|\mathbf{E}\|_2$, can be derived from Figure 4.9. The derivation is given in Appendix B. The distribution has p.d.f.:

$$N\left(r\right) \quad = \quad \begin{cases} -\frac{1}{2}r^7 + 3r^5 - 6r^3 + 4r & 0 < r < \sqrt{2} \\ 0 & \text{otherwise.} \end{cases} \tag{4.4.3}$$

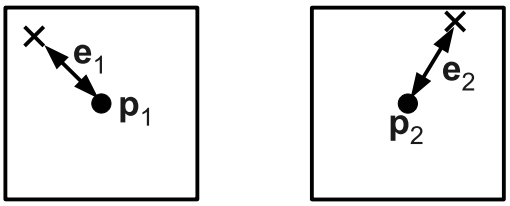


Figure 4.9: A more practical way to measure localisation accuracy is to localise the same feature twice in two images. For an ideal corner detector each true location (marked $\times$) is distributed uniformly within the unit square centred on the detected pixel in each image (marked with dots), for example $\mathbf{e}_1$ from the centre $\mathbf{p}_1$ in the left image and $\mathbf{e}_2$ from the centre $\mathbf{p}_2$. The absolute localisation error is then given by $\|\mathbf{e}_1 + \mathbf{e}_2\|_2$. This quantity may be measured in a real image as the distance between the predicted location of a feature (projected from its detected location in the other image) and its measured location.

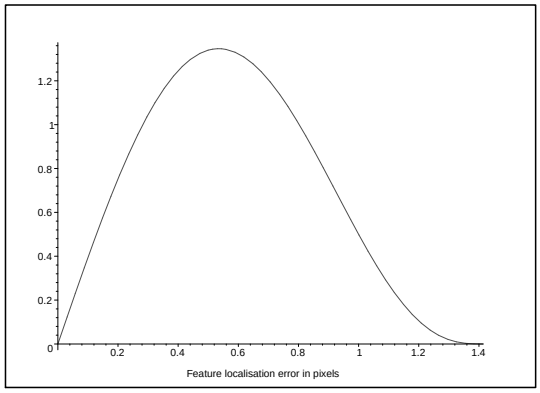


Figure 4.10: p.d.f. of localisation errors which would be measured experimentally for a perfect pixel-resolution corner detector (Equation 4.4.3).

This p.d.f. is plotted in Figure 4.10, and has shape similar to the Rayleigh distribution. Its expectation is $\frac{128}{315}\sqrt{2} = 0.57466$ pixels. This result is used to demonstrate how close pixel-resolution corner detectors are to achieving the highest localisation accuracy they possibly could.

Corner detector evaluation results

Table 4.1 gives the time taken to extract the different features. All except DoH would be feasible for a real-time application, although FAST corners are considerably faster than any other feature. A significant amount of resources could be saved if smaller images were used, as extraction times are approximately proportional to the number of pixels. Subpixel refinement is feasible for a small additional cost, so could be used if it was found to be beneficial.
Tables 4.2, 4.3, 4.4 and 4.5 shows the performance of the tested corner detectors in large images (sized 968×644) and small images (sized 484×322), when the best 250 or 400 features are extracted. In all cases DoH blobs have the greatest distribution across the image, with no significant difference between other detectors. Extracting 400 features rather than 250 increases the distribution by around five percentage points for all detectors and all image sizes.

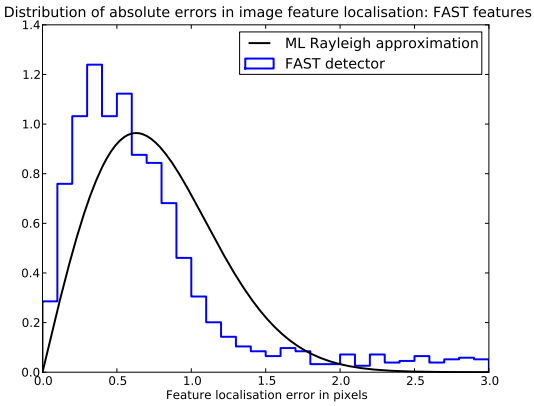


Figure 4.11: Distribution of absolute corner localisation errors from the FAST corner detector; performance is comparable with other detectors.

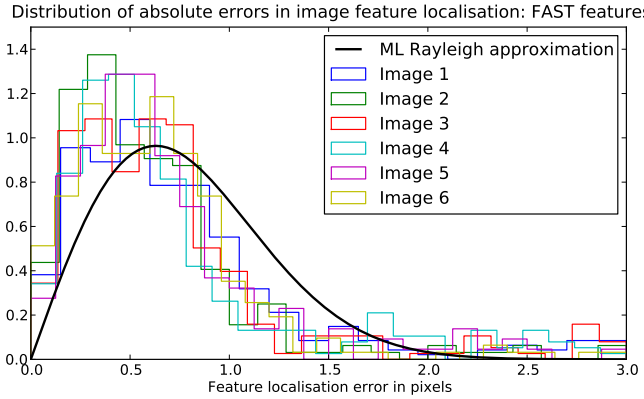


Figure 4.12: Distribution of absolute corner localisation errors from the FAST corner detector. Results from each of the six images from Figure 4.3 are plotted individually, showing there is no significant variation in localisation accuracy between these different images, which show a range of indoor and outdoor locations.

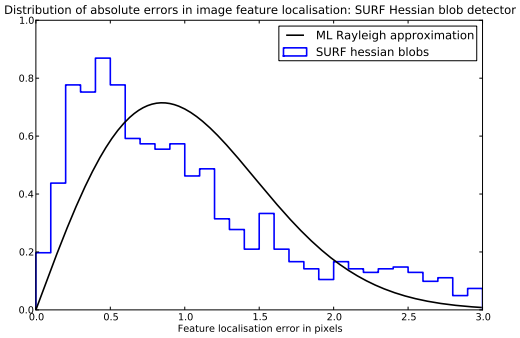


Figure 4.13: Distribution of DoH (SURF) blob localisation errors from experiment. Maximum likelihood (ML) parametrisation of Rayleigh distribution is biased by the heavy tail—either these points are outliers or a heavy-tailed distribution would be more appropriate.

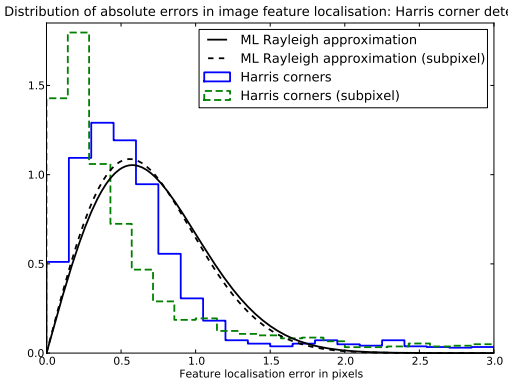


Figure 4.14: Distribution of Harris corner localisation errors from experiment. Note that subpixel refinement gives a significant improvement for the corners that are already positioned reasonably accurately. Maximum likelihood parametrisation of Rayleigh distribution is biased by the heavy tail again.

Table 4.1: The FAST corner detector is considerably faster than other detectors. DoH blobs are barely feasible for real-time use, although Harris-like detectors are, and subpixel refinement adds little additional cost. Features are extracted from 968×644 images with detectors based on OpenCV Computer Vision Library (n.d.) implementations, which are not particularly efficient. All detector implementations are single-threaded.

Detector	Time to compute (ms)
FAST	5
Harris	47
Harris+subpix	60
Shi-Tomasi	48
Shi-Tomasi+subpix	60
DoH (SURF) blobs	310

Table 4.2: Performance at repeatability and localisation in large (968×644) images, top 250 features. Corner features have low localisation errors, close to the best possible for pixel-resolution corner detectors. Performance measures are defined in Figure 4.6.

Detector	Repeatability	Distribution	Localisation error (pixels)
FAST	66%	32%	0.65
Shi-Tomasi	69%	30%	0.74
Shi-Tomasi+subpix	68%	30%	0.61
Harris	73%	30%	0.64
Harris+subpix	72%	30%	0.61
DoH (SURF) blobs	67%	33%	0.97
Theoretical best	100%	100%	0.57

Table 4.3: Performance at repeatability and localisation in large (968×644) images, top 400 features. Compared to the top 250 features, these lower-quality features do not have significantly different repeatability or localisation error.

Detector	Repeatability	Distribution	Localisation error (pixels)
FAST	68%	38%	0.69
Shi-Tomasi	72%	36%	0.75
Shi-Tomasi+subpix	70%	36%	0.63
Harris	75%	35%	0.67
Harris+subpix	73%	35%	0.64
DoH (SURF) blobs	65%	38%	1.01
Theoretical best	100%	100%	0.57

Table 4.4: Performance at repeatability and localisation in small (484×322) images, top 250 features. Localisation errors are slightly worse than in larger images, showing that larger images do lead to considerably more accurate localisation. Otherwise performances are similar to performances in larger images.

Detector	Repeatability	Distribution	Localisation error (pixels)
FAST	70%	32%	0.76
Shi-Tomasi	76%	31%	0.83
Shi-Tomasi+subpix	74%	32%	0.75
Harris	79%	31%	0.74
Harris+subpix	75%	31%	0.74
DoH (SURF) blobs	63%	36%	1.00
Theoretical best	100%	100%	0.57

Table 4.5: Performance at repeatability and localisation in small (484×322) images, top 400 features. Again, introducing more features does not significantly degrade localisation accuracy or repeatability.

Detector	Repeatability	Distribution	Localisation error (pixels)
FAST	70%	39%	0.79
Shi-Tomasi	76%	38%	0.84
Shi-Tomasi+subpix	74%	38%	0.80
Harris	78%	37%	0.77
Harris+subpix	75%	37%	0.78
DoH (SURF) blobs	62%	43%	1.08
Theoretical best	100%	100%	0.57

The Harris detector has the highest repeatability by a small margin in all cases, with all corner detectors outperforming DoH blobs on these images. Repeatability is slightly higher in smaller images, probably partly due to the higher probability of descriptors coincidentally falling near to each other. Correspondingly, localisation errors are slightly worse in smaller images. Repeatability is slightly higher when more features are extracted; using these lower-quality features does not degrade performance.
Subpixel refinement decreases localisation errors by a small amount (5-20%) in most cases, however repeatability is slightly reduced, particularly for lower-quality features, as the refinement sometimes diverges. Detecting this divergence could make subpixel refinement worthwhile. Note that gradient descent-based subpixel refinement does not improve the localisation accuracy of FAST corners, even though the corners that are detected appear similar. All detectors have localisation errors close to the theoretical maximum for a pixel-resolution corner detector, except for DoH blobs which are substantially worse. Subpixel refinement does not push detectors through this boundary, although this may be due to the coinciding-feature problem.
Figures 4.11, 4.13 and 4.14 show the distributions of localisation errors for the FAST detector, the Harris detector and DoH blobs. Each has a heavier tail than the corresponding maximum likelihood Rayleigh distribution, indicating that either a heavy-tailed distribution would be more appropriate, or corners arising from nearby features are being matched incorrectly. Subpixel refinement is clearly improving the localisation accuracy of features which are already localised accurately. Figure 4.12 shows the localisation errors of the FAST detector in the six test images, and shows that there is no significant variation despite the wide range of environments shown.

Corner detector evaluation conclusions

A range of different corner detectors have been evaluated for their suitability for robot localisation. Unlike previous reviews, localisation accuracy and distribution have been measured, and the effect of different image sizes, and of extracting different numbers of features, has been investigated.
Harris corners, Shi-Tomasi corners or FAST corners would all be suitable for robot navigation, of these a major advantage of FAST corners is the low cost of extraction, although at the cost of slightly worse repeatability.
Features are as accurately localised in large images as small images, hence a higher resolution image will provide more accurate measurements for camera motion estimation. In Chapter 6, we note that large localisation errors (with a mean of about three pixels or more in a low-resolution image) can cause a considerable increase in the cost of camera pose computation. In this experiment all of these detectors' localisation errors are small enough to avoid this problem. In real images however, localisation errors are likely to be higher, due to features on 3D objects appearing in slightly different places from different viewpoints, features from slowly moving objects or slowly moving occlusion boundaries, and errors from camera calibration.
This evaluation had several limitations however, in particular that a repeatable corner detector is only useful for wide-baseline matching when paired with a suitable descriptor. It is also unknown whether the sample data is truly representative of real data. Average performances are evaluated on a set of six images, however worst-case performance is often more important, especially when positioning using only a single camera, in which case a single failure out of thousands of frames can introduce gross errors. Worst-case performance is very data-dependent.
In conclusion, any of the corner detectors evaluated could potentially be used in BoWSLAM. Ideally the FAST corner detector will be used, as it is substantially faster than other detectors, although is not quite as repeatable as Harris-like corner detectors on this data. In order to find sufficient matches between frames, an appropriate feature descriptor must also be used, and the performance of this descriptor will depend on the corners detected. In the following section various descriptors are reviewed, then a second experiment builds on the results of this experiment in order to decide an appropriate detector-descriptor combination for BoWSLAM.

Further work in feature detection

The FAST detector was chosen based on the requirement for features to be detected in real-time, and the assumption that changes in scale were typically not too great. However for some applications (Section 8.4) scale can change rapidly with changing view. In this case FAST corners are not necessarily suitable due to their limited scale-invariance. This problem was recently addressed by Agrawal et al. (2008), who propose a new detector, CenSurE (Center Surround Extrema) which aims to be both scale-invariant and to be extractable in real-time. Like DoH features, these features are computed from applying bi-level DoG-like filters to integral images, and scale-invariance is given by detecting features at a range of scales. The result is a scale-invariant detector with comparable performance to DoH, but which can be extracted in about one-third of the time. These descriptors were used successfully by Konolige et al. (2009) who use CenSurE features to register sequences of frames in a visual odometry scheme.
A faster approach to multi-scale feature detection may be to extract FAST corners on multiple scales (Wagner et al., 2008; Newman et al., 2009); the cost of extracting FAST corners in several down-sampled images would still be a fraction of the cost of extracting other scale-invariant features such as DoH or CenSurE blobs. Similarly, investigating the subpixel refinement of FAST corners may improve their localisation accuracy.
Finally, the model of sensor noise and illumination change used are unlikely to be particularly realistic. Using real images would solve this problem, however a better approach may be to compare multiple images from a fixed camera and construct a more accurate error model.

4.5 Image feature description

Once features have been extracted from images the next step is to describe the image patch around each feature in a way that allows the similarity between features to be measured. Ideally, two features arising from the same object visible in two images should have descriptors more similar than either descriptor is to the descriptor of any other detected feature.
As with corner detectors, invariance to changes caused by different viewpoints and imaging conditions is desirable. However too much invariance can degrade performance: the more invariance a descriptor has, the higher the likelihood that multiple features from another image will have very similar descriptors. For example, a rotation-invariant descriptor of corner features might match the top-left corner of a window frame in one image to any one of the window frame's corners in another image, whereas a simpler descriptor may match only the top-left corner. Clearly this rotation invariance is needed if the camera may rotate substantially between images (changing which window corner appears in the top-left for example), however often images are captured from similar viewpoints when a robot visits the same environment repeatedly. Usually perspective or affine invariance would imply scale and rotation invariance, however often a weaker invariance, where the camera is assumed upright, is sufficient (Bay et al., 2006).
An additional invariance needed for descriptors is invariance to errors in feature localisation: if the patches being described are not quite centred on exactly the same point, the descriptors should still be similar.
While many descriptors have been proposed, a major limitation of many is the time needed to extract them. Similarly, some descriptors are considerably more expensive to compare than others, and storing large numbers of descriptors can use a large amount of memory. A range of popular descriptors which may be suitable for image description for robot localisation are analysed in the following sections. Many more are reviewed by Li and Allinson (2008) and Mikolajczyk and Schmid (2003); these include many expensive descriptors of image regions, and several low-dimensional descriptors, which are not considered here.

4.5.1 Image patch descriptors

A simple and often very effective descriptor of the region around an image feature is simply a patch from the image centred on the feature (Szeliski, 2005; Taylor and Drummond, 2009; Davison, 2003). To compare two patches for similarity, the sum of squared differences (SSD; Moravec, 1979), or sum of absolute differences (SAD; Milford et al., 2006) between corresponding pixel values can be used. Examples of patches centred on FAST corners are shown in Figure 4.1.

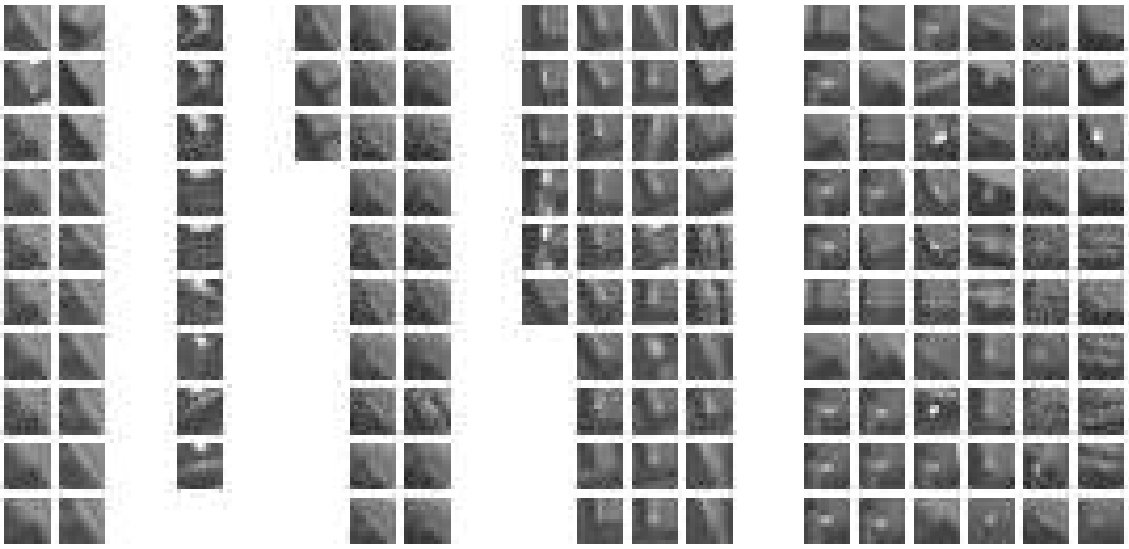


Figure 4.15: Oriented patches are extracted from an indoor dataset. Simple patch descriptors are made orientation-invariant by rotating them to align local image gradients. These oriented patches are grouped by similarity.

These patch descriptors can be made illumination-invariant by normalising the patch (adjusting values to have a particular mean). Alternatively the normalised cross-correlation, which is often used to compare patches (Mikolajczyk and Schmid, 2003), is equivalent to the SSD after normalising the means and standard deviations.
To make patches rotation-invariant they can simply be rotated to be oriented with the direction of the image intensity gradient (Figure 4.15; Brown et al., 2004). Blurring the image (by convolution with a Gaussian kernel) and sub-sampling over larger patches can be a good strategy to improve descriptors' robustness in the presence of noise (Mikolajczyk and Schmid, 2003), and coloured patches can also be used.
The main limitation of using patches as descriptors is their limited invariance to affine or perspective changes in the image (Taylor and Drummond, 2009). In many Augmented Reality applications a known target is located in a scene. In this case, a classifier trained on patches from the target, sampled at many scales and from many viewpoints, can be used to match patch descriptors despite perspective changes (Taylor and Drummond, 2009), however training these classifiers for more than a small number of points is infeasible. A related scheme which can train classifiers for larger numbers of patches is the randomised forest scheme of Lepetit and Fua (2006), although again training data is required and training is computationally expensive.
Many robot positioning schemes assume motion models, and hence can predict the appearance of image patches when they are observed. Klein and Murray (2007) warp patches from the previous image before matching with the latest image, and Chekhlov et al. (2006) store multiple patches subsampled at different resolutions for every landmark; a patch in the new image matching any one of these is likely to be the same landmark.

4.5.2 Histogram descriptors

Instead of using one histogram to describe an entire image (Section 4.1), many individual histograms can be used to describe the patches around each feature in an image. The advantage of using many local histograms is that spatial associations between colours are preserved. Histograms are cheap to extract and hue- and saturation-histograms provide a convenient and reasonably concise descriptor of colour that may be complementary to a descriptor of shape. Local 16-bin hue- and intensity-histograms are used in combination with descriptors of appearance (SIFT) by Filliat (2007) for indoor robot localisation. Similarly, normalised

Table 4.6: Times to extract 400 features from an image sized 968×644. Greyscale image patch features are considerably cheaper to extract than SURF features. The time for SURF descriptors does not include the computation of the integral image, which is computed earlier when DoH blobs are extracted.

Descriptor	Time to compute (ms)
Image patch	5
Oriented image patch	9
SURF	90

local histograms at a range of scales are used for detecting repeated shots of each location in a film, by Chum et al. (2007).

Alternative descriptors of colour are also occasionally used. Yuen and MacDonald (2002) localise a robot in a large office using a combination of vertical edges, which are used to accurately triangulate positions, and local colour descriptors, which provide additional distinctiveness. The vertical edge features from a panoramic image are matched to corners with known positions from a floor plan. The edges can often be matched in many different ways, as edge features are often not distinctive. The most likely of these possible matchings is found by comparing the colours of the regions between each pair of vertical edges. These colours are represented by a single value: the hue when the saturation is high, and the negative intensity when saturation is low (i.e. when hue is not appropriate). The concise representation of colour enables many candidate positions to be evaluated, and the most likely location for the robot to be identified.

4.5.3 Affine- and perspective-invariant descriptors

Affine- and perspective-invariant descriptors aim to describe local planar features in such a way that the same feature viewed from different viewpoints will have similar descriptors. Usually these descriptors encode the shape of a local image patch as a vector; and the similarity of a pair of these features is given by the SSD between their descriptor vectors. Many proposed invariant descriptors are reviewed and compared by Mikolajczyk and Schmid (2003). The major limitations of these descriptors for robot navigation applications however is the high cost of extraction.

The two most popular invariant descriptors are SIFT (Lowe, 1999) and SURF (Bay et al., 2006), which describe image patches around DoG and DoH blobs (Section 4.3.3) respectively. Scale invariance is given by describing patches with the same size as the detected blobs, and rotation-invariance is given by orientating these patches with the local image gradients. Both descriptors then describe the patches in terms of gradient responses at different locations and orientations, leading to a 128-dimensional vector (by default) for SIFT and 64-dimensional vector for SURF. As gradient responses characterise regions of a patch, they are less affected by small perspective changes than raw image patches, while still describing the patch's shape. The main difference between SIFT and SURF is that SURF descriptors are typically faster to extract (Murillo et al., 2007), although extracting them in real-time is still not feasible (Rosten and Drummond, 2006; Cummins and Newman, 2009).

Recent comparisons between SIFT and SURF have indicated that SURF has slightly better distinctiveness when used for robot localisation (Murillo et al., 2007; Valgren and Lilienthal, 2007). Earlier comparisons between SIFT and other proposed descriptors (Winder and Brown, 2007; Mikolajczyk and Schmid, 2003) show SIFT to equal, or outperform other descriptors (which are often even more expensive to extract) on simple wide-baseline matching tasks.

Although expensive to extract, SURF does provide a useful means to test whether a choice of cheaper descriptor is degrading performance. A future, more efficient, invariant descriptor, or SIFT or SURF descriptors extracted in hardware, for example on an FPGA (Se et al., 2004) or GPU (Sinha et al., 2006), would enable the use of these descriptors for real-time applications.

Table 4.7: Performance of descriptors of detected features: proportion and distribution of features which are repeated, and where the matching feature's descriptor is the most similar. Simple patch descriptors are used to describe the image around different corner features. Performance is compared to SURF descriptors. For the small camera motion here image patches have comparable or better performance than SURF. The top 400 features from images sized 968×644 are described. Performance measures are defined in Figure 4.6.

Detector	Efficiency	Match distribution	Match localisation error (pixels)
FAST	47%	32%	0.66
Shi-Tomasi	53%	31%	0.64
Shi-Tomasi+subpix	48%	31%	0.54
Harris	56%	30%	0.61
Harris+subpix	50%	30%	0.50
SURF	49%	34%	0.78
Theoretical best	100%	100%	0.57

Table 4.8: Performance of descriptors of detected features: proportion and distribution of features which are repeated, and where the matching feature's descriptor is one of the two most similar. Significantly more matches are found, even though these images do not appear to be particularly self-similar.

Detector	Efficiency	Match distribution	Match localisation error (pixels)
FAST	52%	34%	0.65
Shi-Tomasi	59%	33%	0.63
Shi-Tomasi+subpix	57%	33%	0.53
Harris	62%	31%	0.61
Harris+subpix	59%	31%	0.51
SURF	51%	35%	0.78
Theoretical best	100%	100%	0.57

4.5.4 Feature descriptor evaluation

In conclusion, many descriptors have been proposed, although few are feasible for use in a real-time application. By far the most widely used descriptors for real-time applications are simple image patches, and it is likely these will also be suitable for BoWSLAM.

Ideally, for recognising when two images show the same place when captured from different viewpoints, scale-, illumination- and perspective-invariant descriptors should be used, although these are typically too expensive to extract from every frame. For matching pairs of frames from similar viewpoints, too much invariance could degrade matching performance.

In this section, the experiment to evaluate corner detectors described in Section 4.4.1 is extended, firstly to parametrise image patch descriptors, and secondly to measure their performance in conjunction with different corner detectors. These simple patch descriptors are compared with more expensive SURF descriptors, which provide a benchmark for matching widely-spaced views.

A perfect descriptor will describe two views of an object appearing in two images as being at least as similar as any other match to one of these descriptors. This allows us to measure the efficiency of a descriptor; we define efficiency as the percentage of features where the most similar descriptor in the other image corresponds to the correct matching point. It is not possible to differentiate between many identical-looking matches purely from descriptors however, so no descriptor is likely to achieve 100% efficiency. As with repeated features, the distribution and localisation accuracy of these efficiently-described features are measured.

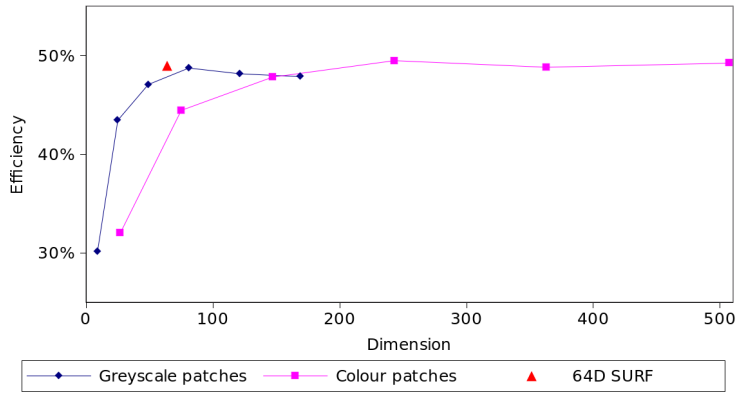


Figure 4.16: Performance of patch descriptors of different sizes, centred on FAST corners. Simple RGB coloured patches perform worse than greyscale patches of the same dimension. Performance increases are marginal for patches sized larger than 9×9 (81D for greyscale patches or 273D for coloured patches), reflecting the fact that some image patches do appear the same. 66% of FAST corners were detected in both images; hence if features were not self-similar then a descriptor could achieve efficiency of 66%. Patch descriptors are parametrised on the images in Figure 4.5 and tested on the images in Figure 4.3.

Greyscale patch descriptors are used for these comparisons. Figure 4.16 shows that they provide considerably better efficiency for the same length of descriptor than coloured patches. Square patches sized 11×11 are used; increasing this size gives no significant increase in efficiency, but at a cost of more expensive comparisons. The descriptors are parametrised by a brute-force search over the parameter space, and are sub-sampled from an area three times as large, and blurred with a Gaussian kernel with standard deviation 2. Interestingly, the most effective area from which to extract each patch (about 33×33 pixels) appears to be independent of image size or patch dimension, suggesting it is a property of the size of features that are detected as corners, rather than the size of objects in the world.

The results of the comparison of patch descriptors centred on different kinds of corner is shown in 4.7. The performance of the different corner detectors is similar, with FAST corners performing almost as well as Harris-like corners. Comparison with Table 4.3 shows that the repeated points with the lowest localisation errors are more likely to be efficiently described, for all descriptors. This indicates that minimising detector localisation error is important for improving the efficiency of descriptors.

In the following chapter we consider the problem of computing camera poses when descriptors are similar to multiple candidate features in the other image. Table 4.7 shows that including descriptors which are correctly matched to one of the top two most similar descriptors in the other image, rather than just the top one, significantly increases descriptor efficiency, by around six percentage points in this case, even though the images (Figure 4.3) are not chosen to be particularly self-similar. Patch descriptors provide more correct candidate matches than SURF descriptors in this case; the performance of SURF features is only slightly improved by considering the second best matches.

Table 4.9 shows the performance of descriptors when images are substantially rotated; in this case the performance of patches oriented to match the local average gradient is comparable with the performance of rotation-invariant SURF descriptors. Descriptors which are not oriented fail entirely in this circumstance. When scale is changed substantially however (Table 4.10), oriented patch descriptors perform very poorly, while SURF still efficiently matches features. The poor performance of patch descriptors is due to the descriptor rather than the detector—the detectors achieve high repeatability despite the change in scale. This indicates the importance of choosing an appropriate invariant descriptor when changes in scale are

Table 4.9: Performance of rotation-invariant descriptors of detected features. Oriented patches are compared with SURF descriptors; the oriented patches have comparable performance and better localisation accuracy.

Detector	Efficiency	Match localisation error (pixels)
FAST	40%	0.66
Harris	45%	0.62
SURF	45%	0.94

Table 4.10: Performance of rotation-invariant descriptors of detected features when there is a significant change in scale (40% rather than about 10%). The repeatability of corner detectors is higher than for SURF blobs, however the oriented patches describe features very poorly compared with SURF descriptors, as these patches are not scale-invariant.

Detector	Repeatability	Efficiency
FAST	67%	3%
Harris	73%	4%
SURF	55%	36%

substantial.

4.5.5 Feature descriptor conclusions

This evaluation has indicated that for robot positioning, simple patch descriptors allow images to be efficiently matched when changes in viewpoint are not too large. Patches centred on Harris corners make the most effective descriptors, although patches centred on cheaper FAST corners also perform well. When substantial changes in scale are observed, patch descriptors are not suitable and scale-invariant descriptors should be used; this is due to the poor invariance of the descriptors rather than the corner detectors.
This evaluation also showed the importance of localisation accuracy in efficiently describing features: boosting localisation accuracy would potentially improve the performance of description. We also showed that features often have multiple potential match candidates. If the correct match from these could be identified, the number of matches found could be significantly increased.

Feature descriptor evaluation limitations

For location recognition, this evaluation is limited as it is not clear how the efficiency of features at matching pairs of images relates to their effectiveness at finding images showing the same place from large sets of images. As with feature detectors, the large numbers of parameter choices, and the range of possible camera motions make choosing a detector-descriptor combination which will work well everywhere challenging.
A further limitation of this evaluation is that no distinction is made between matching nearby (e.g. consecutive) frames, and matching temporally distant frames. When camera motion is likely to be small, the orientations used to orient descriptors, and the scales at which they were detected, can potentially be used to aid matching, for example by only matching features detected at similar orientations in this situation.
The results given are exact for the six images in Figure 4.3, using the parametrisations found. The relative performance of the different detectors is consistent between images and between different experiments, supporting the significance of conclusions we draw from this experiment, however the conclusions drawn are only relevant if the images shown reflect the images a robot will encounter. Clearly these conclusions are not true for all images, for example on the images captured by Rosten et al. (2010) of an artificial environment,

the FAST detector is shown to be more repeatable than other detectors. None of the other review papers discussed address this limitation.

4.5.6 Matching descriptors

Given descriptors extracted from two frames, we aim to identify matches between descriptors in one and descriptors of similar-looking features in the other. Two images containing many similar descriptors are likely to show the same objects and hence may show the same place. For camera positioning, matches between feature locations are also needed, in this case the reliability of matches is important as well: sufficient correct matches must be found that camera motion can be computed efficiently (the identification of a set of correct matches from sets of candidate matches contaminated with gross outliers is analysed in detail in Chapter 6). When the frame-rate is high, and camera motion is smooth, tracking is likely to be an efficient and effective way to find matches between consecutive frames. A feature detected in one frame is tracked into subsequent frames, by using its previous motion to predict where in the new frame it will lie. Either the region around each predicted feature location can be systematically searched (Davison, 2003), or descriptors of each detected corner in this region can each be compared to the descriptor of the tracked corner to find the closest match (Nistér et al., 2006; Klein and Murray, 2007). Alternatively, the point where an image patch is best aligned with a subsequent frame can be found by gradient-descent (Lucas and Kanade, 1981). Tracking is not always possible however; for example when a robot re-visits a region, images captured from substantially different viewpoints must be registered. Matching is also needed when tracking fails due to motion blur, occlusion by moving objects, or erratic camera motion.

The aim of wide-baseline matching algorithms is to find the most similar match to each descriptor out of all of the descriptors in the other image (Lowe, 1999). To improve the quality of these matches, matches can be rejected if there are multiple similar candidates in one image, i.e. the similarities of the best and second best candidates are within some threshold, for example 20% (Brown et al., 2004; Nistér et al., 2006).

To find these matches, all possible pairs of descriptors can be compared (a 'brute-force' approach). This is feasible for small numbers of descriptors which can be compared cheaply, however it becomes costly for larger numbers of descriptors. If C descriptors are extracted from each image, C^2 comparisons will be needed. For example, finding matches between two sets of 300 121-dimensional image patch-descriptors, compared by SSD, takes 50 milliseconds, but comparing 100 from each image takes only 6 milliseconds.

A more efficient method for comparing large numbers of high-dimensional descriptors is to use a kd-tree (Beis and Lowe, 1999). A kd-tree is a balanced binary tree to which descriptors are added. At each node descriptors are partitioned into two sets by thresholding one component of the descriptor, this is repeated at a fixed number of levels. The splitting points are chosen from training data so that the component with the highest variance is split about its median. This strategy ensures that the tree will be approximately balanced, and that similar descriptors are likely to end up at nearby leaves.

To use a kd-tree for matching, the descriptors from one image are first added to the tree. To find a match to a descriptor from the second image, its corresponding leaf node is found. This descriptor is now compared to other descriptors at this leaf. Other leaves that are nearby in the tree are then considered in turn, until some fixed number of leaves have been checked. The most similar descriptor which has been found is very likely to be the most similar overall, and typically only a small number of comparisons are needed, speeding up matching. kd-trees are most likely to find the correct match when one descriptor is significantly more similar than others (Lowe, 2004); these tend to be the most reliable matches.

4.6 Conclusions

In conclusion, appropriate descriptors and detectors for real-time robot positioning are simple greyscale image patches centred on FAST or Harris corners. These descriptors can be extracted cheaply, and for simple camera motion with limited changes in scale they are as-good or better at describing frames than more expensive invariant descriptors such as SURF.

These descriptor and detector combinations are also likely to be as suitable as other simple descriptors for location recognition when changes in scale are small, however in general scale-invariance is likely to be necessary.
Initially, high-resolution images and larger patches should be used to evaluate and demonstrate the capabilities of BoWSLAM, however both of these parameters could potentially be reduced a certain amount to reduce computational costs, with potentially only a small drop in performance.
A minimum separation constraint on features appears to degrade repeatability for all detectors, however many authors have used similar constraints for robot navigation (Strasdat et al., 2010; Mei et al., 2009; Newman et al., 2009), suggesting that in practice this may be necessary, especially when the overlap between matched frames is low.

Chapter 5

The Bag-of-Words algorithm for robot localisation

Abstract

Many schemes have been proposed to recognise when two images show the same place. Of these schemes, the most efficient, and some of the most successful, are those based on the Bag-of-Words (BoW) algorithm. The BoW algorithm represents the descriptors extracted from an image as a set of 'image words' from a discrete 'dictionary'.

In this chapter, location recognition schemes are reviewed, and the BoW model is described in detail. A new BoW scheme based on the approach of Nistér and Stewénius (2006) is proposed. The new scheme creates a hierarchical dictionary of image words online from the features which are observed. This retraining scheme is shown to be more efficient than previous schemes which build dictionaries online. Performance comparable with other contemporary schemes is demonstrated on a standard dataset, and in a pedestrian navigation experiment.

5.1 Introduction

Eventually a robot travelling in an unknown environment will re-visit somewhere it has been before. This event is known as loop closure. If the robot is positioning itself by dead-reckoning (by accumulating incremental position estimates from its sensors, rather than from a absolute positioning sensor such as GPS) then detecting loop closure events is essential for long-term positioning, otherwise accumulated errors will eventually render any position estimate useless. Loop closure also improves the accuracy of the robot's map, as accumulated errors in the map can be corrected around loops, and is needed to ensure that a consistent map is maintained, where each area is mapped once only. In addition, loop closure detection is the only way that an autonomous robot can resume navigation following gross errors or disorientation. Loop closure events must be detected reliably however, as incorrectly detected loops have the potential to corrupt a robot's map and/or its position estimate (Bailey and Durrant-Whyte, 2006).

Sometimes a robot will know approximately where it is, and can use this information to help identify when loop closure will occur, and where it is in its map (e.g. Cole and Newman, 2006; Bosse and Zlot, 2008). Often however, the robot will not know where in the map it could be, because significant errors have accumulated in its position estimate since it last visited the area, because of significant distortions or uncertainties in its map (particularly if the map still consists of disconnected or weakly connected components), because of gross errors in its position estimate, or because of unobservable motion (the 'kidnapped robot' problem (Thrun, 2002); for example a domestic robot that is moved while switched off). In these circumstances the robot must actively search much or all of the map for its current location; this is known as active loop closure detection.

If a robot can predict approximately where in the map loop closure will occur, then loop closure detection simply involves matching current observations with observations made when mapping this region. For example, a wheeled robot in an approximately planar indoor environment, and navigating using a 1D laser scanner, can search the region of its map where it believes it may be for the place where the current scan is best aligned with the previous scan (Folkesson and Christensen, 2007; Zhang and Ghosh, 2000). Alternatively, a robot observing a small set of landmarks (for example a small number of image features observed by a camera) can iterate over possible correspondences between these landmarks and landmarks in the map, and select the most likely alignment, via an algorithm such as Joint Compatibility Branch and Bound (JCBB; Neira and Tardos, 2001), however JCBB has exponential complexity in the number of landmarks, and is only practical for searching small regions of a map (Civera et al., 2009b).

For active loop closure detection, more efficient methods are needed to search the map. One sensor which enables a robot moving on a plane to localise itself in its map efficiently is a laser scanner: the map may be searched by encoding local scans as distinctive descriptors (histograms of gradients), then by comparing these descriptors with a descriptor of the current scan (Bosse and Roberts, 2007).

An additional requirement for loop closure detection is that sufficient information is collected to distinguish between different parts of a map, and to make it unlikely that regions that have not been mapped will be matched to mapped regions. This is challenging in most environments due to different places appearing similar, and due to places changing between visits. These changes include objects such as vehicles, furniture, etc. moving around, and, in the case of computer vision, include changes in illumination. Laser scans are often limited in the amount of distinctiveness they contain, for example many loop closure opportunities will be missed in some outdoor environments (Bosse and Zlot, 2008). In typical indoor environments each laser-scan of right-angle walls and doors could potentially be matched to many different places in the map, although this can sometimes be resolved by making more observations (Zhang and Ghosh, 2000).

One sensor that more often provides the distinctiveness needed is a camera. Cameras are ideal for detecting loop closure as they captures rich and distinctive information about the environment. Enough objects are typically viewed in a scene that small changes can be neglected, and the strong geometric constraint that the two images show much of the same static 3D structure adds to the distinctiveness (Cummins and Newman, 2009; Konolige et al., 2009).

Numerous schemes have been proposed for recognising when two images show the same place. A selection of these schemes, and their use for robot localisation, are described in Section 5.2. Particularly efficient and effective are those schemes based on the Bag-of-Words (BoW) algorithm, described in Section 5.3. Our new BoW scheme for robot localisation, based on these existing schemes, is described in Section 5.4, and validation results for the new scheme are presented in Section 5.5.

5.2 Feature-based location recognition

This section describes contemporary schemes to detect when two images show the same location. These schemes all build on the image feature detection and description techniques discussed in the previous chapter. Two images with many similar-looking descriptors in common are likely to show the same objects, and hence are likely to show the same location. This allows a robot to be localised by extracting descriptors from an image it captures, then searching all previous images for an image containing many similar features.

Finding matching features is not always sufficient for reliable loop closure detection however, as similar-looking objects occur in different locations. Possible matches can be validated by considering the 3D structure of the scenes: once two matching images are found, a geometric test may be used to determine whether a significant proportion of these matched features could have arisen from the same static 3D scene.

In the case of a robot moving on a plane, that visits approximately the same position in the environment every time it visits that location, a simple geometric test is that correctly matched features will have the same configuration in matching images. This works well in the case of a robot moving through a densely-mapped environment, for example a car driving along road which has been mapped at regular intervals (Cummins and Newman, 2009).

For a camera moving in three dimensions, matched frames could contain features in a wide range of configurations. These configurations are constrained by the epipolar geometry of the scene, which is defined by the relative pose of the camera when the images were captured (and is discussed in detail in the next chapter). Methods for computing the epipolar geometry of the scene provide a strong constraint for validating detected loop closure events: if two frames have many features in common, and the relative pose of two cameras can be computed from these feature matches, then the images are likely to show the same objects and the same 3D structure, so are likely to show the same place.

An alternative way to increase the reliability of feature-based loop closure detection is to require that a sequence of images matches a sequence of previously captured images. This method is used by Newman et al. (2006) to reduce the chance of false loop closure detection when visiting areas with repeating architecture. Their method depends on a robot following a previously-travelled route through a densely mapped environment however, otherwise loop closure events may be missed.

For some applications, if only a small number of previous frames could possibly overlap with the current frame, searching for matches between descriptors extracted from each of these frames is feasible. For example, Mahon et al. (2008), who position an AUV using SLAM, detect loop-closure by matching SURF descriptors from the current image with descriptors from frames around the region in the map where their AUV is believed to be. When sufficient matches are found, and the epipolar geometry of the scenes are compatible, loop closure events are detected. This scheme robustly detects loop closure events in an underwater environment, despite small changes in the scene, for example moving starfish (Johnson-Roberson et al., 2010).

Valgren and Lilienthal (2007) capture images of 40 outdoor locations at five different times of the year with a panoramic camera. SURF descriptors from these images are matched with frames captured as a mobile robot explored the same environments. The robot was correctly positioned up-to 90% of the time without using any geometric constraints, despite large changes in foliage, snow cover, cloud cover and illumination, however the image description and brute-force search takes a few seconds to search for the correct location even in this limited dataset.

Probably the largest-scale demonstration of the use of computer vision for location recognition is IM2GPS (Hays and Efros, 2008): six million GPS-tagged images are harvested from the internet, showing a massive range of locations (often tourist destinations) from around the world. 16% of query images are localised to within 200km of the query image's location; this is considerably higher than would occur by chance. These images are represented by a range of descriptors, including 3D colour histograms, descriptors of texture, SIFT-like systematically-sampled GIST descriptors, and tiny thumbnails, and these descriptors are shown to be complementary. This is not a real-time system however; image description took 15 seconds per image, and each query image was compared to every other image's description (which is not infeasibly slow when low-dimension descriptors are used). This task is somewhat different to most robot localisation tasks; as it involves categorising regions rather than providing precise locations within those regions.

While these schemes demonstrate the potential of computer vision for robot localisation, they are limited by their high computational cost. To effectively identify pairs of images with many descriptors in common, a more efficient approach is to use the BoW algorithm, which is described in the following section.

5.3 The Bag-of-Words algorithm

The Bag-of-Words (BoW) image representation is a concise method to represent an image by the set of feature descriptors that it contains, which allows images to be compared efficiently (Sivic and Zisserman, 2003). Each feature extracted from an image is mapped to its closest match from a quantised 'dictionary' of 'image words'—a discrete set of representative features (also known as a 'codebook' or 'vocabulary'). These image words are analogous to written words, where each describes a particular class of features in an image. Image words might represent 'window corners', 'gravel', 'leaves', for example, and all descriptors arising from one of these features in the world should be mapped to the corresponding image word. Figure 5.1 illustrates how an image is represented as a BoW.

This dictionary of image words is chosen with the aim that the same feature viewed in multiple images should map to the same image word each time, and that different-looking features should map to different

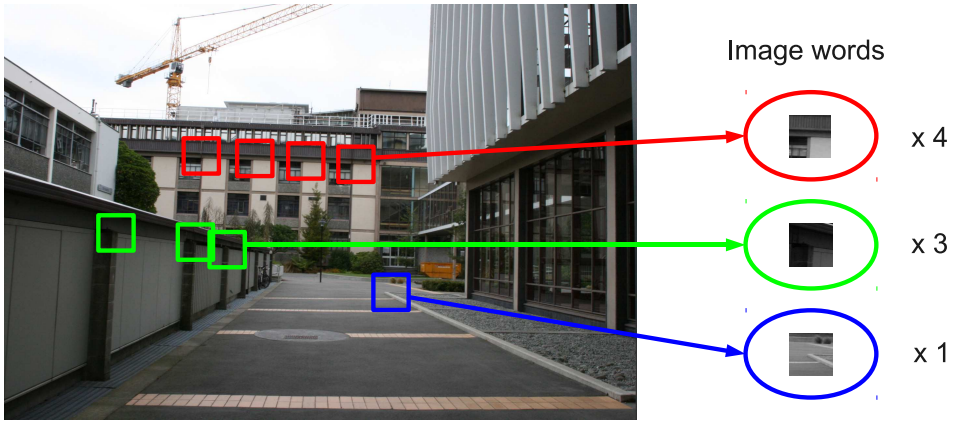


Figure 5.1: To represent an image as a BoW, each descriptor is mapped to the nearest image word (the cluster centres found previously). For each image typically 300 features are mapped to about 150 image words, out of a total of tens or hundreds of thousands.

image words. Two images with many image words in common contain many similar-looking features, and therefore are likely to show the same place. These matches can be found very rapidly as comparing images consists simply of comparing their BoW representations, which are essentially sparse vectors of each word's frequencies.

A suitable dictionary of image words can be found by clustering descriptors extracted from a training set of images, so that each cluster corresponds to a particular class of features which should be described by the same image word. The cluster centres are then used as image words. To cluster the training set into image words an algorithm such as Lloyd's algorithm for k-means clustering is used (as described in more detail in Section 5.4 and Figure 5.6).

Some image words are more useful than others for identifying if two images show the same place (for example many images in the dataset shown later in Figure 5.16 contain image words describing windows and stairs, but only a few contain a vending machine, therefore two images containing words describing the vending machine are likely to show the same place). To take this distinctiveness into account the images' word-frequency vectors can be weighted with a measure of the words' distinctiveness, for example the heuristic Term Frequency-Inverse Document Frequency (TF-IDF) score defined as:

$$w_i = \log \frac{T}{C_i} \tag{5.3.1}$$

where w_i is the weight of image word i calculated from T, the total number of images, and C_i the total number of images where image word i occurs (Nistér and Stewénius, 2006). As the words used, and their TF-IDF weights, are calculated from the training images, it is important that the images in the training set are representative of places the robot will explore.

The most similar images to a query image are found by iterating through the BoW database and comparing each image's weighted word-frequency vector with that of the query image. The images with the most similar weighted word-frequency vectors are most likely to show the same place. Recognition accuracy is high despite geometric information being ignored, and results are robust to small changes in the scene, due to the large number of features that are compared.

The image BoW algorithm was originally developed from document-search techniques by Sivic and Zisserman (2003). Documents with many words in common are likely to describe the same topic, this is analogous to images with many features in common being likely to show the same location. Sivic and Zisserman describe frames from a movie using SIFT, then represent these as a BoW. A user can query this BoW database with

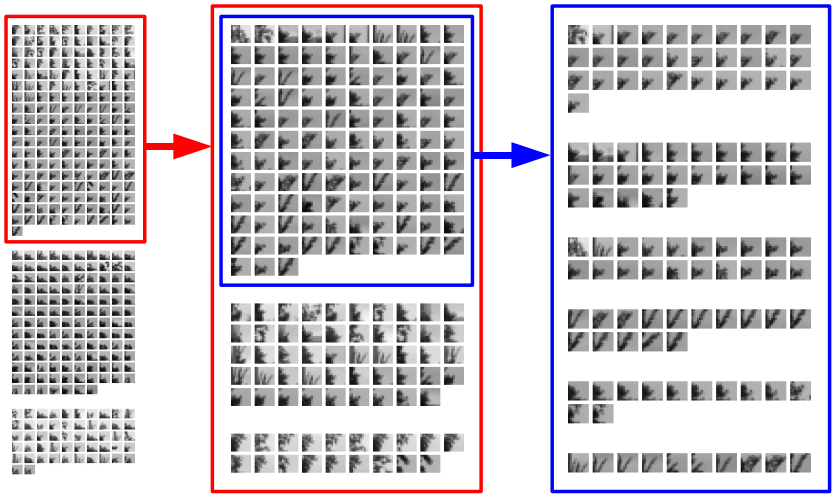


Figure 5.2: Hierarchical clustering: descriptors (image patches in this case) are clustered into a small number of large clusters using a fast approximate clustering algorithm. Each cluster is then clustered again into smaller clusters. Typically this is repeated at about six levels; an example of clusters at the bottom three levels is shown here—the descriptors on the left are clustered into three smaller clusters, each of these smaller clusters is clustered again (the first of these is clustered again into the three smaller clusters shown in the centre), then each cluster is clustered again; the first one gives the six clusters shown on the right. Each of these cluster's centres forms an image word.

an image to find the frames with features in common. This list is retrieved efficiently using an 'inverse file'; a lookup table mapping image words to the frames that contain them, so that the number of comparisons between word vectors are limited. The top matches are refined by requiring that nearby features in the query image are also spatially nearby in the retrieved image. A lists of frames showing the same location as the query frame is then returned.
The dictionary used in many BoW schemes is generated by a k-means clustering of a large subset of the descriptors. Once generated, the most-frequent and least-frequent image words are suppressed; this is analogous to search engines ignoring the most common words in documents, such as 'the', 'a', 'and', which are not useful for characterising a document's contents.

5.3.1 Hierarchical Bag-of-Words dictionaries

A drawback of schemes like that of Sivic and Zisserman (2003), where potentially millions of descriptors are clustered into thousands of image words, is the high cost of clustering, which can take many hours. Once the dictionary is built there is also the problem of efficiently finding the closest image word to a particular descriptor. To solve this second problem Nistér and Stewénius (2006) propose a hierarchical dictionary: the training set is clustered into a small number of image words, then each cluster is recursively clustered into a small number of smaller clusters (Figure 5.2). This is repeated until a sufficient number of image words is found (for example 10^5 image words if descriptors are clustered into ten subsets at each of five levels). Hierarchical clustering is more efficient than clustering once only, as descriptor sets are clustered into a small number of subsets each time. Assigning a descriptor to the closest image word is fast: the closest centre at the top level is found, then the closest centre from its child clusters is found, etc. until the closest image word is found at the bottom level.
The scheme by Nistér and Stewénius (2006) extracts MSER affine-invariant features from each image, then extracts SIFT descriptors oriented with the local gradient at each MSER. The costs of representing frames

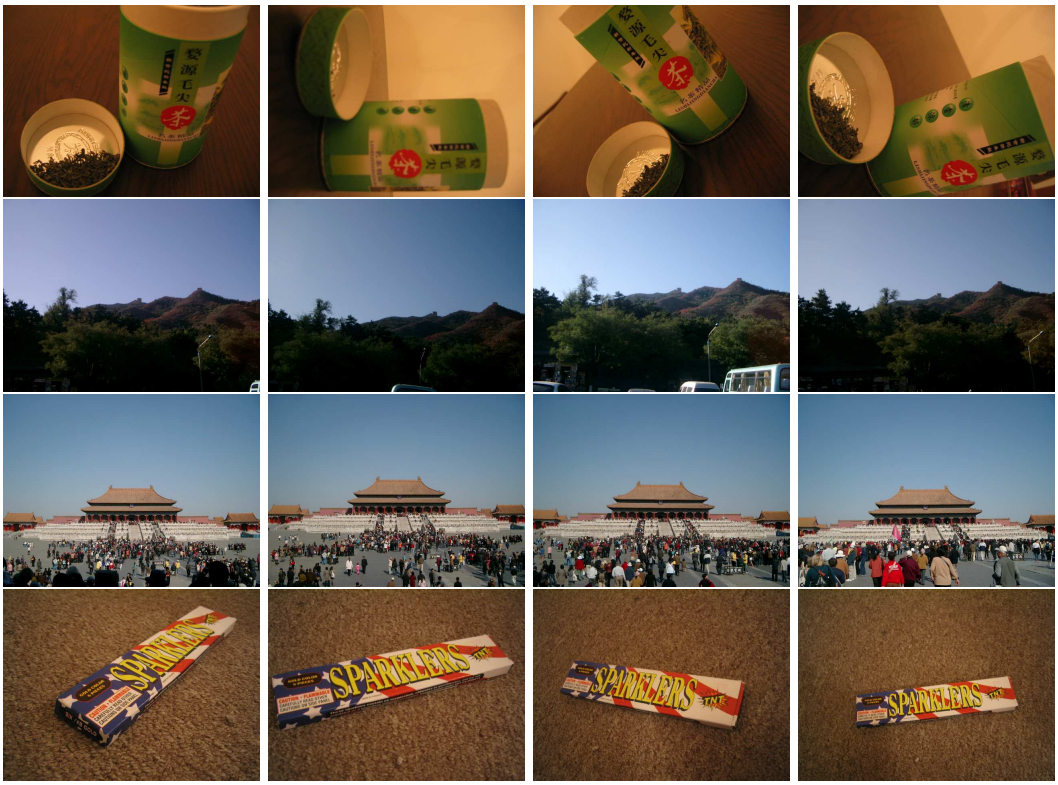


Figure 5.3: Example images from the University of Kentucky dataset for validating the performance of BoW schemes, from Nistér and Stewénius (2006). The database shows four images of each of 2550 objects or scenes. Given one of the images, a good BoW scheme will return all four matches, so the average number of correct matches in the top 4 is a good measure of a scheme's performance.

as BoWs is negligible compared with the cost of extracting these features. To find matches to a query image, the sum of absolute differences (SAD) distance between TF-IDF-weighted vectors is used, with an inverse file used to speed up queries. To test this system, a database of four images of each of 1594 objects is used (now forming part of the University of Kentucky dataset shown in Figure 5.3). With each image used in turn as a query image, the proportion of times that the top four matches returned are correct is used to measure retrieval performance. This dataset is used to demonstrate that increasing the number of words increases recognition performance, with dictionaries of up to 10 million words tested. A higher branch factor (fewer levels in the hierarchical dictionary) results in slightly better performance, but at the expense of increasing the costs of indexing images. The system is also demonstrated recognising images from a database of one million frames, demonstrating that schemes like this are scalable to any environment a robot is likely to need to localise itself in. The stereo camera SLAM scheme by Konolige et al. (2009) uses a hierarchical BoW scheme based on this scheme to reliably close loops in trajectories of several kilometres, in indoor and outdoor environments; a geometric test is used to reject false matches.

To avoid the drop in performance from quantising the dictionary at several levels, Jegou et al. (2008) cluster a training set of descriptors into 200,000 words, then cluster the centres from the first clustering into a smaller number of centres. This is repeated several times, resulting in a hierarchical dictionary which is built from the bottom up. When images are indexed, borderline features are mapped to multiple image words, so that matches between features which happen to fall in neighbouring clusters are not missed. Clustering is considerably more costly, however images can be indexed with comparable speed. Performance on the University of Kentucky dataset (Figure 5.3) is slightly lower than Nistér and Stewénius (2006), however the

scheme outperforms other schemes on a building identification task (Philbin et al., 2007).

5.3.2 Bag-of-Words schemes for image categorisation and object recognition

The BoW model is frequently-used for the related problem of scene categorisation (e.g. Li and Perona, 2005; Csurka et al., 2004). These schemes apply machine learning techniques (including LDA, described in Section 9.2.1) to image's BoW representations in order to identify patterns of co-occurring features which define a small number of categories of scene (e.g. 'mountains' or 'forest'). While not directly related to robot localisation, these techniques do demonstrate how probabilities can be calculated from BoW-represented images. Both schemes try using naïve Bayes classifiers to estimate the probability of observing a particular set of words; the naïve Bayes assumption is that word occurrences are independent. The probability of each word occurring if an image belongs to a category is estimated from the number of times the word occurs in training images from that category. By Bayes theorem, the probability of an image belonging to a category is proportional to the product of the probabilities of observing each word. As the set of features observed in an image, or in training data, is generally incomplete (as features which could be observed will often be missed) any Bayes updates must be damped, for example by Laplace smoothing (Csurka et al., 2004) or a similar technique (Cummins and Newman, 2008a), and negative observations (not observing a particular word) are ignored. These precautions prevent the probabilities of images showing the same place from growing to 1, if an image contains a word found in all training data, or 0, if an image contains a word not found in any training data, or if the training data contains a word not found in the image.
In addition to a Bayes classifier, Csurka et al. (2004) try using a Support Vector Machine (SVM) to categorise images BoW representations, which are treated as vectors. SVMs are machine-learning techniques suitable for partitioning sets of high-dimensional vectors into different clusters, maximising the dissimilarity between clusters. While not really suitable for robot localisation (the SVM would have to be trained for each of the large number of images a robot may observe), the higher performance of this heuristic method, compared to the much-cheaper naïve Bayes classifier, does indicate that the damped estimates of word-occurrence probabilities inaccurately reflect their true values, or that word co-occurrences are important. Both schemes use k-means-clustered dictionaries limited to just a few hundred or thousand words, in order to limit computational costs, even though small improvements in performance come with increasing the number of words. Li and Perona (2005) compare the performance of image patch descriptors and SIFT descriptors, and conclude that when sampled systematically from the image, both have comparable performance, although with just 156 words, the clusters of patches, or of 128D SIFT descriptors, are unlikely to reflect the true range of features from the training sets.
Uijlings et al. (2009) describe a BoW scheme using k-means to cluster SURF descriptors, and find an SVM to perform well for detecting object instances within scenes (for example 'car' or 'chair'). They observe that SURF descriptors are considerably cheaper to compute on a systematic grid, as a pyramid of integral images can be used, and that this systematic sampling performs well for categorisation. Their scheme won the 2008 PASCAL Visual Object Class recognition challenge (Everingham et al., 2008), for identifying instances of one of twenty classes of objects in images. For this task, a relatively small dictionary of 4096 descriptors was sufficient.

5.3.3 Bag-of-Words schemes for robot localisation

The BoW algorithm's ability to allow large image databases to be searched has since been used widely for robot localisation. One such scheme is described by Cummins and Newman (2008a). SURF descriptors extracted from omnidirectional images are k-means clustered into around 11000 words, and a kd-tree (described in Section 4.5.6) is used to match descriptors to these image words. Given a query image the probability of a correct match with each frame is computed. Calculating these probabilities using a naïve Bayes assumption that word co-occurrences are independent is simple and reasonably effective, but slightly better results are given by considering the strongest co-occurrences between image words. From the training data a complete directed graph of the co-occurrence probabilities of all the image words is constructed and the (approximate) maximum-weight spanning tree (Chow Liu tree) of this graph is found. The prior

probability of observing a particular distribution of features anywhere can now be calculated approximately by accumulating the probabilities of observing each image word, conditional on whether its descendent in the Chow Lie tree occurs. These probabilities appear more reflective of the true probability of observing a particular bag-of-words than those calculated using a naïve Bayes assumption. The probability of observing a particular word-bag conditional on a previously viewed scene from the BoW database being viewed can then be computed for each location using a naïve Bayes independence assumption. From these probabilities, Bayes theorem gives the probability of the query image showing the same place as the database image.

The scheme by Cummins and Newman (2008a) is tested on loops of a few kilometres; at each location the image most probable to show the same location as the current image is found, subject to this probability exceeding some threshold. When under 30% of images are matched, the false-positive rate is zero on the datasets which are presented, indicating the power of the BoW model even when no geometric information is taken into account.

Cummins and Newman (2009) describe an optimised version of their previous scheme, which uses a inverse file in order to minimise the number of candidate matches to consider. This scheme is applied on an impressively large scale: loop closure events are accurately detected in a database of over 100,000 frames collected over hundreds of kilometres, captured from a panoramic camera mounted on a car. A dictionary of 100,000 words is shown to significantly outperform a smaller dictionary. A geometric consistency check is used to validate that matched images show the same 3D structure; an assumption that the car is in approximately the same location is used. 8% of images are matched, with 1% of these matches false positives, however the distribution of these matches is not uniform, with no loop closure events detected on one 50km stretch of road when illumination conditions changed between two visits. This scheme is used to successfully detect loop closure events in the stereo camera SLAM scheme by Sibley et al. (2010) over tracks up to 121km in length, and on a smaller scale by Newman et al. (2009).

The schemes described so far all build dictionaries off-line, usually by k-means clustering of descriptors from a training set (Nistér and Stewénius, 2006; Newman et al., 2006; Sivic and Zisserman, 2003; Li and Perona, 2005). Finding a training set representative of all environments a robot will explore may not be possible however, and there is always a risk that a robot may enter an environment containing features which are not represented by the training set. An alternative approach is to dynamically construct a dictionary from the features which are encountered as the environment is explored, so that environments with features not represented in the training set can be recognised effectively. Such a scheme is described by Filliat (2007): when features are observed which differ significantly from any previously-observed features, these features are added as new words. This scheme works well on the scale of a few rooms in a building, but performance is limited by the continual dictionary growth. Angeli et al. (2008a) extend this scheme to add incrementally-calculated TF-IDF weights, to use an inverse file to improve query times, and to estimate the probability that the robot is exploring a new region. This system closes small loops indoors, although real-time (1Hz) performance is limited to about 2000 images.

Similar schemes by Eade and Drummond (2008) and Nicosevici and Garca (2009) also initialise new words dynamically and add them to a dictionary, although in all of these cases the number of words used is of the order of a thousand; orders of magnitude less than schemes working on larger scales (Nistér and Stewénius, 2006; Sivic and Zisserman, 2003; Cummins and Newman, 2009)

SIFT and SURF are the most popular descriptors for BoW scene recognition, used, for example, by Nistér and Stewénius (2006); Newman et al. (2006); Cummins and Newman (2008b); Angeli et al. (2008a,b); Csurka et al. (2004); Li and Perona (2005); Filliat (2007); Eade and Drummond (2008); Nicosevici and Garca (2009); these SIFT and SURF descriptors should allow the recognition of scenes independently of the robot's viewpoint. When these schemes are used for robot localisation they typically index images at around 1Hz (Cummins and Newman, 2008b; Angeli et al., 2008b); this framerate allows dense mapping of the environment, while limiting the cost of extracting SIFT or SURF features. When used for loop closure detection in single camera SLAM, BoW is usually run in isolation on a subset of frames (Eade and Drummond, 2008; Newman et al., 2006, 2009), with the occasional position update provided when a good loop closure candidate is found. Strong viewpoint invariance is not always necessary for recognition however, as often a robot will visit a location which is very close to where it was previously. Cummins and Newman (2008b) impose a geometric

constraint which is only valid when the robot is near to its previous location, and Li and Perona (2005) show that simple image patches with little viewpoint-invariance have comparable performance to SIFT features for characterising images, even when the scales of objects in images within categories clearly vary.

5.3.4 Summary of scene-recognition research

Section 5.3.3 described several schemes which demonstrate the power of cameras as robot localisation sensors. No prior knowledge of a robot's position is needed to reliably recognise a region it has seen earlier. Of these schemes, those using tens or hundreds of thousands of image words are the most successful at scaling to large environments, and the most feasible way to build these large vocabularies is to use hierarchical clustering. Matches found by these schemes are occasionally incorrect, however by filtering results to check that the geometry of matched scenes is compatible, these errors can usually be eliminated. A limitation of these large scale schemes however is that they will only work well when training data is representative of the environments being explored. Alternative approaches attempt to build dictionaries incrementally, however these are limited to considerably smaller dictionaries with around a thousand words.
While the previous robot localisation schemes discussed use SIFT or SURF descriptors centred on blobs, schemes for categorisation and image search applications indicate that systematically sampling features and describing them with a range of descriptors of shape and colour may give better performance.
Some schemes explicitly estimate the loop closure probability (Angeli et al., 2008a; Cummins and Newman, 2008a, 2009), although it is not clear how to accurately estimate word occurrence probabilities. Other schemes achieve high levels of accuracy with heuristic measures of relative image similarity (Newman et al., 2006; Nistér and Stewénius, 2006; Filliat, 2007).

5.4 A new BoW scheme for robot localisation

5.4.1 Requirements

In this section, a new BoW scheme for robot localisation is developed, which is based on the schemes described in the previous section. The BoW scheme allows large dictionaries to be constructed as the robot explores, so that robots can learn to recognise large environments without needing appropriate training data. Preliminary work on this scheme was described by Botterill et al. (2008).
The scheme is developed to meet the following requirements:

- Fast enough for extended real-time operation.
- Large enough vocabulary for robust recognition in large environments.
- No requirement for appropriate training data to be found for each environment which is to be explored.

The scheme best satisfying the first two criteria is the scheme by Nistér and Stewénius (2006), due to the hierarchical vocabulary (Figure 5.2), which supports a large dictionary while enabling fast and efficient mapping of features to the closest image word. A limitation of this scheme is the cost of extracting invariant descriptors, which may be necessary to match widely spaced views, however image patches are much cheaper to extract, and perform as well in many circumstances (as shown in Section 4.5.5).
Several schemes have been proposed which satisfy the third criteria (Eade and Drummond, 2008; Nicosevici and Garca, 2009; Filliat, 2007; Angeli et al., 2008a), however these all build small non-hierarchical dictionaries. Ideally, a large hierarchical dictionary would be built incrementally. Clustering schemes used for building large dictionaries are too slow for online use however, typically taking several hours or days (Nistér and Stewénius, 2006; Cummins and Newman, 2008a; Sivic and Zisserman, 2003).

Table 5.1: Notation used in this chapter.

N	Number of descriptors to be clustered
k	Number of clusters (branch factor in hierarchical dictionary)
D	Descriptor length
I	Number of Lloyd's algorithm/PAM iterations
C	Number of CLARA iterations
S	CLARA subset size
$\delta(i,j)$	Arbitrary distance function between descriptors i and j (a measure of their similarity)
$\delta_{\text{SSD}}(i,j)$	Sum of squared differences between descriptors i and j (a measure of their similarity)
a_i	Index of closest cluster centre to descriptor i
$c_1, ...c_k$	k cluster centres
f	Retraining occurs when the total number of descriptors increases by a factor f

5.4.2 Design of a BoW scheme supporting dynamic retraining

This section describes our new BoW scheme, which is based on the scheme by Nistér and Stewénius (2006), but creates an appropriate dictionary dynamically from the data which is observed, rather than offline from training data. Notation used is summarised in Table 5.1.

The simple method used to create a dictionary dynamically is to periodically cluster all of the descriptors which have been observed so far into a new dictionary, which replaces the old dictionary. Lloyd's algorithm for k-means clustering would be too slow for retraining dynamically in this way, so a faster clustering scheme is required.

In order to speed up the creation of the dictionary, we use the observation that when creating a hierarchical dictionary, the costs of clustering are dominated by clustering large number of descriptors (thousands or millions) in to a small number of centres (e.g. ten or less). In this case, centres chosen by clustering a small random subset of the descriptors, for example less than one hundred, usually define suitable clusters for the entire set (Kaufman and Rousseeuw, 1990, illustrated in Figure 5.4). This observation reduces the time needed to generate an effective dictionary to just a few seconds or minutes, allowing new dictionaries to be created dynamically. For robot localisation the clustering takes place in a separate thread. When a new dictionary and BoW representations are created, the previous dictionary is seamlessly replaced.

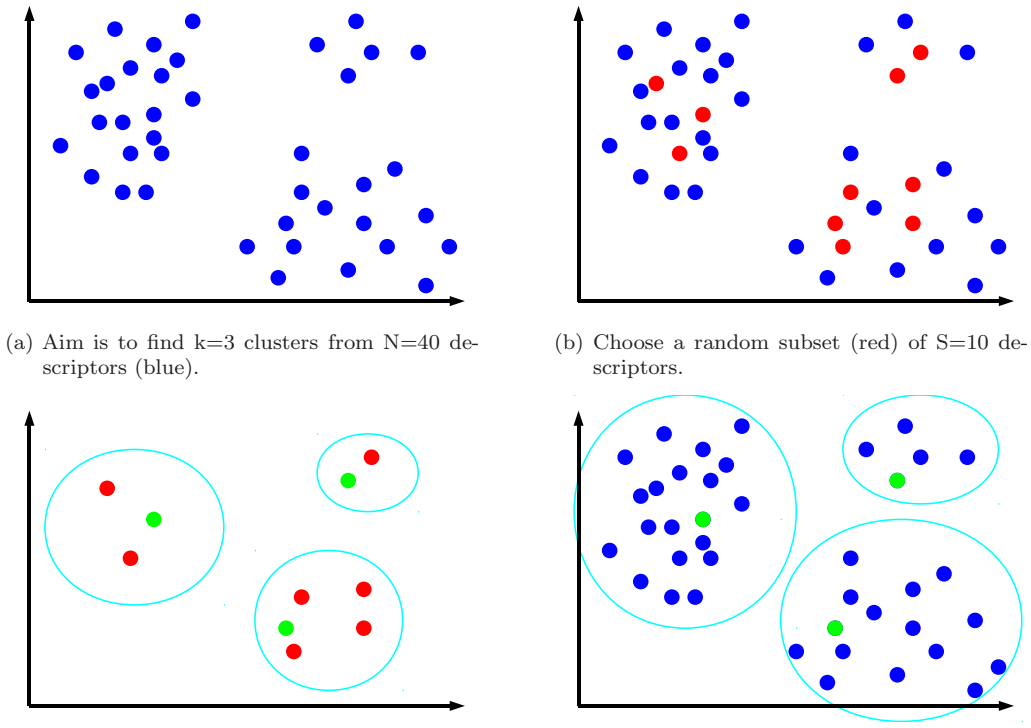
The CLARA algorithm (Clustering LARge Applications, Kaufman and Rousseeuw, 1990) is a simple scheme for clustering large sets in this way, outlined in Figure 5.5. Several small random subsets are each clustered into k clusters. Of these clusterings, the best clustering for the whole set is chosen. This avoids the occasional poor clusterings that can occur when clustering subsets.

Many previous schemes use k-means clustering to generate dictionaries, and this is suitable for clustering subsets as well. However, as only small subsets are being clustered, the computational cost of clustering is dominated by assigning descriptors to cluster centres, rather than by the clustering algorithm used to cluster subsets. In this case any clustering algorithm can be used.

Given N vector-descriptors, $v_1, ..., v_N$, a k-means clustering algorithm aims to find k vectors (cluster centres; $c_1, ..., c_k$) which minimise the total squared Euclidean distance between descriptors and their closest centres. This is referred to as the residual; a high residual indicates a poor clustering.

$$\text{Residual} = \sum_{i=1}^{N} \|v_i - c_{a_i}\|_2^2 \tag{5.4.1}$$

where $a_i \in 1, ..., k$ is the index of the closest centre to vector v_i. A popular algorithm for k-means clustering is Lloyd's algorithm, a simple version of which is outlined in Figure 5.6. The simplest version of Lloyd's algorithm has complexity $O(DIkN)$ where D is the cost of comparing two descriptors (their length), and I is the number of iterations (in practice low, but at worst exponential in N; Vattani, 2009). Lloyd's algorithm



(a) Aim is to find k=3 clusters from N=40 descriptors (blue).

(b) Choose a random subset (red) of S=10 descriptors.

(c) Cluster random subset into k=3 clusters (cluster centres are marked green).

(d) These k=3 centres are often good centres for clustering the entire dataset (the clusters they define are circled).

Figure 5.4: Illustration of one iteration of CLARA clustering: 40 points in 2D space are clustered into 3 clusters by clustering a subset of size 10. When a large number of points are being clustered into a small number of clusters, clustering a small random subset is likely to give a good overall result. Repeating this with another random subset (including the centres found so far) often improves the clustering.

Input: N descriptors, k: number of centres, C: number of CLARA iterations, S: subset size. $k \ll S$. Ideally $S \ll N$

Output: $\{c_1, c_2, ..., c_k\}$: List of k descriptors forming good cluster centres

```
Randomly choose S descriptors:
    Cluster these S descriptors into k clusters

Repeat C times:
    Randomly choose S-k descriptors:
        Cluster S-k descriptors, plus the k best centres found so far, into k clusters
        Remember centres with lowest total residual so far

Return centres with lowest total residual so far
```

Figure 5.5: CLARA: Clustering for LARge Applications algorithm for clustering a large number (N) of descriptors into a small number of centres, k.

Input: N vector descriptors, k: number of centres I: number of k-means iterations
Output: $c_1, ..., c_k$: Set of k vectors forming good cluster centres

```
Choose k initial cluster centres (e.g. randomly)

Repeat for I iterations or until no change in assignment:
    Assign every descriptor to closest centre
    Replace each centre with mean of descriptors assigned to it
```

Figure 5.6: Lloyd's algorithm for k-means clustering (many variations exist)

is fast in practice, but can converge to false minima (Kanungo et al., 2002), particularly with a poor choice of initial centres. 'Swapping heuristics' where some cluster centres are moved to where centres appear to be missing can reduce this problem, but current algorithms still converge in exponential time in the worst case (Kanungo et al., 2002). A further limitation of k-means clustering is that the residual which is minimised is the sum of squared differences (SSD), which is not necessarily a good metric with which to compare descriptors, in particular histograms. A wide range of distance measures are used to compare histograms, many of which are described by Shahrokni (2004); Brunelli and Mich (2001). Of these, one of the most popular is the Chi-squared distance, used by Filliat (2007) and Hays and Efros (2008) for example, which is an estimate of the probability that two histograms represent data drawn from the same distribution.
One algorithm addressing the problem of appropriate distance measures, and including a systematic swapping heuristic to ensure all points are considered as centres, is the Partitioning About Medoids, or PAM, algorithm for k-medoids clustering, proposed by Kaufman and Rousseeuw (1990). A k-medoids clustering algorithm finds a subset of k of the N descriptors which form cluster centres. The PAM algorithm chooses k centres which minimises the following residual:

$$\text{Residual} = \sum_{i=1}^{N} \delta(i, a_i) \tag{5.4.2}$$

for any measure of distance, δ, where $a_i \in 1, ..., N$ is the index of the closest centre to descriptor i. The PAM algorithm, outlined in Figure 5.7 and illustrated in Figure 5.8, first computes all pairwise distances between descriptors, then makes an initial guess for the cluster centres. This initial guess is refined by finding pairs of centres and non-centres which lower the total residual when swapped. The swapping heuristic works well in practice; typically few iterations are needed (although worst-case performance is unknown). Each time the ith centre is swapped, the current k-subset of centres is replaced by the k-subset with the lowest residual out of any k-subsets which can be constructed by replacing centre i with a different descriptor. Although PAM's complexity of $O(IkN^2 + DN^2)$ is higher in N than Lloyd's algorithm, it is still feasible for subsets of thousands of descriptors, which is larger than is needed (Section 5.4.4).
Kaufman and Rousseeuw (1990) suggest than only descriptors forming a metric space can be k-medoids clustered, although the algorithm they describe will find a clustering for any distance function[1]. Every time the set of centres is updated the total residual decreases, and as there are a finite number of sets of centres the algorithm will always converge to a minimum (although possibly a local minimum).
Note that clustering may not be necessary to generate a dictionary; Nowak et al. (2006) found that a random selection of descriptors may be used as a BoW dictionary. A random set of descriptors provides a suitable benchmark for verifying that clustering is worthwhile at all.
Other large scale BoW schemes (Cummins and Newman, 2009; Nistér and Stewénius, 2006; Sivic and Zisserman, 2003) use an inverse file to speed up queries. This is a lookup table used to map words in a query image to a list of images containing that word, minimising the number which need to be searched. Managing the memory use of inverse files is challenging however, and sparse vector comparisons between dissimilar

[1]If the distance function ever takes negative values then the centres found may not be distinct, i.e. there may be fewer than k centres found.

Input: N descriptors, k: number of centres, I: number of k-medoids iterations
Output: $\{c_1, c_2, ..., c_k\} \subseteq \{1, ..., N\}$: set of k of the descriptors forming good cluster centres

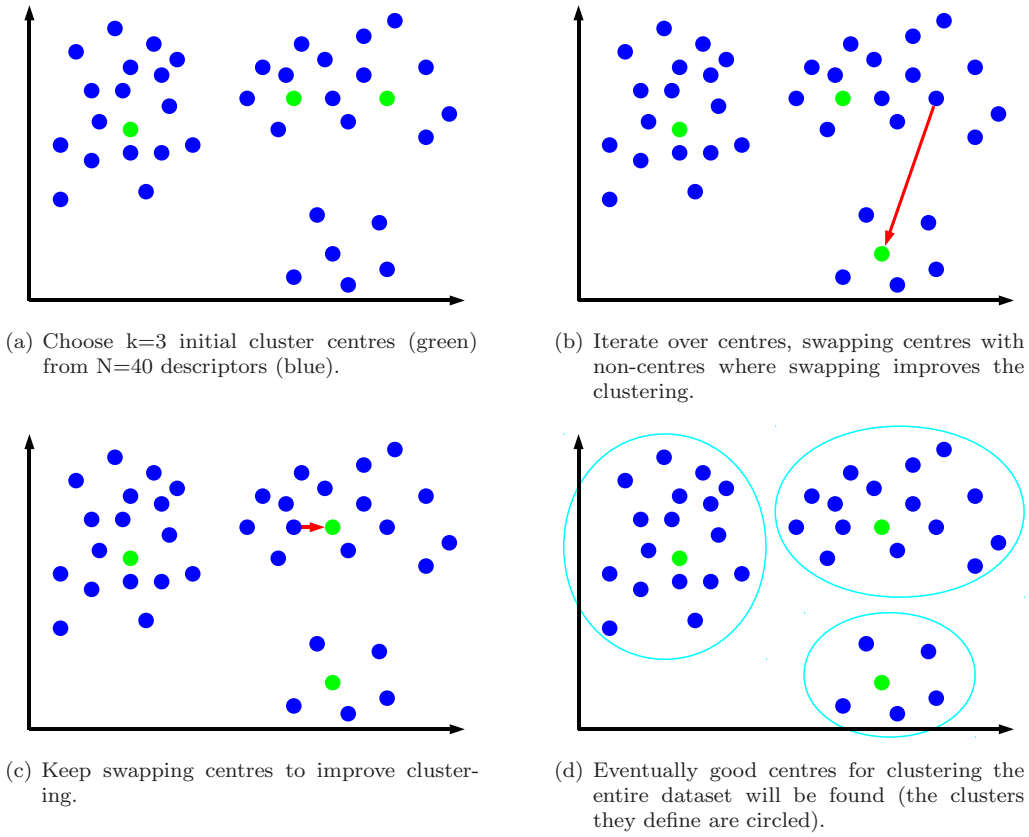
```
// Precompute distances
For all pairs of descriptors i,j in 1, ..., N:
    Compute and store distances d(i,j)

// Add initial k centres one at a time
Repeat k times:
    Iterate over all descriptors to find point i where the reduction
      in residual from adding i as a centre is greatest
    Add i as a centre
    Assign each descriptor to closest centre

// Refine centres to lower residual
Repeat for I iterations or until no change:
    Consider moving each centre. For each centre c:
        Iterate over all descriptors which are not centres to find point i where the
          reduction in residual from replacing centre c with descriptor i is greatest
        If swapping i and c would reduce the residual then perform the swap
        Assign each descriptor to closest centre
```

Figure 5.7: Partitioning Around Medoids (PAM) algorithm for k-medoids clustering.

BoW representations are fast (and in practice can be stopped early if few matches are found). We have not implemented an inverse file as queries are currently considerably faster than retraining costs, and rebuilding the inverse file when retraining occurs would add to these retraining costs. If the cost of searching the entire database became too great however, an inverse file could be implemented. Alternative possibilities to speed-up queries include using the scheme by Fraundorfer et al. (2007) to cluster bags-of-words by similarity, allowing searches to be limiting to a small cluster of similar scenes, or to search a set of whole-image descriptors as a first-pass measure of similarity (Hays and Efros, 2008).

5.4.3 Choosing k

The branch-factor of the hierarchical dictionary, k, and the number of levels, determines the eventual number of image words found. Ideally each image word should correspond to one class of feature in the environment, however in practice similar-looking descriptors are sometimes split between image words, especially when viewed from different viewpoints, and sometimes one image word describes multiple classes of feature. Currently, one word for every 20-30 descriptors for the first few thousand frames (up-to a million descriptors) is a suitable heuristic, although the parameter sensitivity is low. This parameterisation is verified by inspecting the clusters found, with the aim that each cluster corresponds to a different class of feature (Figures 4.1 and 5.2 for example). Typically five to eight levels in the hierarchical dictionary are used, using more degrades the performance.
Theoretically, the number of words needed should increase more slowly as the robot explores, or as the robot re-visits known environments, however additional words may be needed to provide sufficient distinctiveness as more similar-looking environments are explored. In future we plan to estimate the numbers of words needed by examining how efficiently new frames are being described.



(a) Choose k=3 initial cluster centres (green) from N=40 descriptors (blue).

(b) Iterate over centres, swapping centres with non-centres where swapping improves the clustering.

(c) Keep swapping centres to improve clustering.

(d) Eventually good centres for clustering the entire dataset will be found (the clusters they define are circled).

Figure 5.8: Illustration of PAM k-medoids clustering: 40 points in 2D space are clustered into 3 clusters.

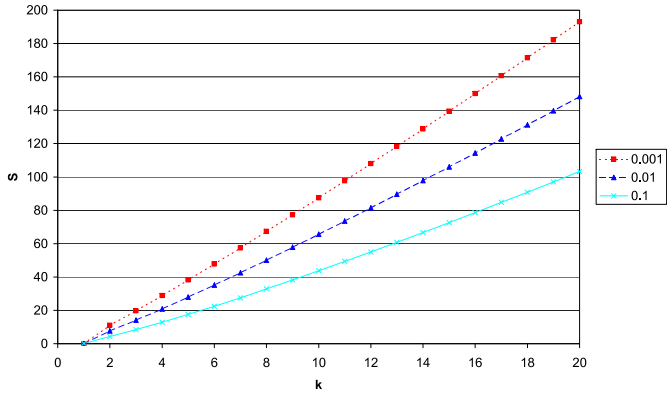


Figure 5.9: Subset sizes necessary to ensure the probability of missing a cluster is less than ϵ, $\epsilon = 0.001, 0.01, 0.1$. This indicates that only small subsets need to be clustered. Lines all converge to $k \log k$ as $k \to \infty$.

5.4.4 Choosing subset sizes

When choosing subsets of descriptors to cluster, the probability that the subset does not include one of the clusters found in the underlying data should be minimised. We assume that a suitable number of clusters, k, is known.

If each observed descriptor is assumed to fall into one of k feature classes, and that the probability of an observed descriptor falling into any one feature-class is $1/k$, then the probability of there being no descriptors in the first feature-class in a random subset of size S is $(1 - \frac{1}{k})^S$. Therefore the probability that a subset of size S misses at least one feature-class is:

$$P(\text{ 1 or more feature-classes missed }) < k(1 - \frac{1}{k})^S \tag{5.4.3}$$

(with '$<$' because probabilities are not disjoint; multiple clusters may be missed). A value of S that ensures this probability is less than a threshold, ϵ, is given by:

$$S = \frac{\log \epsilon - \log k}{\log(1 - \frac{1}{k})} \tag{5.4.4}$$

Examples of suitable values of S for different k are shown in Figure 5.9.

This is a best-case analysis, as the distributions of feature-classes will vary in practice, however it indicates that the probability of missing clusters by clustering only subsets is small, particularly for small k. The probability of these clusterings which miss centres being used is even lower if multiple CLARA iterations are used to select the best of several clusterings (Figure 5.5).

5.4.5 Complexity analysis of online re-training

While completely rebuilding a dictionary periodically may appear inefficient, in fact it has the lowest complexity possible for a scheme building a hierarchical dictionary incrementally, and has lower complexity than previous proposed schemes. This section proves this result.

If N descriptors in total are observed, and descriptors are clustered each time the number of descriptors increases by a factor of f, then rebuilding will have occured at most $\log_f N$ times, with f^i descriptors at the ith time.

We assume that $\log_A n$ image words are sufficient to describe n descriptors, for some constant A. The ith time clustering occurs, we cluster f^i descriptors into $\log_A(f^i)$ words.

Next we assume a balanced tree is built with branch-factor k. The ith time clustering occurs, a descriptor can be mapped to the correct word with $k \log_k \log_A(f^i)$ comparisons[2], so $f^i k \log_k \log_A(f^i)$ comparisons are required to map every descriptor to a word each time clustering occurs. Clustering occurs approximately $\log_A(f^i)/k$ times, with clustering subsets requiring $C(k)$ operations for some function C (i.e. constant time when the branch-factor is fixed).
When N descriptors have been observed, the total number of comparisons needed is:

$$\sum_{i=2}^{\log_f N} \left[f^i k \log_k \log_A(f^i) + C(k) \log_A(f^i)/k \right] \tag{5.4.5}$$

$$= \sum_{i=2}^{\log_f N} \left[f^i k \log_k i + f^i k \log_k \log_A f + C(k) \log_A(f^i)/k \right] \tag{5.4.6}$$

As A and k are fixed, the first term dominates. Applying the following result:

$$\sum_{i=1}^{n} \log(i)^c \cdot i^d \cdot b^i = \Theta(n^d \cdot \log(n)^c \cdot b^n) \tag{5.4.7}$$

from Summation; Wikipedia (2010), verified with the symbolic algebra software package Maple (n.d.), the total number of comparisons has complexity:

$$\Theta(f^{\log_f N} \log \log N) \tag{5.4.8}$$

$$= \Theta(N \log \log N) \tag{5.4.9}$$

For comparison with our scheme, we consider a scheme building a hierarchical dictionary incrementally, so that every descriptor is either mapped to an existing word, or added to a new word, with the same assumption that N descriptors can be described with $\log_A N$ words. If a balanced tree with branch-factor k is built incrementally, with $\log_A j$ words when descriptor j is added, then $k \log_k \log_A j$ comparisons are needed to add the jth descriptor. This gives the total number of descriptor comparisons as:

$$\sum_{j=2}^{N} k \log_k \log_A j \tag{5.4.10}$$

$$= \Theta(N \log \log N) \text{ (for fixed } k \text{ and } A) \tag{5.4.11}$$

Remarkably, the complexity of our scheme equals the complexity of the theoretically most efficient scheme for incrementally building a hierarchical dictionary. In practice, contemporary schemes building dictionaries incrementally use simple single-level dictionaries, with complexity which is potentially worse than this case (Filliat, 2007; Angeli et al., 2008a; Eade and Drummond, 2008; Nicosevici and Garca, 2009), although if a balanced kd-tree was built incrementally, and used to match descriptors to words, then the problem is very similar to the theoretically most efficient scheme described.
The total cost of our scheme is also comparable with any other schemes building a dictionary incrementally, because the total cost after observing N descriptors is dominated by the most recent clustering event. Asymptotically, the cost of the most recent event, compared to all previous clustering events (those before time N/f), is given by:

$$\lim_{N \to \infty} \frac{N \log \log N - \frac{N}{f} \log \log \frac{N}{f}}{N \log \log N} = \frac{f-1}{f} \tag{5.4.12}$$

For a typical value for f, for example $f = 1.5$, our scheme can be made no-worse than three times more costly than any other method which involves building a hierarchical dictionary.
This analysis assumed that the number of words was logarithmic in the number of descriptors. Alternatively, the worst possible case is that the number of words necessary is proportional to the number of descriptors. In this worst possible case, both our scheme, and the theoretically most efficient scheme, have complexity $\Theta(N \log N)$.

[2] A balanced tree with L leaves, and branch factor k, has height $\log_k L$. Inserting a descriptor into this tree requires $k \log_k L$ comparisons.

A drawback of our scheme however is that novel words may be introduced somewhat later than when they are first encountered; a word encountered at time t may not be added until time ft. A further drawback to the current implementation is that all descriptors are stored, and these dominate the memory required (and also BoWSLAM's memory requirements). This is not a limitation of the approach however, as descriptors could be merged when similar, reducing costs. Storing descriptors from a full day of exploration in memory is within the capabilities of a modern computer; for example, if 200,000 images are each described with 250 64D SURF descriptors, then 3GB of memory is needed.

5.4.6 Implementation details

As stated earlier, assigning descriptors to clusters is the most costly part of rebuilding a dictionary. Several optimisations allow this to be speeded up; in this section we describe one important optimisation. The cost of assigning descriptors to clusters is dominated by the cost of computing the distance between pairs of descriptors. This optimisation reduces the number of comparisons needed to assign a descriptors to the closest of k centres.

When descriptors belong to a metric space, the distances between cluster centres can be used to speed up the assignment of a descriptor to its nearest centre. A set of descriptors, with an associated distance function δ, form a metric space if and only if δ satisfies the following properties:

$$\begin{aligned} \delta(x,x) &= 0 \,\forall\, x && (5.4.13)\\ \delta(x,y) &> 0 \,\forall\, x,y \text{ where } x \neq y && (5.4.14)\\ \delta(x,y) &= \delta(y,x) \,\forall\, x,y && (5.4.15)\\ \delta(x,y)+\delta(y,z) &\leq \delta(x,z) \,\forall\, x,y,z \text{ (triangle inequality)} && (5.4.16) \end{aligned}$$

The fourth property, the triangle inequality, allows a descriptor to be matched to a set of cluster centres more efficiently. Conventionally, to match a descriptor to the closest cluster centre requires it to be compared to each centre. When a descriptor is matched to a small set of centres however, the costs of storing the distances between centres is low (and these distances are already available if descriptors have been k-medoids clustered), allowing the following optimisations to be made.

If a descriptor x is distance $\delta(x,c_i)$ from centre c_i, and another centre, c_j, is distance $\delta(c_i,c_j)$ from c_i, then the following test checks if c_j can ever be the closest descriptor to x:

$$\begin{aligned} \text{if} \quad & 2\delta(x,c_i) \leq \delta(c_i,c_j) \text{ then } \delta(x,c_i) \leq \delta(x,c_j) && (5.4.17)\\ \text{because} \quad & \delta(c_i,c_j) \leq \delta(c_i,x)+\delta(x,c_j) \text{ (by triangle inequality)} && (5.4.18) \end{aligned}$$

Therefore, if $2\delta(x,c_i) \leq \delta(c_i,c_j)$ then the distance $\delta(x,c_j)$ does not need to be computed, as c_j must be further from x than c_i (Figure 5.10).

A slightly stronger test is possible when we have already found a centre c_k which is closer to x. In this case:

$$\begin{aligned} \text{if} \quad & \delta(x,c_i)+\delta(x,c_k) \leq \delta(c_i,c_j) \text{ then } \delta(x,c_k) \leq \delta(x,c_j) && (5.4.19)\\ \text{because} \quad & \delta(x,c_j) \geq \delta(c_i,c_j)-\delta(x,c_i) \geq \delta(x,c_k) \text{ (by triangle inequality)} && (5.4.20) \end{aligned}$$

This test can be adapted for when the SSD, δ_{SSD}, is used to measure descriptor similarity. The SSD does not satisfy the triangle inequality, so is not a metric, but its square root, the Euclidean distance, does. Therefore:

$$\text{if } \sqrt{\delta_{\text{SSD}}(x,c_i)}+\sqrt{\delta_{\text{SSD}}(x,c_k)} \leq \sqrt{\delta_{\text{SSD}}(c_i,c_j)} \text{ then } \delta(x,c_i) \leq \delta(x,c_j) \quad (5.4.21)$$

To avoid evaluating the square root (which is expensive on some systems) the following approximation can be used:

$$\text{if } 2\delta_{\text{SSD}}(x,c_i)+2\delta_{\text{SSD}}(x,c_k) \leq \delta_{\text{SSD}}(c_i,c_j) \text{ then } \delta(x,c_i) \leq \delta(x,c_j) \quad (5.4.22)$$

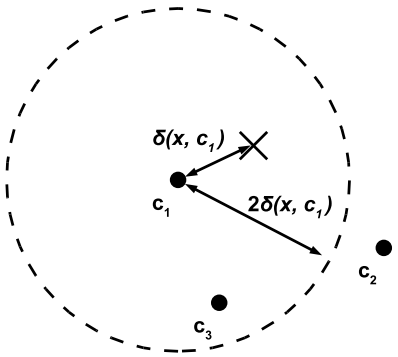


Figure 5.10: When descriptor x (marked $\times$) is measured to be $\delta(x, c_1)$ from a centre c_1, other centres further than $2\delta(x, c_1)$ from c_1 (i.e. centres outside of the dashed circle) cannot be the nearest neighbour of x. In this example c_2 cannot be the closest centre. c_3 could be however.

This approximation is derived using the following inequality:

$$0 \leq (d_1 - d_2)^2 = d_1^2 - 2d_1d_2 + d_2^2 = (d_1 + d_2)^2 - 4d_1d_2 \tag{5.4.23}$$

$$\Rightarrow d_1 + d_2 \geq 2\sqrt{d_1d_2} (\text{assuming } d_1, d_2 \geq 0) \tag{5.4.24}$$

From Equation 5.4.22:

$$\text{if } \delta_{\text{SSD}}(x, c_i) + 2\sqrt{\delta_{\text{SSD}}(x, c_i)\delta_{\text{SSD}}(x, c_k)} + \delta_{\text{SSD}}(x, c_k) \leq \delta_{\text{SSD}}(c_i, c_j) \text{ then } \delta_{\text{SSD}}(x, c_i) \leq \delta_{\text{SSD}}(x, c_j) \tag{5.4.25}$$

Combining this with the inequality $\delta_{\text{SSD}}(x, c_i) + \delta_{\text{SSD}}(x, c_k) \geq 2\sqrt{\delta_{\text{SSD}}(x, c_i)\delta_{\text{SSD}}(x, c_k)}$ gives:

$$\text{if } 2\delta_{\text{SSD}}(x, c_i) + 2\delta_{\text{SSD}}(x, c_k) \leq \delta_{\text{SSD}}(c_i, c_j) \text{ then } \delta_{\text{SSD}}(x, c_i) \leq \delta_{\text{SSD}}(x, c_j) \tag{5.4.26}$$

This test is almost as powerful as the test for metric spaces (Equation 5.4.19), because $\delta_{\text{SSD}}(x, c_i) + \delta_{\text{SSD}}(x, c_k) \approx 2\sqrt{\delta_{\text{SSD}}(x, c_i)\delta_{\text{SSD}}(x, c_k)}$. This approximation is good, because c_i was not eliminated when the closer centre c_k was found, therefore $\delta_{\text{SSD}}(x, c_i)$ falls in the range:

$$\delta_{\text{SSD}}(x, c_k) \leq \delta_{\text{SSD}}(x, c_i) \leq 3\delta_{\text{SSD}}(x, c_k) \tag{5.4.27}$$

In the worst case ($\delta_{\text{SSD}}(x, c_i) = 3\delta_{\text{SSD}}(x, c_k)$) the approximate bound is $8 - 4\sqrt{3}$ times, or 7.2% higher than the exact case.

This test takes negligible time compared with descriptor comparison, and reduces the number of comparisons needed by around 25% when 121D patches are clustered by SSD. Cluster centres are sorted by the number of descriptors assigned to them (when clustering a subset), so that descriptors are compared to the centres they are most likely to be close to first. This order improves performance slightly compared to a random order, or sorting by each clusters' residual (which each reduce the number of comparisons by a smaller amount; about 16% rather than 25%).

A similar test is possible, as shown in Figure 5.11, allows descriptors which are near to one centre, c_k, and are observed to be far from another centre, c_i, to determine that other descriptors close to c_i are not closer than c_k.

$$\text{if } \delta(c_i, c_j) \quad \leq \quad \delta(x, c_i) - \delta(x, c_k) \text{ then } \delta(x, c_j) \geq \delta(x, c_k) \tag{5.4.28}$$

so c_j cannot be closer than c_k. In a high-dimensional descriptor-space with D dimensions however, this test will almost never eliminate centres. If this test can be applied, then

$$\delta(x, c_i) - \delta(x, c_k) \leq \delta(x, c_i)/2 \tag{5.4.29}$$

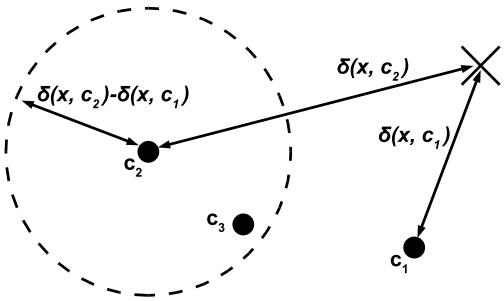


Figure 5.11: When the closest centre so far to descriptor x is c_1, and x is measured to be $\delta(x, c_2)$ from a centre c_2, other centres within $\delta(x, c_2) - \delta(x, c_1)$ of c_2 cannot be the nearest neighbour of x. In this example c_3 is inside this circle, so cannot be the closest centre.

otherwise c_i would have been eliminated by the earlier test. The volume of the hypersphere in which a point must lie in order to be eliminated is less than $\frac{1}{2^D}$ of the volume of a hypersphere including both descriptors, so the probability of it containing a centre is extremely small.
In the future two further optimisations are planned: firstly, the entire descriptor, or occasionally multiple descriptors are compared at each level of the hierarchical dictionary. A more efficient kd-tree like approach would be to partition the descriptors, so for example only eight dimensions of a 64D descriptor would be used for clustering at each of eight levels. This could potentially speed up retraining and indexing by a factor of L, the number of levels, however it may also degrade the quality of the clustering.
The second planned optimisation is to predict where descriptors will fall during re-training, based on their previous cluster assignment. This prediction could potentially be used together with the triangle inequality result detailed above to assign many descriptors to a new dictionary with considerably fewer comparisons.

5.5 Results

In this section experimental results from testing our BoW scheme are presented. Firstly, our scheme is validated and parametrised on an object recognition task. Secondly, the ability to localise a pedestrian in a large indoor environment is demonstrated.

5.5.1 Validation

Firstly, as described in our paper (Botterill et al., 2008), the BoW scheme is tested on the 10,200-image University of Kentucky dataset, of which example images are shown in Figure 5.3. This dataset consists of four photos of each of 2550 objects. Each image is added to the BoW database, after which a dictionary is built. To measure retrieval performance, each image is used as a query image in turn, and the average number of correct matches (those showing the same object) out of the top four returned is computed. A score of 4 represents perfect performance, whereas a score of 1 represents only the ability to detect duplicate images.
To test if re-building the dictionary is worthwhile, rather than using an earlier dictionary, or a dictionary built from training data, recognition performance is measured when training is carried out using only a subset of the images. 64D SURF descriptors are extracted from each image, and dictionaries of 10,000 words are built using CLARA k-medoids clustering. Figure 5.12 shows that training on all of the images gives a clear performance improvement compared with re-training on only a subset of images, hence re-training when new environments are encountered is worthwhile.

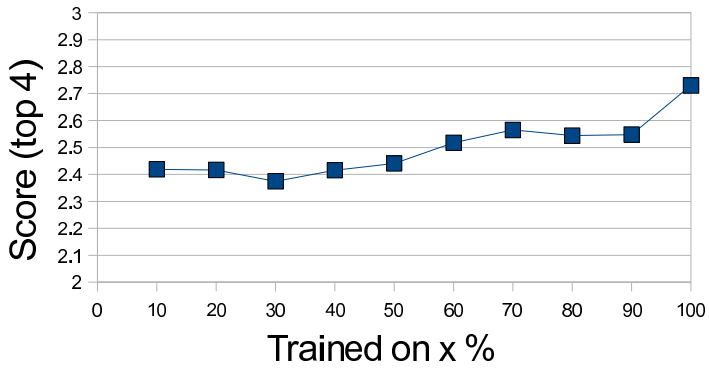


Figure 5.12: Benefits of re-training when new environments are encountered.

Table 5.2: Results from clustering 100k 64D SURF descriptors into 21 clusters at 3 levels (giving 9261 words in total), as presented in Botterill et al. (2008). The mean and sample variance of 5 runs with each parametrisation are given. Times are indicative only; our current implementation is considerably faster. Borderline descriptors are those where the second closest centre is less-than 10% further away than the closest centre. Residuals are in arbitrary units (descriptor-space distance squared).

Clustering method	**CLARA Iterations**	**Residual**	**% borderline**	**Time (s)**	**Score (top 4)**
k-means, 20 iterations	-	39.5 ± 0.0	11.2 ± 0.1	3.6	2.00 ± 0.05
CLARA k-medoids	6	44.2 ± 0.5	8.5 ± 2.2	1.8	1.93 ± 0.02
CLARA k-means	1	40.2 ± 0.0	12.2 ± 3.7	0.4	1.92 ± 0.02
CLARA k-means	6	39.9 ± 0.1	16.0 ± 5.0	3.7	1.90 ± 0.02
CLARA k-medoids	1	44.8 ± 0.1	14.8 ± 2.0	0.4	1.84 ± 0.07
Random set of centres	-	44.0 ± 3.1	22.7 ± 12.0	0.3	1.71 ± 0.08
Random separated centres	-	46.0 ± 4.1	14.7 ± 9.9	0.9	1.71 ± 0.09

The University of Kentucky dataset is also used to evaluate the performance of the approximate clustering scheme, and to choose an appropriate clustering algorithm. Table 5.2 shows analysis of the clustering of a small training set of 100,000 64D SURF descriptors into 21 clusters at each of three levels. The quality of the clustering is evaluated for several different clustering algorithms: full k-means, CLARA k-means, and CLARA k-medoids. The performance of clustering algorithms is compared to the use of random descriptors as centres. It is possible that all clustering is doing is finding a set of descriptors distributed throughout the descriptor space, so sets of random descriptors chosen with a heuristic minimum separation constraint are also considered. Each test is run five times and the mean score and sample standard deviation are given.
Both random centre-selection schemes perform significantly worse than dictionaries generated by any of the clustering algorithms, indicating that . Applying k-means clustering to all descriptors gives the best performance; this is significantly better than any other method, although this is slow, so is not feasible for generating dictionaries online. CLARA k-means and CLARA k-medoids, with subsets sized 300, give clusterings of similar quality. Either scheme could be used for building a dictionary incrementally. With subsets of this size, multiple CLARA iterations are not necessary to ensure that good clusterings are found. The residuals of different clusterings found, and the number of borderline descriptors, appear to have little effect on the effectiveness of dictionaries found.
The University of Kentucky dataset is also used to evaluate several descriptors, and descriptor combinations. The descriptors considered include 64D SURF descriptors, shorter 36D SURF descriptors (the patch described by each descriptor is divided into 9 patches rather than 16, resulting in a shorter descriptor),

Table 5.3: Performance of various descriptors at object recognition, using the University of Kentucky dataset. Around 600 descriptors are extracted from each image.

Descriptor combination	Score (top 4)	Descriptor size (bytes)
36D SURF+Histograms	3.00	64
Random Histograms	2.91	28
64D SURF	2.45	64
11×11 oriented RGB patch	2.35	363
11×11 oriented greyscale patch	2.35	121
64D SURF (downsampled)	2.25	64
9×9 oriented greyscale patch	2.22	81
36D SURF	1.62	36

and 64D SURF descriptors extracted from downsampled images Clarke (as proposed by 2009, to speed-up extraction and enable real-time use). Extracting 600 36D or 64D descriptors from a 640×480 image takes 180ms, and extracting 200 features from an image downsampled to 320×240 image takes 45ms. While SURF descriptors are too expensive to extract from every frame for real-time positioning, these descriptors provide a benchmark for evaluating cheaper descriptors. If used only for location recognition, it might only be necessary to extract SURF descriptors from a subset of frames (Eade and Drummond, 2008). Alternatively, SURF descriptors extracted from downsampled images can feasibly be extracted in real-time.

The second type of descriptors considered are colour histograms. Descriptors of colour are often found to be effective for object recognition, and to be complementary to other descriptors (Hays and Efros, 2008; Filliat, 2007). In addition, they are cheap to extract. The histograms are centred on DoH blobs when combined with SURF descriptors, or are centred at points chosen randomly when used alone. Each histogram consist of a 16 bin 1D hue histogram and a 12 bin 1D saturation histogram, giving a 28 byte descriptor in total. Histogram similarity is measured using SSD, and when combinations of descriptors are used, the different distance measures are simply added together.

The third type of descriptors considered are simple image patches centred on FAST corners. Patch descriptors are often effective for matching features between frames (Section 4.5.5), and are cheap to extract.

For each type of descriptor, a dictionary is built using CLARA k-medoids from all of the descriptors extracted from all of the images (up to six million in total), which takes 200s on a single processor. Each query takes 36ms, which includes the cost of comparing the query image to all 10,200 images in the BoW database.

Table 5.3 shows the object recognition performance with various descriptor parametrisations and combinations. Using a combination of histograms and 36D SURF descriptors gives the best performance, with 3.00 correct matches found on average. Figure 5.13 shows that for this descriptor combination, 95% of the correct matches are returned in the top 5% of query results. Nistér and Stewénius (2006) retrieve 3.29 correct matches on average in a time of about 0.0027s per query (using an inverse file), however they use larger numbers of long and expensive-to-extract 128D SIFT descriptors, and use a slow k-means clustering to build a dictionary.

Using histograms alone is surprisingly effective: 2.91 correct matches are found on average. 36D SURF descriptors perform relatively poorly alone, these short descriptors are not sufficiently distinctive. Of the other descriptors, 64D SURF are the most effective, although only by a small margin over greyscale patches. Again, greyscale patches match the performance of colour patches of the same area, while requiring one third as much memory. Greyscale patches outperform SURF descriptors extracted from downsampled images, indicating that this optimisation is not worthwhile.

In conclusion, colour histograms are the most suitable low-cost descriptors for object recognition, although greyscale image patches are also effective. CLARA k-means or k-medoids clustering successfully builds dictionaries from all observed descriptors, and these dictionaries are more effective than dictionaries built from only a subset of descriptors. Performance is parametrisation-dependent however, and results based on this object recognition task are not necessarily applicable to other tasks, such as robot localisation. In the

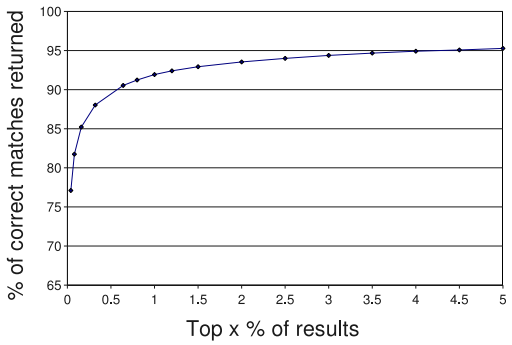


Figure 5.13: 95% of correct matches are returned within the top 5% of results.

following section we describe a pedestrian navigation experiment which demonstrates the location recognition ability of our BoW scheme in a more realistic environment.

5.5.2 Pedestrian navigation application

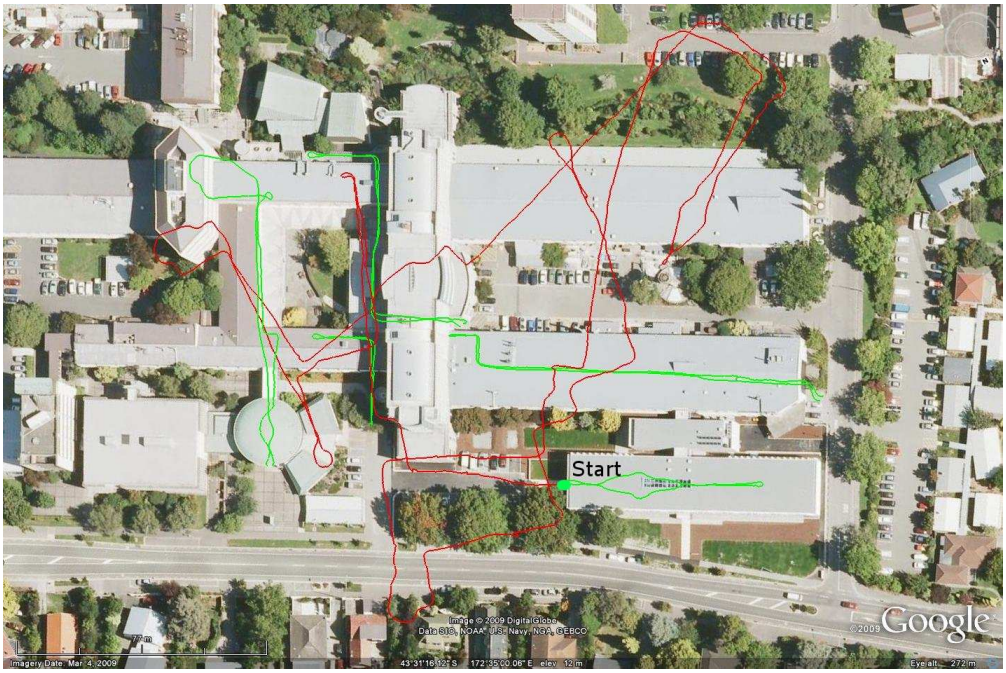
One use for BoW location recognition is localisation in a previously-mapped environment. This section presents results from integrating the BoW scheme into a pedestrian navigation system, which provides validation that it is suitable for robot localisation. My BoW implementation is used, and I co-authored the paper on the system (Hide, Botterill, and Andreotti, 2009), however the remainder of the pedestrian navigation system was developed and demonstrated by Chris Hide, who also prepared the figures which are reproduced in this section. This system integrates measurements from a low-cost foot-mounted IMU (Section 2.3.1) with the occasional GPS signal (when the pedestrian ventures outdoors) using a Kalman filter, and allows the pedestrian's path to be reconstructed as they walk around a building. Without GPS updates, the position estimate will slowly drift.

For many pedestrian navigation applications (for example a 'pedestrian satnav' scheme to help a visitor navigate a university campus), it is feasible to survey some of the areas which the pedestrian will visit. This survey consists of walking around the area with a handheld Sony Handycam video camera and an expensive 'navigation grade' IMU. The IMU data is post-processed to produce accurate position estimates of where images were captured. Greyscale patch descriptors are extracted from the images, and these are used to build a BoW database using k-medoids clustering.

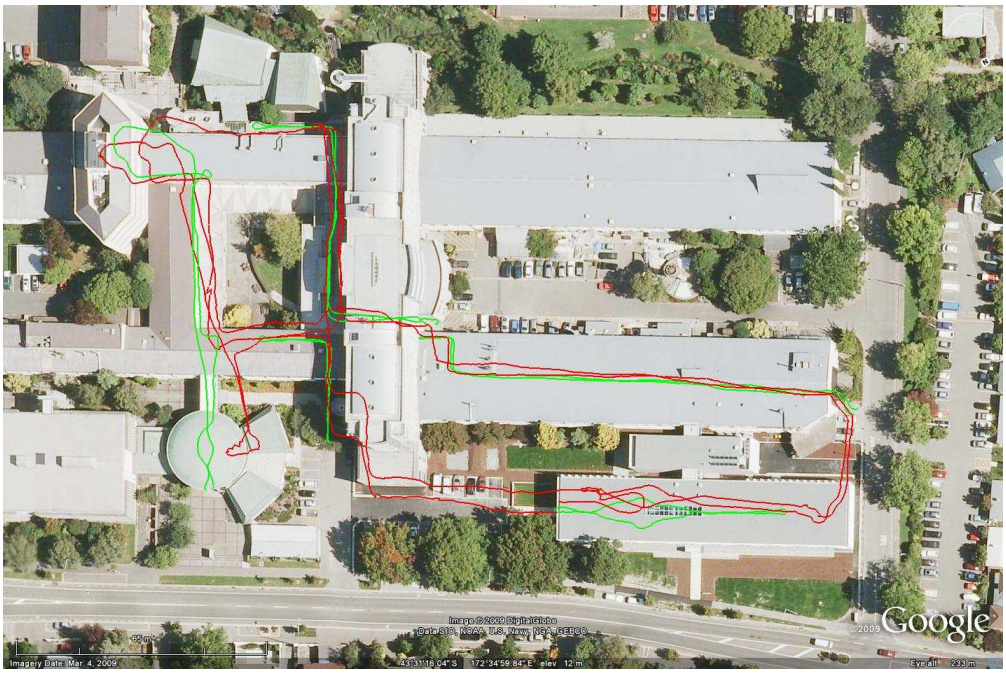
When a pedestrian navigates the building, they also carry a camera. Images captured are represented as BoWs, and the database is searched for matches. A match is used when the top match is above a threshold, and four of the top five matches come from approximately the same location (i.e. sequentially nearby in the survey images). The top matches are compared to the position estimate from the IMU; matches indicating that the pedestrian is further than three standard deviations from their estimated position are rejected as outliers.

Figure 5.16 shows matched images that identify the robustness of the BoW scheme to small changes in scale and rotation, and large changes in the environment. As the environment is densely mapped, the images which match most strongly are generally captured from very similar locations, so coordinate updates from image matches are relatively accurate.

The image database that was constructed using the survey data consisted of 4322 images, and their coordinates. The navigation trial spanned a duration of 21 minutes, over which 5066 images were collected, at



(a) Trajectory computed from IMU alone (red), and ground-truth survey track (green).



(b) Trajectory computed from IMU with BoW absolute position updates (red).

Figure 5.14: (a) Positions computed from a low-cost foot-mounted IMU alone show substantial drift. 5.14(b) When updates from using BoW scene recognition are used, the track (marked in green) closely matches the survey track (marked in red). (from Hide et al., 2009).

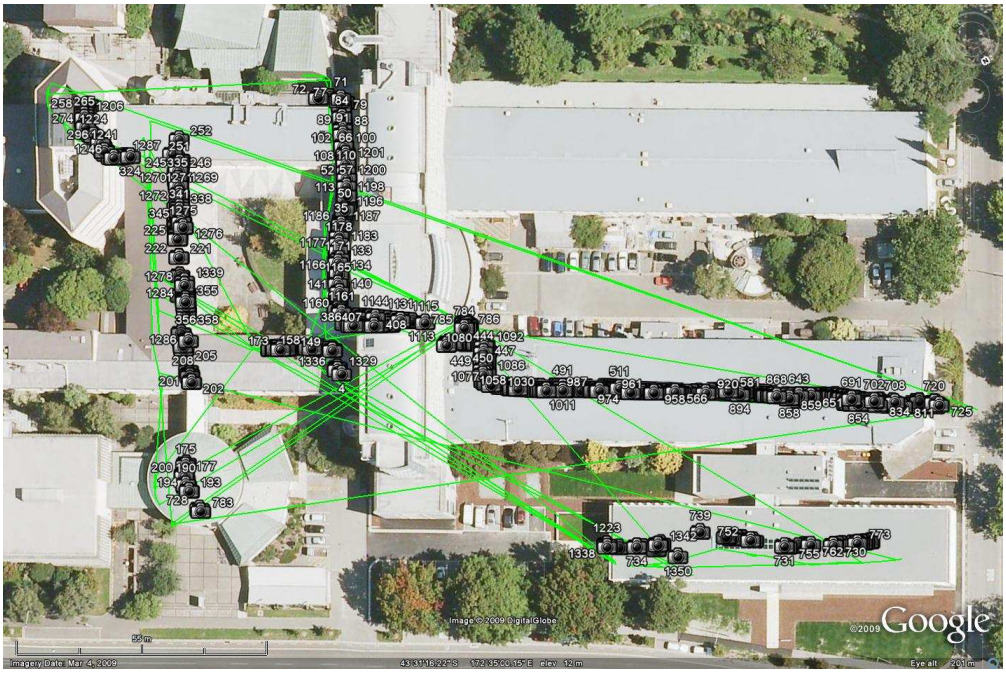


Figure 5.15: Image matches are found throughout the surveyed regions (marked with camera icons). Green lines join them to their correct location; longer green lines indicate incorrect matches, which are removed if they are far from the current estimated position (from Hide et al., 2009).

4Hz. Out of the 5066 images, the BoW scheme matched 1374 images (27%) to survey locations. From the 1374 matched images, 51 images (4% of matched images) were incorrectly matched, which corresponds to a precision of 96% at a recall rate of 27%.
Figure 5.14 shows the integrated IMU and BoW position solution, compared to the survey data, and the position from the IMU alone. The BoW updates clearly improve the position updates. The area that has most notable position drift is on the left hand side of the image and corresponds to an area where fewer BoW position updates are found. Through knowledge of the user's true trajectory, and by measuring position error in Google Earth, the largest known position error is measured as 14m.
This experiment is larger than the BoW loop closure experiment described by Cummins and Newman (2008a), where 2474 images are captured over two loops of a 1km track, captured in a single survey. While our precision is slightly lower than their 100% figure (they report 96% precision at 40% recall, and 100% at 27%), the distribution of matches that we observe (Figure 5.15) is at least as good: in surveyed areas, periods where no matches are found are 100m long at most. In Section 7.4.2 we describe how geometry can be used to eliminate almost all of the few false matches we observe.

5.6 Correspondences from the Bag-of-Words algorithm

The BoW representation can also be used to extract cheap correspondences between pairs of images, which can then be used for both validation of BoW matches, and for camera positioning. In this section we describe how these are obtained.
If there is exactly one of a particular word in each of two images, then ideally this will correspond to a matching feature. Finding these matching words is extremely fast, however variability in cluster sizes leads to many poorly conditioned correspondences being introduced, while good correspondences are missed when descriptors are split between two nearby clusters. Typically only 40% of the correspondences found

Figure 5.16: Examples of matches found when using our BoW scheme for pedestrian navigation. Images from the navigation trial (left); image matched from the database (right). Changes include people and objects visible in one image which are not visible in others, small changes in orientation and scale, and changes in illumination (the survey was conducted on a cloudy day, and the experiment on a sunny day) (from Hide et al., 2009).

by a brute-force matching are found by BoW feature matching. To avoid this problem we use the coarser partitioning of the descriptor space at a higher level of the hierarchical dictionary, and find matches within each cluster by brute-force descriptor comparison. As clusters are still small, with typically fewer than 20 descriptors in each, this is not prohibitively expensive (taking typically 2ms per image pair with 300 features from each image), and typically over 80% of the correspondences found by a brute-force approach (which requires around 50ms) are found, with very few additional outliers.
When the descriptors of a small number (N and M) of a particular word appearing in two images are all sufficiently similar-looking be matched, all possible pairs can be considered as candidate matches ('$N - M$ correspondences'). This is useful to ensure sufficient match-candidates are found in self-similar environments. Chapter 7 describes how the subset of these that are correct is determined.
This BoW feature matching method is similar to the kd-tree approach described in the previous chapter. BoWSLAM uses the partitioning found by clustering descriptors rather than an arbitrary partitioning about descriptor components. Clustering descriptors minimises the number that will fall near the boundary of two clusters (Table 5.2), so unlike the kd-tree approach, by choosing a sufficiently coarse partitioning we can avoid searching neighbouring bins for matches.
One enhancement which may boost BoW feature matching's performance is to assign descriptors falling near the border of clusters to multiple image words, as is done by Jegou et al. (2008).

5.7 Conclusions

In this chapter we have developed a new BoW scheme for localising a robot in a previously unknown environment. This scheme is based on the hierarchical dictionary model of Nistér and Stewénius (2006), which allows images to be indexed and matched efficiently. Unlike previous large scale BoW schemes, our new scheme allows the dictionary to be rebuilt dynamically, allowing image words appropriate for describing any environment to be found. This simple retraining scheme is shown to attain the theoretical best complexity for a scheme rebuilding a dictionary dynamically, unlike previous schemes. The total cost can be made no worse than a small constant factor more than any other scheme to rebuild a dictionary dynamically.
In a large scale pedestrian navigation application, our BoW scheme is shown to have similar performance to leading contemporary BoW schemes, while requiring only computationally cheap image patch descriptors rather than expensive invariant descriptors. Regular position updates are provided, which are used to correct the drift in a position estimate from an IMU.

Chapter 6

Background to robust relative camera pose computation

Abstract

Robustly computing the relative pose of the robot when two images were captured is essential for BoWSLAM's operation. This chapter reviews suitable algorithms for this estimation, and gives an overview of the RANSAC framework for robust estimation. Experiments are conducted on simulated data in order to select a suitable combination of algorithms.

6.1 Introduction

An integral part of BoWSLAM is the framework which computes relative poses from a sequence of images. This chapter describes contemporary algorithms suitable for this task, then selects an appropriate combination of these algorithms for robustly computing relative poses from feature correspondences between pairs of images.

As a robot explores, it captures images of its environment. When a pair of images overlap, the relative pose of the two camera positions when they were captured can be computed. From a sequence of overlapping images, a sequence of relative pose estimates can be added together to reconstruct the trajectory of the camera. The accuracy of the trajectory depends on the accuracy of the relative poses used to compute it. If any of these relative poses contain errors, then these errors will propagate into the estimate of the trajectory. Hence it is important that as many as possible of the relative pose estimates are free of gross errors, and ideally that the remaining errors are minimised.

This chapter is organised as follows: the following section gives an overview of the combination of algorithms used, then describes each algorithm in detail. The efficient implementation of these algorithms is discussed in Section 6.3. Section 6.4 evaluates the performance of the algorithms described on simulated data, in order to inform implementation decisions and to quantify uncertainty. Finally, Section 6.5 summarises and discusses our choice of algorithms.

6.2 Algorithms for relative pose computation

This section gives a brief overview of the combination of algorithms used, before describing each algorithm in detail. The combination of algorithms used is summarised in Figure 6.1.

The relative pose of two camera positions can be computed from a set of feature correspondences. These correspondences can be obtained by matching descriptors between pairs of images, as described in Section 4.5.6, or from the images' Bag-of-Words (BoW) representations, as described in Section 5.6. If the correspondences are correct matches between static features in the world, then least-squares methods can be used to estimate

the relative pose of the two camera positions. Exact methods, and linear least-squares methods for relative camera pose computation are described in Section 6.2.1, and nonlinear least-squares methods are described in Section 6.2.2. In practice however, these correspondences contain many gross errors, caused by incorrectly-matched features in self-similar environments, from corners arising from occlusion boundaries intersecting edges, and from moving objects in the scene; these outliers must be removed before least-squares methods can be applied. To remove outlier correspondences, while simultaneously computing a relative pose from inliers, the RANSAC framework (Fischler and Bolles, 1981) can be used. While BoWSLAM actually uses the BaySAC framework described in the following chapter, BaySAC is based on the RANSAC framework, as described in this chapter in Section 6.2.3.
Once RANSAC has identified a set of correspondences containing mostly inliers, and a relative pose estimate which is approximately correct, a top-down refinement procedure (Rousseeuw and Leroy, 1987) is applied to remove remaining outliers and to improve the accuracy of the pose estimate. Two possible top-down refinement algorithms are described in Section 6.2.4.
Many of the algorithms described use the Singular Value Decomposition (SVD), or Levenberg-Marquardt's method for least-squares optimisation, which are described in Appendix A.

For each pair of frames:

1. **Find Matches**
   - Find a set of possible correspondences, e.g. from the Bag-of-Words representations of the two images (from 15%-75% of about 200 matches will be inliers)
2. **Find inliers and approximate solution**
   - Use RANSAC/BaySAC to identify which of the possible correspondences are inliers, while simultaneously estimating a first approximation to the essential matrix (inlier set now contains over 95% inliers)
3. **Refine inliers and solution**
   - Top-down refinement finds more inliers and rejects more outliers, while simultaneously fitting an increasingly accurate essential matrix
   - Choose camera matrix from E and reconstruct 3D point positions (up to a scale factor); reject points falling behind camera (inlier set now contains over 97% inliers)
   - Refine relative pose by Levenberg's nonlinear least-squares optimisation to minimise Sampson's error (errors in relative orientation reduced by typically 30%)
4. **Estimate uncertainty in relative orientation**
   - Variance estimated from number of inliers and feature localisation accuracy.

Figure 6.1: Procedure for estimating the relative pose of two cameras

6.2.1 Exact and linear least-squares computation of the essential matrix

The relative pose of two cameras consists of a 3D rotation and translation. From correspondences between matching features in two frames this rotation and the direction of the translation can be estimated, however the magnitude of this translation cannot be determined without further information. In this chapter, only the estimation of the rotation and translation direction is discussed, and the procedure used to recover the translation magnitude is described later in Section 8.3.2.
Algorithms to compute the relative pose from feature correspondences often work by estimating the essential matrix (Hartley and Zisserman, 2003, Chapter 9). The essential matrix, $\mathbf{E}$, is a 3×3 matrix encoding the

rotation and translation direction. If the rotation is expressed as a matrix, $\mathbf{R}$, and the translation as a unit vector, $\mathbf{t}$, then the essential matrix, $\mathbf{E}$, is defined by

$$\mathbf{E} = [\mathbf{t}]_\times R \tag{6.2.1}$$

where $[\mathbf{t}]_\times$ is the matrix-representation of the vector cross-product, with the property that $[\mathbf{t}]_\times \mathbf{x} \equiv \mathbf{t} \times \mathbf{x}$. $[\mathbf{t}]_\times$ can be written:

$$[\mathbf{t}]_\times \equiv \begin{pmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{pmatrix} \tag{6.2.2}$$

where $\mathbf{t} = (t_1, t_2, t_3)$.

This section describes two algorithms for estimating the $\mathbf{E}$: firstly the five-point algorithm, by Stewénius et al. (2006), for computing a set of possible essential matrices from five correspondences, and secondly, the eight-point algorithm (Hartley and Zisserman, 2003, Chapter 11), a linear least-squares algorithm for fitting an essential matrix to eight or more correspondences. While the workings of the algorithms are not central to the thesis, understanding their operation and characteristics are important, as they are required as part of the RANSAC framework, and are often the single most costly part of BoWSLAM (Figure 8.25).

$\mathbf{E}$ has the property that, for a noiseless correspondence between points $\mathbf{p}$ and $\mathbf{p}'$ in the image ($(p_x, p_y, 1)$ and $(p'_x, p'_y, 1)$ as homogeneous coordinates):

$$\mathbf{p}'^T \mathbf{E} \mathbf{p} = 0 \tag{6.2.3}$$

Expanding this equation gives a single linear constraint in the nine elements of $\mathbf{E}$ for every correspondence. As $\mathbf{E}$ has five degrees of freedom, from the 3 DOF rotation and 2 DOF translation direction in Equation 6.2.1, at least five correspondences are needed to estimate it. Given exactly five correspondences, there are up to ten possibilities for $\mathbf{E}$, which can be computed from Equation 6.2.3 using the five-point algorithm of Stewénius et al. (2006). The five-point algorithm uses the five linear equations in the nine elements of $\mathbf{E}$ from Equation 6.2.3, to compute four basis vectors for the elements of $\mathbf{E}$, by SVD. Any linear combinations of these basis vectors which satisfy Equation 6.2.1 is a possible essential matrix. These possibilities are found by solving cubic constraints on the elements of $\mathbf{E}$ from Equation 6.2.1.

$\mathbf{E}$ is uniquely determined by eight correspondences, which give eight linear equations in the nine elements of $\mathbf{E}$, with an additional constraint given by $\det \mathbf{E} = 0$ (because $\det [\mathbf{t}]_\times = 0$). $\mathbf{E}$ could be computed from the matrix $\mathbf{F}$ that minimises:

$$\sum_{i=1}^{C} \mathbf{p}_i'^T \mathbf{F} \mathbf{p}_i \tag{6.2.4}$$

for $C \geq 8$, however this cost function is biased towards $\mathbf{p}$ with high magnitude. To reduce the effect of this bias, the normalised eight-point algorithm (Hartley and Zisserman, 2003, Chapter 11) is used. The normalised eight-point algorithm first transforms the set of point locations from each image with two transforms, $\mathbf{T}$ and $\mathbf{T}'$, in order to centre them and to normalise their variance.

$$\mathbf{q}_i = \mathbf{T}\mathbf{p}_i, \qquad \mathbf{q}'_i = \mathbf{T}'\mathbf{p}'_i. \tag{6.2.5}$$

The SVD is then used to estimate the elements of the matrix $\mathbf{F}'$ best-satisfying the C linear constraints from Equation 6.2.3 for these normalised points:

$$\sum_{i=1}^{C} \mathbf{q}_i'^T \mathbf{F}' \mathbf{q}_i. \tag{6.2.6}$$

The normalising transforms are then applied to $\mathbf{F}'$ to find a matrix $\mathbf{F}$ which approximately satisfies Equation 6.2.3 for the original points:

$$\mathbf{F} = \mathbf{T}'^{-1} \mathbf{F}' \mathbf{T}^{-1}. \tag{6.2.7}$$

This matrix $\mathbf{F}$ is an approximation to the fundamental matrix (a better estimate is given by imposing the constraint that $\det F = 0$; Hartley and Zisserman (2003), chapter 11). $\mathbf{F}$ will not exactly satisfy

Equation 6.2.1 however. Equation 6.2.1 implies that the singular values of E are 1, 1, and 0, and this property can be used to calculate E as follows:

$$\mathbf{E} = \mathbf{U} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{V}^T \tag{6.2.8}$$

where $\mathbf{UDV}^T = \mathbf{F}$ is the SVD of $\mathbf{F}$ (Hartley and Zisserman, 2003, Chapter 11).

6.2.2 Nonlinear refinement of relative poses

The normalised eight-point algorithm finds the essential matrix from the matrix minimising Equation 6.2.6. This equation is not a good measure of the error in relative pose despute the normalisation however, as it is biased by the distances of points from their centroid. Ideally the transformation and 3D structure should be found that minimises the squared distances between where reconstructed 3D points are projected into the image and where they are measured to lie (the reprojection error; Hartley and Zisserman, 2003, Chapter 11). This estimate is the MLE of the relative pose (Triggs et al., 1999).
The orientation, translation direction, and reconstructed 3D point locations can be refined iteratively to minimise the reprojection error using Levenberg's method for nonlinear optimisation (Appendix A.2). Convergence typically requires few iterations. Refining a large number of reconstructed points is costly however; an alternative approach is to refine only the relative pose, to minimise Sampson's approximation to the reprojection error (Hartley and Zisserman, 2003, Chapter 11), given by

$$\sum_{i=1}^{C} \frac{\mathbf{p}_i'^T E \mathbf{p}_i}{(E\mathbf{p}_i)_1^2 + (E\mathbf{p}_i)_2^2 + (E\mathbf{p}_i')_1^2 + (E\mathbf{p}_i')_2^2}. \tag{6.2.9}$$

This optimisation is fast as only the relative pose is refined, and few iterations are needed to achieve convergence.
These least-squares methods minimise the effects of small errors in feature localisation, however they are sensitive to incorrect correspondences (outliers), which can lead to a completely incorrect solution. The following section describes how outliers are removed, while using the exact and least-squares methods to compute an accurate camera pose from the inliers.

6.2.3 RANSAC for essential matrix estimation

E is computed from outlier-contaminated correspondences, while simultaneously identifying inlier correspondences, using the RANSAC framework (Fischler and Bolles, 1981, summarised in Figure 6.2), as demonstrated by (Torr and Murray, 1997). In RANSAC, many small hypothesis sets of data points are chosen randomly. Each hypothesis set is used to generate candidate models, and the model consistent with the largest number of data points is selected. Assuming the selected model is correct, the data points consistant with this model are inliers.
To estimate **E** by RANSAC, random hypothesis sets of five correspondences are selected. Each hypothesis set is used to compute up to ten essential matrices, using the five-point algorithm described in Section 6.2.1. Each of these essential matrices is tested against every correspondence, and correspondences compatible with the model being correct are counted (those where Sampson's error is below a threshold). This is repeated for many hypothesis sets, until an essential matrix compatible with many correspondences is found. If the model is correct, then these correspondences are inliers.
While five correspondences is the minimum needed to generate a relative pose hypothesis in general, SLAM schemes often assume a motion model, which allows an approximate relative pose to be predicted by extrapolating from previous motion. Single camera EKF-SLAM schemes (Section 3.3) can update a predicted pose using just a single correspondence. Civera et al. (2009b) describe a 1-point RANSAC scheme for visual odometry, where random hypothesis sets consisting of just a single correspondences are used. Each

Repeat for I iterations, or until a large inlier set is found:

Hypothesise

- Choose a random hypothesis set of 5 correspondences, H
- Calculate 2 to 10 essential matrices from H using the five-point algorithm

Test

For each of these essential matrices:

- Find the subset of correspondences where Sampson's error falls below a threshold (inliers if this model is correct).

Return model with largest inlier set, and the corresponding inlier set.

Figure 6.2: RANSAC for essential matrix computation.

of these correspondences is evaluated by updating a copy of the EKF's state. Only the updates which are also compatible with the observed motion of many other tracked points are used to update the SLAM EKF. Hundreds of features can be matched between frames in real-time, however this framework is only suitable for the situation where camera motion between frames is small, and could not be adapted to register frames without prior relative pose estimates, for example when closing loops or when tracking fails.

6.2.4 Top-down outlier removal

RANSAC efficiently finds models which are approximately correct, however usually a few outliers will remain, and some inliers will be missed. Even with low outlier rates, RANSAC can require a large number of iterations to find a solution including all of the inliers (Lacey et al., 2000), as even when the hypothesis set contains only inliers, errors in localising these points introduce errors into the model found, and these errors are sufficient that the model is likely to miss inliers. In addition, outliers are likely to be found which are compatible with the model. For this reason a combination of approaches is needed: RANSAC to find an approximate inlier set containing at most a few-percent outliers, then a top-down refinement to remove these outliers while finding more inliers (Chum, 2005).

To remove outliers and compute a more accurate relative pose estimate by top-down refinement, Lacey et al. (2000) suggest using the approach of Trivedi (1987), who use nonlinear optimisation to minimise a error function which is robust to mismatched features. This error function is very similar to Sampson's error when points are compatible with the relative pose estimate, i.e. where Sampson's error is less than 3 pixels, and is constant for points which are not compatible (outliers). The true minimum of the error function is likely to correspond to the model compatible with most data points. The relative pose is refined using the downhill-simplex algorithm (an iterative gradient descent method similar to Levenberg-Marquardt). A suitable initial estimate is required to avoid local minima.

An alternative, simpler, top-down approach is proposed by Rousseeuw and Leroy (1987). A model is fitted to all data points believed to be inliers, then the points with the largest residual errors are removed, and other points with smaller residual error are added to the inlier set. For essential matrix estimation, an essential matrix is fitted to the correspondences believed to be inliers (following RANSAC) using the normalised eight-point algorithm, then Sampson's error is computed for every data point. Points where Sampson's error exceeds a threshold are removed from the inlier set, and other points where Sampson's error falls below the threshold are added. This refinement is repeated several times. Normally, a large inlier set containing few outliers is found after few iterations, as illustrated in Figure 6.3. The threshold is determined empirically; when outlier rates are high, and few inliers are found, a threshold which is either too high or too low causes the initial solution to diverge to one with fewer inliers.

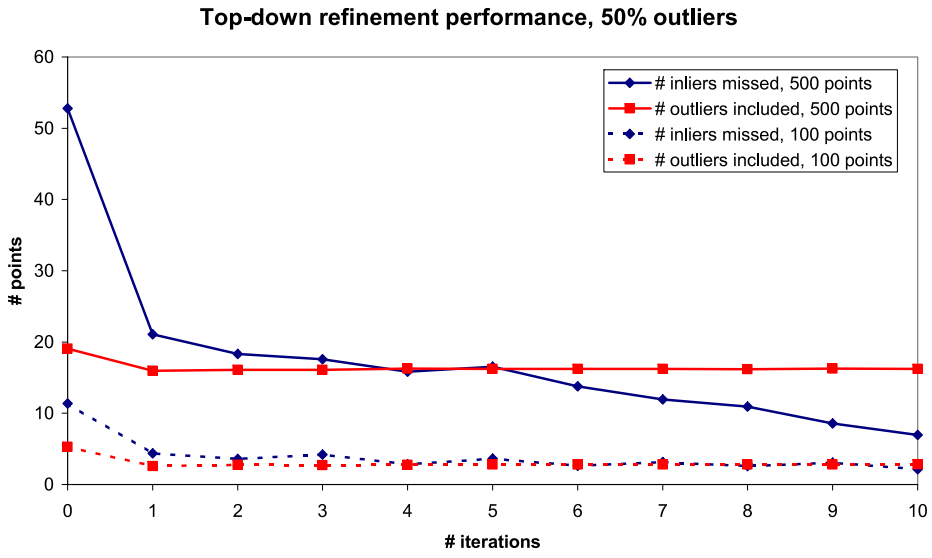


Figure 6.3: Graph showing the effects of each topdown refinement iteration on inliers missed and outliers found following RANSAC. Top-down refinement rapidly excludes many outliers, except for those which are coincidentally compatible with the best model found. Although parametrisation dependent, only a small number of iterations are necessary. Sets of 100 or 500 correspondences with 50% outlier rates, and localisation errors with a standard deviation of 0.0025 radians are simulated, and RANSAC is used to fit an essential matrix.

Once the essential matrix and inlier set are found, the essential matrix is decomposed to give four possibilities for the relative camera pose; the correct relative pose is chosen to be the one leading to most reconstructed points falling in front of the cameras (Hartley and Zisserman, 2003, Chapter 9). Points lying behind the camera are now discarded as outliers, and remaining points are used for nonlinear refinement of the camera pose, as described in Section 6.2.1. Given this optimised relative pose, 3D point positions can be reconstructed up to an unknown scale factor (Hartley and Zisserman, 2003, Chapter 10).

6.2.5 Alternative frameworks for robust relative pose computation

While RANSAC is easily the most popular framework for relative pose estimation from outlier-contaminated correspondences, other approaches have been proposed for the case when an approximate relative pose is available, for example from a motion model in SLAM. In this case, an alternative to the classic pipeline of extracting features, describing these features, matching pairs of descriptors, then using these matches for structure and motion estimation, is to couple these stages together, so that the current estimated pose is used to predict where in the image additional matches are likely to be found, with the new matches then used to further improved the pose estimate. Chli and Davison (2008) describe such a scheme for efficient matching in an EKF-SLAM framework, known as Active Matching. When a single feature match is found, this match is used to update the pose estimate, so that more matches can be found. As the pose estimate improves, the area which is searched for each feature decreases. Each feature is selected to maximise the amount of mutual information which will be gained from its measurement, relative to the cost of making the measurement, so that the accuracy of the pose estimate rapidly improves as more features are observed. Overall only a small area of each image must be searched, so when the number of features is small, Active Matching is more efficient than simpler EKF-SLAM tracking schemes, which search for every tracked feature

before making any pose update.
A limitation of Active Matching is the cubic complexity in the number of matched features, which limits real-time use when larger numbers of matches are needed. This complexity arises from the process used to identify the feature which should be searched for next at each stage. Handa et al. (2010) propose an extension to Active Matching, SubAM, which addresses this issue. From the mutual information of pairs of features, a balanced approximation to the Chow Liu tree is constructed. The Chow Liu tree is used to find approximations to the mutual information between pairs of nodes efficiently. SubAIM can match 400 features between a pair of images from a slowly moving camera in 170ms. While somewhat higher than the typical cost of using RANSAC for this matching, the success of this novel approach indicates that alternatives to RANSAC could be used. A limitation of this approaches is the requirement for an initial relative pose estimate; while an initial estimate is often available, wide-baseline matching without a motion prior would still be required for when loops are closed, or to resume positioning after sequences of unusable frames in challenging environments.

6.3 Implementation details

Our five-point algorithm implementation is based on code by Stewnius (2006), ported to C++. The 'Eigen' matrix library (Guennebaud and Jacob, n.d.) is used for linear algebra; this provides a three-fold speedup compared with the Lapack++ LAPACK wrapper (Stimming, 2004), as small, fixed-size systems are solved, which Eigen specialises in solving efficiently. 50% of the computational cost consists of computing the eigendecomposition of a 10×10 matrix; an alternative five-point algorithm by Nistér (2004) avoids this eigendecomposition (and solves a 10th degree polynomial instead), so may be more efficient.
Levenberg's method for nonlinear optimisation (Appendix A.2) is also implemented using Eigen. 3 DOF rotations are represented as 4D quaternions (as is common when refining orientation; Grisetti et al., 2010), and translation directions are represented as 3D unit vectors. These overdetermined representations, together with very high correlations between certain measurements (e.g. sideways translation versus yaw), mean that the Jacobian matrix is often near-singular. In this case, Levenberg's damping method, which tolerates a near-singular Jacobian, is more suitable than the method by Marquardt (1963) (in addition, for many problems, these approaches converge at similar rates anyway; Lampton, 1997). Typically, convergence takes two to six iterations.

6.4 Selection of camera pose refinement algorithm

The eight-point algorithm, and the algorithms to minimise Sampson's error and the reprojection error are tested and evaluated on simulated data in order to identify which are suitable for use in BoWSLAM. In simulated data, the camera motion, feature localisation errors, and outlier rate are known, allowing an algorithm's performance to be evaluated relative to any of these factors.

6.4.1 Simulating data to evaluate algorithms

Images are simulated by first simulating a cloud of N points and projecting into two cameras. Points have a depth of 3 to 4 times the baseline length (the distance between the camera centres). The relative pose of the cameras is chosen with motion in a uniform random direction, and a rotation with angle chosen uniformly from $[-0.75, 0.75]$, about a uniform random axis[1]. This range of angles ensures points stay in front of the cameras. This random relative pose is used to generate a camera matrix which is used to project points into the simulated images. Measurement noise is simulated by adding Gaussian noise. Cameras have a FOV of approximately one radian square. When a relative pose is estimated, its error is given by the angular

[1] Uniform random directions are chosen by choosing three i.i.d. normally distributed random variables with zero mean, then normalising.

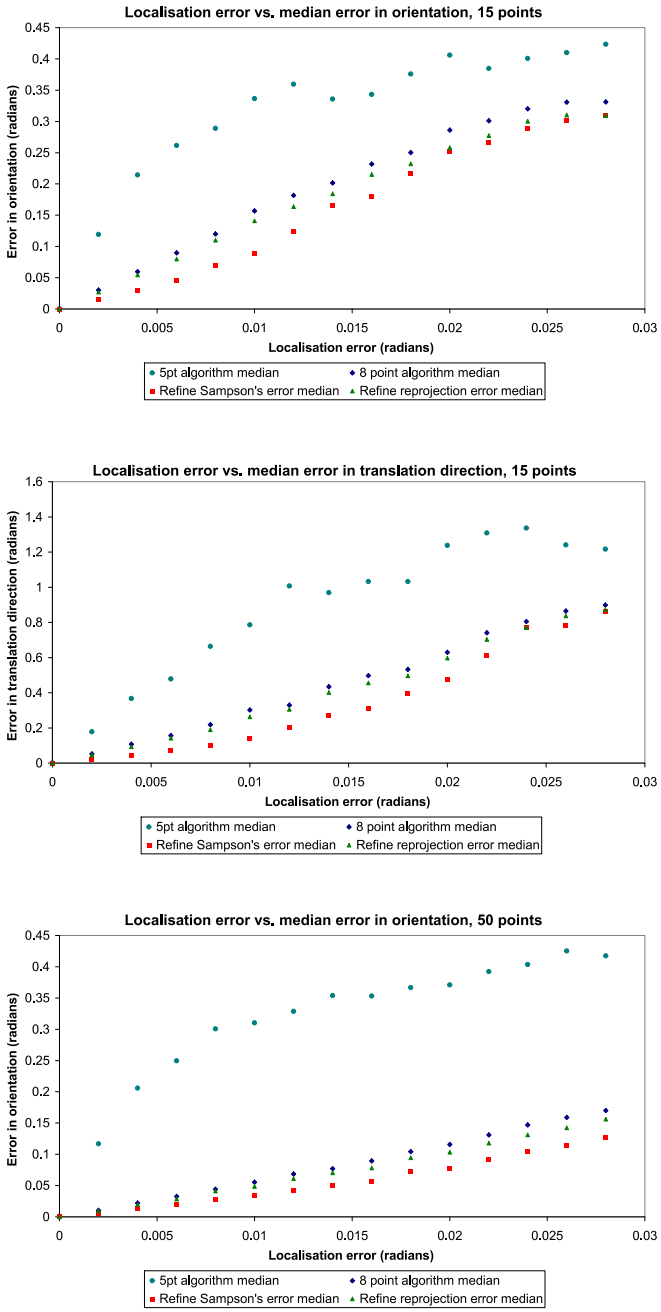


Figure 6.4: Graphs showing the effects of feature localisation error on the accuracy of relative pose computation algorithms. Median errors in orientation and translation direction increase with increasing feature localisation error. Medians are used as all methods suffer from gross outliers when points have high localisation errors. The 99% confidence bound is 0.06 at worst for each point.

differences between the estimated relative orientation and translation direction, and the known orientation and translation direction.
Each point on the graphs shown in this section is the average of thousands of runs, giving each point a tight 99% confidence bound, e.g. 0.06 radians at worst, and typically around 0.02 radians for Figures 6.4 and 6.5. This simulated data is designed so that points at infinity are almost never observed. Points at infinity are points far from the camera, the depths of which cannot be estimated accurately (often they will appear to have a large negative depth), as even small localisation errors have a large effect on the reconstructed depths of distant points. In real data, points at infinity are frequently encountered. BoWSLAM uses them for camera pose refinement, but not for subsequent 3D reconstruction.

6.4.2 Experimental results on simulated data

Figure 6.4 compares the performance of the different algorithms with varying feature localisation errors. The relationship between localisation errors and errors in orientation and translation is approximately linear in all cases. Unsurprisingly, the five-point algorithm is substantially less accurate than least-squares approaches, with large model errors occurring even when localisation errors are relatively small. The eight-point algorithm finds a considerably more accurate solution, but this is improved substantially by nonlinear refinement. Refining the pose to minimise Sampson's error gives consistently more accurate results than minimising the reprojection error. It is not clear why this is; if the reprojection error refinement is started from the solution of the Sampson's error refinement, it still diverges to a point with lower reprojection error and higher model error. The same observation is made regardless of point depths or FOV, or if squared angular errors are used to evaluate the algorithms. This is a welcome observation however as Sampson's error is considerably cheaper to refine. A similar result was found by Chum et al. (2005) in the related problem of perspective homography recovery (described in Chapter 10.1): Sampson's error is as good for homography fitting as the reprojection-error equivalent for homographies.
Figure 6.5 shows the relationship between the number of correspondences and the accuracy of refined solutions. Gross errors are observed with small numbers of points; models computed using few points should be avoided where possible. Increasing the number of points decreases errors in localisation, and again, refining poses to minimise Sampson's error leads to the most accurate reconstruction. Even with 100 correspondences between each pair of frames however, significant errors will accumulate when hundreds of poses are added together, again indicating the importance of good feature localisation accuracy.
RANSAC's performance is evaluated by comparing its success rate with its theoretical success rate. If correspondences are inliers with probability p, then the probability of having observing an all-inlier hypothesis set after I RANSAC iterations is given by

$$\begin{aligned} P(\text{ success in } I \text{ iterations }) &= 1 - P(\text{ one hypothesis set contains outliers })^I && (6.4.1) \\ &= 1 - (1 - p^5)^I. && (6.4.2) \end{aligned}$$

Figure 6.6 shows the success rate of 250 RANSAC iterations, with a range of outlier rates, and also shows its theoretical success rate. For this simulated data, RANSAC is close to matching its theoretical success rate, although the performance drops slightly when only a small number of all-inlier hypothesis sets are seen (at outlier rates of around 50%). RANSAC performs well with outlier rates of up to 60%, in which case it succeeds 70% of the time, however performance drops sharply for higher outlier rates. Note that at high outlier rates, RANSAC sometimes finds an inlier set even when no hypothesis set contains only inliers, as so many models are tried that a model compatible with many inliers is often found by chance. Figure 6.7 shows that when RANSAC is successful in finding an inlier set, errors in computed orientations are low, even though outlier rates are high. Even though many outliers remain in the inlier sets found by RANSAC, these are successfully removed by top-down refinement (Figure 6.3).
Table 6.1 shows indicative computational costs of the algorithms discussed. The least-squares eight-point algorithm, and nonlinear optimisation of Sampson's error, are cheap compared with the cost of even just 100 RANSAC iterations, indicating the importance of making RANSAC converge rapidly. The cost of RANSAC

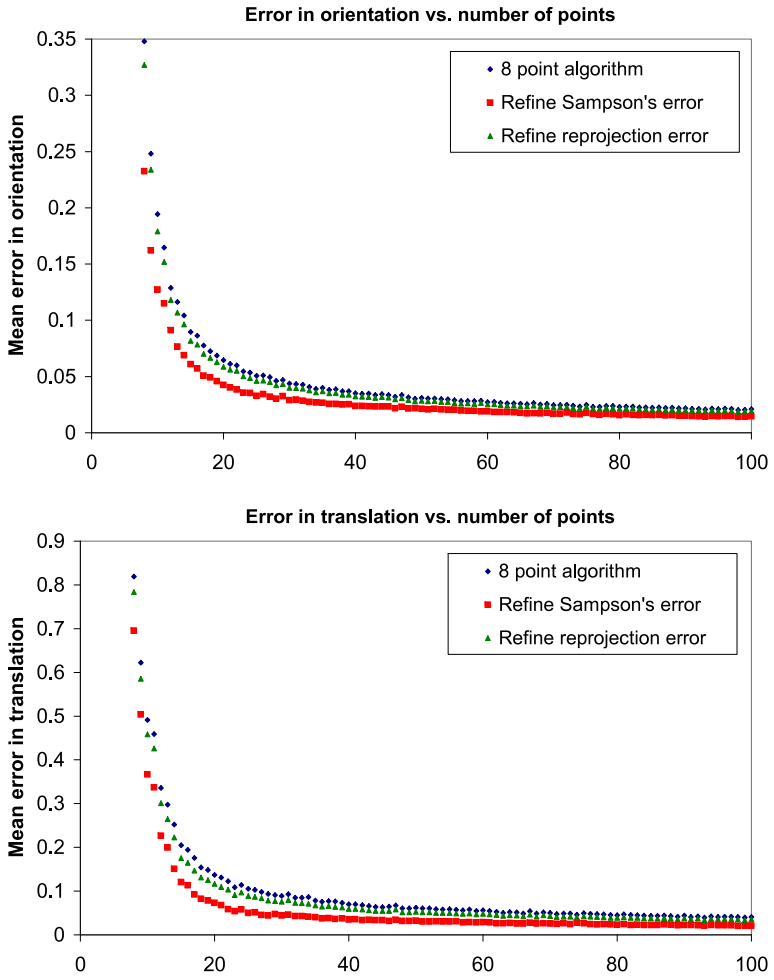


Figure 6.5: Graphs showing the accuracy of relative pose computation with increasing numbers of points. Mean errors in orientation and translation (in radians) estimated are from simulated correspondences with localisation errors of 0.005 radians.

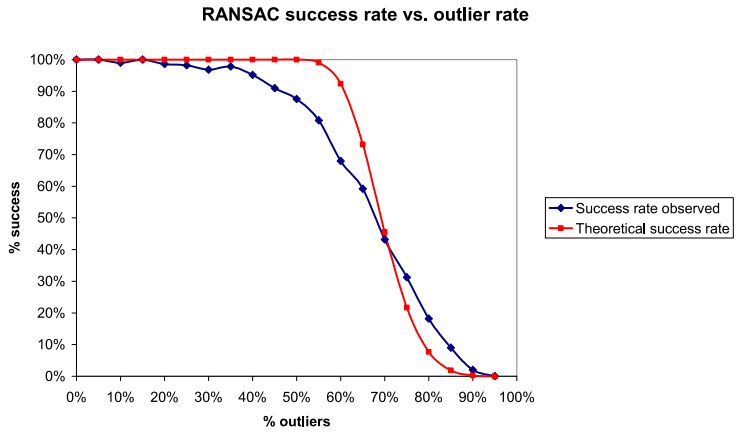


Figure 6.6: Graph showing the success rate of RANSAC with different outlier rates. 200 correspondences are simulated with localisation errors of 0.0025 radians, and RANSAC is terminated after 250 iterations. A solution with an inlier-rate of 10% is considered successful. RANSAC's success rate falls as the outlier rate increases, roughly in proportion to the probability of observing an all-inlier hypothesis set, although at higher outlier rates, models close-enough to match many inliers are often found by chance.

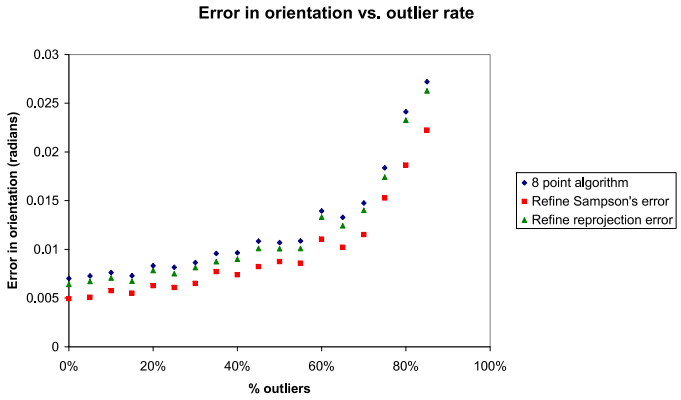


Figure 6.7: Errors in orientation increase only slowly with increasing outlier rates, as when RANSAC succeeds, remaining outliers have a minimal effect on the solution. 200 correspondences are simulated with localisation errors of 0.0025 radians, and with a maximum 250 iterations. A solution with an inlier-rate of 10% is considered successful.

Table 6.1: Times taken to compute E and refine relative pose (single thread). Times for refinement are parametrisation dependent, and refinement to minimise reprojection error (marked *) would benefit significantly from a Levenberg-Marquardt implementation using sparse matrices, however this would still be more expensive than other least-squares algorithms.

Number of points	5/7	15	50
Five-point algorithm	0.10ms	-	-
Seven-point algorithm	0.07ms	-	-
Eight-point algorithm	-	0.10ms	0.16ms
Refine Sampson's Error	-	0.14ms	0.24ms
Refine Reprojection Error	-	3.0ms*	57ms*
100-iteration RANSAC/BaySAC	-	12ms	12ms

is dominated by the cost of computing hypothesis sets using the five point algorithm, so minimising the cost of this algorithm, or the number of RANSAC iteration needed, is important.
Fischler and Bolles (1981) recommend using minimal hypothesis sets, and the five-point algorithm uses the minimum number of correspondences from which a discrete set of essential matrices can be computed, however other algorithms could also be used, for example the seven-point algorithm (Hartley and Zisserman, 2003, Chapter 11). The seven-point algorithm finds one or three essential matrices and is widely used for RANSAC hypothesis generation (e.g. by Torr and Murray, 1997; Mouragnon et al., 2009; Ogale, 2010), so timings for this algorithm (using the implementation from the OpenCV Computer Vision Library, n.d.) are also given. While slightly faster than our five-point implementation, the probability of a seven-point hypothesis set containing only inliers is usually considerably lower than for a five-point hypothesis set, so use of the seven-point algorithm would only be worthwhile when outlier rates were very low (for example if less than 15% of correspondences were believed to be outliers). Alternatively, a gradient descent-based five-point solver could potentially be more efficient than the closed-form approach used, although finding all minima (corresponding to up-to 10 possible essential matrices) may be challenging.

6.4.3 Quantifying errors in camera motion

Once the relative motion between two frames has been estimated, an estimate of its uncertainty is often needed, for example to choose the best of multiple relative position estimates, to integrate multiple relative position estimates, or to integrate a relative position estimate with a measurement from a different sensor. Position estimates and their uncertainty are very often expressed as a multivariate Gaussian distribution (i.e. a position and orientation estimate with an estimated covariance matrix).
In the case of a simple linear model fitted to i.i.d. measurements with Gaussian errors, the variance of the estimated model is proportional to the variance of the measurements, σ^2, divided by the number of degrees of freedom in the measurements. As relative pose estimation can be approximated locally by a linear model then it is reasonable to expect errors to vary in proportion to $\frac{\sigma}{\sqrt{N-(n-1)}}$, where N is the number of inliers and n is the minimum number of points required to fit a model (5 for the case of essential matrices). This model is a good approximation to the errors observed in Figures 6.4 and 6.5. By fitting the model to the errors observed when minimising Sampson's error, the following approximations are obtained: angular errors in orientation approximated by $\sigma_R \approx \frac{28\sigma}{\sqrt{N-4}}$, and angular errors in translation direction are approximated by $\sigma_{\mathbf{t}} \approx \frac{40\sigma}{\sqrt{N-4}}$.
If the estimated magnitude of the translation, $\|\mathbf{t}\|$, is also normally distributed, with $\|\mathbf{t}\| \sim N(s, \sigma_s^2)$ (Figure 6.8), a covariance matrix for the camera translation can be fitted. Covariance matrices have three perpendicular eigenvectors, with eigenvalues equal to the variance in each direction. This enables a covariance matrix for a translation estimate to be computed from its variance in three perpendicular directions, which include the direction of the greatest and smallest variance, as follows. A set of three orthonormal

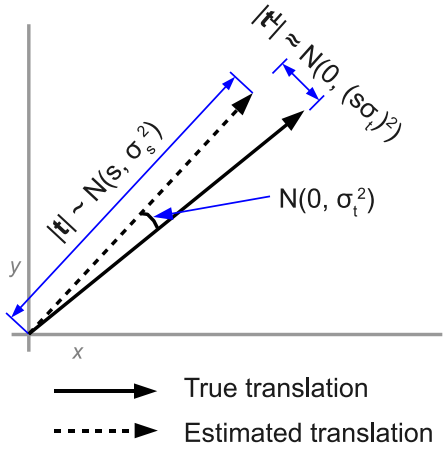


Figure 6.8: Diagram showing how errors in translation direction are related to errors in translation once the magnitude of the translation is known. The uncertainty in the estimated translation is estimated from the uncertainty in translation direction, and the uncertainty in the translation's magnitude (the third dimension is not shown).

vectors including $\mathbf{t}$, $\{\mathbf{t}, \mathbf{t}_1^\perp, \mathbf{t}_2^\perp\}$, is chosen. Expected errors in the directions $\{\mathbf{t}_1^\perp, \mathbf{t}_2^\perp\}$ are approximately $\sigma^\perp = s\sigma_\mathbf{t}$ (Figure 6.8). This breakdown of errors allows a covariance matrix $\mathbf{C}$ to be constructed from its eigendecomposition:

$$\mathbf{C} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T \tag{6.4.3}$$

$$\text{where } \mathbf{Q} = \begin{bmatrix}\mathbf{t} & \mathbf{t}_1^\perp & \mathbf{t}_2^\perp\end{bmatrix} \tag{6.4.4}$$

$$\text{and } \mathbf{\Lambda} = \begin{bmatrix}\sigma_\parallel & 0 & 0 \\ 0 & \sigma_\perp & 0 \\ 0 & 0 & \sigma_\perp\end{bmatrix} \tag{6.4.5}$$

This error model is very simple, and ignores many significant correlations (such as the strong correlation between errors in pitch/yaw and errors in motion parallel to the image plane), however this single measurement of orientation uncertainty, and a covariance matrix for uncertainty in translation, are all that are needed for BoWSLAM, which uses a pose graph optimisation based on the TORO framework (Grisetti et al., 2007a). Errors caused by the condition of the problem are also ignored: one particular source of error is the large uncertainties in estimating the translation direction when observed points are all far away. For a small cloud of points, errors are low for a range of point depths up-until a certain threshold, after which errors grow rapidly. This model would be hard to fit to real data. Similarly, reconstruction is ill-conditioned if all point matches fall in a narrow strip of an image (as there is little overlap with a neighbouring image). For a related problem in GPS, where visible satellites are close together in the sky, a measure known as dilution of precision is used to estimate errors (Langley, 1999). Dilution of precision measurements are often fitted to observed errors as we have done here. Dilution of precision is used by Kelly (2003) to estimate robot localisation errors with respect to measurements of pairs of landmarks; this procedure could potentially be adapted to larger numbers of points to improve these error estimates.

An alternative way to compute covariances of relative poses is to use the condition of the Jacobian matrix from fitting a transformation to inliers (Konolige et al., 2009; Eade and Drummond, 2007). When relative poses are computed using RANSAC however, errors in position and orientation are usually caused by remaining outlier correspondences, either from movement in dynamic environments, or when a poorly-conditioned or small inlier set is found. We have found the Jacobian is often better conditioned when more outliers are

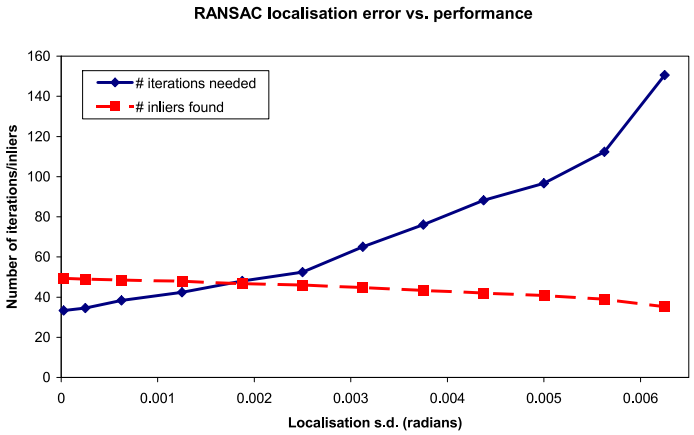


Figure 6.9: The number of iterations required by RANSAC to fit a model to 100 simulated correspondences, of which 50% are inliers, with different localisation errors. There are 50 inliers in 100 data points, and the inlier threshold is tuned to find a solution containing no more than 5% outliers on average.

present, so is a poor measure of the quality of a transformation. A potentially more accurate, although also potentially costly, approach would be to estimate the covariance for each transformation when it is computed, by propagating small changes in point locations through the relative pose computation, and observing the change in translation and orientation.

6.4.4 Effects of feature localisation accuracy on the cost of RANSAC

Feature localisation errors do not only increase errors in models computed from feature locations, they also increase the cost of computing these models by RANSAC. Figure 6.9 shows that higher feature localisation errors increase the time needed to find inliers, and decrease the number of inliers found. For each value of the localisation error, the inlier threshold (for Sampson's error) is tuned to minimise the time taken to find a hypothesis set, subject to the inlier set containing less than 5% outliers (values close to 0.02 radians are found in all cases). In the example in Figure 6.9, an all-inlier hypothesis set will be found in 32 iterations on average, however as localisation errors increase, errors in the models fitted to these all-inlier hypothesis sets increase, which reduces the number of inliers compatible with them.

Chapter 4 presented results from testing feature detectors on synthetic images for which features located using the Harris corner detector, with subpixel refinement have localisation errors with a standard deviation of 0.25 pixels. Similarly, DoH SURF features had localisation errors of around 0.39 pixels. These are probably underestimates, as real features arise due to 3D objects which appear differently from different viewpoints, and because features arise around slow-moving occlusion boundaries or objects.

Table 6.2 shows the effect of localisation errors of this magnitude in images of different sizes. In high resolution images, point localisation errors will have a negligible effect on the cost of RANSAC. In lower resolution images, as used by many SLAM schemes (Davison, 2003; Civera et al., 2009b), localisation errors from applying RANSAC would increase the number of RANSAC iterations required by possibly 20%.

While often very effective for object recognition, features sampled systematically from the image are unlikely to be usable for relative pose computation in a RANSAC framework—if 1000 features were extracted from a 1000 pixel-per-radian image, and features were matched to the closest corresponding feature in the other image, the localisation errors would be approximately 0.034 radians, however refining the positions of matched features may improve the localisation accuracy.

An additional source of localisation errors is camera calibration. While negligible for high-quality cameras,

Table 6.2: Feature localisation errors in images of various resolutions (from results in Chapter 4).

Feature detector	Harris+subpix	DoH SURF blobs
High-res image (1000 pixels/radian)	0.00025	0.00039
Low-res image (300 pixels/radian)	0.00075	0.0012
1000-pixel panoramic image	0.0015	0.0025

low-cost cameras require careful calibration to keep localisation errors low—in Section 8.5.2, even a fourth-order radial-distortion correction leaves errors of 0.015 radians near to the image's edge.

6.5 Conclusions

In this chapter, a robust procedure to estimate the relative pose of two cameras from correspondences contaminated with gross outliers has been described. Inliers are identified using RANSAC, remaining outliers are removed by top-down refinement, the relative pose of the cameras is refined to minimise Sampson's error, and the errors in the relative pose computation are quantified by fitting a model to simulated data.
In Chapter 4, the localisation errors of various corner detectors was measured. While high localisation errors could increase the cost of relative pose computation, the accuracy of these detectors is sufficient for this to be unlikely to be a problem.
As part of a real-time system, a major limitation of this combination of algorithms is its high computational cost, which is dominated by the cost of the large number of RANSAC iterations needed in some circumstances. In the following chapter a modification to the hypothesis set selection stage of RANSAC is proposed in order to reduce this cost.

6.5.1 Further work

This section concentrated on two-view relative pose computation, and in some cases multi-view methods would be more appropriate, and could result in lower errors, however BoWSLAM does not currently use multi-view methods to compute relative pose. The largest errors from pose computation are seen when reconstructing trajectories with rapid cornering; in this situation frames may not overlap sufficiently for multi-view methods.
Many other implementation choices are possible, for example Mouragnon et al. (2009) follow RANSAC with top-down refinement using a nonlinear optimisation rather than the 8-point algorithm; this would be worth investigating in order to reduce errors from refinement, as the additional cost would be marginal. Other improvements we plan to investigate include improving relative pose accuracy by actively searching for extra correspondences when these are needed (as low inlier rates are very often associated with gross errors in pose), and minimising robust functions of Sampson's error (Trivedi, 1987) as an alternative to top-down outlier removal.

Chapter 7

The BaySAC sampling strategy for speeded-up RANSAC

Abstract

RANSAC (Random Sample Consensus) is a popular algorithm in computer vision for fitting a model to data points contaminated with many gross outliers. Traditionally many small hypothesis sets are chosen randomly; these are used to generate models and the model consistent with most data points is selected. Instead we propose that each hypothesis set chosen is the one most likely to be correct, conditional on the knowledge of those that have failed to lead to a good model. Two novel algorithms are described, BaySAC and SimSAC, to choose this most likely hypothesis set. Both algorithms outperform previous improved sampling methods on both real and synthetic data, sometimes halving the number of iterations required. In the case of real-time essential matrix estimation, BaySAC can reduce the failure rate by 78% with negligible additional cost.

7.1 Introduction

RANSAC (Random Sample Consensus; Fischler and Bolles, 1981) is a popular algorithm in computer vision for fitting a model to a set of data points contaminated by outliers. A minimal number of points necessary to generate a model is randomly selected (a hypothesis set), a model is generated from these points, and the number of other points consistent with this model are counted. This is repeated for many randomly selected hypothesis sets until a model consistent with a sufficient number of data points is found.

In the previous chapter, we described how RANSAC is used in BoWSLAM for computing relative camera poses. RANSAC can be computationally expensive however, due to the large number of iterations which are often required. In this chapter we propose improvements to the hypothesis set selection stage of RANSAC, in order to reduce the number of iterations needed.

Traditionally, hypothesis sets are chosen randomly. Instead, at each time we select the hypothesis set most likely to lead to a good model. This deterministic selection process uses information gained by having tried and failed to find a good model from previous hypothesis sets, together with any prior information we might have on inlier probabilities. This innovation reduces the number of iterations needed to find a good model, hence reducing the computational cost. For real-time applications where the number of iterations is bounded, the probability of failing to find a model is reduced. Results in this chapter were originally published by Botterill et al. (2009b).

This chapter is organised as follows: Section 7.2 describes RANSAC, its applications in computer vision, and previous attempts to improve hypothesis set sampling, Section 7.3 describes our new algorithms for generating hypothesis sets, Section 7.4 presents results on simulated and real data, and Section 7.5 discusses these results.

D	Number of data points
n	Hypothesis set size
$\mathbb{I}$	Set of all inliers
t	Time (number of sets that have been tested)
H_i	Hypothesis set of n data points used at time i
$\mathbb{H} = \{H_1, H_2, ..., H_t\}$	History of failed hypothesis sets
$\Pi = \{\pi_1, \pi_2, ..., \pi_D\}$	Prior inlier probabilities

Table 7.1: Notation

7.2 Background

The original RANSAC algorithm (Fischler and Bolles, 1981) proceeds as follows:

1. **Hypothesise:** Randomly select n data points, where n is the minimum number needed to generate a model. Generate model from these points.

2. **Test:** Count the number of data points consistent with this model.

3. Repeat until a model consistent with a large number of data points is found. If the model is correct these data points are inliers.

In this chapter we select hypothesis sets of size n from a set of D data points. $\mathbb{I}$ is the set of all inliers that we aim to find, and H_t the hypothesis set at iteration t. $\Pi = \{\pi_1, \pi_2, ..., \pi_D\}$ denotes the prior inlier probabilities of each data point. This notation is summarised in Table 7.1.
There are many applications for RANSAC, these include finding the homography relating two views of a planar scene for image mosaicing (Chapter 10.1), fitting 3D models to 3D point clouds from a laser scanner (Schnabel et al., 2007), robot localisation from ambiguous landmark observations (Yuen and MacDonald, 2005), and linear regression (Rousseeuw and Leroy, 1987, for example to find a ground plane). As described in the previous chapter, one of the best known applications is for simultaneously finding the fundamental matrix (or essential matrix in the case of calibrated cameras) relating correspondences between two images with the relative pose of the cameras, and to identify and remove bad correspondences.
Several variations on RANSAC have been proposed, where RANSAC is run for a very large number of iterations, or on data when outlier rates are low. In this situation, many all-inlier sets are found, and the best of these inlier sets is selected. However, as shown in the previous chapter, and by Lacey et al. (2000) and Chum (2005), an approximately-correct inlier set can be improved efficiently using a top-down approach. In practice this is considerably faster than RANSAC, so for real-time applications, RANSAC should only be used to find an approximately-correct inlier set as quickly as possible, and this solution should be refined.
Examples of these RANSAC variations include MLESAC (Maximum Likelihood Estimation Sample Consensus; Torr and Zisserman, 2000), where the model with the highest estimated likelihood is selected, rather than the one with the highest inlier count, and LMS (Least-Median of Squares; Rousseeuw and Leroy, 1987), where the model minimising the median error of the data is selected. Both of these algorithms are variations on the model selection stage of RANSAC. Results in this chapter apply only to the hypothesis generation stage, so are equally valid for MLESAC and LMS.

7.2.1 Previous hypothesis set selection schemes

The computational cost of RANSAC is proportional to the number of iterations, which is the number of hypothesis sets that are chosen before a good enough model is found. The original RANSAC hypothesis set sampling strategy assumes that all data points are equally likely to be inliers, and hence all hypothesis sets are equally likely to contain only inliers, so the order in which they are considered is irrelevant. In practice however, the probability of data points being inliers can often be estimated prior to carrying out RANSAC.

When matching descriptors between images for example, correspondences are less likely to be correct if there are other potential matches with points that appear almost as similar. This information can be used to assess the relative likelihood of hypothesis sets. To minimise the time taken to find an all-inlier set leading to a good model, hypothesis sets should be evaluated in order of decreasing likelihood. Alternatively, the same ordering will also maximise the probability of finding an all-inlier set for real-time applications, when the time available is bounded.

The Guided-MLESAC algorithm (Tordoff and Murray, 2005) uses a function of feature correlation strength (derived by fitting a heuristic model to test data) as a measure of prior inlier probability. When choosing random hypothesis sets each correspondence is chosen with selection probability proportional to its prior probability. Together with other innovations this is shown to reduce the time to find a model.

Chum and Matas (2005) present an alternative sampling strategy known as PROSAC (Progressive Sample Consensus) based on the same idea. Data points are sorted by some measure of inlier likelihood, avoiding the need to estimate actual probabilities. First the n most likely data points are chosen, then the point $n+1$ and the $\binom{n}{n-1} = n$ possible sets of $n-1$ data points of the first n in some random order, and so-on. This allows hypotheses to be chosen in approximate order of their prior likelihood. In one feature-matching experiment, where a small number of data points are very likely to be inliers, and a very large number are not, PROSAC reduces the number of iterations needed by a factor of 10,000.

These sampling strategies are heuristic and are based on data point inlier probabilities rather than the likelihood of hypothesis sets consisting entirely of inliers. In Section 7.3 we describe our new sampling algorithms which combine inlier probabilities with information from previous outlier-contaminated hypothesis sets to allow the (usually) deterministic choice of the most likely hypothesis set on each iteration, and which determine the best order in which hypothesis sets should be tried.

7.3 Proposed conditional sampling methods

Previous sampling methods fail to take into account the information gained by testing hypothesis sets and finding them to be contaminated by outliers. In this chapter we propose two methods to take this into account, based on the following observation: given a hypothesis set H that leads to a model that is consistent with few data points, this is assumed to be because it contains one or more outliers (the possibility that it contains a degenerate configuration of inliers will be considered later). Trying the same set again is a waste of time (although this is unlikely to happen with any reasonable number of likely data points). However, hypothesis sets with one or more data points in common with H are also now less likely, as they are likely to include the same outlier(s) that contaminated H. Note that this is true for any set of prior probabilities, in particular the original RANSAC random sampling algorithm which assumes constant prior probabilities.

Ideally, at every iteration the hypothesis set that is *most likely to contain no outliers* based on the prior probabilities Π and the history of contaminated samples $\mathbb{H}$ will be chosen. Unfortunately a closed-form solution for this posterior probability is algebraically intractable, even for the simplest non-trivial case $n = 2$ (as every intersecting pair of contaminated hypothesis sets introduces covariances between the inlier probabilities of all of their data points). Instead we present two methods of approximating this probability, both of which are shown to work well in practice. The aim of each of these methods is to choose H_t at time t such that $P(H_t|\mathbb{H}, \Pi) \geq P(H|\mathbb{H}, \Pi) \; \forall \, H$. This strategy minimises the number of hypotheses that must be tested before finding one consisting entirely of inliers.

7.3.1 Naïve Bayes method—BaySAC

The first proposed method, BaySAC, assumes independence between inlier probabilities of data points in the same hypothesis set. On the first iteration, the n data points with the highest inlier probabilities are selected as the hypothesis set. This is the most likely under the independence assumption. After testing a hypothesis set H_t and finding that it is contaminated with outliers, the inlier probabilities of each data point is updated using Bayes' rule. The same procedure is repeated to select the next hypothesis set, using the

new inlier probabilities. Probabilities are updated as follows:

$$\begin{aligned} P_t(i \in \mathbb{I}) &= \text{Inlier probability for data point } i \text{ at time } t \\ &= \begin{cases} \frac{P_{t-1}(i \in \mathbb{I}) P(H_t \not\subseteq \mathbb{I} | i \in \mathbb{I})}{P(H_t \not\subseteq \mathbb{I})} & i \in H_t \\ P_{t-1}(i \in \mathbb{I}) & i \notin H_t \end{cases} \qquad (7.3.1) \\ \text{where } P(H_t \not\subseteq \mathbb{I}) &= 1 - P(H_t \subseteq \mathbb{I}) = 1 - \prod_{j \in H_t} P_{t-1}(j \in \mathbb{I}) \qquad (7.3.2) \\ P(H_t \not\subseteq \mathbb{I} | i \in \mathbb{I}) &= 1 - P(H_t \subseteq \mathbb{I} | i \in \mathbb{I}) = 1 - \prod_{j \in H_t : j \neq i} P_{t-1}(j \in \mathbb{I}) \qquad (7.3.3) \\ &= 1 - P(H_t \subseteq \mathbb{I}) / P_{t-1}(i \in \mathbb{I}) \qquad (7.3.4) \\ \Rightarrow P_t(i \in \mathbb{I}) &= \begin{cases} \frac{P_{t-1}(i \in \mathbb{I}) - P(H_t \subseteq \mathbb{I})}{1 - P(H_t \subseteq \mathbb{I})} & i \in H_t \\ P_{t-1}(i \in \mathbb{I}) & i \notin H_t \end{cases} \qquad (7.3.5) \end{aligned}$$

BaySAC uses these equations to update inlier probabilities:

Do

1. Choose n data points most likely to be inliers
2. Use Equations 7.3.2 and 7.3.5 to update the inlier probability of these points

until a sufficiently large inlier set is found

BaySAC is fast, and works well when there are few large intersections between sets. It works poorly in some cases when there are a small number of data points of which many are of very similar or equal probability, as following a Bayes update they're still equiprobable, and are therefore likely to be selected in the same hypothesis set again. This is not a problem for larger D as long as equiprobable data points are selected at random when they make the hypothesis set choice ambiguous.

7.3.2 Simulation method—SimSAC

An alternative way to compute data point inlier probabilities is by simulation. Choose random inlier/outlier statuses for each data point, check whether these statuses are compatible with having failed to observe an inlier set so far, and if so, accumulate a histogram of inlier counts for each of the D data points. The n highest peaks in this histogram form the most likely inlier set. SimSAC proceeds as follows:

1. Do

   - Simulate random inlier/outlier statuses $x_1, ..., x_N \in \{0, 1\}$ based on prior inlier probabilities (so that $P(x_i = 1) = \pi_i$).
   - If $\forall\, H \in \mathbb{H}\ \exists\ h \in H$ such that $x_h = 0$ then increment a counter for each simulated inlier.

   until we have found T compatible simulated sets.

2. Choose the n data-points that were most frequently inliers as our next hypothesis set.

SimSAC works by sampling from the prior distribution of inlier/outlier status vectors, then updating this sample conditional on observing hypothesis sets that contain outliers (indicating that many possible status vectors were not correct). Finding the peaks in a histogram from accumulating these status vectors gives us the n data points that are individually most likely to be inliers, conditional on the failed hypothesis set history. Assuming independence between inlier probabilities for the most likely inliers at time t (a much weaker assumption than the Naïve Bayes assumptions in BaySAC), this algorithm will give us the actual

most likely hypothesis set at each time, as $T \to \infty$. A simulation scheme avoiding this assumption would be to find the inlier/outlier status vector that is most frequently observed conditional on it being drawn from the prior distribution, and being compatible with the observed history. An implementation of this scheme as described would take time and memory $\Omega(exp(D))$, as a large fraction of possible status vectors (of which there are 2^D) must be simulated many times, which would be feasible for only the smallest problems.
While SimSAC is much slower than other approaches, and still has high complexity (Section 7.4.3), for some practical applications it may be considerably faster than model generation. For these applications, choosing $T = 10$ samples provides a method to choose one of the more likely hypothesis sets. A value such as $T = 1000$ provides a 'ground truth' reference—it is unlikely any feasible algorithm can perform substantially better than this.

7.3.3 N–M correspondences

When searching for correspondences between two frames, multiple (N) features in one image often appear sufficiently similar to multiple (M) features in the other that they could be the same feature. The standard approach to this situation is to limit the image distance between features in the two images, and/or to reject all correspondences with multiple similar-looking potential matches (Torr and Murray, 1997; Nistér et al., 2006; Brown et al., 2004). The problem with this approach is that in visually poor or self-similar environments too few good correspondences may remain for scene geometry to be recovered accurately, if at all. Also limiting track lengths introduces difficulties in registering frames during rapid camera motion or following loop closure. Figure 7.1 shows two images where 2–2 and 3–3 correspondences help to obtain enough inliers over a wide enough area to recover camera motion. As a result, visual navigation systems are rarely demonstrated in highly self-similar, or visually poor environments.
A better solution to this problem is to introduce correspondences between all possible pairs of points. This leads to $N \times M$ correspondences of which at most $min(N, M)$ are correct. If $N \leq M$, and the probability of one of these points i in the first image matching any of the M points in the second is p, then the probability that correspondence (i, j) is an inlier is assumed to be p/M. In general, the probability that (i, j) is an inlier is $p/max(N, M)$.
To make use of these probabilities, a simple adaption to avoid choosing incompatible correspondences is made: if (i, j) is an inlier, (i, k) and (l, j) are not inliers $\forall\, k, l$. Note that unlike other sampling strategies (RANSAC and Guided-MLESAC) it does not matter that many correspondences with low probabilities are introduced, as they will only be selected as part of hypothesis sets once sufficient evidence has been observed to indicate they are more likely to be correct than correspondences with higher prior probabilities.
If $(i, j) \in H_t$, and H_t is contaminated by outliers, the posterior probabilities of $\{(i, k), (l, j)\ \forall\, k, l\}$ are slightly increased, as these inlier probabilities are disjoint. The probability of other correspondences that are incompatible with $\{(i, k), (l, j)\}$ should also be slightly reduced but this amount is negligibly small. Equation 7.3.5 is adapted to take this into account as follows:

$$\begin{aligned} P_t((i,k) \in \mathbb{I} | (i,j) \in H_t) &= P((i,k) \in \mathbb{I} | (i,j) \in \mathbb{I}) P((i,j) \in \mathbb{I} | (i,j) \in H_t) && (7.3.6) \\ &+ P((i,k) \in \mathbb{I} | (i,j) \notin \mathbb{I}) P((i,j) \notin \mathbb{I} | (i,j) \in H_t) \\ &= \frac{P_{t-1}((i,k) \in \mathbb{I})}{(1 - P_{t-1}((i,j) \in \mathbb{I}))} (1 - P_t((i,j) \in \mathbb{I})) && (7.3.7) \\ \text{because } (i,j) \in \mathbb{I} &\Rightarrow (i,k) \notin \mathbb{I} \Rightarrow P((i,k) \in \mathbb{I} | (i,j) \in \mathbb{I}) = 0 && (7.3.8) \end{aligned}$$

7.4 Results

This section presents results to validate SimSAC and BaySAC using both simulated data and real data (for the case of essential matrix estimation). When considering N–M correspondences, all algorithms are modified to avoid selecting hypothesis sets which cannot possibly contain only inliers: correspondences are not selected if a previously selected correspondence shares an endpoint.

Figure 7.1: Frames from a video sequences used for essential matrix estimation with insufficient 1-1 correspondences for these to be used alone. Of the 6 potential correspondences shown at most 3 are correct.

7.4.1 Results on simulated data

To test our new sampling algorithms, the inlier/outlier statuses of D data points are simulated with some prior probabilities Π. We then sample repeatedly using one of our sampling strategies until a set consisting entirely of inliers is found. Table 7.2 shows the mean number of iterations (and a 99% confidence bound on this mean) needed to find a hypothesis set containing only inliers, using 50 datapoints, and choosing sample sets of size 5.
Three prior probability distributions are used:

Constant(p) Original RANSAC—prior probabilities all equal a constant p ($p = 0.5$ is shown in Table 7.2, similar results are observed for $p = 0.25$ and $p = 0.75$).

Unif(a, b) Prior probabilities are uniformly distributed in the range (a, b). Results using Unif(0.25, 0.75) are shown in Table 7.2—this distribution may apply to the situation where correspondences are correct with probabilities of up to 0.75 if the feature matcher response is strong, together with a lower bound on allowed match quality. Very similar results are observed for probabilities from Unif(0, 1).

N-N(p, N) Prior probability of one data point having a match is constant ($p = 0.6$ for these experiments), but sets of k points each have k match candidates ($P(\text{match}) = p/k$, $k \sim \text{Unif}[1, ..., N]$).

Additionally, as prior probabilities are hard to estimate in advance (Chum and Matas, 2005), simulations are carried out where true prior probabilities are uniformly distributed about the estimated values from 0.25 below to 0.25 above (subject to staying within $(0, 1)$). There is also a possibility that degenerate configurations, incorrectly assumed to contain outliers, will cause a sampling strategy to fail. To simulate this, all-inlier hypothesis sets are assumed to fail with probability 0.25. The number of iterations needed under these assumptions are shown in column 'Uncertain Π' of Table 7.2.
Table 7.2 only shows results for the values $D = 50$ and $n = 5$. Results are comparable for values of D from 15 to 1000 and n from 2 to 10; Figure 7.2 shows the consistently strong performance of BaySAC and SimSAC for values of D from 15 to 100, and $n = 8$, with prior probabilities from N-N$(0.8, 2)$. Note the periodicity in the BaySAC performance—when n divides D (or they have a high common factor) sets of n equiprobable data points remain equiprobable, and hence are selected at the same time again following a Bayes update. Run times are given–these are only directly comparable for a particular number of iterations (before either succeeding or terminating) as different sampling strategies have different complexity, as discussed in Section 7.4.3. In these simulations RANSAC is terminated after 250 iterations if no inlier set is found (termination is essential for real-time applications), and the success rate is recorded.
The results presented include the mean number of samples drawn when a hypothesis set is found, a 99% confidence bound on this mean and the proportion of samples where no inlier set was found. SimSAC with $T = 1000$ outperforms all other methods, but is considerably more costly. BaySAC performs almost as well, and performs considerably better that Guided-MLESAC sampling, RANSAC sampling or PROSAC sampling

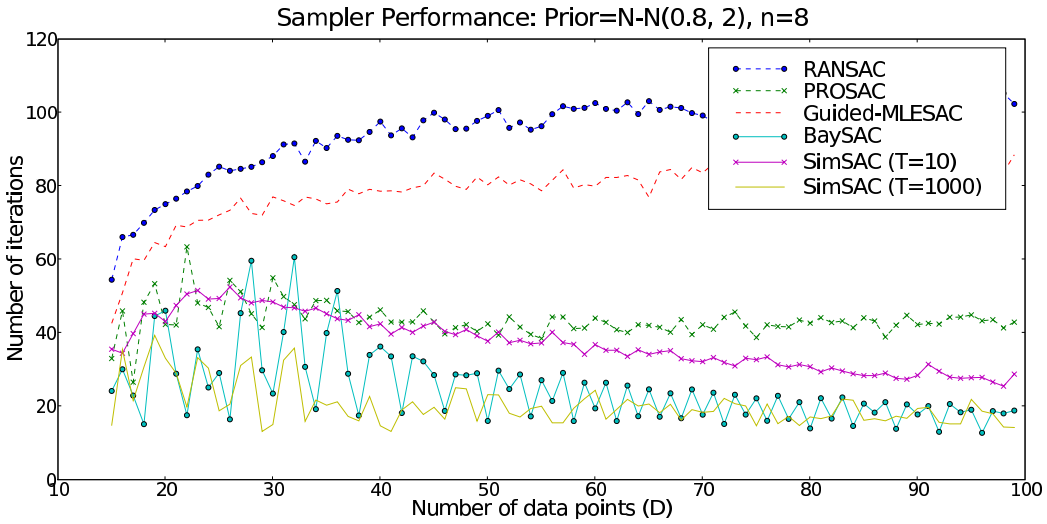


Figure 7.2: BaySAC and SimSAC with $T = 1000$ reduce the number of iterations needed to find an all-inlier hypothesis set.

Sampling algorithm	Prior probabilities	Mean iters. (99% bound)	Success (%)	Time ($\times 10^{-6}$s)	Mean iters. Uncertain Π
RANSAC	Constant(0.5)	43.28 ± 0.16	96	0.34	51.77 ± 0.18
PROSAC	Constant(0.5)	66.85 ± 0.29	53.2	0.447	70.1 ± 0.3
Guided-MLESAC	Constant(0.5)	43.14 ± 0.16	95.9	0.661	51.57 ± 0.18
BaySAC	Constant(0.5)	41.74 ± 0.16	96.2	1.04	50.41 ± 0.18
SimSAC ($T = 10$)	Constant(0.5)	42.76 ± 0.16	96	7.26	50.82 ± 0.18
SimSAC ($T = 1000$)	Constant(0.5)	$\mathbf{41.71 \pm 1.1}$	96.2	579	$\mathbf{47.93 \pm 3.9}$
RANSAC	Unif(0.25, 0.75)	43.34 ± 0.16	96	0.341	51.8 ± 0.18
PROSAC	Unif(0.25, 0.75)	30.96 ± 0.16	90.9	0.625	34.21 ± 0.17
Guided-MLESAC	Unif(0.25, 0.75)	32.39 ± 0.13	98.2	0.687	40.03 ± 0.15
BaySAC	Unif(0.25, 0.75)	18.99 ± 0.12	96.4	1.42	23.37 ± 0.14
SimSAC ($T = 10$)	Unif(0.25, 0.75)	21.51 ± 0.12	98.2	6.47	27.86 ± 0.14
SimSAC ($T = 1000$)	Unif(0.25, 0.75)	$\mathbf{16.47 \pm 0.68}$	99	479	$\mathbf{19.9 \pm 2.3}$
RANSAC	N-N(0.6, 3)	109.1 ± 0.43	29.5	0.386	112.7 ± 0.48
PROSAC	N-N(0.6, 3)	48.61 ± 0.29	40.6	0.687	51.86 ± 0.31
Guided-MLESAC	N-N(0.6, 3)	96.32 ± 0.35	44.5	1.07	101.2 ± 0.38
BaySAC	N-N(0.6, 3)	58.8 ± 0.38	40.9	3.59	57.61 ± 0.38
SimSAC ($T = 10$)	N-N(0.6, 3)	61.7 ± 0.29	56.3	10.1	65.82 ± 0.31
SimSAC ($T = 1000$)	N-N(0.6, 3)	$\mathbf{19.65 \pm 1.4}$	31.9	509	$\mathbf{28.79 \pm 6.1}$

Table 7.2: Results on simulated data with $D = 50$ and $n = 5$.

Figure 7.3: Frames from the indoor and outdoor sequences used for essential matrix estimation

Sampling algorithm	Outdoors		Indoors	
	Success rate (s.d.)	Inliers	Success rate (s.d.)	Inliers
RANSAC	144.0/203 (1.0)	151	16.7/19 (0.5)	131
PROSAC	179.3/203 (5.0)	130	17.0/19 (0.0)	130
Guided-MLESAC	186.3/203 (3.4)	129	16.3/19 (2.6)	124
BaySAC	189.0/203 (0.0)	130	18.3/19 (1.1)	129
SimSAC (10)	185.0/203 (2.0)	130	17.7/19 (1.1)	122
SimSAC (1000)	**191.3/203** (1.9)	131	**18.7/19** (0.5)	131

Table 7.3: Essential matrix estimation—$N-M$ correspondences from pairs of consecutive frames from indoor and outdoor video sequences

in most cases. Inaccurate prior probability estimates have surprisingly little effect on results. Improvements from our new algorithms are marginal if prior probabilities are constant (although are more significant when D is small, e.g. $D = 20$), but large improvements are gained when prior probabilities can be estimated.
The case when BaySAC does not performs quite as well is with $N - N$ correspondences. This is the one case where there is significant room for improvement—no other algorithm comes close to matching the 20 iterations needed by the simulated sampler with $T = 1000$.
Note that PROSAC sampling often performs poorly, particularly with constant prior probabilities, where its performance is considerably worse than with random sampling. This is because it repeatedly samples from the same small set of data points, and when these are contaminated with outliers, many iterations are wasted trying small variations on contaminated hypothesis sets. PROSAC does not take into account the information provided by previous failures.

7.4.2 Results on real data

In order to evaluate our new RANSAC variations on real data we use them to find the essential matrices from the N–M correspondences ($N, M \leq 3$) between consecutive pairs of frames from a 20 frame indoor and a 204 frame outdoor video sequence (Figure 7.3). The prior probability model described earlier is used, with $p = 0.5$. Each video was processed three times, and the mean number of times a good essential matrix was found (one with at least 15 inliers, and within a threshold of the known camera motion) is given. The number of iterations is limited to 250, and in practice we rarely terminate much earlier than this (as inlier rates are not high enough). As a result, run times are almost identical (0.04s per frame pair). Table 7.3 shows that BaySAC finds the essential matrix more often than other methods, other than SimSAC with $T = 1000$, however this takes 0.3s per frame pair so is too slow for real-time use. Note that ignoring prior probabilities entirely (original RANSAC) results in poor performance.
We find around 400 potential correspondences per frame with $N, M \leq 3$. Using only 1-1 correspondences results in around 50-150 correspondences, and regularly less than 25 inliers are found from these (an average

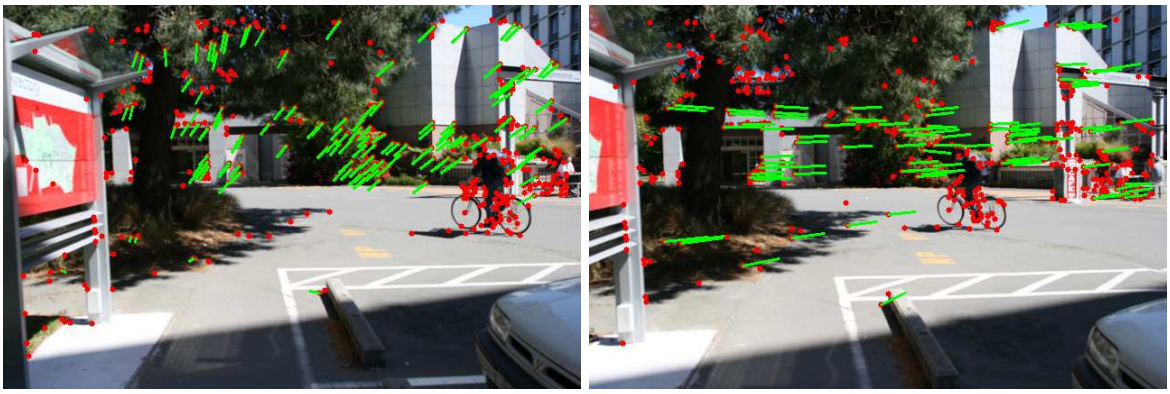


Figure 7.4: BaySAC successfully avoids outlier correspondences from moving objects. The red dots indicate detected features, and the green lines indicate the locations in the previous frame which features are matched to. Despite the high density of features on the cyclist, following BaySAC no matches are found here.

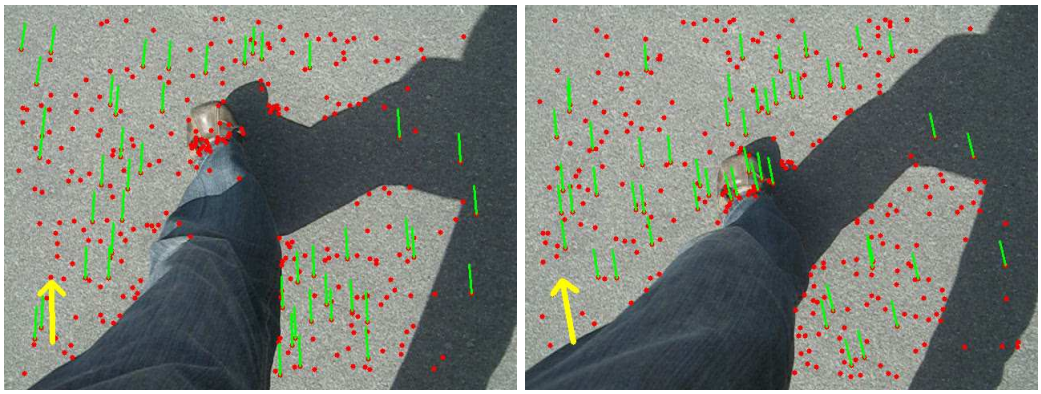


Figure 7.5: BaySAC used to estimate the perspective transformation between pairs of images of the ground. Green lines show where features were matched to in the previous frame. Stationary features are matched despite the pedestrian's moving legs and shadow. Features on the pedestrian's shoe are avoided while it is moving, then matched when it is stationary on the ground.

of 49 are found in the indoor sequence). We have found this level too low to recover an accurate camera motion estimate from these videos.

BaySAC for robust sparse optical flow computation

Another application of BaySAC is to compute the perspective homography mapping a plane visible in one image (for example the ground plane) to another. This perspective homography computation is described in detail in Chapter 10.1. One application, which is still under development, and which is described in Hide et al. (2010), is dense optical flow extraction to aid a position estimate computed from an IMU. A pedestrian carries an IMU and an approximately down-pointing camera (these sensors are increasingly found in modern mobile phones). From the perspective transformation found between a pair of frames, and the height of the camera above the ground, the velocity and orientation change of the camera can be estimated, and these can be combined in a Kalman filter with acceleration estimates from an IMU to enable accurate pedestrian navigation with low drift.
The ground plane viewed by a pedestrian is often featureless or self-similar however, and the shadow and legs of the pedestrian introduce moving objects. From $N - M$ correspondences between patch descriptors

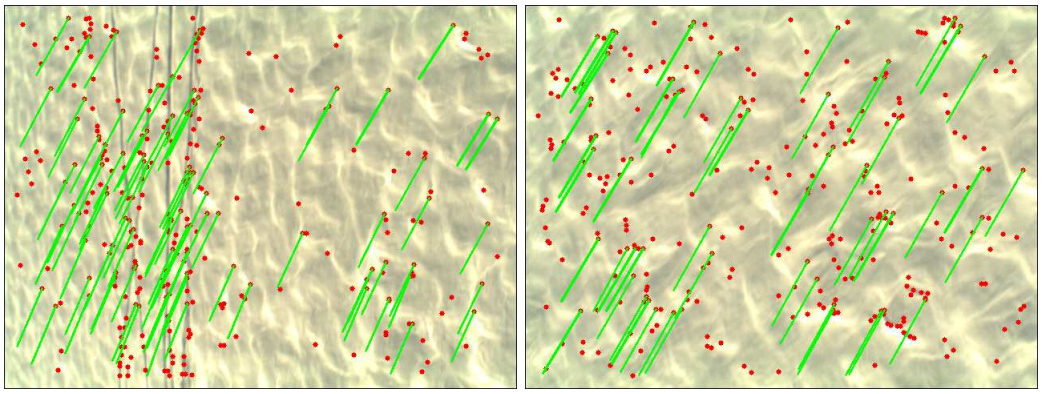


Figure 7.6: BaySAC used to estimate the perspective transformations between undistinctive images of Antarctic ice. $N - M$ correspondences between patch descriptors are used for all of these figures.

from a pair of frames however, BaySAC can successfully estimate the perspective transformation, as shown in Figure 7.5.
A related application is UAV positioning using a camera and IMU. Figure 7.6 shows that BaySAC can successfully identify inlier correspondences between photos of highly self-similar Antarctic snowfields, captured from a UAV.

Application for pedestrian navigation

Section 5.5.2 described a pedestrian navigation sensor which used the BoW algorithm to match images captured by a pedestrian to images with known locations. Incorrect matches were occasionally found however. Hide and Botterill (2010) propose to use BaySAC to estimate the essential matrix relating the image captured by the pedestrian, and its match from the database of surveyed images. This provides validation that the match is correct (if it is incorrect, it is likely that no essential matrix will be found), and provides the relative orientation of the two images, which may be used to correct heading errors in the IMU's pose estimate. In a small experiment, 500 frames captured by a pedestrian are matched to 500 frames in a heavily-overlapping 'survey' video of the same route, using the BoW algorithm. Out of 373 matches found, only one is incorrect. This is also the only pair where BaySAC cannot find an essential matrix (Figure 7.7), hence validating that the match is incorrect.

7.4.3 Run-times and complexity

Drawing a hypothesis set of size n using BaySAC has $O(n \log D)$ complexity (maintain a list of D data points, sorted by their probabilities, update the probabilities of around n points at each iteration, and select the top n). There is an initial setup cost of $O(D \log D)$.
Simulating inlier/outlier statuses for D data points takes time of $O(D)$. Testing compatibility with a history of size t takes time of $O(tn)$. If prior probabilities truly reflect the inlier/outlier probabilities then few status sets need to be generated to find one compatible with having not found an all-inlier set so far. This would give SimSAC a total complexity of $O(Dtn)$. However overly optimistic inlier probabilities lead to large numbers of status sets being rejected, greatly increasing this complexity.
In BoWSLAM, RANSAC is used for real-time essential matrix estimation. The five-point essential matrix estimation algorithm by Stewénius et al. (2006), which was described in the previous chapter, takes an average of 1.0×10^{-4}s to estimate an average of 5 matrices compatible with a hypothesis set. Validating a model for one data point takes 2.5×10^{-8}s, and typically few data points need to be compared in practice (Matas and Chum, 2005; Capel, 2005), making this a small component of the total cost. For this application,

Figure 7.7: BaySAC fails to find a match between two images of different places which were incorrectly matched using the BoW algorithm. Essential matrices are found between other matches.

BaySAC's increased sampling costs are insignificant compared to the cost of model generation and testing (taking about 1.5×10^{-6}s per sample with $D = 50$, $n = 5$; about 1% of the total cost).
For applications where model generation and verification is more expensive (for example trifocal tensor estimation; Hartley and Zisserman, 2003) minimising the number of iterations required by using SimSAC may be worthwhile, however the improvement in the case of essential matrix estimation is not great enough to justify the increased cost of generating samples.

7.5 Conclusions

BaySAC is an effective way to speed up RANSAC when prior probabilities can be estimated. In some cases, BaySAC halves the number of iterations required by simple RANSAC sampling, and it significantly outperforms other sampling algorithms based on estimated prior probabilities. Performance is good even when prior probability estimates are only approximately correct.
In the case of essential matrix estimation, BaySAC and SimSAC can make effective use of $N - M$ correspondences between sets of multiple points. This allows improved performance in visually poor and self-similar environments, decreasing the probability of failing to find an essential matrix connecting two cameras by 78% in one example. BaySAC and SimSAC can also reduce the number of iterations needed when data points are assumed equally likely to be inliers, as in the original RANSAC algorithm, although the speed-up is marginal unless model generation is particularly expensive.

7.6 Further work

In this chapter hypothesis sets of correspondences were chosen which were most likely to be correct, assuming that $1 - 1$ correspondences' inlier probabilities were independent. When matching invarient descriptors such as SURF or oriented patches however (described in Chapter 4), these probabilities are not independent, and this dependence could potentially be used to further improve BaySAC. In the case of rotation-invariant descriptors, the orientation at which a feature was extracted is known, and the differences in orientations between correctly-matched descriptors should be similar. Similarly, with illumination-invariant descriptors,

the differences in illumination between matched features are likely to be similar for all of the correct correspondences. We plan to incorporate these dependencies into BaySAC in order to increase the probability that hypothesis sets consist only of inliers.

Chapter 8

BoWSLAM: Bag-of-Words-driven Single Camera SLAM

Abstract

This chapter describes BoWSLAM, a scheme for a robot to reliably navigate and map *a priori* unknown environments, in real-time, using only a single camera. BoWSLAM can navigate challenging dynamic and self-similar environments, and can recover from gross errors. BoWSLAM combines the robust methods developed throughout this thesis with new uses for the Bag-of-Words image representation; this is used to select the best set of frames to reconstruct positions from; and to give efficient wide-baseline correspondences between many pairs of frames, providing multiple position hypotheses. A graph-based representation of these position hypotheses enables the modeling and optimisation of errors in scale in a dual graph, and the selection of only reliable position estimates in the presence of gross outliers. BoWSLAM is demonstrated mapping a 25 minute 2.5km trajectory through a challenging and dynamic outdoor environment without any other sensor input; considerably further than previous single camera SLAM schemes.

8.1 Introduction

This chapter describes BoWSLAM: a single camera SLAM scheme which enables a mobile robot moving in three dimensions to position itself in a previously unknown environment. BoWSLAM combines the new robust techniques developed in this thesis to enable the successful navigation of dynamic environments despite erratic camera motion, corrupted position estimates, total disorientation, or large accumulated errors. A map of the environment that has been explored is built and this map is refined and improved when places are re-visited. To validate this approach, BoWSLAM is demonstrated mapping a 2.5km trajectory in real-time through a dynamic environment, with no other sensor input, motion assumptions or odometry input. This chapter is based on the paper by Botterill et al. (2010).

Two additional components are described in this chapter, which are developed in order to allow the navigation of difficult environments. Firstly, the Bag-of-Words (BoW) algorithm (Chapter 5), normally used for detecting loop closure by recognising when locations are re-visited, is used in two new ways: for fast feature matching, allowing multiple position hypotheses to be generated for each camera location, and for smart selection of frames from which to compute each pose. Secondly a graph-based map of these relative position hypotheses allows the selection and optimisation of only the most reliable position estimates, and allows the perceived scale of the world to be modeled, and scale drift to be corrected, via a dual graph.

BoWSLAM is substantially different from previous single camera navigation schemes: a high framerate is not required, enabling a high-level representation of each frame to be built and multiple position hypotheses to be calculated; few assumptions about motion are made, allowing navigation despite erratic camera motion

or gross errors in dynamic environments; and a novel graphical representation of relative positions and scales allows selection of only the best position hypotheses in the presence of many gross outliers.
This chapter is organised as follows: the next section summarises contemporary approaches to navigation and SLAM using computer vision, and describes the aims of this new approach; Section 8.3 describes BoWSLAM, the new uses for the BoW algorithm, and how to use information from the additional position hypotheses it generates; Section 8.4 presents results from positioning a camera with videos from difficult indoor and outdoor environments; and the final section discusses our conclusions and future plans. Notation used in this chapter is summarised in Table 8.1.

8.2 Summary of prior research

In Chapter 3 we reviewed a selection of the many schemes which enable robots to position themselves using a single camera alone. These schemes often work in real-time, and over tracks of hundreds of metres, while often producing maps accurate to within a few percent of ground truth. There are two major limitations of these approaches however: firstly the environments explored are largely static, and rapid cornering (during which single camera geometry is poorly conditioned) is largely avoided, however typical indoor or urban environments where robots will one day operate contain moving people and vehicles; a robot moving through these environments must cope not only with these dynamic objects, but also with erratic motion including rapid cornering. To overcome these limitations, a successful SLAM scheme must function despite these challenges, and critically, must be able to recover from gross errors in positioning.
The second limitation of many of these schemes (Lemaire and Lacroix, 2007; Dellaert and Kaess, 2006; Eade, 2008; Strasdat et al., 2010) is their high algorithmic complexity—the SLAM algorithms soon become too costly for real-time operation as the mapped area grows. SLAM methods with lower cost and complexity should be used for longer term navigation.
In Chapter 5, we observed that the key strength of a digital camera as a navigation sensor is the ability to recognise when locations are re-visited, and that the BoW algorithm is generally very successful at efficiently recognising when two images show the same scene. This observation is key to the operation of BoWSLAM, which is described in the following section.

8.3 The BoWSLAM Single Camera SLAM Scheme

This section describes our new scheme, BoWSLAM. BoWSLAM uses new techniques, enabled by adding every image to a BoW database, to overcome some of the limitations of previous single camera SLAM schemes, allowing robust long-term positioning in visually challenging dynamic environments. This section first gives an overview of BoWSLAM, then describe its individual components in detail.
All previous visual navigation systems operate in basically the same way: the camera captures an image showing part of the same environment that was viewed previously; matches are found between features in the new image and the same features in previous images; the camera's new pose is calculated from these matches, based on the assumption that some of the environment is static; and finally a map is built from the observed features. BoWSLAM's basic mode of operation is exactly the same; in addition to allow long-term robot positioning in real-world environments, where moving objects and erratic camera motion make navigation difficult, we have the following requirements:

Active loop closure detection To recover from gross errors, and to maintain a consistent and accurate map for extended periods, the robot must detect when known locations are re-visited.

Wide-baseline matching The capability to register frames captured from significantly different viewpoints is essential where sequences of frames are unusable due to occlusion, motion blur, or erratic camera motion, and following loop closure.

Robust estimation Position estimation must be robust in the presence of moving objects and in self-similar environments.

Multiple position hypotheses Despite robust position estimation, errors will inevitably still occur. The ability to reject these erroneous position estimates is essential for long-term navigation.

The leading framework for active loop closure detection is the BoW algorithm, as described in Chapter 5. To use the BoW framework for loop closure detection some frames must be represented as bags-of-words. In BoWSLAM all frames are represented as BoWs, this enables wide-baseline correspondences between any pair of frames to be obtained cheaply and effectively from their BoW representations (Section 5.6). As wide-baseline matching allows us to efficiently register any pair of overlapping frames, we can use any such sequence of frames to position the camera. Frames from which to compute the relative position of the latest frame are selected efficiently using the BoW algorithm and include loop closure candidate frames from previous visits to a location, as well as the most suitable recent frames, hence allowing recovery after periods of erratic motion or occluded frames. These new applications for the BoW algorithm are described in Section 8.3.1.

When a frame is captured, feature points are detected and descriptors of the image around these points are extracted (simple image patches centred on corners found using the FAST corner detector are used, as described in Chapter 4). These descriptors are used to describe the frame as a BoW. We now use the BoW database to find both nearby frames, and feature matches with these frames.

This set of correspondences will contain many outliers, for example from self-similar features and from moving objects. Chapter 6 described how to compute and refine the relative pose of the cameras from these outlier-contaminated correspondences by combining least-squares approaches with the RANSAC framework. This approach can be slow however when outlier rates are high, so instead of RANSAC the BaySAC framework described in Chapter 7 is used; this is shown to reduce the time needed to find an inlier set.

Given these relative pose estimates between pairs of frames, we can reconstruct the camera's trajectory by adding a sequence of these relative poses together. Any errors in one pose however will also affect subsequent pose estimates. While small accumulated errors can be dealt with via the SLAM framework (Section 8.3.4), gross errors will inevitably also occur. In order to maintain consistent maps and accurate pose estimates it is essential to avoid using these erroneous pose estimates.

To eliminate erroneous pose estimates we compute the pose of each frame relative to each of several nearby frames, leading to a graph of pose estimates (Figure 8.8). Two frames can be positioned relative to each other in many different ways, via all of the different paths between them. From this graph we choose a small subset of possible position hypotheses containing only the most reliable position estimates, hence minimising the chance of incorporating erroneous position estimates into our map. This graph of multiple position hypotheses, and the dual graph in which the scale drift that occurs in single camera SLAM is modeled, are described in detail in Section 8.3.3.

Finally, in order to create a globally accurate map, we optimise a graph of independent relative position estimates using the efficient TORO framework (Grisetti et al., 2007a). The TORO framework, and the generation of the graph optimised by TORO, are described in Section 8.3.4.

In summary, BoWSLAM works by representing every frame as a BoW, then using the BoW representation to select multiple nearby frames from which to compute relative positions. The latest frame is positioned relative to each of these frames using the robust BaySAC framework together with correspondences from the BoW algorithm. A subset of the multiple position hypotheses which is unlikely to contain any gross errors is then selected and refined to accurately position the camera in a global map. An overview of the BoWSLAM algorithm is given in Figure 8.1, and the following sections describe each component in detail.

8.3.1 Feature description and the Bag-of-Words image representation

As described in Chapter 4, image patch descriptors centred on corner features provide good features for finding matches between frames, and in Chapter 5 we showed that these are also effective features for loop closure detection. We use patch descriptors centred on FAST corners, these are cheap to extract while providing almost as good localisation accuracy and repeatability as more expensive features, such as Harris corners. Patches sized 33×33 pixels are down-sampled to 11×11.

For each frame:

1. **Add new frame to BoW database:**
   - Capture new frame from camera
   - Detect corners and extract features
   - Add image to BoW database
2. **Select recent frames for relative positioning, and potential loop closure candidates. For each of these frames:**
   - Find BoW feature correspondences with current frame
   - Compute pose relative to several previous frames
   - Reconstruct 3D landmark positions (up to scale ambiguity)
   - Align landmark positions with earlier observations to resolve scale change and reconstruct camera positions
3. **Update graph to select best pose estimate for every frame**

Figure 8.1: Overview of the BoWSLAM Algorithm

Table 8.1: Notation used in this chapter. See also Figure 8.7.

Camera i	Pose of the robot's camera at time i
T	Total time, equivalent to the total number of frames
s_{ij}	'Scale'–magnitude of motion between cameras i and j; related to velocity
δ_{ijk}	Distance moved between times j and k relative to distance i to j; related to acceleration
d_{ijk}, g_{ijk}	Parameters of the distribution of $\delta_{ijk} \sim \text{Log-}\mathcal{N}(d_{ijk}, g_{ijk})$
D_{ijk}, G_{ijk}	Parameters of the distribution of $s_{ij} \sim \text{Log-}\mathcal{N}(D_{ij}, G_{ij})$
Q	Pose graph: nodes are camera positions and edges are relative pose estimates
$E(Q)$	Edge-vertex dual of Q; nodes are relative poses and edges are relative scale estimates
$S_t(Q)$	t-spanner of Q; a subgraph of Q which preserves large cycles in Q

The highest-scoring FAST corners are chosen subject to a minimum separation constraint. Although minimum separation constraints degrades repeatability, this turns out to be necessary to find sufficient features to register images during rapid cornering. In some environments featuring rapid changes in scale, orientation-invariant SURF descriptors may be used; however these are expensive to extract, preventing real-time operation.
These features are extracted from every frame, which is then added to the BoW database, as described in Chapter 5. This database can now be queried to find the most similar images to the latest image, and also to find correspondences between pairs of these images.

8.3.2 Computing relative positions

This section describes how BoWSLAM computes the position of a camera based on a sequence of frames (the selection of this sequence is described in Section 8.3.3). First the relative position and orientation of two frames is estimated, as described in Chapter 6. Next, the 3D positions of landmarks observed in these frames are reconstructed. Finally the true scale of these landmarks, and hence the true motion of the camera between frames, is computed by aligning these landmark positions.
Given two frames represented as BoWs, correspondences between them can be found efficiently by comparing

their BoW representations. These correspondences are contaminated by many gross outliers however, so to compute the relative pose of the two camera positions the procedure described in Chapters 6 and 7 must be used. This procedure uses the BaySAC framework to compute the relative pose of the cameras while identifying points which are inliers. This inlier set is refined by top-down refinement (Section 6.2.4), then a nonlinear optimisation is used to improve the relative pose estimate (Section 6.2.2).

Resolving Scale Change

Given a feature viewed in two frames, and the relative pose between the two cameras, the 3D position of that feature can be reconstructed in the local coordinate frame of one of the cameras. As the cameras' relative position is known only up to an unknown scale factor, this feature's 3D reconstruction is also known only up to an unknown scale factor. If the feature is also viewed in an earlier frame we can align this reconstructed feature with its earlier reconstructed position, hence calculating the ratio of these scale factors (Figure 8.2). This scale factor is proportional to the distance between the two camera positions (the 'baseline length'), so estimating it relative to the previous scale factor gives us the distance moved by the camera relative to the camera's previous motion.

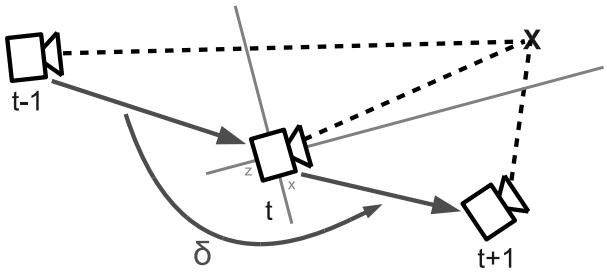


Figure 8.2: The point X is observed at times t and $t+1$. We know the relative pose of these cameras up to an unknown scale factor, so can reconstruct X in the coordinate frame of camera t, up to an unknown scale factor. If this point was also observed in an earlier frame $(t-1)$, we can calculate the ratio of the two scale factors, δ, by aligning the points' 3D reconstructions.

In the noise-free case, two reconstructions of the same feature in the same coordinate frame, X_i and X'_i, will fall on the same ray, i.e. $X_i = \delta X'_i \ \forall \ i$, where δ is the ratio of the two baselines, however in practice reconstructed points contain substantial errors from several sources. These sources of error include small errors in localising features in the image, which are amplified by stereo reconstruction; small errors in computing the cameras' relative poses; and errors from incorrectly-matched features that have not been discarded, as they were approximately compatible with the relative poses computed earlier. To estimate the scale change accurately despite these errors, we must combine multiple measurements while discarding any outliers.

Given multiple matches between reconstructed 3D points, we first remove outliers by discarding point pairs not within a certain angle of each other, as these do not fall on the same ray. We then calculate the scale factor for each pair X_i, X'_i, giving a set of K scale measurements $\{\delta_i, i = 1, ..., K\}$. Typical distributions of scale measurements in these sets are shown in Figures 8.3 and 8.4.

The change in scale between two frames is modeled with a lognormal distribution, $\delta \sim \text{Log-}\mathcal{N}(d, g^2)$ (defined by $\log \delta \sim N(d, g^2)$; Croarkin and Tobias, 2010, Section 8.1.6.4). This distribution is a good approximation for observed data (much better than the normal distribution; Figures 8.5 and 8.6), and has the useful properties that the product and inverse of lognormally-distributed variables are also lognormally distributed. d and g are estimated from $\{\log \delta_i\}$ by first removing any remaining gross outliers using Grubb's test (Croarkin and Tobias, 2010, Section 1.3.5.17), then taking d as the sample mean. As we have assumed $\{\log \delta_i\}$ are normally

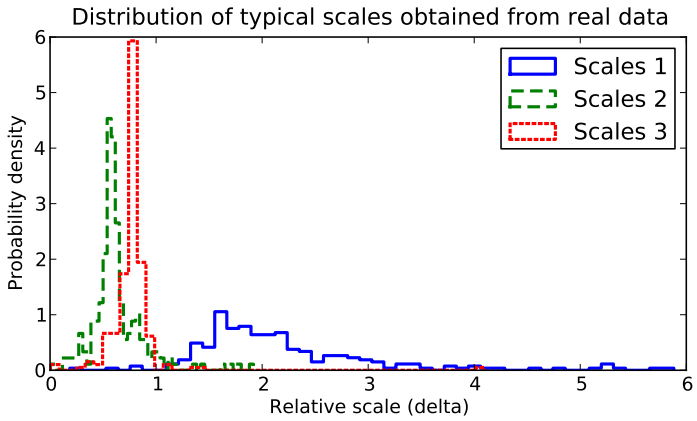


Figure 8.3: The multiple relative scale estimates computed from aligning reconstructed 3D points form a unimodal distribution with large variance, and a few gross outliers. This figure shows the distribution of relative scale measurements for three different triples of frames. These distributions are assumed to be approximately lognormal.

distributed, then d is approximately normal (d actually has a T-distribution), with variance:

$$g^2 = \frac{1}{K(K-3)} \sum_{i=1}^{K} (\log \delta_i - d)^2 \tag{8.3.1}$$

Accumulating relative positions and scales

The previous section describes how to calculate the speed of a camera relative to its previous speed, however its absolute speed is unknown, as a global scale ambiguity exists when navigating using only a single camera (without making assumptions about the speed of the robot or the size of observed objects). For now we assume the first time the camera moved it moved a unit distance; all other motion is now represented in terms of this distance. Other distances can now be calculated by multiplying sequences of relative scales (Figure 8.7). As these relative scales are modeled as lognormally distributed random variables, then their product s_{jk} is also lognormal:

$$s_{jk} \sim \text{Log-}\mathcal{N}(D_{jk}, G_{jk}^2) \tag{8.3.2}$$

$$\Leftrightarrow \log s_{jk} \sim N(D_{jk}, G_{jk}^2) \tag{8.3.3}$$

$$\text{where } D_{jk} = D_{ij} + d_{ijk},\ G_{jk} = G_{ij} + g_{ijk} \tag{8.3.4}$$

We have now calculated the relative pose of each pair (Section 8.3.2) and have parametrised the length moved at each time by $s_{jk} \sim \text{Log-}\mathcal{N}(D_{jk}, G_{jk}^2)$. Now every frame can be positioned relative to the first frame's position by accumulating these relative pose estimates, with lengths s_{jk} given by the median of the lognormal distribution, $e^{D_{jk}}$. This method allows a map to be built from any sequence of poses.

8.3.3 Graph-based representation of multiple position hypotheses

This section describes how wide-baseline feature matching and the BoW algorithm allow BoWSLAM to compute multiple position hypotheses for each frame, and how the graph these position hypotheses form is used to select only the most reliable position estimates, in the presence of gross outliers.

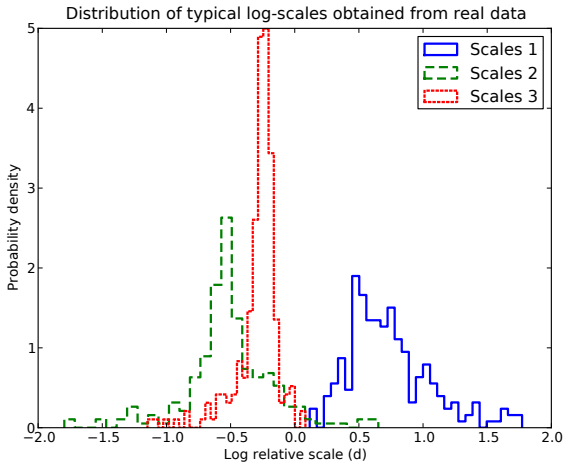
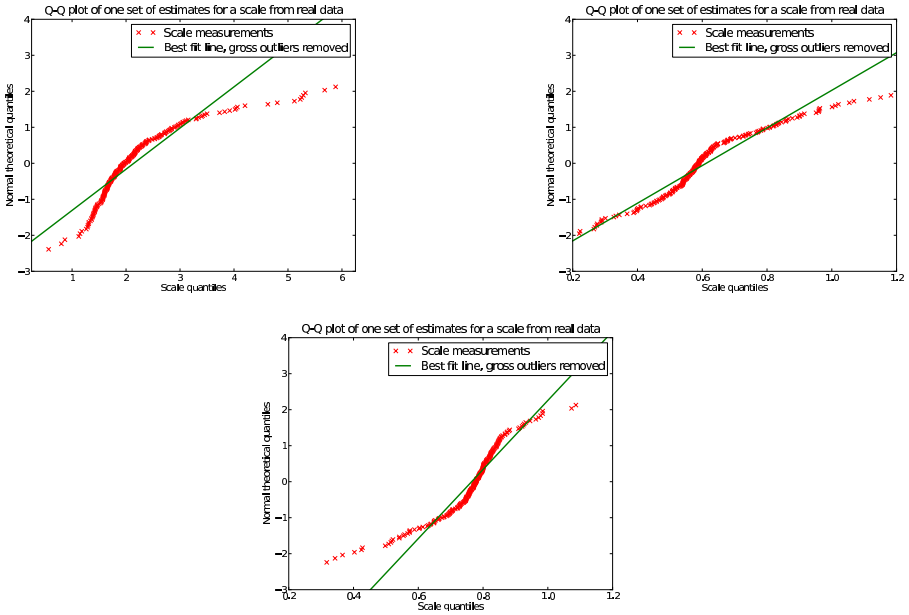


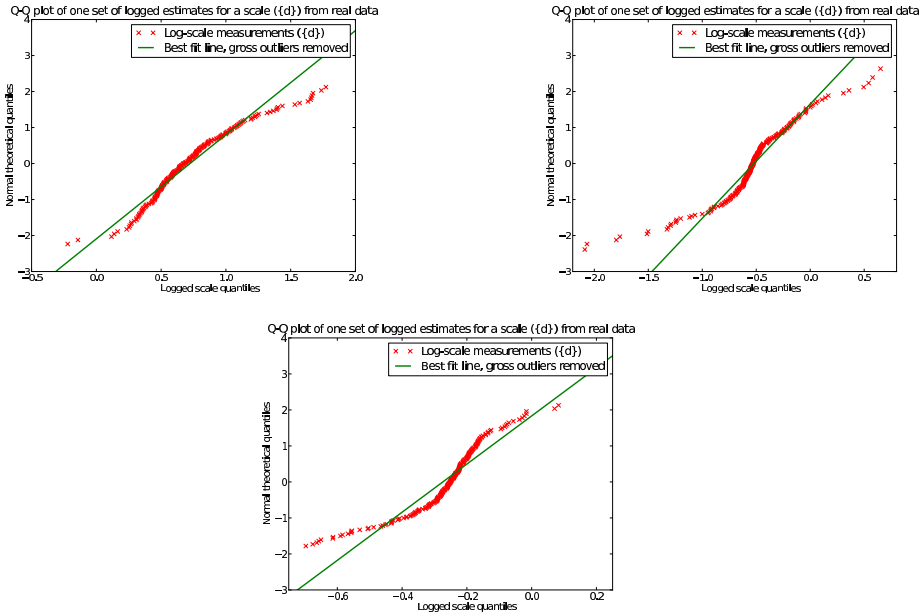
Figure 8.4: As relative scale estimates are assumed to be approximately lognormal, their log is approximately normal. This figure shows the distribution of logged relative scale measurements for the same three triples of frames as Figure 8.3.

Most contemporary visual navigation schemes track features over consecutive pairs of frames (e.g. Nistér et al., 2006; Davison, 2003; Eade and Drummond, 2008; Strasdat et al., 2010). If camera motion is smooth and the frame rate is high enough there will be large overlap between images, and features move only a short distance, allowing feature tracking by searching a small area where they are predicted to lie. These assumptions are not true in general however; real video sequences often contain sequences of frames with insufficient information for accurate matching (for example blank walls), sequences corrupted by dynamic objects (people walking in front of the camera), or sequences of rapid movement with either motion blur or insufficient overlap for reconstruction. Ideally each frame should be positioned relative to a frame with which it has large overlap (and many of the same landmarks visible), significant camera motion (for well-conditioned reconstruction), and few dynamic objects. These frames are chosen by using the BoW algorithm to select frames with many features in common with the current frame. From these candidate frames we choose those that already have accurate position estimates. This BoW sampling strategy avoids selecting frames to register to that contain too many dynamic objects or are blurred (as these could not be positioned previously), and avoids frames with insufficient overlap as these do not generate a strong BoW match. At the same time potential loop closure candidates are chosen—these must match the current frame at least as strongly as several recent frames. When the camera is stationary, moving slowly, or revisiting an earlier location, the baseline (the distance between two poses) can be too short for accurate 3D reconstruction. In this case, no new position estimates are calculated until the camera has moved sufficiently for an accurate position to be calculated. This automatically keeps the map sparse when repeatedly visiting the same place, an important requirement for long-term navigation in a bounded environment (Mouragnon et al., 2009). While these nearby frames are currently identified after the transform between them has been estimated, they could be found cheaply with a near-duplicate image detection scheme, for example the histogram-based scheme by Chum et al. (2007).

This strategy results in each camera position being computed relative to several (typically one to four) other camera positions, giving several position hypotheses. These multiple position hypotheses naturally form a graph, with video frames corresponding to nodes and position hypotheses relative to other frames' locations corresponding to edges (Figure 8.8). Each edge has a set of associated 3D landmarks which are visible in the two frames it connects. Every path connecting two frames in this graph gives a sequence of relative position estimates; the relative position of these two frames is given by accumulating the relative position estimates along one of these paths. This graph is similar to the graph in the Atlas framework (Bosse et al.,



(a) Q-Q plots showing how well scales are approximated by a normal distribution.



(b) Q-Q plots showing that logged scales are approximately normal; hence the lognormal distribution is an appropriate model.

Figure 8.5: The lognormal distribution is a good approximation for relative scale estimates. Q-Q plots (Thode, 2002) show how well data is approximated by a particular distribution. The quantiles of the data are plotted against the quantiles of a normal distribution; the closer points are to a straight line, the better the approximation. For these three examples of real data, the log of the scale measurements (b) is slightly better approximated by a normal distribution than the scale measurements (a).

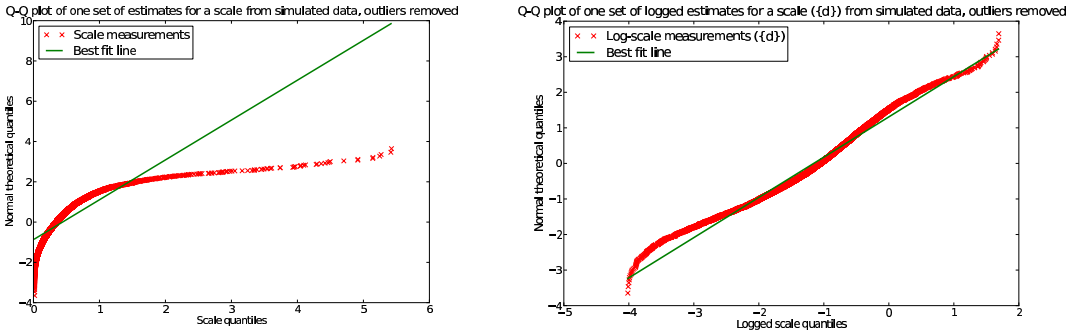


Figure 8.6: Q-Q plots of relative scale estimates and logged relative scale estimates from aligning simulated 3D points (10,000 points before outliers were removed). The lognormal distribution is a much better approximation than the normal distribution for the scales.

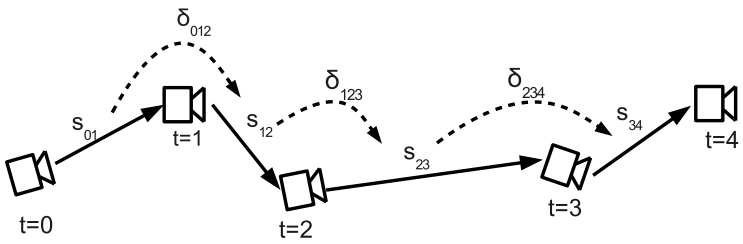
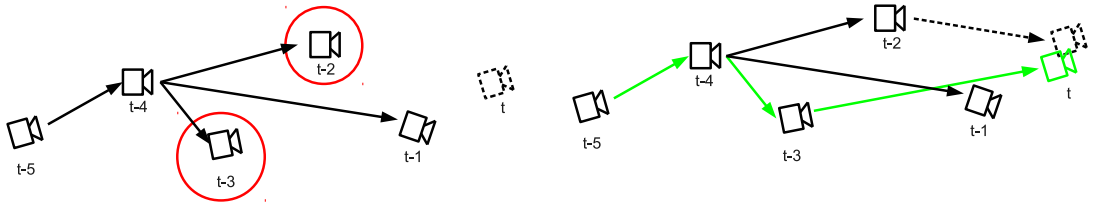


Figure 8.7: The distance between each pair of cameras j and k, s_{jk}, is calculated by accumulating relative scale estimates, δ_{ijk}. Note that errors in the estimate of δ_{ijk} will accumulate in subsequent scale estimates (and hence scale estimates are correlated). The dashed edges form the edges in the Edge-Vertex dual $E(G)$.



(a) When a new frame is captured first choose candidate frames from which to compute relative position hypotheses.

(b) Add position hypotheses—the sequence of relative positions most likely to be outlier-free (highlighted) is used to position the new frame; another position hypothesis (dashed) is not used for now.

Figure 8.8: Camera positions form a graph with edges representing relative position hypotheses. (a) To add a new frame at time t (dashed) suitable candidates to position it relative to are selected. (b) The new frame is positioned relative to several other frames, leading to several position hypotheses. Of these hypotheses the position most likely to be outlier-free is selected. These positions are updated as more position hypotheses become available.

2004) with edges representing transformations. In BoWSLAM every video frame corresponds to a node (as in Newman et al., 2009), so that transformations connect robot poses, in contrast to Bosse et al. (2004); Eade and Drummond (2007) where nodes represent local coordinate frames with associated landmarks.
We now have a pose graph, Q, with each frame positioned relative to several nearby frames. This provides a large number of possible paths by which two frames can be positioned relative to each other. The Atlas framework provides an efficient way to choose a good path between any two frames by assigning appropriate quality weights to each edge then finding the 'shortest' (highest quality) path between the frames, however in the case of single camera SLAM, estimates of relative scale must also be taken into account. These relative scale estimates are dependent on sequences of relative poses, leading to constraints between pairs of edges in Q. These relative scale estimates are modeled using the dual-graph approach described in the following section.

Dual-graph representation of relative scale

When positioning using a single camera, we find the principal sources of error are gross outliers (usually due to dynamic environments, or from failing to detect remaining outlier correspondences) and scale drift, as errors in relative positions are proportional to errors in the estimated scale, and errors in orientation are typically low by comparison. When gross errors in relative pose do occur, they normally lead to gross errors in the corresponding relative scale estimate. Therefore we aim to choose the path through the pose graph Q that is least likely to contain gross errors in relative scale estimates. Relative scale estimates are not associated with edges in Q however; instead a relative scale estimate is associated with every pair of edges meeting at a node (Figure 8.7), i.e. is associated with edges in a new graph, the 'edge-to-vertex dual' $E(Q)$ of the pose graph Q (or 'line graph'; Balakrishnan, 1997, Chapter 2). Vertices in $E(Q)$ correspond to edges in Q, and edges in $E(Q)$ connect two vertices if the corresponding edges in Q meet at a vertex. Each edge in $E(Q)$ has an associated relative scale $\delta_{ijk} \sim \text{Log-}\mathcal{N}(d_{ijk}, g_{ijk}^2)$, and connects vertices corresponding to relative positions from i to j and j to k.
Edges in $E(Q)$ are assigned weight g_{ijk}^2, then the 'shortest' path P between two nodes in $E(Q)$ provides a sequence of relative scales from which the relative lengths of the two relative poses is estimated. This scale estimate has distribution $\text{Log-}\mathcal{N}(D, G^2)$ where $G^2 = \sum_{e \in P} g_e^2$ by Equation 8.3.4. Note that P is the sequence of edges minimising G^2. The following observations justify this weighting:

1. If a scale $s_{ijk} \sim \text{Log-}\mathcal{N}(d_{ijk}, g_{ijk}^2)$ is parametrised from two point sets X and Y, then the scale computed in the opposite direction has the distribution $s_{kji} = \frac{1}{s_{ijk}} \sim \text{Log-}\mathcal{N}(-d_{ijk}, g_{ijk}^2)$. Therefore edges have

the same weight regardless of the order or direction they are traversed[1].

2. If the measured relative scales were truly lognormal random variables $s_{ijk} \sim \text{Log-}\mathcal{N}(d_{ijk}, g^2_{ijk})$, then P is the path giving an estimate of the log of the true scale with expected error less than the estimate along any other path.

3. Gross errors in relative position and orientation normally lead to few accurately reconstructed points from which to recover scales, leading to higher g^2 estimates by Equation 8.3.1. These edges will only be used when no better relative position estimates exist.

4. An alternative weighting scheme would be to weight each edge in $E(Q)$ by the negative log of its inlier probability. The shortest path would then be the one least likely to contain an outlier, i.e. the most robust position estimate. These weights would be equal if $P(\text{scale estimate is an inlier}) = e^{-g^2}$. While the true outlier probability is hard to measure and is threshold-dependent in real data, coincidentally this equation does give realistic values of $P(\text{no outliers})$ of about 0.6-0.9 after travelling for 500 frames without loop closure, and hence we believe it is a good strategy for minimising the probability of our position estimates being contaminated by outliers.

As at most a fixed number of relative positions is introduced to Q at each time, the number of edges in Q, and hence the number of relative-scale-edges in $E(Q)$, is proportional to the number of frames, T. Therefore a shortest path tree can be found in $E(Q)$ in time $O(T \log T)$ using Dijkstra's algorithm. This shortest path tree gives a good position estimate for every frame and hence a global map. Alternatively a locally accurate map could be found in constant time by recursing to a fixed depth from the current node, in the same manner as Mei et al. (2009).

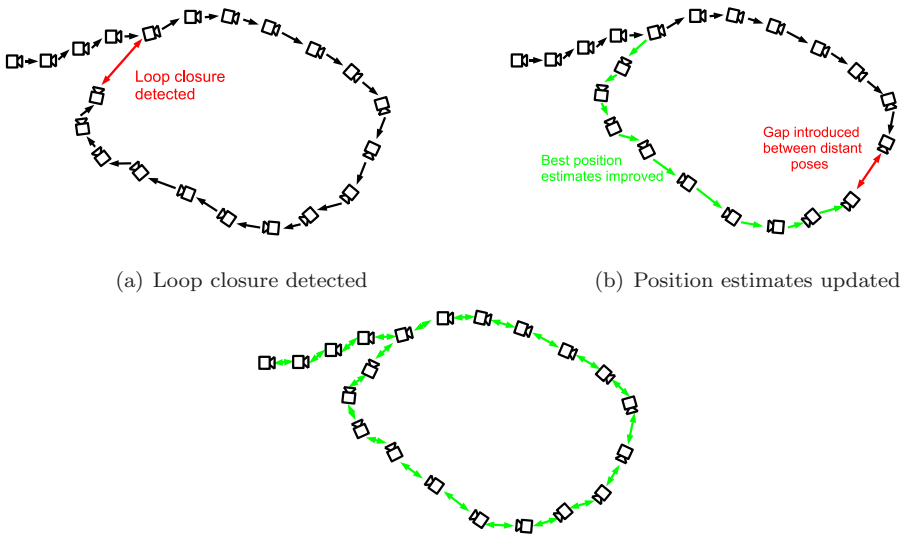
Motion models and scale drift

While BoWSLAM can position a robot for extended periods without making assumptions about the robot's motion, in some visually challenging environments significant scale drift accumulates and introduces errors into global maps (for example Figure 8.16(a)). This scale drift arises primarily from a small number of relative scale estimates where few 3D points are matched between successive frames; these unreliable scale estimates are incorporated into the map only when no better scale estimates have been found (usually when cornering rapidly). One solution to reduce this scale drift is to impose a weak motion model on relative scale estimates. Two possible motion models are proposed in this section. The first models the acceleration, and the second models the perceived depth of points in the world.
The first motion model is based on the assumption that the camera velocity is approximately constant over short periods of time. This motion model is implemented by assuming that the relative acceleration over short period of time is lognormally distributed about zero.
Relative scale estimates are modeled as lognormally distributed variables, which are parametrised by aligning reconstructed 3D points (Section 8.3.2). When a scale is computed from three frames, i, j and k, which are temporally close, we assume these relative scales are lognormally distributed about the scale corresponding to zero acceleration as follows:

$$\delta_{ijk} \sim \text{Log-}\mathcal{N}(log\frac{\Delta_{jk}}{\Delta_{ij}}, G^2_{MM}\max(\Delta_{ij}, \Delta_{jk})) \tag{8.3.5}$$

where Δ_{ij} is the time difference between frames i and j, and G_{MM} controls how tightly speeds are constrained. This distribution is combined with the distribution for δ_{ijk} from aligning points. G_{MM} is chosen so that when many points are observed, the scale from observing the points dominates, and when few are observed, the motion model dominates. This motion model is weaker than other single camera SLAM motion models as only the camera's relative acceleration is constrained. Even so, like other SLAM schemes, it could potentially

[1]When D is calculated along a path in E(G), and rotations and translations are accumulated along a path in Q the direction edges are traversed is relevant, this is implemented in an undirected graph simply by assigning each edge a relative pose/scale in one direction (the direction of increasing time) then reversing it if traversed in the opposite direction.



(a) Loop closure detected

(b) Position estimates updated

(c) t-spanner optimised

Figure 8.9: When loop closure is detected the latest frame can be positioned relative to a much earlier frame (a). After loop closure, the 'best' position estimate for each frame is updated. This improves the position estimates of a sequence of nearby frames relative to the latest frame (b). When a global map is required a t-spanner of good position estimates preserving cycles is selected and optimised, closing any distant gaps (c).

limit camera acceleration after the camera has slowed. In this circumstance however, position estimates from many recent frames are normally available, as there is high overlap between consecutive frames when the camera moves slowly. If sufficient 3D structure is aligned over any of these triples of frames then this scale estimate will dominate.

This motion model reduces scale drift, but does not eliminate it; errors will still accumulate. An alternative motion model is based on the observation that reconstructed points in the world typically have depths in a particular range, for example 0.5m to 20m for a pedestrian outdoors. The mean depth of reconstructed points is proportional to the baseline length (which is assumed to be lognormally distributed), therefore by modelling the depth of reconstructed points with a lognormal distribution and combining this estimate with the depths from SLAM an absolute constraint on scales can be imposed. As sequences of scales are all dependent (by Equation 8.3.4), an uninformative distribution can be used, leading to many small updates that together eliminate scale drift over long sequences of frames. This constraint is similar to constraints on the depth of observed points imposed by some other single camera SLAM schemes (Davison, 2003; Lemaire and Lacroix, 2007).

8.3.4 Loop closure and map optimisation

Loop closure in a graph-based map is very simple—an edge is added in the same way as for other relative position hypothesis. However when a loop is closed and a new global map is computed a break in a continuous chain of positions in the middle of the loop often forms when poses for frames at either side of the break are computed via different paths (Figure 8.9). This break will usually be far from the root edge from which the graph's shortest path tree is generated so a locally accurate map can always be constructed by setting the root near to the current robot position, however sometimes a global map is required.

Our goal is to compute an accurate map from a set of relative pose estimates. Several solutions have been proposed for the case when errors in relative poses are Gaussian and independent (described in detail

Input: Weighted undirected graph Q
Output: Weighted spanning subgraph $S_t(Q)$

$d(u, v, G) \equiv$ distance from u to v in graph Q

Find minimal spanning tree (Kruskal's algorithm):

1. Create a forest, S_t, of T disconnected components, one per vertex
2. Create a list, E, of edges sorted in order of increasing weight
3. For each edge $e \in E$ in order:
   - If e connects two components, remove e from E and add to S_t to join components.
4. Now S_t is the minimal spanning tree of Q

Find t-spanner:

1. For each edge $e = (u, v)$ in E in order:
   - if $d(u, v, S_t) > td(u, v, Q)$ then add e to S_t

S_t is now a t-spanner of Q with total weight close to the minimum possible.

Figure 8.10: Algorithm to find an approximate minimum-weight t-spanner in a graph (Althöfer et al., 1993). This algorithm has complexity $O(T \log T)$, as the number of edges is proportional to T, and determining whether $d(u, v, G)$ is less than a threshold has constant cost.

in Section 3.2.5). Those with lowest complexity (as low as $O(T)$) are based on relaxation, where nodes positions' are incrementally updated to reduce the total error in all relative pose estimates. BoWSLAM used the 3D version of the TORO framework (Grisetti et al., 2007a) to generate global maps, as this is faster than other relaxation frameworks, and can optimise 3D pose graphs.

There are two issues with using TORO with BoWSLAM's maps however, the first is that TORO does not model outliers, and one bad relative pose estimate can corrupt the entire global map. Optimising the entire pose graph Q rarely results in a map resembling the ground truth, so instead we must select an outlier-free spanning subgraph of Q. The most important property of this subgraph is that large cycles in Q should be preserved. A suitable subgraph with this property is known as a t-spanner, $S_t(Q)$. $S_t(Q)$ is defined as a spanning subgraph where two nodes separated by a distance k in Q are separated by a distance of no more than tk in $S_t(Q)$. For our pose graph, which has T nodes and $O(T)$ edges, a t-spanner with low total cost (close to the minimum possible) can be found efficiently in time $O(T \log T)$ using the algorithm by Althöfer et al. (1993), outlined in Figure 8.10. A value of around 20 for t gives a subgraph consisting of the minimal spanning tree, plus a small number of edges sufficient to preserve cycles larger than t; this is the subgraph we optimise using the TORO framework.

A second problem with using TORO for single camera SLAM is that scale drift means that the independence assumption between poses does not hold. Graphs containing significant scale drift remain distorted after TORO optimisation. In Section 8.7 plans to develop a pose graph optimisation framework which optimises scale estimates is discussed; for now a cyclic constraint on scales is imposed by SVD: around a cycle C, $\sum_{(ijk)\in C} d_{ijk} = 0$. While this optimisation has poor theoretical complexity, in practice it is fast, taking just a few milliseconds in t-spanners, as long sequences of edges typically belong to the same set of cycles, so can be temporarily replaced with a single edge.

The final requirement for optimisation are variances for each relative pose (full covariances are not used by the 3D version of TORO). These are estimated by assuming pose estimation is locally linear, and fitting a

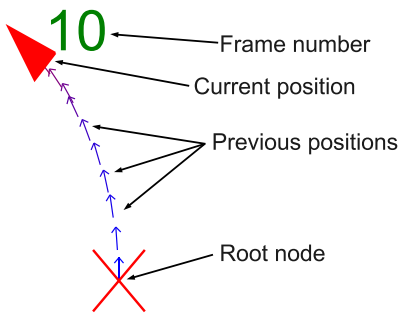


Figure 8.11: Key to maps: each frame with a pose estimate is marked with a small arrow showing its orientation and position. The current robot position is marked with a red triangle, and the map's root node is marked ×; if the camera is upright, then the maps show the ground plane oriented with the root pose pointing towards the top of the map. The maps' scales are arbitrary.

simple model to errors in simulated noisy data, as described in Section 6.4.3.

8.4 Experimental results

In this section we validate the BoWSLAM algorithm on simulated data, then demonstrate BoWSLAM's ability to position a camera in challenging indoor and outdoor environments on three datasets which include long, complex paths with erratic motion, total occlusion, moving objects, and motion blur. The maps presented show the robot's trajectory; Figure 8.11 provides a key to these maps.

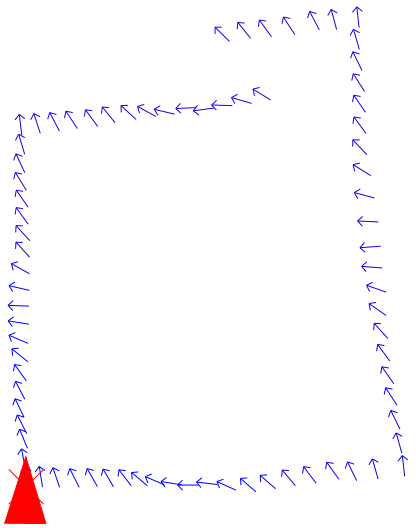
8.4.1 Validation on simulated data

Simulated data was used extensively in the development of BoWSLAM. In this section, simulated data is used to demonstrate BoWSLAM's ability to recover from positioning failure, and to demonstrate BoWSLAM's ability to position a camera in a small-scale feature-rich environment, with a range of speeds and imaging conditions.

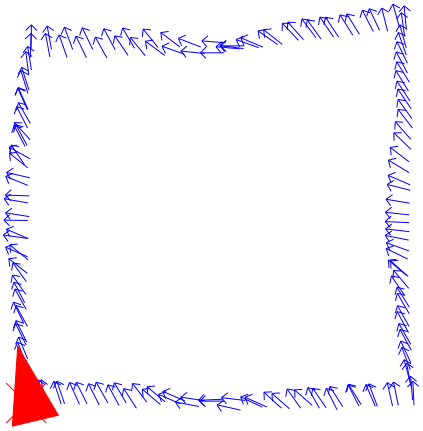
Firstly, we demonstrate 5 DOF positioning, and that the accuracy of position estimates does improve with time. Sets of descriptors and locations are generated by projecting a random cloud of points into a sequence of simulated images, adding point localisation noise of 2 pixels, then simulating descriptors (from 100 distinct descriptors in total, so each will exactly match 1% of other descriptors incorrectly).

Figure 8.12 shows maps generated with simulated data from a camera which spins around all three axes, while travelling around a square at variable speeds. Typically between 20 and 100 features are matched between each pair of consecutive frames, depending on the velocity. Maps generated using only a spanning tree are shown (i.e. maps generated without any optimisation). After the first lap, a significant gap opens, due to accumulated errors. On the next lap, better relative pose estimates are selected, and this gap is closed. The gap remains closed for 50 more laps, despite occasional gross errors occurring in relative pose computation. Slight systematic errors remain in the optimised trajectory.

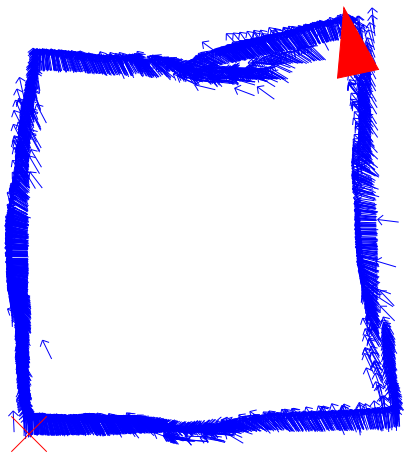
Figure 8.13 shows maps generated on the same dataset, when 50% of matches are gross outliers, and point localisation errors are increased to 3 pixels (0.01 radians). BaySAC now fails to find a relative pose containing 10% inliers approximately 50% of the time, and some poses which are computed contain gross errors (due to outliers being included when inlier rates are low). The map after the first lap of the square consists of two disconnected components, which are then fused when the loop is closed. The gap in the map from a spanning tree is never closed, although TORO generates a consistent, although distorted map, which is free of gross outliers.



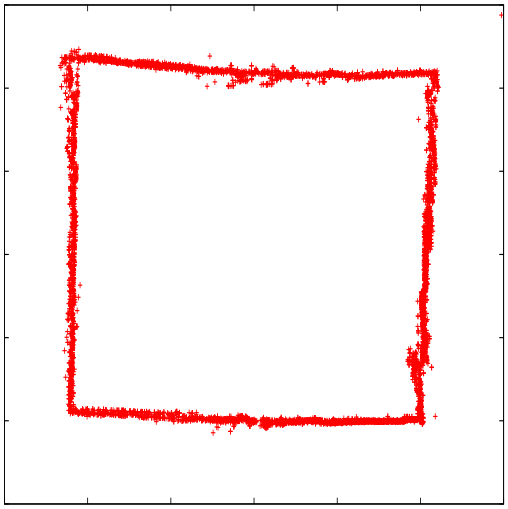
(a) Significant gap after first lap



(b) No gap after second lap

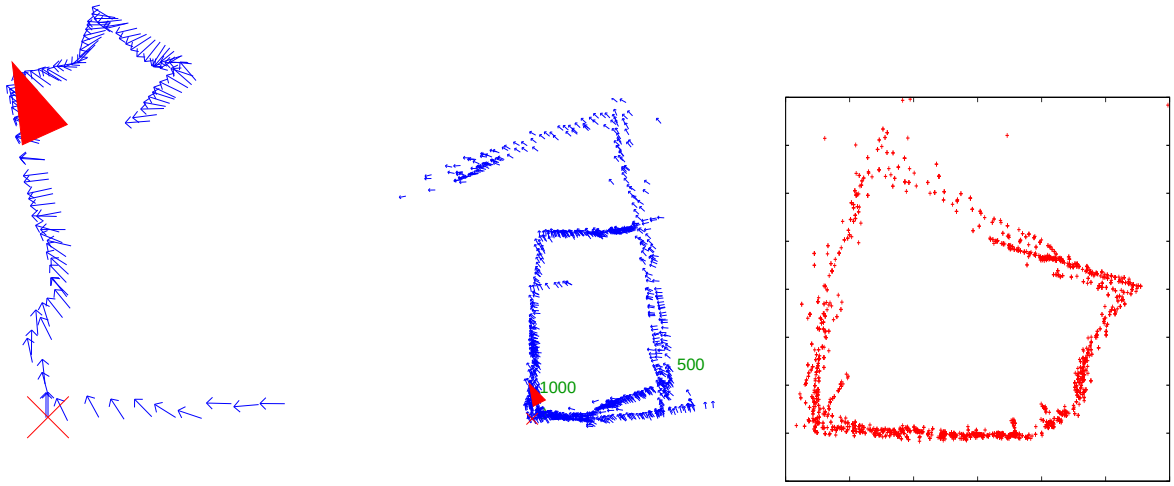


(c) Still no gap after 50 laps



(d) Optimised map after 50 laps

Figure 8.12: Estimated positions of a simulated camera which travels around a square path, at variable speed, while spinning about all three axes. A gap visible after the first lap is closed after the second, as more accurate relative pose estimates become available. The final optimised map closely matches the square path.



(a) After 100 frames, disconnected map components have been fused, but scale drift is high.

(b) A gap remains after 1000 frames.

(c) The optimised 20-spanner is distorted, but consistent.

Figure 8.13: Estimated positions of a simulated camera. Despite very high error rates, a consistent map is maintained.

In order to obtain images with known ground-truth, Funke and Pietzsch (2009) build a simple model of a room (a cube with the same image pasted onto every wall), then use ray-tracing software to project images of the environment into a simulated camera, which travels on a small loop. Effects such as motion blur can be simulated (Figure 8.14). On a 4m, 288-frame loop with added motion blur, their EKF-based stereo camera SLAM scheme accumulates errors of about 0.45m while negotiating the loop, then corrects these errors when the loop is closed. Figure 8.15 shows maps generated by BoWSLAM on this dataset. Comparable errors accumulate (up to 0.5m) using just a single camera, the loop closure is rarely detected, as the frames contain repeated images of banana boxes, so the frames where loop closure should occur do not match any more strongly than other frames. A loop closure framework which incorporates the estimated camera position would be able to detect this loop closure, however the repeated environment shown is not particularly realistic. The smooth camera motion, and feature-rich images indicate that a motion model and feature tracking would perform well, however BoWSLAM operates successfully without these. This dataset shows that BoWSLAM has comparable performance to another SLAM scheme, in a situation where a conventional single camera SLAM scheme would be expected to perform well.
These simulated images are unlike many real datasets however. In this simulated data we consistently find large numbers of good correspondences, and if gross errors occur, they arise from repeated structures. Data could be simulated to contain moving objects, or erratic motion, but it is never certain that this simulated data is really representative of real data. Hence, we believe that the results on real data presented in the remainder of this section provides a much better demonstration of BoWSLAM's capabilities.

8.5 Results on real data

In this section we demonstrate that BoWSLAM can robustly navigate difficult environments with three datasets, each processed off-line.
The first dataset is captured at 2.8Hz from a handheld Canon 400D SLR camera in a suburban environment (Figure 8.17). Images are greyscaled, normalised, and down-sampled to 640×426. After travelling 1.6km the camera is stopped, then restarted elsewhere. A couple of minutes later the new path re-joins the original

Figure 8.14: Simulated frames from the SLAMDUNK dataset with added motion blur, covering a small loop. The repeated image of banana boxes occasionally triggers false loop-closure.

Table 8.2: RMS errors in global maps optimised by TORO. Errors are relative to GPS positions after scaling and aligning maps; RMS errors in the non-differential GPS position are estimated to be 5m. These figures are parametrisation-dependent and mainly reflect the amount of scale drift present. The total track length is 2.5km.

Motion model	RMS error	Error relative to distance travelled
No motion model	198m	7.9%
Constrained acceleration	83m	3.3%
Constrained depths	56m	2.2%

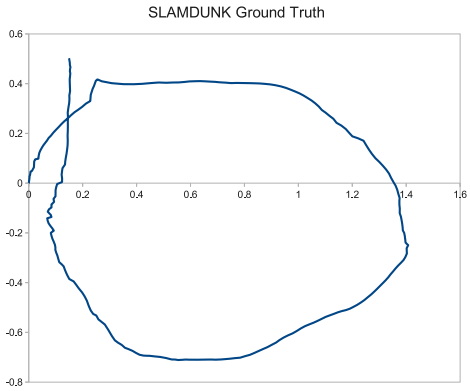
path. The sequence includes many pedestrians and cars, and three frames show only of the side of a passing bus.

Without assuming any motion model, BoWSLAM successfully positions the robot along the first path, producing locally consistent maps (Figure 8.18a). Loop closure is detected every time the path re-visits an earlier location. When the camera is stopped and re-started, a new map component is started (Figure 8.18b), however significant scale drift accumulates during rapid cornering. When the new path returns to a previously visited location, loop closure is detected and the maps are fused. The scale from the first component is propagated along the new path, however a loop closure opportunity was missed when the camera travels in the opposite direction to the previous visit, leading to large uncorrected scale drift, as seen on the right of Figure 8.18c. This loop is hard to detect as features common to both frames are visible from very different angles (possible solutions are discussed later, and the frames concerned are shown in Figure 9.12). Even viewpoint-invariant SURF descriptors are not sufficient to detect this loop closure.

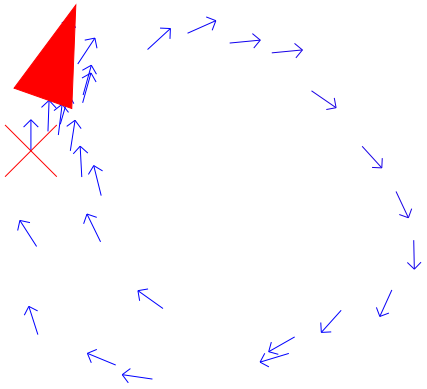
Initially a framerate of 16Hz is maintained; after 24 minutes this has dropped to 11Hz; still almost four times the rate images were captured. Correcting scale drift in 1546 edges, around 7 cycles, by SVD takes just 60 milliseconds.

To produce the optimal map in Figure 8.16 a 25-spanner of the pose graph is found. This contains 14 more edges than the shortest path tree used to find locally consistent maps (Figure 8.18). A reasonable map with only small gaps is obtained after 500 iterations (6 seconds). After 20,000 iterations (234 seconds) the global error is reduced four-fold and gaps are completely closed. This slow convergence is due to variation in relative position variances; by setting all variances equal a reasonable map is obtained in 50 iterations. While most details of the ground-truth track from GPS are recreated, in two places (at either end of the second track) uncorrected scale drift corrupts the position estimate.

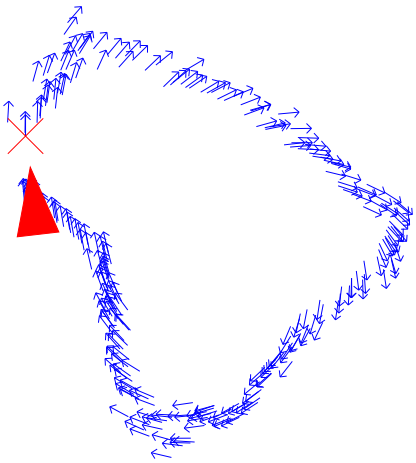
We have also processed this dataset with the two motion models proposed in Section 8.3.3. The first of these motion models, constraining accelerations, reduces scale drift throughout the map (Figure 8.16(c)), however some scale drift still distorts the map during tight, rapid cornering. The constraint on acceleration does not



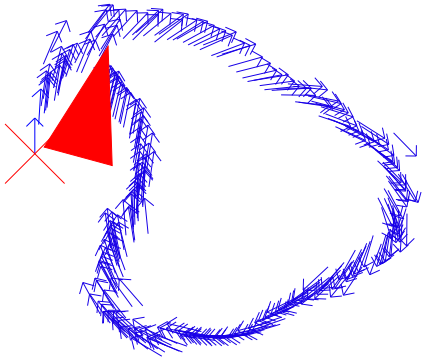
(a) Ground-truth trajectory from the SLAMDUNK dataset.



(b) Trajectory computed using every 10th frame, motion blur added.

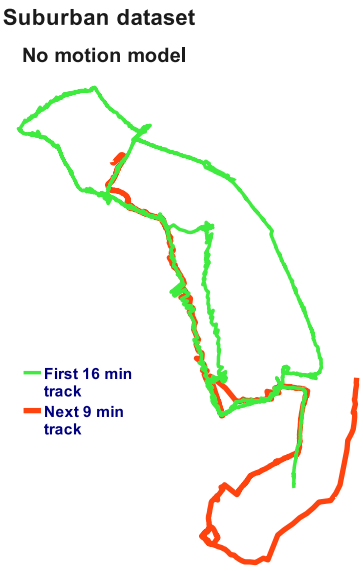


(c) Trajectory computed using every frame, motion blur added.

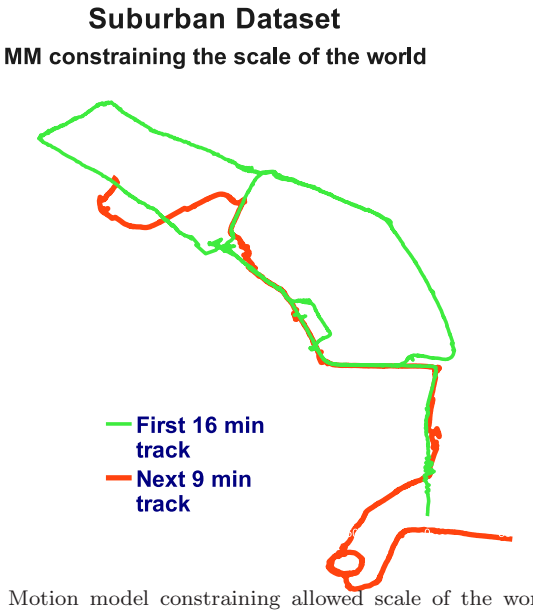


(d) Trajectory computed using every frame, no motion blur.

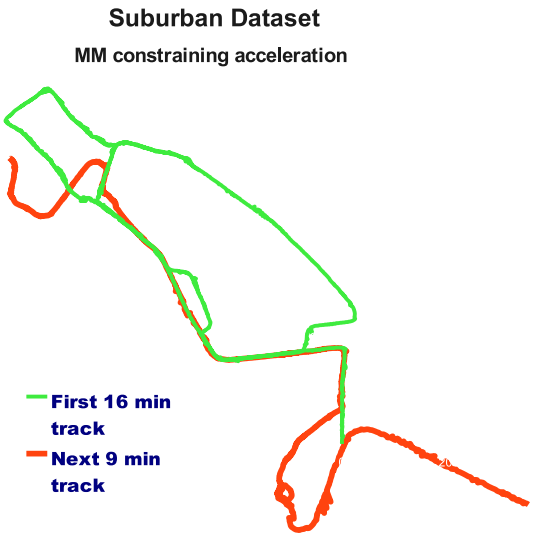
Figure 8.15: Camera trajectory computed from the simulated SLAMDUNK dataset. No motion models are assumed, and no optimisation is used. The repeated images of banana boxes make loop-closure detection challenging, and occasionally trigger false loop-closure. Adding motion blur degrades positioning slightly, as does using only every 10th frame.



(a) Global map, no motion model. The path is generally reconstructed accurately, except at either end of the second trajectory where loop closure events are missed and scale drift accumulates.



(b) Motion model constraining allowed scale of the world. Scale drift is greatly reduced, although the map is distorted where the true scale of the environment does not reflect the scale assumed by the motion model.

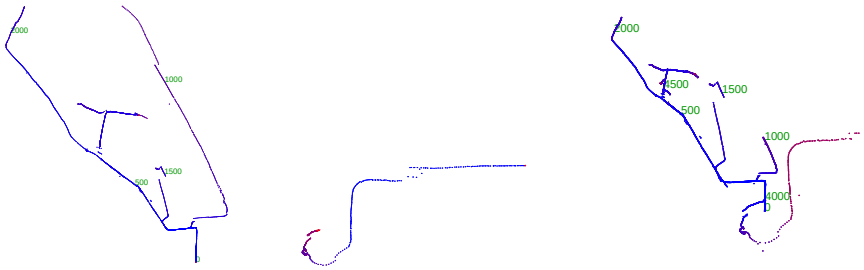


(c) Motion model constraining acceleration. Some scale drift still occurs at either end of the second track.

(d) Ground truth data from GPS

Figure 8.16: Maps of robot poses compared with ground truth from GPS, from the Suburban dataset (Figure 8.17). The path starts at **S** and traverses the large loop twice, with various diversions. After 16 minutes the camera is stopped, then restarted at **R**. Later the original path is re-joined.

Figure 8.17: Frames from the Suburban dataset, covering 2.5km over 25 minutes. Dozens of pedestrians and cars pass by, and one sequence of three frames is completely obliterated by a passing bus.



(a) A locally-consistent map is built as the robot explores.

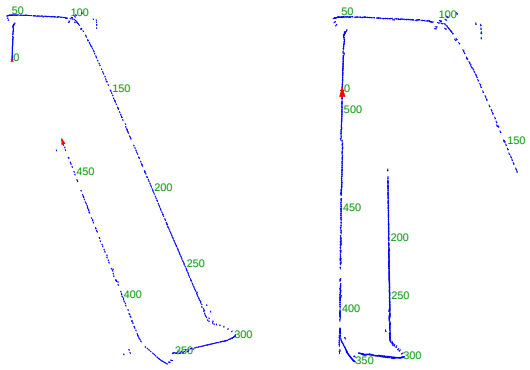
(b) The camera is stopped after 16 minutes and started elsewhere. A new map component is started (rotated to match Figure 8.16).

(c) Loop closure is detected as soon as path crosses previous path and the two map components are fused.

Figure 8.18: The robot needs to identify which frames have reliable position estimates as it explores. To do this it maintains maps where each frame is positioned along the 'shortest' path back to a root node (the first node in this example). These maps are locally consistent (near the root) but contain significant scale drift and gaps in loops; these are corrected when a globally accurate map is optimised using TORO.

Figure 8.19: Frames from the University of Alberta CS Centre dataset—a 512 frames captured around a 75m indoor loop from The Robotics Data Set Repository (Radish) (2003).

affect BoWSLAM's ability to stop or start moving at the three locations where the camera is stationary.
The second motion model, constraining the depths of reconstructed points, greatly reduces scale drift and all features of the original map are accurately reconstructed, however some distortions are introduced where the camera passes through a narrow passage where the point depth model does not reflect the true point depths (Figure 8.16(b)). The RMS errors between these three runs and the GPS ground-truth are given in Table 8.2, showing that both motion models improve BoWSLAM's accuracy.
The second dataset demonstrates BoWSLAM navigating a typical indoor environment. The dataset, from the University of Alberta (The Robotics Data Set Repository (Radish), 2003), consists of a 75m rectangular loop around corridors and landings. 640×480 greyscale images were captured every 15cm from a wheeled robot (Figure 8.19). Images are processed at 16Hz, corresponding to a robot velocity of 2.4m/s. Positioning is accurate along straight corridors, despite a large number of self-similar features, and features reflected from the uneven floor. Again when cornering rapidly the rapid rotation relative to forward motion make tracking scale difficult, as few points with accurate 3D positions are tracked into subsequent frames (using a wide-angle lens would greatly reduce this problem). Positions of a few frames with gross errors are visible near corners; these errors are successfully avoided when computing subsequent positions.
When the start point is revisited loop closure is detected (Figure 8.20). Optimisation with the TORO framework produces the map shown in Figure 8.21, needing 50 iterations and 0.08 seconds. Artefacts from the errors during rapid cornering are still visible, in particular the zig-zagging where a sequence of alternate frames is positioned relative to two different earlier frames.
The final dataset demonstrates BoWSLAM navigating a challenging outdoor dataset. The dataset is captured from a Contour HD mountain bike helmet camera, and features full six degree-of-freedom motion, large changes in scale, rapid cornering, erratic motion over rough ground, and sequences of frames with motion blur (Figure 8.22). 1.5km are covered in 5 minutes at speeds ranging from 5 to 30 km/h. The original video consists of frames sized 1280×720 captured at 30Hz; image features accelerate from zero to 140 pixels-per-frame across sequences of three frames, and sequences of up to six consecutive frames contain motion blur.
BoWSLAM uses one-in-three frames from the original video, downsampled to 640×360. A motion model constraining the world's scale, and scale- and rotation-invariant SURF descriptors are used; these are necessary to register poor-quality images despite rapid changes in scale, but cut performance to around two frames-per-second.
BoWSLAM successfully recreates the double-loop and many features of the trajectory despite numerous small errors (Figure 8.23). Overall the map is warped from many small errors in relative orientation, and errors in scale when the camera moves between forest and open areas. Corrections around loops mean changes in altitude are recreated successfully, despite the warped map (Figure 8.24).



(a) Initially navigation is by dead-reckoning.

(b) Once robot returns to starting point loop closure is detected. A locally accurate map is built.

Figure 8.20: Local maps of robot poses before pose graph optimisation, University of Alberta dataset (Figure 8.19).

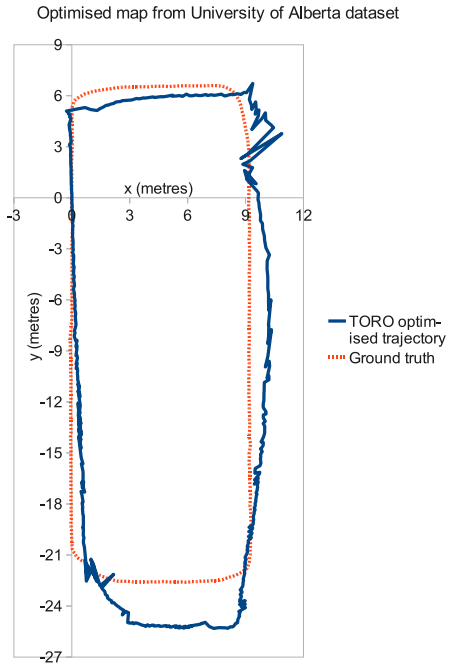
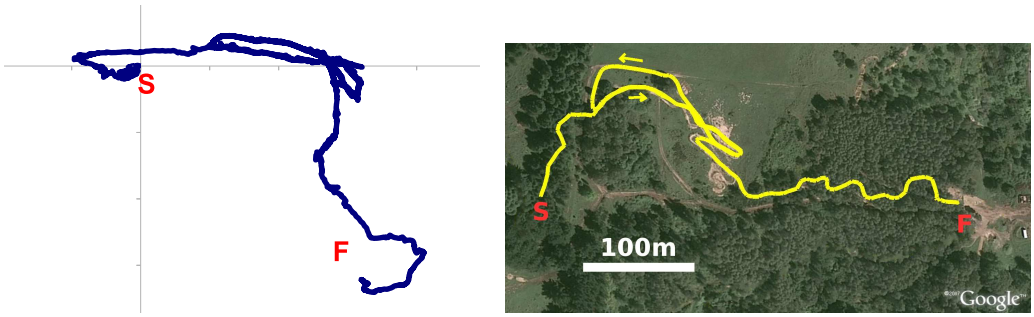


Figure 8.21: Global map of robot poses compared with ground-truth, University of Alberta dataset (Figure 8.19), after optimisation with TORO framework. A scale is applied to the optimised map to match ground-truth scale.

Figure 8.22: Frames from the Helmet Camera dataset, a challenging sequence characterised by erratic motion and motion blur.



(a) BoWSLAM's optimised map of the Helmet Camera dataset.

(b) Ground-truth data from GPS.

Figure 8.23: Optimised map of the Helmet Camera dataset. BoWSLAM successfully re-creates the structure of the trajectory, although errors accumulate around tight corners and distort the optimised map.

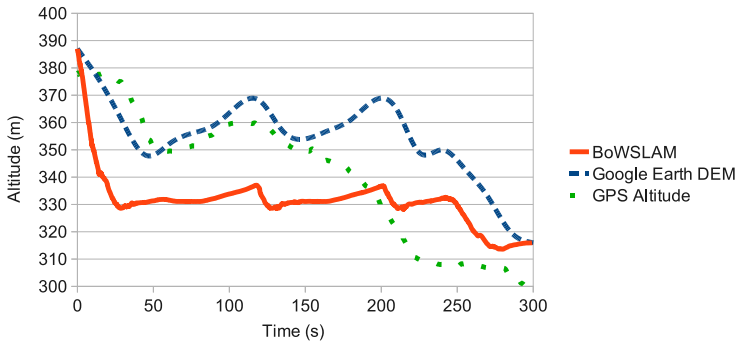


Figure 8.24: Camera altitude on the Helmet Camera dataset from GPS, from Google Earth's Digital Elevation Model (DEM), and from BoWSLAM (shifted and scaled to match the range of the DEM data). BoWSLAM's altitude estimate contains systematic errors from a slightly warped map, however details are recreated more successfully than by GPS (where the same location has a 30m height difference on two visits).

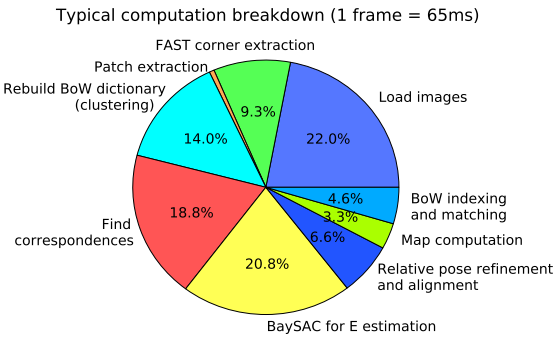


Figure 8.25: Breakdown of execution time per frame. Computed on University of Alberta dataset, excludes rendering of maps and TORO optimisation.

8.5.1 Long-term real-time operation

A breakdown of typical processing times is given in Figure 8.25. Note that once the inlier correspondences and epipolar geometry are determined, the positioning and mapping parts of the algorithm are very fast. Also computing four position hypotheses per frame rather than one only increases the total cost by about 50%. For this dataset, using RANSAC instead of BaySAC almost doubles the time needed to find essential matrices (increasing the total time by 20%), although this is parametrisation-dependent.
For the first few thousand frames the execution time is dominated by constant-time operations: extracting features from each frame and computing positions relative to around four nearby frames. However, the following parts of BoWSLAM currently have higher complexity in the number of frames T, and will eventually become the most expensive operations:

- Loop closure takes time up to $O(T \log T)$ (Section 8.3.3)
- Finding BoW matches takes time $O(T)$ (comparison with every other frame)
- Building a new BoW dictionary (clustering) takes time $O(T \log T)$, although per-frame this is only $O(\log T)$
- Finding a t-spanner for optimisation by TORO takes time $O(T \log T)$, and TORO takes time $\Omega(T)$, although this is performed only occasionally when a globally-accurate map is needed.

For long-term operation, constant time costs are necessary, however in a bounded environment a robot will eventually map everywhere it ventures, reducing the SLAM problem to a much simpler localisation problem. In the mean time the costs listed here could still be reduced further: the complexity of BoW matching can reduced to $O(\log T)$) by clustering the BoW image representations and only searching in nearby clusters (Fraundorfer et al., 2007), and BoW clustering can be avoided entirely once a dictionary appropriate for the environment is found. Alternatively, for some environments, a dictionary built offline from training data could be used (as used by Cummins and Newman, 2008a; Konolige et al., 2009), which would avoid the need to retrain while exploring entirely.
Memory usage is linear in the number of frames and is dominated by storing descriptors, requiring around 40KB per frame. This gives about 25,000 frames/two hours of operation per GB of memory. This could be reduced greatly if correspondences directly from the BoW representation were used, as proposed in Section 5.6; this would remove the need to store descriptors once a BoW dictionary representative of all the environments that might be encountered was found.

8.5.2 Discussion

BoWSLAM can navigate in three dimensions with very few assumptions about motion, however a motion model helps to reduce scale drift. Ideally rotation-invariant descriptors should be used, as some robots (for example climbing robots) may visit the same location with a different orientation. Rotation-invariant SURF descriptors perform as-well or better than patch descriptors, however they are too costly to extract in real-time.

We attempted to parametrise MonoSLAM (Davison, 2003) to work on the University of Alberta dataset, however features near to the camera move too far between frames to be tracked (or when allowed motion is increased the tracking jumps between nearby similar-looking features). When a corner is reached, rapid feature motion causes tracking to fail, and the estimated path continues straight ahead due to the constant velocity motion model. These difficulties are largely related to the data used however; other single camera SLAM schemes generally require considerably higher framerates.

Like all SLAM schemes with global loop closure detection, there is a risk that BoWSLAM may close loops incorrectly in self-similar environments, corrupting part of the map, however this is only a problem if the geometry of the scenes is also compatible. So far the only incorrect loop closures observed have been in simulated data where the same image is observed in different locations. Other schemes (Cummins and Newman, 2009; Konolige et al., 2009) have also found geometric constraints sufficient to prevent false loop closure. If incorrect loop closure did occur, the incorrect link would be incorporated into the t-spanner, corrupting the optimised map, however by actively detecting these incorrect loop closure events, the damage could be undone simply by deleting the incorrect link.

In Chapter 1, it was proposed that a single camera SLAM scheme should be capable of operating indefinitely in a bounded environment, i.e. that the complexity should grow only with the area explored. BoWSLAM does not yet achieve this aim, as new relative position estimates are added as previously-explored areas are traversed, however by dropping frames which are in the approximately the same place as previous frames, the cost does grow more slowly. In addition, the number of non-leaf edges in the t-spanners which are optimised only grows when new areas are visited, or loops are closed. In future we plan to build a t-spanner incrementally, allowing a global map to be generated online. The cost of this optimisation would only grow when new areas are explored, or where new loops are closed.

Cameras used to capture image sequences used in this chapter were calibrated using the MATLAB camera calibration toolbox Bouguet (2010), which uses the calibration-pattern-based method of Zhang (1999). For the low-cost wide-angle Contour HD helmet camera, uncertainty in radial distortion coefficients equivalent to a localisation error standard deviation of 0.015 radians at the image edge, or six pixels, are observed, even though a fourth-order correction model is used. This demonstrates one of the difficulties of implementing SLAM on low-cost platforms. Errors are substantially lower towards the centre however. For the Canon 400D SLR camera however, a second-order correction with errors of 0.001 radians (less than a pixel at worst), is found.

Points at infinity are used for relative pose estimation, but not for relative scale estimation. These points arise when distant points are viewed, and have localisation errors large enough that their depth cannot accurately be recovered. The threshold at which points are classified as being 'at infinity' appears to affect the level of scale drift observed. Improving localisation errors (Chapter 4), and ensuring cameras are accurately calibrated, could have a significant impact on scale drift observed.

8.5.3 Limitations of BoWSLAM

In this section I discuss the limitations of BoWSLAM, and describe how further development could prevent these problems. While successful in a wide range of environments, there are still situations in which BoWSLAM fails. When positioning fails, a new map component is usually started, and this may be fused to the original map on loop closure, however multiple failures lead to multiple disconnected components.

One situations where positioning fails is when featureless walls are viewed when cornering rapidly. This situation sometimes occurs when negotiating doorways and stairwells (Figure 8.26). In this case overlapping regions of consecutive frames containing few reliable point features, and even fewer features are matched

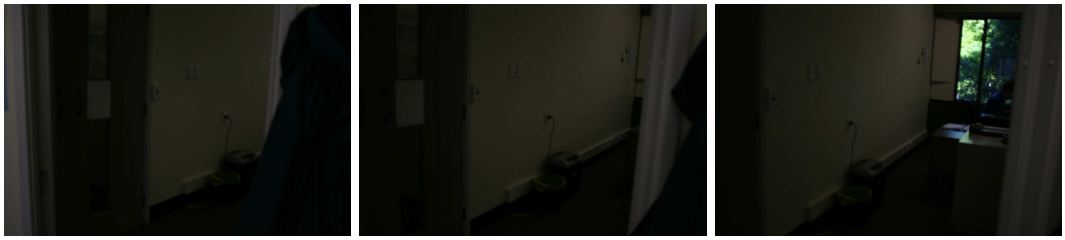


Figure 8.26: Environments where positioning is challenging include locations where few point-features are tracked between frames, particularly in locations such as doorways. Possible solutions to this problem include higher-level features (line features, or even the concept of a doorway), or omnidirectional images.

across triples of frames. One way to resolve this limitations would be to incorporate topological constraints into the map, for example identifying that locations are spatially nearby, but with unknown relative scale or orientation, however loop closure would be necessary (and not always sufficient) to produce consistent global maps, and loop closure is unlikely to occur if an environment is only accessible by the challenging door or stairwell. An alternative way to resolve this situation would be to recognise the features of typical buildings, so that the robot (or pedestrian navigation system) can infer its relative position from knowledge that it has, for example, climbed stairs with two right-angle corners, or passed through a door (as proposed by Flint et al., 2010).
The other situation when positioning fails is when a robot stops and rotates, then moves away. Tracking scale is impossible in this situation, however orientation can be recovered accurately, so, as with the case of a stationary camera, this could be handled as a special case, with only a topological constraint introduced between poses. Scale can be resolved later by closing a loop, or by recognising an object of known size (Chapter 9). Alternatively, both of these situations could be resolved by using omnidirectional images.

8.6 Conclusions

This chapter describes BoWSLAM, a scheme combining techniques developed throughout this thesis in order to enable real-time navigation of dynamic environments using a single camera alone. An accurate map is generated after travelling 2.5km over 25 minutes; considerably further than the tracks over which previous single camera SLAM schemes have been demonstrated. BoWSLAM demonstrates that autonomous mobile robots may be positioned for extended periods in large scale environments using just a single camera.
Key to this performance is ensuring that positioning does not fail even in the most challenging situations. Three innovations make this possible. Firstly, the BoW algorithm together with BaySAC, are used for fast, wide-baseline feature matching. This enables positioning despite substantial camera movement, and outliers from moving or repeated features, and is necessary when computing position estimates relative to more distant frames. Secondly, the BoW algorithm is used to select good candidate frames from which to compute positions, this allows positioning to continue despite sequences of frames being unusable, for example due to moving objects. Thirdly, a graph-based representation of multiple position hypotheses allows subsets of good position estimates to be found, despite the presence of gross outliers. Scale estimates are modeled as a dual graph, which is optimised to correct scale drift when loops are closed.

8.7 Future development

We have demonstrated BoWSLAM's robustness to gross errors and challenging environments, however there are several areas where accuracy can be improved. Firstly, scale drift is still a large source of errors in maps, particularly when navigating by dead-reckoning (far from previously-visited locations). One obvious solution is to incorporate measurements from additional sensors; in particular the complementary measurements from

low-cost IMUs (like those found in many modern phones). An alternative solution however, which we explore in the next chapter, is to extend the high-level representation of each frame by using the BoW database to learn and recognise classes of objects. Measurements of the size of these objects can then be used to estimate the true scale of the world.

We also plan to improve positioning accuracy by using a bundle adjustment-based approach (Triggs et al., 1999; Mouragnon et al., 2009). Position estimates over good sequences of frames could be refined, and information from multiple good tracks could be combined (Section 8.3.3), however a better approach may be to focus on those regions with the highest uncertainty. Strasdat et al. (2010) observe comparable levels of scale drift in relatively small and static loops, despite the use of incremental bundle adjustment to refine the map, indicating that the simple two-view relative pose computation is not necessarily the factor limiting BoWSLAM's accuracy.

A more profitable way to improve the accuracy of global maps may be to improve BoWSLAM's optimisation of scale drift. BoWSLAM currently uses the 3D version of TORO, by Grisetti et al. (2007a), to generates global maps offline, taking typically a few seconds. These maps are only approximations to the true ML maps however, and currently scale drift within cycles is not optimised. Ideally an accurate global map would be available online, and this map would be more accurate if errors in scale were incorporated (as shown by Strasdat et al., 2010). Following the recent trend away from global Euclidean coordinate frames, a new pose graph optimisation scheme, HOG-Man (also by Grisetti et al., 2010) has recently been developed, featuring online optimisation of pose graphs, where each pose is optimised in its own local coordinate frame. In addition, a hierarchy of subgraphs (similar to the hierarchy of graphs used by Frese et al., 2005, in multi-level relaxation), allows optimisations to be performed online, with only poses which will be adjusted significantly on each update being updated. Substantially more accurate maps than those generated using TORO are presented. We plan to investigate the modification of HOG-Man to incorporate the distribution of changes in scale between poses. One approach would be to alternately optimise scale estimates and relative positions, however it may be possible to incorporate relative scale estimates into the transformations between poses. A scheme to incrementally estimate the outlier-free pose graph to be optimised would then enable online real-time global mapping.

Acknowledgement

The University of Alberta CSC data set was obtained from The Robotics Data Set Repository (Radish) (2003). Thanks go to Jonathan Klippenstein for providing this data.

Chapter 9

SCORE2: Scale Correction by Object Recognition

Abstract

This chapter proposes a novel solution to the problem of scale drift in single camera SLAM, based on recognising and measuring objects. When reconstructing the trajectory of a camera moving in an unknown environment the scale of the environment, and equivalently the speed of the camera, is obtained by accumulating relative scale estimates over sequences of frames. This leads to scale drift: errors in scale accumulate over time. The proposed solution is to correct the scale estimate by recognising objects of known size. A Bag-of-Words-based scheme to learn object classes, to recognise object instances, and to use these observations to correct scale drift is described, and is demonstrated reducing accumulated errors by 64% while navigating a large outdoor environment.

9.1 Introduction

This thesis describes the development of BoWSLAM, a scheme allowing a mobile robot to position itself in an *a priori* unknown environment, using only a single camera. BoWSLAM builds a map of the environment as it is explored. The map is optimised to correct some of the errors that accumulate, enabling long-term positioning, and as long as the trajectory regularly re-visits previously mapped areas, accurate maps can be generated. Not all errors can be corrected however, in particular errors which accumulate within large loops, and in tracks without loop closure. These errors can severely distort global maps, so should be minimised to ensure position estimates are still useful.

A significant source of error unique to single camera SLAM is scale drift. A robot with a single camera can resolve the scale of the world, and hence its speed, by identifying objects of known size, such as the calibration objects used to initialise some single camera SLAM schemes (Davison, 2003; Eade and Drummond, 2006), or previously mapped landmarks. As the robot explores away from previously mapped areas, small errors in the robot's scale estimate accumulate, eventually rendering position estimates useless. Even when a loop is closed, substantial uncorrected errors in scale can remain, due to the trade-off between adjusting rotations, translations, and scales in order to reduce some global error function, therefore it is always worthwhile to minimise this scale drift where possible.

In an earlier paper (Botterill et al., 2009a), we described a new algorithm, SCORE (Scale Correction by Object Recognition), that exploited a novel solution to this problem. SCORE attempts to reduce scale drift by firstly identifying object classes from the robot's internal SLAM map, secondly by measuring the distribution of size within classes, and thirdly by using measurements of these objects at later times to improve the robot's scale and speed estimates. Results were mixed; while object classes and size distributions

were learned and measured in a variety of environments, the speed updates provided by later measurements of these objects only reduced scale drift significantly in one indoor dataset containing many repetitive structures. The reason for this failure was partly the simple model of scale used in the early version of BoWSLAM, and partly that SCORE approximated object class size distributions with a normal distribution, which was often a poor approximation. Since then, BoWSLAM has been improved in many ways, in particular by modeling scale estimates as lognormally-distributed random variables. Correspondingly, the SCORE2 algorithm developed in this chapter has been improved in many respects, in particular the model for object class size distributions is now lognormal, this proves to be a considerably better model than the normal distribution and allows scale estimates to be integrated efficiently with the SLAM map. In this chapter, SCORE2 is demonstrated on indoor and outdoor datasets: in a large outdoor environment, errors in the optimised map are reduced by 64%, and in one indoor dataset, scale drift is reduced by 75%.

In some situations, scale drift can be eliminated by other means, in particular by using measurements from other sensors, or by making assumptions about properties of the environment or of the robot's motion. Measurements from other sensors, including IMUs, wheel encoders or additional cameras, provide complementary measurements which can be used to estimate the true scale of the world at any time (Hide et al., 2010). Alternatively, as described in the previous chapter, an assumption that reconstructed 3D points in the world have depths from a particular distribution allows scale drift to be eliminated. That assumption improves the accuracy of maps, despite distortions where it does not hold. For a single camera attached to a wheeled vehicle, such as a car or bicycle, the distance of the camera above the ground plane is approximately constant, so the scale of the world can easily be identified by identifying points on the ground plane (Scaramuzza et al., 2009). Such assumptions are not always true however, and alternative sensors are not always available. SCORE2 demonstrates that these assumptions, or additional sensors, are not necessary to correct scale drift.

SCORE2 is not only useful for scale drift correction: the object recognition (OR) component of SCORE2 could also be applied to other robot tasks, such as robust loop closure (Section 9.6), and task execution (Jensfelt et al., 2005), and the capability to estimate the sizes of objects seen in only a single image (or from a distance) could potentially be used to aid path planning. SCORE2 also provides a mechanism for additional measurements of scale from other sensors to be incorporated into BoWSLAM, or for measurements from these sensors to be validated, in order to remove gross errors.

A related problem to estimating a camera's speed from object observations is the problem of reconstructing a scene observed with a fixed camera by making observations of moving objects. Jacobs et al. (2010) reconstruct a depth map for a scene imaged with a fixed camera by tracking the shadows of clouds moving across it. The true motion of a point on the edge of a shadow is assumed to match the wind speed, allowing the depth of the point to be inferred.

The following section describes the OR and machine learning techniques that SCORE2 is based on, and previous use of OR by mobile robots. Section 9.3 analyses the problem in detail, and describes the SCORE2 algorithm. Section 9.4 demonstrates SCORE2 successfully reducing scale drift in single camera SLAM, and the final sections discuss our conclusions, and SCORE2's possible application to other navigation problems.

9.2 Background

This section describes previous use of OR by mobile robots, and the OR schemes SCORE2 is based on. Note that OR schemes either aim to recognise classes of objects (e.g. trees), or aim to identify particular objects (e.g. the small plane tree outside our office). Both OR and object class recognition are discussed here.

9.2.1 Real-time object recognition

For recognising objects from a small set of distinctive items, local area descriptors such as SIFT (Lowe, 1999) or SURF (Bay et al., 2008), or descriptors of colour (Jensfelt et al., 2005) are commonly used. A descriptor in an image which matches the descriptor of an object from the database indicates that the image contains an object resembling the database object. OR schemes based on individual descriptors do not scale to large

object databases however, as individual descriptors are not sufficiently distinctive. Instead, combinations of descriptors are used, and the most popular model to represent these descriptors is the BoW model. The BoW model, described in detail in Section 5.3, describes an image by mapping each descriptor to its closest match from a 'dictionary' of 'image words'; a discrete set of representative features. Testing whether an image contains a particular set of image words associated with an object is very fast, allowing large image databases to be searched for large numbers of objects. The BoW model also enables the distinctiveness of each word to be estimated, normally using the TF-IDF score (Section 5.3). A distinctive (rare) feature associated with an object class is more useful for OR than a common feature which occurs in many different contexts.

The BoW image representation also facilitates the automated learning of object classes or categories. Classes are often defined by a weighted set of image words that tend to co-occur, and hence are assumed to arise as a result of an object in that class being visible. An image containing many of the words defining an object class, particularly those that are most distinctive, is likely to show an instance of an object from that object class.

Object classes can be learned automatically from the BoW representations of images of these objects. One popular algorithm for automatically identifying object classes is Latent Semantic Analysis (LSA; Dvorsky et al., 2003, popular variants include Probabilistic Latent Semantic Analysis, Sivic et al. (2005); and Latent Dirichlet Analysis, Li and Perona (2005)). In LSA, a sparse matrix is constructed where rows represent images, columns represent image words, and elements are the occurrence counts of each word in each image. Each occurrence count is weighted by the word's distinctiveness (TF-IDF score). The principal components of the matrix are then found by SVD. These principal components represent sets of co-occurring features that partition images into those containing particular classes of objects; the three strongest components might characterise for example 'cars', 'trees' and 'bikes', depending on the contents of the training images. LSA is an unsupervised learning algorithm. The principal components will identify sets of features, the presence or absence of which will which partition the training images. No *a priori* knowledge of the contents of the training images is required (when training data is already categorised, e.g. into 'cars' and 'trees', then a Support Vector Machine, or similar algorithm can be used; Section 5.3.2).

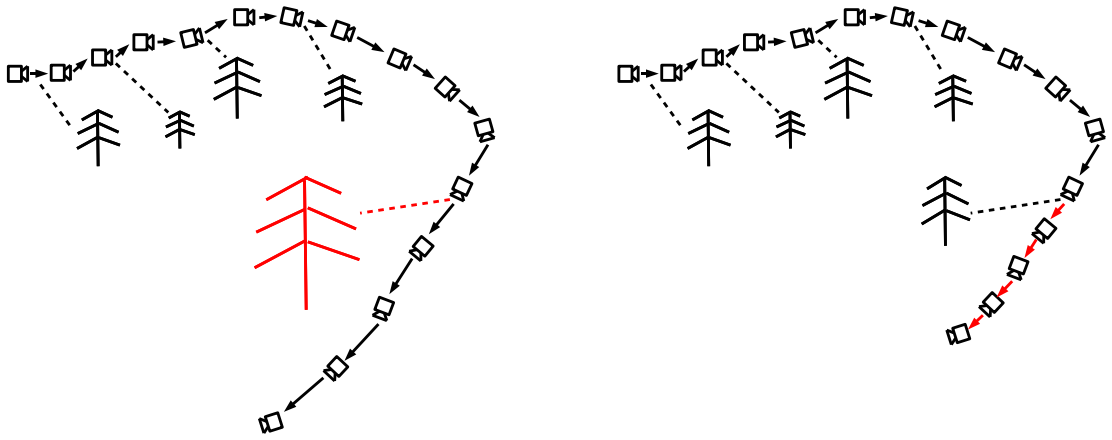
BoW-based OR is particularly suited for autonomous mobile robots as many SLAM schemes already maintain BoW databases in order to detect global loop closure (Eade and Drummond, 2008; Newman et al., 2009; Konolige et al., 2009; Sibley et al., 2010), and because BoW schemes are capable of real-time recognition (Sivic and Zisserman, 2003; Nistér and Stewénius, 2006; Cummins and Newman, 2008a), and occasionally real-time training (Section 5.4, and Angeli et al., 2008a; Eade and Drummond, 2008).

Many other schemes for OR have been demonstrated, for example by matching the 3D structure of objects (Brown and Lowe, 2005), by texture analysis (Serre et al., 2007), or with hybrid approaches (Rothganger et al., 2006). These approaches are often highly accurate but are rarely fast enough for real-time OR (taking several minutes per object in these examples).

9.2.2 Object recognition by mobile robots

Several mobile robots have been equipped with OR capability, but to our knowledge, all recognise particular object instances from a manually-trained database. OR is used to improve the performance of visual SLAM by Ahn et al. (2006) and Castle et al. (2007). These schemes both manually construct databases of planar objects (posters and photographs), then recognise these (using simple SIFT matching) and use them as robust landmarks for stereo EKF-SLAM, and MonoSLAM respectively. Castle et al. extended their scheme to incorporate the known size of particular objects into the SLAM solution (Castle et al., 2009), however the major limitation of these schemes is that they are only beneficial in environment containing these objects, making the schemes of limited use for exploring unknown environments.

Another application for OR by mobile robots is for task execution. Ramisa et al. (2008) use a BoW model to recognise instances of various household objects while a robot explores. A similar scheme by Jensfelt et al. (2005) recognises a pre-defined set of objects, and adds their positions to the SLAM map. Objects in training images are characterised by the set of colours they contain, then as the robot explores, occurances



(a) A robot observes objects as it explores. Later, substantial scale drift accumulates. A subsequent object observation is perceived to have a different size (highlighted).

(b) The assumption that the size of the observed objects come from the same distribution allows scale drift in the highlighted edges to be corrected.

Figure 9.1: Figure showing how SCORE2 corrects scale drift. (a) As the robot explores it observes and measures objects. Later, scale drift degrades its position estimate. (b) A subsequent observation of the object allows scale drift to be corrected.

of the same sets of colours are found in captured images. The robot can later be commanded to fetch one of the objects it has identified.
These schemes demonstrate some uses of OR for mobile robots, however all are limited to recognising a small number of objects from a pre-defined database, so are only useful when the environment is likely to contain those particular objects. To overcome this limitation, SCORE2 is designed to learn to recognise the object classes which are encountered as the robot explores, so that no prior training data, or prior knowledge about the environment is needed.

9.3 Scale Correction by Object Recognition

This section describes SCORE2, our new scheme to learn classes of objects, to measure instances of these objects, to estimate the distribution of sizes of each class of objects, and to use later object size measurements to correct accumulated errors in scale. This section is organised as follows: firstly, Section 9.3.1 analyses of the problem of correcting scale drift from object observations. Secondly, Section 9.3.2 gives an overview of the proposed solution. Thirdly, Section 9.3.3 describes how objects are measured and how object classes are learnt. Fourthly, Section 9.3.4 describes how these uncertain object size measurements are used to parameterise a size distribution for each class. Finally, Section 9.3.5 describes how measurements of objects from these distributions are used to correct accumulated errors in scale.

9.3.1 Analysis of scale correction problem

A robot identifies classes of objects as it explores an unknown environment. The distribution of sizes in each class is measured. As the robot travels into a previously unmapped area, its estimate of scale drifts, and as a result its position and speed estimates deteriorate. When the robot observes and measures objects belonging to classes observed earlier, it can use these measurements to improve its scale estimate. This idea is illustrated in Figure 9.1.

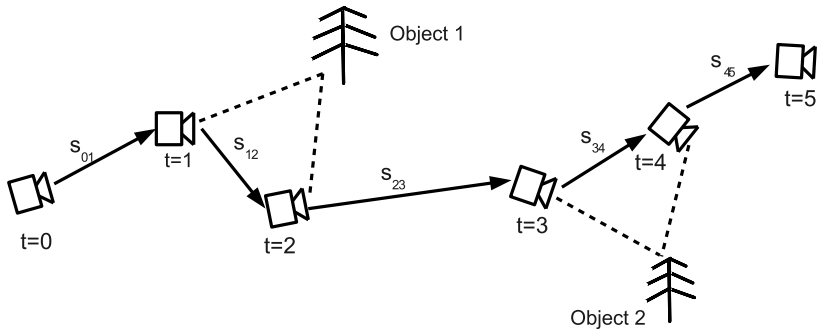


Figure 9.2: An object observed at times $t = i$ and $t = j$ is measured by reconstructing features on it, then measuring the distances between these features. The size of this measurement is proportional to the baseline length, s_{ij}. As baseline length estimates are correlated, different object measurements are also correlated.

For each frame:

When re-training:

1. Identify (learn) the B object classes
2. Measure each object (subject to scale estimate from SLAM)
3. Estimate distribution parameters
4. For each edge combine the most likely scale given the observations with the measured scale

When a new edge (relative pose estimate) is added:

1. Observe any objects reconstructed here
2. Update object size distributions
3. Combine the scale estimated from the observations with the measured scale

Figure 9.3: Overview of the SCORE2 algorithm

An object is measured by reconstructing features on the object which are viewed from two frames, then measuring the distance between the features. The major source of errors in an object measurement is usually the error in the estimated baseline between the pair of frames (which is equivalent to the estimated speed of the robot). This source of error applies equally to all objects measured in the same two frames. In addition, as scales are accumulated sequentially, errors in an estimated scale are correlated with errors in other scale estimates from which this scale was calculated (or will be used to calculate), and hence errors in all object measurements made along a trajectory are correlated (Figure 9.2).
As with the SLAM problem (Chapter 3), where correlations between landmark positions and the robot pose should be modelled (Smith et al., 1990; Durrant-Whyte and Bailey, 2006), modeling the correlations would be useful for estimating object class size distributions. Maintaining these correlations would be challenging however. Unlike SLAM measurements, which can be partitioned into local submaps, the most suitable object classes for scale drift reduction are likely to be encountered throughout the robot's environment. Currently measurements are assumed independent, to simplify the problem. This approximation is most appropriate when scale drift is low, which is when the most reliable measurements of objects (those contributing most to their estimated size distribution) are made.

Table 9.1: Selected notation used in this chapter.

B	Number of objects
w_1, w_2	Two co-occurring words defining an object class
M	Term-document matrix, with elements m_{ij}
m_{ij}	The number of times word j occurred in image i, weighted by TF-IDF
λ_b	Measurement of object b (distance between features defining b), assuming baseline 1
γ_b	Underlying measurement of object b if scale was known
r_b, s_b	Scaled and logged measurement of a measurement, with variance from uncertainty in scale
μ_b, σ_b	Parameters of the lognormal distn. of sizes of objects in class b
$\mathbb{M}_b = \{(r_{bi}, s_{bi}), i = 1, ..., N\}$	N measurements of object B
D_{SLAM}, G^2_{SLAM}	Parameters of a scale estimate from SLAM; $s \sim$ Log-$\mathcal{N}(D_{SLAM}, G^2_{SLAM})$
D_{OR}, G^2_{OR}	Parameters of a scale estimate from SCORE2; $s \sim$ Log-$\mathcal{N}(D_{OR}, G^2_{OR})$
$D_{combined}, G^2_{combined}$	Parameters of combined scale estimate; $s \sim$ Log-$\mathcal{N}(D_{combined}, G^2_{combined})$
$M_i, 1/t_i^2$	Parameters of conjugate distn. for μ; $\mu \sim N(M_i, 1/t_i^2)$ after i observations
α_i, β_i	Parameters of conjugate distn. for $\tau = 1/\sigma_b^2$; $\tau \sim \Gamma(\alpha_i, \beta_i)$ after i observations

9.3.2 Overview of solution

The SCORE2 (Scale by Object Recognition) algorithm is outlined in Figure 9.3, and is detailed in the following sections. In summary, when retraining occurs (when a new BoW dictionary is created), a set of object classes is identified. The measured sizes of objects in these classes are used to improve existing and new scale estimates.

SCORE2 is designed to integrate with BoWSLAM, the single camera SLAM scheme described in this thesis. BoWSLAM represents every frame as a 'Bag-of-Words' (BoW), using the scheme described in Chapter 5. This high-level representation is used for active loop closure detection, fast feature matching, and to select the sequence of frames used to position each new frame relative to. The representation is also ideal for OR. Although SCORE2 is designed to integrate with BoWSLAM, it could easily be adapted to other single camera SLAM schemes which index a subset of frames into a BoW database (for example Eade and Drummond, 2008).

The effect of SCORE2 is to propagate reliable scale estimates from better mapped areas to areas where the scale is much less certain, but where the same kinds of objects are visible. In practice there is often a large difference between the uncertainty in scale estimates in different areas, for example scale estimates are accurate when the robot is moving is a straight line in a feature-rich environment, then deteriorate when the robot corners. SCORE2 is also able to initialise a scale estimate when a new map component is started, after the robot has become lost.

9.3.3 Classes of measurable objects

This section describes how SCORE2 defines a class of objects, and how objects in that class are measured. Notation used is summarised in Table 9.1.

SCORE2 requires that object classes can be learnt and identified in real-time, and that identified objects can be measured. The object classes recognised in real-time by contemporary OR schemes (Section 9.2.1) consist of occurrences of one or more of a set of features defining that class. The most obvious measure of an object in one of these classes is the distance between two features on the object. Measurements of more than two features are not considered, as this would add to the complexity (there are $\binom{n}{2}$ possible measurable distances between n points), and would introduce difficulties in coping with partially-observed and partially-reconstructed objects. As a result, SCORE2's object classes are each defined by the co-occurrence of two image words. Multiple instances of the same object are likely to be visible in many scenes; however by

assuming objects are separated by more than the separation of the features within them, only the least separation between all possible pairs of two features visible in a scene must be measured.
To identify co-occurring image words, the same term-document matrix as LSA is used. This sparse matrix, M, has elements m_{ij} representing the number of times word j occurred in image i, multiplied by word j's TF-IDF score. Each row of this matrix, $\mathbf{r}_i$, is the BoW representation of image i.
LSA finds co-occurring features by computing the principle components of the matrix M by SVD. M could realistically have 10,000 rows, 50,000 columns, and 5 million non-zero entries however, making the computation challenging for a real-time application. In addition, only co-occurring pairs of features are of interest, rather than co-occurring sets of features.
The principle component $\mathbf{p}$ of M is a unit vector maximising $\|M'\mathbf{p}\|$, where M' is given by subtracting the column's mean from each column of M. As co-occurring pairs of features are required, vectors $\mathbf{p}_2$ with two equal nonzero elements are found, which each corresponds to one co-occurrence. For each co-occurrence of two words w_1 and w_2, we calculate the sum:

$$\sum_{\text{Images } i \text{ containing } w_1, w_2} m_{i\,w1}^2 + m_{i\,w2}^2 \tag{9.3.1}$$

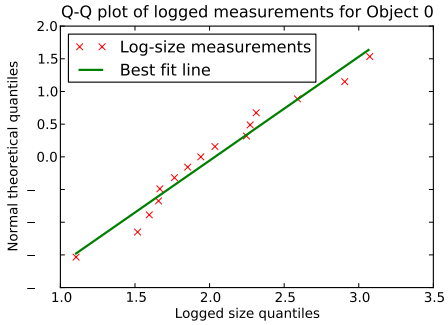
If co-occurring words only occurred in these pairs, then the pair of words maximising $\|M'\mathbf{p}_2\|$ would be the pair of words for which this sum is greatest. The B pairs of words with the highest value for this expression are chosen as our B object classes. Only features that are successfully reconstructed are counted, which limits the number of co-occurrences that must be considered, and avoids learning object classes which cannot often be measured.
Equation 9.3.1 is a heuristic, as is LSA itself, however the experiments in Section 9.4 show it to work reasonably well at identifying pairs of features on objects which tend to co-occur. Many alternative schemes for choosing the word pairs could be used, for example choosing the pairs of features with the highest mutual information. The mutual information of two words is a measure of the amount of information provided about the occurrence or non-occurrence of one word, given that the other has been observed, and the co-occurrences with the highest mutual information are selected by Cummins and Newman (2008b) in order to build a Chow Liu tree (Section 5.3). Computing mutual information in this scheme requires heuristic estimates of word co-occurrence probability however.
Once this set of object classes has been identified, the instances of the objects are identified (by searching each of the sets of 3D points reconstructed between pairs of frames) and measured.

9.3.4 Estimating object class size distribution parameters

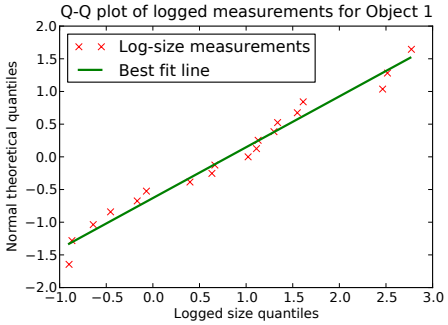
This section describes how a distribution is fitted to the noisy measurements of object sizes, as described in the previous section. These measurements incorporates both measurement error, and variability in object sizes within each class. There are five sources of variability in the observed object sizes:

1. Uncertainty in the baseline length (scale) from which objects are reconstructed.
2. Variation in true size of objects (e.g. cars are 1.5 to 2.5m high).
3. The same two-word combination occurring in multiple contexts.
4. Errors in reconstructing 3D point positions.
5. Errors from measurements of multiple partially-visible objects, or features occurring in multiple objects.

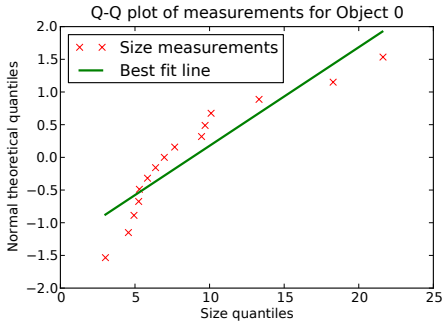
The combination of these sources of variability is assumed lognormal, as this fits observations well. Figure 9.4 shows that the first two object classes identified in one outdoor dataset have measurements which are considerably better approximated by a lognormal distribution than a normal distribution. The lognormal assumption makes integration with the lognormally-distributed scales estimated by BoWSLAM simple, and



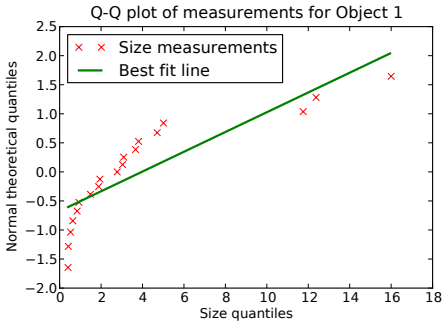
(a) The lognormal distribution is a good model for size measurements of Object 0.



(b) The lognormal distribution is a good model for size measurements of Object 1.



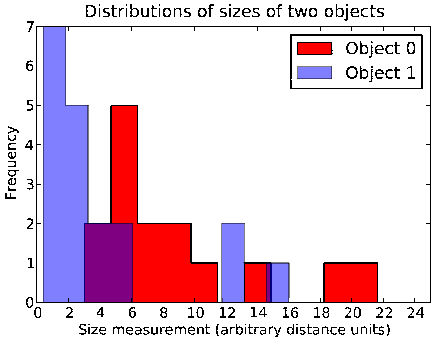
(c) The normal distribution is not as good a model as the lognormal distribution for size measurements of Object 0.



(d) The normal distribution is a poor model for size measurements of Object 1.

Figure 9.4: Q-Q plots (Thode, 2002) of distributions of object size measurements for the first two objects identified in an outdoor dataset, which indicate that the lognormal distribution is a good model.

(a) Distances measured of two objects, 'round shadows', and 'car wheels'.



(b) Distributions of the sizes of these objects (15 and 19 measurements are made respectively).

Figure 9.5: Two object classes measured in an outdoor dataset have significantly different size distributions. The lognormal distribution is a good model, as it is unimodal, heavy-tailed, and non-negative.

is underpinned by evidence that the lognormal distribution is very often a good model for size measurements including the heights of plants or people, or the length of words (Limpert et al., 2001). Object size measurements are also assumed to be independent.
An object b is observed in two frames, and is measured to have length λ_b when points are reconstructed with baseline 1. If the baseline length was known, then the object would be measured to have size γ_b, drawn from the underlying object class size distribution Log-$\mathcal{N}(\mu_b, \sigma_b^2)$, which includes variation in object sizes, and measurement error.
From SLAM, the baseline length is estimated to be lognormally distributed with parameters D_{SLAM} and G^2_{SLAM}, therefore λ_b is lognormally distributed about γ_b, and the logged object size measurement $r_b = \log(\lambda_b) + D_{SLAM}$ is normally distributed about $\log \gamma_b$ with variance $s_b^2 = G^2_{SLAM}$.
The joint likelihood of the object class size distribution parameters, μ_b, σ_b^2, given the observation r_b, s_b, is therefore:

$$L(\mu_b, \sigma_b^2; r_b, s_b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp(-\frac{1}{2}\frac{(r_b - \mu_b)^2}{\sigma_b^2 + s_b^2}). \tag{9.3.2}$$

We make N observations of each object, $\mathbb{M}_b = \{(r_{bi}, s_{bi}^2), i = 1, ..., N\}$. Note that the only assumption regarding the measurements m_{bi} is that they are drawn from the same distribution, unlike measurements in SLAM which should be distributed about their actual values.
For each object class, b, the parameters μ_b and σ_b^2 of the underlying distribution of sizes within the class are now estimated from these N object measurements.
As the measurements are assumed independent, the likelihood of any particular parameters, μ_b and σ_b^2, is given by the product of the p.d.f.s:

$$L(\mu_b, \sigma_b^2; \mathbb{M}_b) \propto \prod_{i=1}^{N} \frac{1}{\sigma_b^2 + s_{bi}^2} \exp(-\frac{1}{2}\frac{(r_{bi} - \mu_b)^2}{\sigma_b^2 + s_{bi}^2}) \tag{9.3.3}$$

The log of this likelihood function is twice differentiable, so a maximum can be found by Newton's method. The function often has local maxima where $\mu_b = r_{bi}, \sigma_b^2 = 0$ for some i however. While a MLE for μ_b, σ_b with $\sigma_b > 0$ can usually be found, it appears an MLE is not appropriate in this situation.
Instead, a Bayesian approach is used to estimate μ and $\sigma^2 = \frac{1}{\tau}$ (dropping subscript bs for clarity). Appropriate conjugate prior distributions for each parameter are the normal distribution for μ and the gamma distribution for τ (Weerahandi, 1995). When N measurements have been made:

$$\mu \sim N(M_N, 1/t_N^2) \tag{9.3.4}$$
$$\tau \sim \Gamma(\alpha_N, \beta_N) \tag{9.3.5}$$

where parameters are given by:

$$M_i = \frac{t_{i-1}M_{i-1} + \frac{r_i}{s_i^2}}{t_i + \frac{1}{s_i^2}} \tag{9.3.6}$$
$$t_i = t_{i-1} + \frac{1}{s_i^2} \tag{9.3.7}$$
$$\alpha_i = \alpha_{i-1} + \frac{1}{2} \tag{9.3.8}$$
$$\beta_i = \beta_{i-1} + \frac{1}{2}(M_N - r_i)^2 \tag{9.3.9}$$

Initially, uninformative parameters $M_0 = 0, t_0 = \epsilon, \alpha_0 = 1, \beta_0 = \epsilon$, for some small $\epsilon > 0$, are used. After N observations, the parameters for the distribution of object sizes are taken to be the mean of these distributions:

$$\mu = M_N \tag{9.3.10}$$
$$\sigma^2 = \frac{\alpha_N}{\beta_N} \tag{9.3.11}$$

Figure 9.6: Objects from the same class observed on several similar-looking doors, University of Alberta dataset. (Each object is measured in one place per pair of frames, however in BoWSLAM each frame is registered to many others, so the same type of object may be measured in several places in the same frame.)

σ^2 is initially very large, but typically after four or five observations is low enough that subsequent object observations can substantially affect scale estimates. By Equation 9.3.6, the most accurate measurements (those where s_i^2 is low) have the greatest effect on the estimated parameters.

9.3.5 Object observation and scale updates

When the BoW dictionary is re-created (Section 5.4), every scale estimate (between pairs of frames) is updated with information from measurements of objects reconstructed between the two frames. The same method is used to update new scale estimates as new edges are added.
The relative pose of two cameras has a baseline length (scale) estimate modeled by a lognormal distribution with parameters D_{SLAM} and G^2_{SLAM}. A set of M objects, with size distributions parametrised by $\{(\mu_j, \sigma_j), j = 1, ..., M\}$, is observed in a frame, with logged measurements $\{\log \lambda_j\}$. The scale of the edge $s \sim \text{Log-}\mathcal{N}(D, G^2)$ satisfies $s\lambda_j \sim \text{Log-}\mathcal{N}(\mu_j, \sigma_j)$, so $\log s \sim N(\mu_j - \log \lambda_j, \sigma_j)$. The MLE of the scale of these object observations is given by differentiating the log of this likelihood function and setting equal to zero:

$$D_{OR} = \sum_{j=1}^{M} \frac{\mu_j - \log \lambda_j}{\sigma_j^2} \tag{9.3.12}$$

$$G^2_{OR} = \frac{1}{\sum_{j=1}^{M} \frac{1}{\sigma_j^2}} \tag{9.3.13}$$

Combining these parameters with the parameters from SLAM gives:

$$D_{combined} = \left(\frac{D_{OR}}{G^2_{OR}} + \frac{D_{SLAM}}{G^2_{SLAM}}\right) / \left(\frac{1}{G^2_{OR}} + \frac{1}{G^2_{SLAM}}\right) \tag{9.3.14}$$

where $D_{combined}$ is the scale estimate combining the scale estimate from SLAM, and the scale estimate from observing the objects.

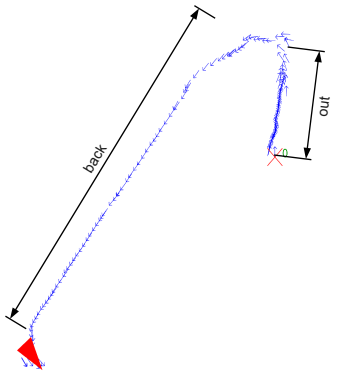


Figure 9.7: Map of robot poses, NZi3 dataset. Significant scale drift accumulates when cornering rapidly. Scale drift is measured as the ratio of the estimated lengths of two sections of the trajectory which have equal length (marked 'out' and 'back').

In summary, scale parameter estimates from SLAM alone are used to estimate the size distributions of objects. These size distributions are used to compute the most likely scale parameters for each edge, given the objects that have been observed. Observations of objects only substantially affect scale estimates when scale estimates from SLAM have high uncertainty compared to the scale estimates when object sizes were measured.
The computational cost of SCORE2 is dominated by identifying the B best object classes following retraining. This takes time $O(BT)$, as a fixed number of feature co-occurrences occur in each image. When processing the NZi3 dataset (described in Section 9.4), 4% of the total computational cost is related to SCORE2. As BoWSLAM has $O(T \log T)$ complexity per frame (Section 8.5.1), SCORE2 does not add significantly to the total cost.

9.4 Results

This section describes results obtained from running BoWSLAM, with SCORE2, on outdoor and indoor datasets. First, the 2.5km, 3662 frame, outdoor 'Suburban' dataset, described in Section 8.5, is used. About 2000 pairs of co-occurring features are identified and measured; of these, 455 have distributions with $\sigma_b < 1$ (higher values are too uncertain to have a significant effect on scale estimates). The 455 objects are each measured between 5 and 31 times (typically one or two per frame are measured); the object with the tightest distribution has $\sigma_b = 0.47$, whereas the least accurate scales from SLAM have distributions with $G^2_{SLAM} \approx 1$. Examples of objects found in a similar outdoor environment are shown in Figure 9.5.
Without using SCORE2, RMS errors of 198m remain in BoWSLAM's optimised map (Table 8.2). SCORE2 reduces the RMS errors to 71m; a 64% reduction in error. A model constraining the depths of reconstructed points in the world (described in Section 8.3.3) reduces these errors to 56m (although significant variation between different runs and parametrisations is observed).
Two indoor datasets are also used to evaluate SCORE2. The first dataset is captured at 2.8Hz with a Canon 400D SLR camera in the NZi3 office building at the University of Canterbury. Examples of frames from this dataset, and objects detected, are shown in Figure 9.9. The dataset consists of a 20m straight line, two sharp right-angle corners, then another 20m straight line. Scale drift often accumulates when cornering rapidly, and is measured by comparing the estimated lengths of the two 20m sections: the longer length, as a fraction of the shorter length, provides a measure of scale drift (Figure 9.7).
On this dataset, SCORE2 is compared to BOWSLAM alone, and the two motion models proposed in Section 8.3.3 (Figure 9.11). As levels of scale drift vary between runs (due to random BoW clustering), thirty runs are made in each case. Figure 9.8 shows the distribution of scale drift observed for each motion model. SCORE2 substantially outperforms a motion model constraining acceleration between frames ("Acceleration

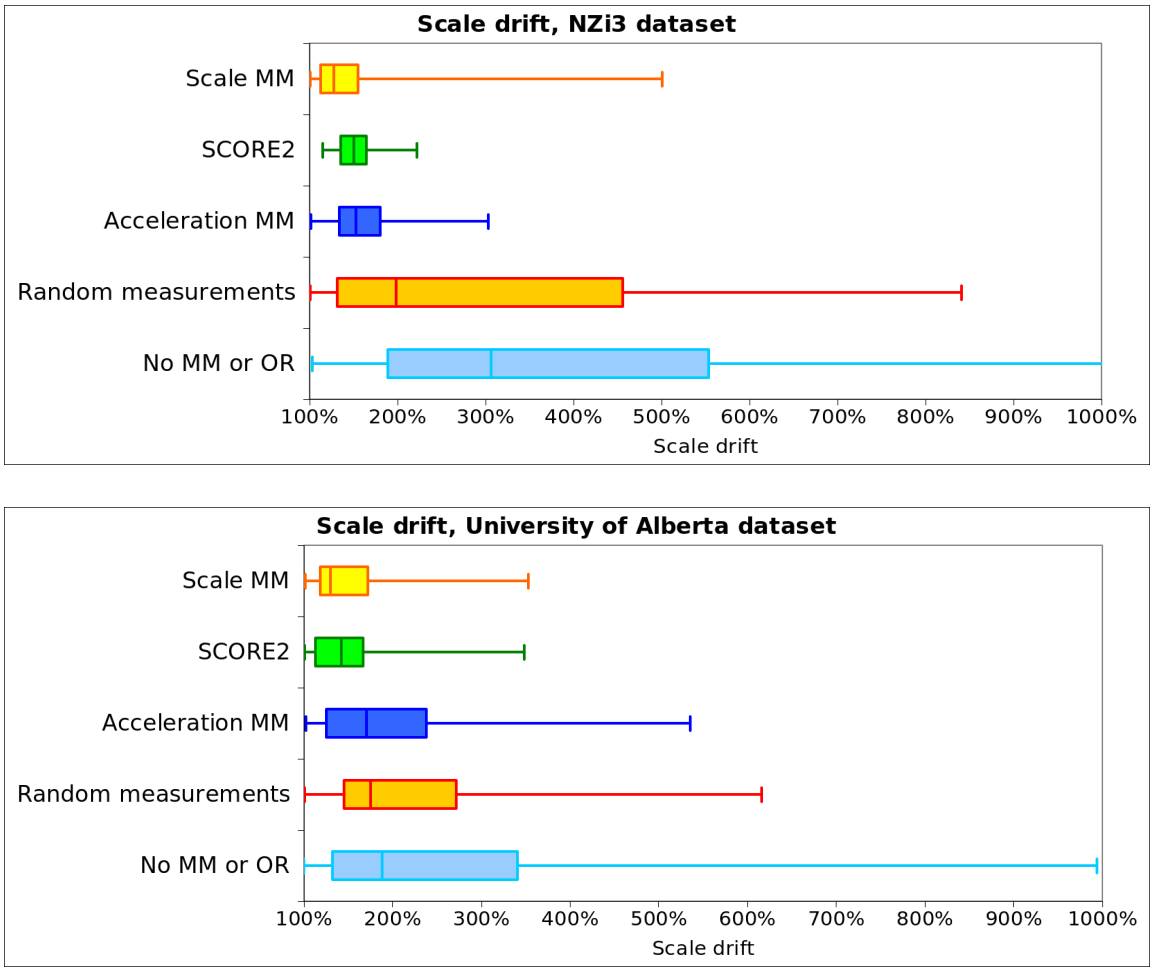


Figure 9.8: Box plots showing the distribution of scale estimates from 30 runs for each model on each dataset. The range, quantiles and median scale drift is shown. A score of 100% would indicate no scale drift. SCORE2 greatly reduces scale drift in both the NZi3 dataset and the University of Alberta dataset. SCORE outperforms a motion model constraining acceleration, and matches the performance of motion model constraining the scale of the world.

Figure 9.9: Frames from the NZi3 dataset. This dataset is captured in a modern office environment, including many repeated features, large amounts of reflective glass, and people. The distances between ceiling lights and sprinklers (Object 4), and the distances between ceiling beams and top windows (Object 6) are measured throughout. Some other detected objects do not appear to correspond to particular objects; measurements of these may have a similar effect to constraining the depth of reconstructed points.

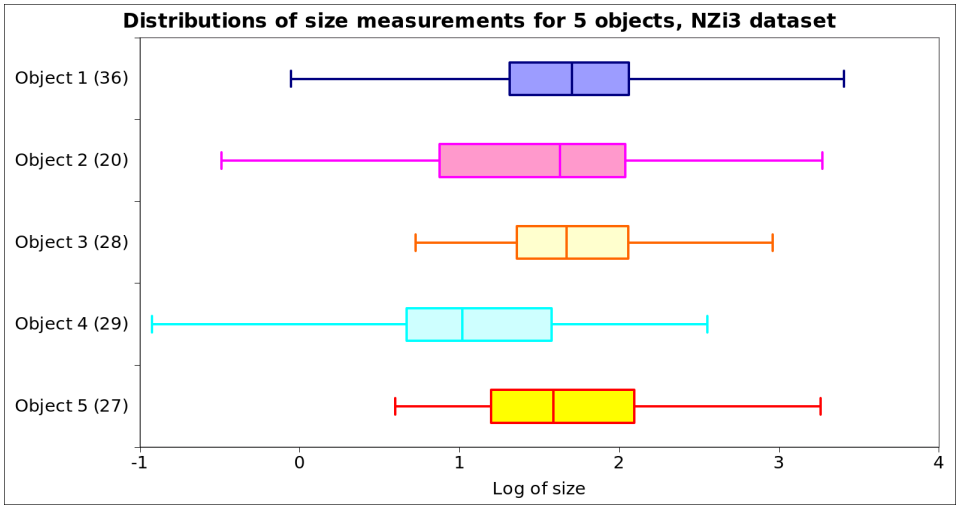


Figure 9.10: Distributions of sizes of the first five of the 19 object classes detected in the NZi3 dataset. Some have significantly different size distributions, indicating that the assumption that co-occuring features correspond to object classes with distinct size distributions is valid. The numbers in brackets indicate the number of measurements of each object.

MM"), and is close to matching the performance of the model constraining point depths ("Scale MM"). SCORE2 reduces scale drift by an average of 75% when no motion model is used.
It is possible that the objects chosen by SCORE2 are irrelevant; i.e. constraining any measurements of the world would be equally effective. To verify that this is not the case, SCORE2 is run where each object measurement is replaced with a measurement of a random pair of reconstructed points. This scheme ("Random measurements") only reduces scale drift by a small amount; considerably less than the reduction when using SCORE2. This verifies that SCORE2 works because object instances are being measured, rather then simply that the scale of the world is contrained. Secondly, we verify that objects detected do have significantly different size distributions. Figure 9.10 shows the distribution of the measured sized of the first five object classes detected in the NZi3 dataset (out of 19 in total). Several object classes detected have significantly different distributions of sizes.
A similar experiment is conducted with the University of Alberta dataset (described in Section 8.5). This dataset is captured from a wheeled robot following a rectangular loop along corridors. Frames from this dataset, and examples of objects detected, are shown in Figure 9.6. Again SCORE2 is compared to other motion models (Figure 9.8), and again, SCORE2 substantially reduces scale drift, and matches the performance of a constraint on the depths of observed points. Frames are rearranged in the test dataset, so that loop closure never occurs, however in the original dataset, loop closure substantially reduces scale drift, which can be partially corrected around the loop.

9.5 Conclusions

This chapter has demonstrated a novel solution to the problem of scale drift in single camera SLAM. BoW OR is used to learn object classes and to recognise objects. The distributions of object sizes in each class is estimated, and this information is used to correct scale drift. Scale drift can be corrected over indefinitely long tracks, and results in only a small increase in the computational cost.
SCORE2 reduces scale drift by 75% in one indoor dataset, and reduces the total accumulated error by 64% in a large outdoor dataset. SCORE2 has similar performance to a heuristic model of the scale of the world

SCORE2 BoWSLAM run with SCORE2 to recognise objects and correct scale drift.

Scale MM Reconstructed 3D points are assumed to have depths from a particular distribution, as described in Section 8.3.3.

Acceleration MM Constant velocity motion model, as described in Section 8.3.3.

Random measurements SCORE2 modified to make a measurement between two random points, rather then points on the object. Verifies that SCORE2 works because measurements of instances from object classes are made, rather than because it constrains the scale of the world.

No MM or OR BoWSLAM run without SCORE2 or any motion model.

Figure 9.11: Motion models used for comparison with SCORE2.

at reducing scale drift.
Experiments on indoor and outdoor datasets demonstrate that object classes with a variety of different size distributions are found, and SCORE2 is verified to use the different sizes of these different objects to correct scale drift.

9.6 Future work

There are many small improvements that would improve the performance of SCORE2, in particular more sophisticated methods for choosing and robustly measuring more complex objects. In this section some planned improvements and extensions are detailed.
Section 8.5.1 discussed plans to modify the pose graph optimisation framework used by BoWSLAM so that relative scale estimates are also refined. This modification could also be used to improve SCORE2. While SCORE2 currently assumes that measurements of objects are independent, this is not true in general, as errors in scale estimates are correlated with each other. While SCORE2 works well despite this assumption, the assumption could be avoided if distributions of relative object sizes were maintained instead. An object measured in two locations would then introduce a constraint between scale estimates in the two different locations, although at the expense of increasing the cost of the pose graph optimisation, as different sections of the map would no longer partition so easily.
In addition to the scale, many objects also occur in particular orientations (i.e. 'upright'). By identifying classes of objects which usually have the same orientation, errors in orientation could also be corrected.
The main limitation of SCORE2 results from the emphasis in this thesis that a robot must be able to operate without any prior knowledge of the environment to be explored. As a result, our BoW scheme is designed to build a dictionary dynamically, to avoid the need for training data. To integrate with BoWSLAM and this BoW scheme, SCORE2 learns classes of objects dynamically. For many practical applications however, training data, and prior knowledge about the robot's environment and the objects it contains, is available. In this case, both the BoW dictionary, and classes of objects which are likely to be encountered, could be learned in advance. Examples of object classes might include cars, bicycles, people, and household objects; these are amongst the 20 categories which are recognised by schemes competing in the PASCAL Visual Object Class challenges Everingham et al. (2008); many of these schemes are BoW based and some achieve recognition rates of around 30% of object occurances, in real-time (Uijlings et al., 2009).
Pre-defined object classes could provide absolute scale estimates wherever they were observed, and in addition could assist navigation in several other ways. Objects could be used to aid feature matching, for example by identifying which features are likely to be on moving objects (such as people), or alternatively could be used as features, and be matched between frames. As many OR schemes aim to recognise objects regardless of pose or scale, these features would form highly effective perspective-invariant descriptors, and possibly an even lower framerate could be used, leaving more resources for the processing required for OR.

Figure 9.12: An example of two frames where loop closure is not detected. Loop closure is very challenging when frames are captured from completely different viewpoints, and fails for these frames, resulting in uncorrected scale drift (Figure 8.16). High-level representations of these frames in terms of the objects they contain may allow frames like these to be registered however.

BoW location recognition could also be improved if frames were represented as sets of objects. In this thesis, almost all loop closure events in the datasets presented are detected, however one significant loop closure event is missed in the Suburban dataset, when the camera travels down a road in the opposite direction to the previous visit (Figure 9.12). In this case few features are visible from both directions, but a representation of the frames in terms of the objects they contain (which includes bikes, a glass building and road markings) could allow them to be matched. Detecting this loop closure event would enable the most significant error encountered when using the Suburban dataset to be corrected (Figure 8.16(a)).
The BoW representation used in BoWSLAM is not quite the same as those used for OR. In BoWSLAM, the BoW scheme uses images described by simple patch descriptors around FAST corners. These descriptors are suitable for relative pose computation, enabling the BoW representation to be used to obtain cheap correspondences. When BoW schemes are used for OR however, it is often found that descriptors sampled systematically from across images are more effective, as are combinations of descriptors (Uijlings et al., 2009; Hays and Efros, 2008; Li and Perona, 2005). To develop a SLAM scheme based around OR, these representations should be investigated further.

Chapter 10

Real-time aerial image mosaicing

Abstract

This chapter describes a scheme for seamlessly stitching together images captured from an aerial platform, in real-time, in order to provide an operator with a larger field-of-view. Both recent images, and images from earlier in a flight are used. To obtain real-time performance several of the latest computer vision techniques are applied: firstly the Bag-of-Words image representation allows overlapping images to be found efficiently, and provides cheap wide-baseline correspondences between them. Secondly the BaySAC robust estimation framework allows images to be registered efficiently from a prior motion model combined with large numbers of potential matches between cheap image patch descriptors. Thirdly an efficient seam-placement algorithm allows the rendering of a visually attractive mosaic. Results are presented on two sequences of high-resolution images, captured from a microlight and a UAV.

10.1 Introduction

This thesis has described a set of robust techniques which allow a robot to position itself from a stream of images. This set of techniques is not only applicable to robot positioning however; a related problem with very similar demands is real-time aerial image mosaicing, and in this chapter we describe how the techniques developed to enable BoWSLAM to robustly navigate visually challenging environments can instead be applied to this problem.

Image mosaicing is the process of joining overlapping images together to form a larger image, hence enlarging a cameras' field-of-view. One application for this is to increase the field-of-view of a camera streaming images back from an aerial platform, such as an aeroplane or UAV (Unmanned Aerial Vehicle), without the additional cost, and in the case of a UAV lens weight and/or bandwidth requirements of capturing larger images in the first place. This aids an operator by keeping features of interest in view for longer.

This chapter describes our scheme for real-time mosaicing of aerial images. High-resolution images are stitched together around the latest image, providing a seamless wide-angle view of the surrounding area, in real-time. Three innovations make this possible: firstly actively searching for adjoining images using the Bag-of-Words (BoW) algorithm, enabling the use of both recent images and images from earlier in the flight. Secondly, efficient computation of perspective transformations between images, combining correspondences provided by BoW with the BaySAC framework and a prior motion model where available. Thirdly, an efficient shortest-path algorithm that finds visually unobtrusive seams between frames, enabling a seamless mosaic to be rendered. No previous scheme has had this capability to generate large seamless mosaics in real-time.

This chapter is organised as follows: the following section describes the latest contemporary mosaicing techniques and their limitations; Section 10.3 describes our more efficient approach, Section 10.4 presents experimental results; and Section 10.5 discusses these results.

10.2 Background

There has been extensive research into image mosaicing; the review papers by Szeliski (2006) and Zitová and Flusser (2003) describe this literature in detail. Many of these methods however make assumptions that are not true for aerial images, or are computationally expensive batch processes for producing massive Google-Earth-style mosaics, so in this section we will concentrate on those techniques that are most likely to be suitable for real-time mosaicing of aerial imagery.
Zitová and Flusser (2003) identify four stages that most image mosaicing and registration processes have in common. These are:

1. **Feature detection** Salient or distinctive elements of each image are identified and located.
2. **Feature matching** Correspondences between features are established, typically by comparing feature descriptors.
3. **Transform estimation** The correspondence between features is used to determine a transform that maps one image to the other.
4. **Image Composition** The images are transformed and aligned with one another. Some form of interpolation is often applied to blend the images.

In general to align two images of the same scene the images must be mapped onto a model of the 3D surface shown, which may be computed dynamically from the images or obtained from other sources. Systems building these 3D mosaics include the offline processes used to generate mosaics used in applications such as Google Earth[1] or Microsoft Photosynth[2], and incremental procedures which are not currently practical in real-time, but eventually aim to produce mosaic-based 3D maps of a robot's environment, such as the underwater mapping scheme by Johnson-Roberson et al. (2010) and the aerial image mosaicing schemes by Bryson et al. (2010) and RavAcha-etal-2008. For images from an aerial platform however, we can generally assume that the ground is approximately planar, i.e. it is viewed from sufficient distance that relative depth variation is low and parallax is small. This allows a simpler model to be used: two images of a planar surface captured through a pinhole camera are related by a perspective transformation (Hartley and Zisserman, 2003, Chapter 4): if two images each show the same plane, then there is a homography (3×3 matrix) H such that for all $\mathbf{x}$ in the first image and $\mathbf{x}'$ in the second image (represented in homogeneous coordinates), $H\mathbf{x} = \mathbf{x}'$.
For many mosaicing applications simpler models are appropriate: for a rotating camera at a fixed location (all points are on the plane at infinity), a simple 2D translation and 1D rotation is sufficient to align two images. For a strictly down-pointing camera on an aerial platform a similarity transform (rotation, translation and scale) is sufficient. For a camera with a small field-of-view the perspective transformation may be approximated with an affine transformation: $A\mathbf{x} + \mathbf{t} = \mathbf{x}'$ for a 2×2 transformation matrix A and 2D translation $\mathbf{t}$.

Feature detection and matching

The first stage in estimating the transformation between two images is to find matches between features in one image and the same features in the other image. These features are usually points within each image chosen to be localisable and repeatable, such as Harris Corners (Harris and Stephens, 1988) or SIFT's Difference-of-Gaussian blobs (Lowe, 1999). A descriptor vector describing the area around each feature is then extracted, for example a SIFT feature or just a simple image patch (Szeliski, 2006). Possible matches between these descriptors are then found, for example by pairwise comparison.

[1] `http://earth.google.com`
[2] `http://photosynth.net`

Transform estimation

Given at least four point correspondences, a perspective transform can be estimated via the Discrete Linear Transform, or DLT (Hartley and Zisserman, 2003, Section 4.1). This linear least-squares method reduces the effects of noise in the image measurements. Least-squares solutions are, however, susceptible to being corrupted by outliers, which are common amongst correspondences due to repeated similar-looking features and because of moving objects in the scene. To overcome this Random Sample and Consensus (RANSAC) approaches are commonly used (Fischler and Bolles, 1981). RANSAC works by repeatedly selecting small random subsets of correspondences (hypothesis sets) from which to compute candidate solutions. Each candidate solution is compared with the entire data set until a solution compatible with a large number of correspondences is found. These are assumed to be inliers, and a least-squares solution is computed from this inlier set.

One mosaicing scheme closely following this pattern is the scheme described by Brown and Lowe (2007) for producing panoramic photographs from a set of still images. SIFT features are used to firstly to identify which frames overlap, then RANSAC is used to compute perspective transforms between these overlapping images. Image boundaries are blended together with a multi-scale blur to smooth edges in areas of low detail, while preserving structure in areas of higher detail. While results are visually impressive, performance is far from real-time, taking several minutes to mosaic 10 images.

Similar schemes include the system by Rong et al. (2009), which matches SURF features between microscope slides to produce one image of a larger sample, however despite the simple motion model this still requires tens of seconds per frame; and the system by Turkbeyler and Harris (2009) which registers batches of images using RANSAC, producing impressive globally consistent mosaics, although not in real-time, partly due to the expensive optimisation needed to ensure globally consistent transforms.

An alternative to feature-based image matching is to directly align the images to one another, by using correlation across entire images to estimate the transformation between them, thus replacing the first three steps of the process. Nielsen et al. (2008) use a Fourier transform to calculate the correlation between images from a camera spinning about the vertical axis, and render a cylindrical mosaic at 30Hz, however this simple model has predictable motion with only one DOF (the camera orientation), and generates a mosaic of fixed size. Similarly Tegolo and Valenti (2001) find similarity transforms between pairs of aerial images by brute-force search, however this takes 88 seconds per 256×256 frame due to direct alignment's complexity exponential in the number of parameters in the model being fitted, and hence we do not consider a direct approach feasible for a real-time system where perspective effects are significant.

As is the case of SLAM, the camera's motion can often be used to track features from one frame to the next. For example, the scheme described by Civera et al. (2009a), which stitches together images captured by a stationary rotating camera to produce a spherical mosaic. A Kalman filter and a model of the camera rotation predicts the region of a future frame in which each tracked feature will lie. This region is then searched for the corresponding features. This approach works well at high frame-rates, where the regions searched for each feature are small, however at lower frame-rates or during rapid image motion large regions must be searched for each feature, making a tracking approach considerably more costly and error-prone than a wide-baseline approach, and in addition a wide-baseline approach is still needed if images from earlier passes are to be registered. In aerial images large image motion often occurs due to camera rotation, and it is more desirable to use available bandwidth for boosting the total area imaged, rather than for transmitting large numbers of almost-entirely overlapping images, making tracking unsuitable for our application.

Image composition

Once transforms between images have been computed a mosaic can be rendered by choosing one image, then warping every other image to the same coordinate frame. However the two images will rarely line up exactly, due to a combination of small errors remaining in the transform estimate, uncorrected lens distortion, and intensity variation between images. In the case of aerial images the fact that the ground is not exactly planar adds to this misalignment. A naïve approach to composition, laying each image over the existing mosaic results in visible and potentially distracting artefacts due to straight edges in the image (such as roads and

For each frame:

1. **Add new frame to BoW database:**
   - Capture new frame from camera
   - Detect corners and extract features
   - Add image to BoW database
2. **Select both recent frames and earlier frames overlapping current frame. For each of these:**
   - Find BoW feature correspondences with current frame
   - Compute perspective homography aligning image with current frame
3. **Render mosaic**
   - Recurse to find many images surrounding current frame
   - Warp each image to align with current frame
   - Identify optimal seams between image pairs
   - Copy warped images into a global mosaic

Figure 10.1: Overview of RT-AIM, a real-time aerial image mosaicing scheme

boundaries) being broken at the join, and the new straight edges appearing along borders (Figure 10.2). Two common solutions to this problem are firstly to hide seams by interpolating between overlapping images, or secondly to choose seams minimising visual discontinuities.
A variation of the first approach is used by Brown and Lowe (2007), which successfully hides seams, although moving objects or poor alignment can result in multiple images of objects appearing ('ghosting'). A similar approach could be feasible in real-time. Alternatively Levin et al. (2004); Zomet et al. (2006) estimate a final mosaic by minimising a cost function based on image gradient similarities, however at several minutes per-frame this would not be feasible for a real-time application.
The second approach is to place a seam between images chosen to be visually inconspicuous. This seam should cut natural edges in the images at places where they line up (or ideally not at all if possible). Various methods for finding optimal graph-cuts minimising dissimilarity functions between pixels (Agarwala et al., 2004; Kolmogorov and Zabih, 2004) or segments (Gracias et al., 2006) have been proposed, however these methods are currently far-from practical in real-time. An alternative approach is to use Dijkstra's algorithm to find the seam (path) where the difference between the two images is minimised (Davis, 1998); this efficient approach is feasible for real-time processing.
In summary, while numerous mosaicing schemes have been proposed, most are targeted at producing extensive, globally consistent mosaics and are unsuitable for real-time processing. Those systems that do run in real-time assume camera motion models inappropriate for aerial images.

10.3 Video mosaicing in real-time

The previous section described contemporary approaches to image mosaicing, and their limitations. In this section we describe our new scheme for real-time aerial image mosaicing, RT-AIM, which follows the same basic pattern as many existing schemes (summarised in Figure 10.1), but uses the latest computer vision algorithms to enable real-time performance.
The following sections describe the individual components of RT-AIM: firstly Section 10.3.1 describes the BoW image representation and how it is used both to find adjoining frames, and to find correspondences

between images; Section 10.3.2 describes how these correspondences are used for image registration; and Section 10.3.3 describes how seams between registered images are found, and how we efficiently render a seamless mosaic.

10.3.1 Feature description and the Bag-of-Words image representation

As with BoWSLAM, RT-AIM uses simple descriptors based on image patches centred on corners from the FAST corner detector (as described in Chapter 4). The highest-scoring FAST corners are chosen subject to a minimum separation constraint, as often the overlap between pairs of frames is small, so a good spread of features is necessary to find enough matches. For large, high-resolution images patches sampled from across large patches of the image provide more distinctiveness than smaller patches; typically patches sized 66×66 pixels are down-sampled to 11×11. Extracting FAST corners from a 800×600 image takes 30 milliseconds, and extracting 450 patch descriptors and adding them to a hierarchical BoW database takes typically 4 milliseconds per frame, allowing every frame to be indexed (Figure 10.3). In order to register images from earlier passes, when features were viewed at different orientations, oriented patches are used. In aerial images from approximately down-pointing cameras, the changes in perspective and scale between images is usually small, so perspective-invariant descriptors such as SURF or SIFT are not necessary, and anyway are too computationally expensive to extract from large images in real-time.

As with BoWSLAM, every image is added to a BoW database, as described in Chapter 5. The BoW database is used to recognise when two images overlap, so that images from previous visits to a region may be registered, allowing the cameras' FOV to be expanded in two directions. Again, the requirement that a transform is computed is sufficient to ensure that images are not registered incorrectly.

Once images are represented as a BoW, the most similar matches from earlier in the flight are found. To find all available images in the vicinity of the latest frame we recurse over the images similar to these matches (including the previous frame). Transformations are computed between pairs of the top few matches. Note that as large BoW databases can be searched efficiently and effectively, no navigation data is necessary to find these matches.

10.3.2 Computing relative positions

For aerial images we assume that the ground is approximately planar, and hence images can be aligned with a perspective transformation (the affine approximation is not sufficiently accurate—alignment errors of about 5% are typical for aerial images). This transformation is approximately linear so can be estimated from feature-correspondences via the DLT. Ideally, as with relative pose estimation, a nonlinear refinement would be used to reduce errors further, however in practice transformations appear sufficiently accurate already.

As with BoWSLAM, the BoW image database is used to find correspondences between pairs of frames, and BaySAC framework is used to fit a transformation to these correspondences. This allows the correct model to be found quickly, and inliers to be found from amongst a large number of low-quality N–M matches without degrading performance. Large numbers of inliers are desirable to minimise errors caused by small or poorly-conditioned inlier sets. This is important when mosaicing aerial images as there is often only a small overlap between pairs of consecutive images; this leads to initially high outlier rates, and often poorly conditioned geometry when matched features lie only in a narrow strip of the image.

To assign prior probabilities to correspondences the following assumptions are made: firstly, without any knowledge of the camera's motion, the prior probabilities of each point in an image correctly matching any point in the other image are uniform (with probability p). Therefore if point i in one image is matched to M_i points in the other image, the probability of each of the M_i candidate correspondences being correct is $P(i \text{ inlier}) = p/M_i$, and these probabilities are disjoint (each point can be matched to only one other).

Secondly, when we are registering consecutive frames (at times t and $t+1$) we can use information from the transformation found between frames $t-1$ and t, $H_{t-1,t}$ to predict the locations of each feature in image $t+1$. The assumption here is that the velocity at time t is a good initial estimate for the velocity at time $t+1$; a good assumption for fixed-wing aerial platforms. In addition we can estimate the acceleration of image

features[3]. Probabilities of correspondences being inliers can now be updated to incorporate the following motion model using Bayes theorem: given a correspondence between two points $i = (\mathbf{x}_i, \mathbf{x}'_i)$ we assume that if i is an inlier, $\mathbf{x}'_i$ is distributed normally about $H_{t-1,t}\mathbf{x}_i$ with standard deviation s pixels (about 150 pixels works well), and if i is an outlier $\mathbf{x}'_i$ is distributed uniformly over the image. This gives i the following p.d.f.:

$$f(i) = f((\mathbf{x}_i, \mathbf{x}'_i)) \quad = \quad \begin{cases} \phi_{(0,s^2)}(\mathbf{x}'_i - H_{t-1,t}\mathbf{x}_i) & i \text{ inlier} \\ 1/A & i \text{ outlier} \end{cases} \tag{10.3.1}$$

where ϕ is the 2D normal p.d.f. with each component i.i.d. with mean 0 and variance s^2, and A is the image area.

$$P(i \text{ inlier}|H_{t-1,t}) = \frac{f(i|i \text{ inlier})P(i \text{ inlier})}{P(i \text{ inlier})f(i|i \text{ inlier}) + P(i \text{ outlier})f(i|i \text{ outlier})} \tag{10.3.2}$$

When N–M correspondences are used, the fact that each point in one image matches to at most one in the other must also be considered.

$$P(j \text{ inlier}|j \text{ incompatible with i}) = P(j \text{ inlier})\frac{1 - P(i \text{ inlier}|H_{t-1,t})}{1 - P(i \text{ inlier})} \tag{10.3.3}$$

10.3.3 Rendering seamless mosaics

Three alternative strategies for image composition were implemented: firstly images are simply warped into the frame of a single image, producing a mosaic very rapidly but with obvious seams remaining. Secondly a simple interpolation scheme combines overlapping images with weights in proportion to each pixel's distance from the frame's centre. Thirdly cuts are found between pairs of images that minimise the total squared difference between the two frames along the cut, using Dijkstra's algorithm (Davis, 1998).
Figure 10.2 illustrates that the third option is most successful at eliminating artefacts in both man-made and natural environments; however a search over all paths through the image is costly. Instead we sub-sample the area to be searched and find a path at this courser resolution. This path is usually visually acceptable, especially at higher frame-rates when it is only briefly visible.

10.3.4 Optimisations for real-time operation

A typical breakdown of computation times is shown in Figure 10.3. Once optimised no particular areas dominate, however many optimisations were necessary to achieve this performance, in particular to speed-up image warping.
To warp one image into another, we predict where the source image will map to in the destination image, then iterate over an appropriate area in the destination image, calculating where in the source image each pixel should come from (using the inverse of the perspective transformation). Only the nearest source pixel is considered. To make this work in real-time the following optimisations were necessary: firstly, while in general a perspective transform is computed, often the elements $H_{3,1}$ and $H_{3,2}$ are close to zero (i.e. the transform is approximately affine), and are exactly zero for one image in each rendered mosaic. In this case a slightly faster affine warp is applied. Secondly, if $H_{3,1}$ is not too large, the division needed to compute the source pixel location can be approximated with a binomial expansion: each source pixel location is given by $(x'_s, y'_s) = (x_s/t_s, y_s/t_s)$ where $(x_s, y_s, t_s) = H^{-1}(x_d, y_d, 1)$. Given t_s^{-1} at (x_d, y_d), ${t'}_s^{-1}$ at $(x_d + 1, y_d)$ is given by

$${t'}_s^{-1} = ((H^{-1})_{3,1}(x+1) + (H^{-1})_{3,2}y + 1)^{-1} \quad = \quad t_s^{-1}(1 + (H^{-1})_{3,1}t_s^{-1})^{-1} \tag{10.3.4}$$

$$\approx \quad t_s^{-1}(1 - (H^{-1})_{3,1}t_s^{-1}) \tag{10.3.5}$$

[3]As with single camera incremental structure-from-motion this is not quite the same as modeling the acceleration of the camera, as image feature motion depends on the plane's altitude as well as its velocity, so an uninformative value should be used that suitable for any height the plane is likely to fly.

(a) No seam removal

(b) Interpolation between images

(c) Optimal seam between images

Figure 10.2: Simply warping images over each other leads to visible seams and broken edged when images do not quite align (a). The seams are successfully hidden by interpolating between frames, but artefacts around man-made straight edges still occur (b). An optimal seam between images usually removes both kinds of artefact (c).

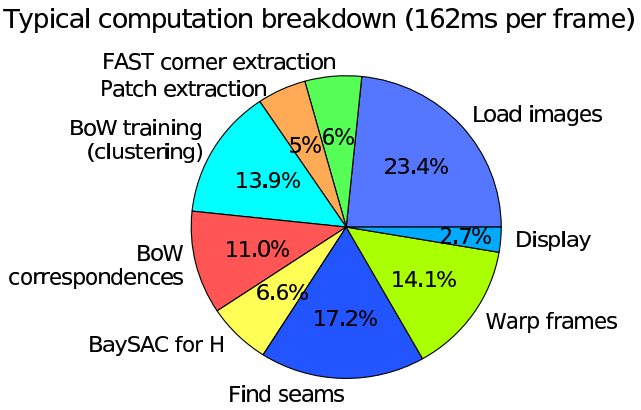


Figure 10.3: Breakdown of execution time—one 1000×1000 mosaic containing an average of 6.8 frames is rendered and displayed per-frame.

Thirdly, four-channel images are used, so that each three-byte RGB triple is aligned with a 32-bit word, allowing faster copying of pixels throughout the program, despite the increase in memory required. It may be possible to improve performance using SIMD instructions or to carry out some processing on the GPU, however no single optimisation would greatly affect the performance.
The only parts of out scheme with complexity increasing with time are querying the BoW database, and re-building the dictionary. BoW queries have complexity linear in the number of images, although in practice the cost of these queries is negligible for up-to tens of thousands of images Re-building the dictionary has complexity log-linear in the number of images, however this is only necessary while the environment keeps changing in appearance. Eventually a dictionary suitable for all environments likely to be encountered will be found.

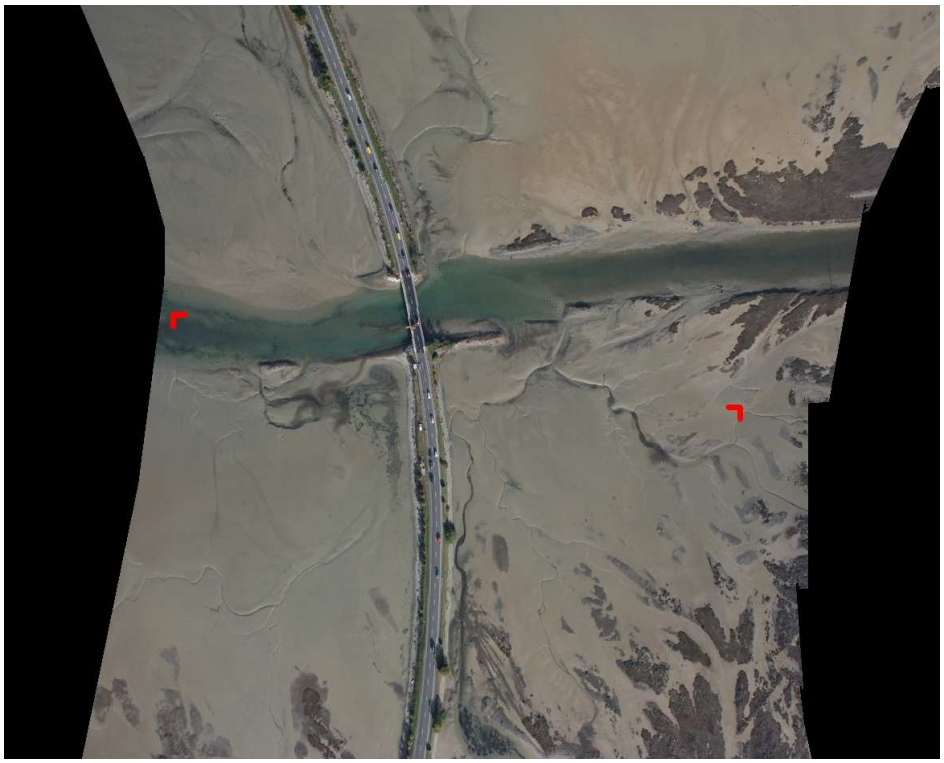
10.4 Results

RT-AIM is tested with an aerial image sequence from a down-pointing Canon 400D camera attached to a microlight. 3220 images sized 800×532 and covering farmland, forest, coastline and urban areas were captured at 2.8Hz, then processed sequentially to produce a video-mosaic sized 1200×1000 (Figure 10.4). One mosaic is generated for every frame, and a frame-rate of 6.2 Hz is maintained, with each mosaic containing an average of 8 images, often including frames from earlier in the flight.
Table 10.1 shows that BaySAC with a simple prior probability model is no more successful than RANSAC at finding transformations, but finds considerably larger inlier sets, as it makes use of N–M correspondences. However when a prior motion model (MM) is used, BaySAC clearly outperforms RANSAC, enabling transformations to be found 80% of the time, compared with 69% of the time for RANSAC. The motion model particularly helps during rapid motion when image overlap is small and a low proportion of potential correspondences are correct. As a result, mosaics rendered when BaySAC is used contain an average of 8 frames, compared with 5.2 frames when using RANSAC.
The second dataset used is an image sequence captured from a UAV flight in the Antarctic (Barnsdale, 2010). 1115 images sized 640×480 were captured at 2Hz. Despite the self-similar and sometimes apparently-featureless environment, frames were mosaiced successfully throughout the flight (Figure 10.5). An average

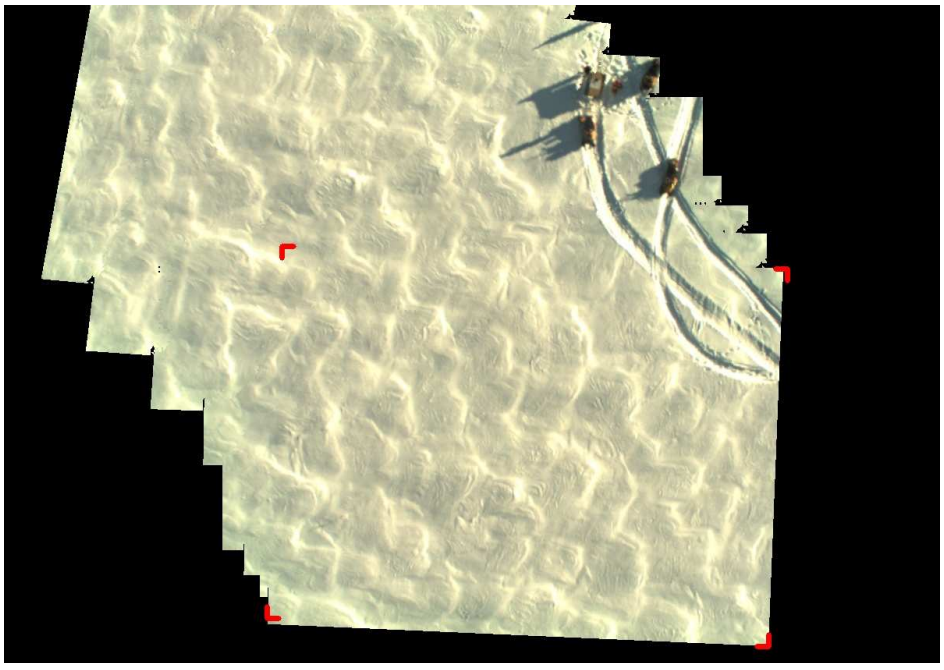
(a)

(b)

Figure 10.4: These mosaics include images from earlier in the flight, found by BoW.



(a)



(b)

Figure 10.5: Mosaicing succeeds in a wide range of environments, including difficult visually sparse and self-similar regions.

Table 10.1: Success at finding transformations between image pairs from a stream of 3220 images with different parametrisations, terminating after 250 iterations.

Method	Av. # iterations	% Success	# inliers on success
BaySAC+MM	172	80%	67
BaySAC	193	69%	73
RANSAC+N–M	245	23%	76
RANSAC+no N–M	209	69%	53

of 5.2 images form each mosaic, an average of 136 inlier correspondences were found between pairs of frames, and 93% of image pairs where registered successfully.
For higher frame-rate video, images sized 320×212 can be mosaiced at 20Hz. Alternatively a larger 1440×2000 video-mosaic, filling two computer monitors, is rendered at 3.6Hz, which is still faster than images were captured. Towards the corners of this mosaic however images are warped with accumulated sequences of several transformations, and become distorted in places. In future we will investigate optimising transformations across multiple images in order to prevent this.

10.5 Conclusions

A mosaicing scheme has been described that enables the mosaicing of images captured from an aerial platform, hence providing a larger field-of-view for the operator. Images from throughout the flight are registered and stitched together seamlessly in real-time. This real-time performance is enabled by the following four implementation choices: firstly expensive invariant descriptors are not necessary for aerial images; simple oriented image patches chosen around FAST corners provide good distinctiveness at a fraction of the cost. Secondly, in order to robustly match sufficient numbers of features between frames, the BaySAC framework is used, which reduces costs compared to conventional RANSAC and enables an accurate perspective transformation to be found from large numbers of poor-quality correspondences. Thirdly, the BoW algorithm allows images near to the current frame to be found from throughout the flight, and in addition provides cheap wide-baseline correspondences between pairs of frames. Finally, an efficient optimal cut-finding algorithm applied to subsampled images, together with an optimised image warping procedure allow large mosaics to be rendered and displayed at 6Hz, over twice the framerate at which images were captured.
This chapter has demonstrated an alternative application of techniques developed in this thesis for robust real-time robot positioning. The BoW scheme used to detect loop closure for mobile robot positioning has allowed overlapping images to be detected, and the BaySAC framework has been used to make effective use of N–M correspondences, with fast and reliable matching aided by incorporating prior probabilities from an image-feature motion model.

Chapter 11

Conclusions

This thesis has described a new single camera SLAM scheme, BoWSLAM, which can accurately position a camera over long tracks through a range of visually challenging environments. In one experiment, BoWSLAM is demonstrated mapping a 25 minute 2.5km trajectory through a dynamic outdoor environment in real-time, without any other sensor input; considerably further than previous single camera SLAM schemes. Key to this performance is the development and selection of a combination of schemes which are robust to the errors which occur when positioning using computer vision. Positioning is possible even with low cost cameras, including a Contour HD mountain bike helmet camera, and a Sony Handycam digital camcorder, demonstrating that a single camera is a suitable low-cost sensor for robust long-term SLAM.

To enable BoWSLAM to navigate previously unknown environments, several new algorithms were developed. Firstly, a BoW scheme which reliably detects when two images show the same place, despite substantial changes in the scene, was developed. This scheme requires no prior training, enables wide-baseline correspondences between frames to be found efficiently, and is used in BoWSLAM both to detect loop-closure, and to select nearby frames from which to compute relative positions.

Secondly, to robustly compute relative camera poses from pairs of images, a new RANSAC sampling strategy, BaySAC, was developed. BaySAC successfully registers images despite high rates of outliers from moving objects, or limited overlap between frames, and reduces the time needed to compute relative poses. BaySAC makes effective use of matches between multiple similar-looking features, and by combining BaySAC with the cheap, wide-baseline correspondences available from the BoW algorithm, multiple position hypotheses can be computed for each frame.

Thirdly, to enable accurate global maps to be constructed from these multiple position hypotheses, despite the occasional gross error, a novel graph-based map representation was developed. From this map, only the most reliable relative pose estimates are selected. This outlier-free submap is optimised to generate an accurate global map, using the TORO framework.

Fourthly, to address the problem of scale drift which occurs when navigating using a single camera, the SCORE2 scheme was developed. SCORE2 uses the BoW image representation to learn classes of objects present in the environment. Objects are measured, then the distributions of object class sizes are estimated. Subsequent observations of the same classes of objects enable estimates of scale to be improved. SCORE2 successfully reduces scale drift by 75% while navigating an indoor environment, and reduces accumulated errors by 64% while navigating a large outdoor environment.

The schemes which have been developed for BoWSLAM have many additional applications. One application of the combination of robust algorithms proposed is the video mosaicing scheme described in Chapter 10.1, in which high resolution images from a UAV are stitched together in real-time in order to enlarge the camera's field of view. An additional application for the BoW scheme is in the pedestrian navigation system described in Section 5.5.2. In this system, the BoW scheme provides regular absolute position updates by recognising the pedestrian's location in a previously-mapped building. These updates are used to correct drift in the position estimates obtained by integrating measurements from a foot-mounted IMU.

In the first chapter of this thesis, five requirements for robust positioning using a single camera were proposed.

Firstly, I proposed that a robust single camera SLAM scheme should be able to actively detect loop closure by recognising places which have been visited previously. Chapter 8 demonstrated that the BoW component of BoWSLAM can reliably detect loop closure events, and can even re-localise the robot after it has become completely lost.
The second proposed requirement was that the robot must be able to register a sequence of frames from a video, even when many of these frames are corrupted by occlusion, motion blur, or erratic camera motion. Again, Chapter 8 shows that positions can be computed despite sequences of frames containing motion blur, total occlusion, and erratic motion. Some environments still present significant challenges however, and during rapid motion, expensive scale-invariant descriptors are required, which increases processing requirements.
The third proposed requirement was that position estimation must be robust in the presence of moving objects, and in self-similar environments. The BaySAC robust estimation framework, described in Chapter 7, is shown to successfully compute relative poses in the presence of moving objects, and can efficiently find inliers amongst large numbers of matches between similar-looking features in images of Antarctic ice, and in repetitive indoor environments. As part of BoWSLAM, position estimates are computed despite multiple moving objects, including cars and pedestrians.
The fourth proposed requirement for successful single camera SLAM, was the ability to reject the erroneous position estimates which will inevitably occur, before they corrupt the global map. This is demonstrated by generating global maps using the TORO framework using only a subset of the most reliable position hypotheses. Even though some position hypotheses do contain gross errors, these are successfully avoided when position estimates to optimise are selected. Even if erroneous position hypotheses were selected and did corrupt the optimised map, future optimised maps will not contain these position hypotheses if more reliable relative positions become available, allowing complete recovery.
The fifth criteria for long-term operation is that the cost and complexity of SLAM must be low enough that a usefully large environment can be explored, and costs must grow only as a function of the area which is explored. BoWSLAM has been demonstrated positioning a camera in real-time along a 2.5km path, and could scale to environments several times that size. This is comparable to the size of many environments in which robots will one day operate, for example inside buildings. The next challenge is to switch between SLAM and localisation-only when moving in or out of previously mapped areas. Currently frames are occasionally discarded when captured from the same location as previous frames, and the t-spanning subgraphs which are optimised grow only slowly when re-visiting areas, but for long term operation the complexity should not grow at all when previously mapped areas are visited.
In summary, BoWSLAM has demonstrated that all of these criteria can be met for a single camera SLAM system, however additional work on all of BoWSLAM's components would always improve performance, and allow more challenging environments to be navigated.

11.1 Further work

Each component of BoWSLAM has been developed to the level necessary to successfully navigate challenging environments, however each could be improved further, as described in each chapter. I believe that the most significant improvement in robustness could come from a higher level representation of the world, where images are represented in terms of the objects they contain, and a map is built consisting of these objects.
In the mean-time, BoWSLAM can already reliably navigate many environments, and I plan a live implementation, to position either a remote-control robot platform, or a pedestrian streaming back images from a camera-phone.

Appendix A

Least-squares optimisation

The Singular Value Decomposition, and Levenberg-Marquardt's algorithm, are used frequently throughout this thesis for linear and nonlinear least-squares optimisation respectively.

A.1 The Singular Value Decomposition for linear least-squares optimisation

The Singular Value Decomposition (SVD), described in Hartley and Zisserman (2003), Appendix 5, is a widely-used matrix decomposition which enables least-squares solutions to overdetermined systems of linear equations to be found efficiently. A $n \times m$ matrix $\mathbf{A}$ is decomposed as:

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T \tag{A.1.1}$$

where $\mathbf{D}$ is a $n \times m$ matrix with the singular values of $\mathbf{A}$ on the diagonal, in order of decreasing size; $\mathbf{U}$ is a $n \times n$ orthonormal matrix; and $\mathbf{V}$ is a $m \times m$ orthonormal matrix. The SVD is useful for finding least-squares solutions to systems of equations of the form $\mathbf{A}\mathbf{x} = 0$; if $\mathbf{A}$ is overdetermined then the right-most column of $\mathbf{V}$ is the least-squares solution for $\mathbf{x}$, and if if $\mathbf{A}$ is underdetermined then the right-most $m - n$ columns of $\mathbf{V}$ form a basis for the space of solutions for $\mathbf{x}$. The SVD is used for computing first-approximations to essential matrices and perspective transformations between images, and to correct scale drift around cycles.

A.2 Levenberg-Marquardt nonlinear least-squares optimisation

As SLAM is fundamentally a nonlinear optimisation over landmark observations and the robot pose (or trajectory), a good ground-truth solution can be obtained by a full nonlinear optimisation. One algorithm which can be used for this optimisation is Levenberg-Marquardt's algorithm for least-squares optimisation. Levenberg-Marquardt's algorithm finds parameters which minimise functions which can be written as a sum-of-squares, for example to find the parameter vector $\mathbf{x}$ minimising the scalar function $e(\mathbf{x})$ defined as:

$$e(\mathbf{x}) = \sum_{i=1}^{N} [f(\mathbf{x})]_i^2 \tag{A.2.1}$$

Functions of this form include expressions for the weighted sum-of-squared errors; in this case, if errors are Gaussian, and are weighted by their information, the global minimum of the function is the MLE of the parameters.
Levenberg-Marquardt's algorithm, described in Hartley and Zisserman (2003), Appendix 6, is an iterative gradient descent algorithm based on the Gauss-Newton method. The Gauss-Newton method works as follows:

starting from some approximate initial solution, $\mathbf{x}_0$, the Jacobian matrix J of partial derivatives is computed (either numerically or analytically). The Hessian matrix of second derivatives is also estimated (usually approximately, from the Jacobian; $H \approx 2J^T J$). From these matrices an update $\mathbf{d}$ to the vector $\mathbf{x}$ in the direction of steepest descent is computed, so that $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \mathbf{d}$. This process is repeated, and $\mathbf{x}_i$ usually converges to a minimum of e.

Levenberg's improvement to this algorithm (Levenberg, 1944) is to damp the update vectors, by adding a dynamically-varying damping factor to the Hessian, to increase the chances of convergence when far from the minimum, while still converging quickly while nearby. Levenberg's method is particularly useful when the Hessian is near singular (so the update required is poorly determined), however in other situations the solution converges too slowly. Marquardt (1963) proposed that the damping factor chosen is proportional to the curvature, and this speeds up convergence in some situations (Lampton, 1997). In some situations, including SLAM, a reasonably accurate initial solution is needed in order to avoid false minima.

Each iteration of Levenberg-Marquardt has complexity cubic in the number of parameters, as a linear system is solved to find the update. In the case of SLAM, the Jacobian matrices are often sparse, so a sparse matrix implementation can reduce costs (although loop closure introduces correlations between distant poses, so the algorithm still has cubic complexity). For problems with fixed numbers of parameters, for example optimising relative poses to minimise Sampson's error, Levenberg-Marquardt optimisation can be very cheap.

Appendix B

Derivation of corner localisation error p.d.f.

This appendix derives the distribution of feature-point localisation errors that would be obtained from a perfect pixel-resolution corner detector, as shown in Figure 4.9.
In the experiment described in Section 4.4.1, the distance between the detected location of a corner in one image transformed into the other image, and the location of the same corner detected in the other image, $\|\mathbf{E}\|_2$, is measured. Figure 4.9 shows this situation. A perfect pixel-resolution corner detector will measure the corner's location to be the nearest pixel centre to the location of the corner in the world, projected into the image. In this case, the distribution of errors observed is derived from Figure 4.9, and the properties of the p.d.f. (Probability density function; Wikipedia, 2010) as follows:

$$\mathbf{e}_i = (e_{ix}, e_{iy}) \text{ where } e_{ix}, e_{iy} \sim \text{Unif}\left(-\frac{1}{2}, \frac{1}{2}\right), \ i = 1, 2 \tag{B.0.1}$$

$$\Rightarrow \text{ total error } \mathbf{E} = \mathbf{e}_1 + \mathbf{e}_2 = (e_{1x} + e_{2x}, e_{1y} + e_{2y}) \tag{B.0.2}$$

Therefore E_x, E_y each have p.d.f. f given by:

$$f(t) = \begin{cases} 1 - |t| & -1 < t < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{B.0.3}$$

Therefore E_x^2, E_y^2 each have p.d.f. F given by:

$$F(t) = \begin{cases} 3(1-t)^2 & 0 < t < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{B.0.4}$$

The p.d.f. of the total absolute squared error, $|\mathbf{E}|_2^2 = E_x^2 + E_y^2$, is then given by convolving F with itself:

$$(F * F)(s) = g(s) = \begin{cases} -\frac{1}{4}s^3 + \frac{3}{2}s^2 - 3s + 2 & 0 < s < 2 \\ 0 & \text{otherwise.} \end{cases} \tag{B.0.5}$$

Integrating to $s = r^2$ gives the c.d.f. of errors we would expect to observe from testing an ideal corner detector:

$$P\left(|E|_2^2 < r^2\right) = \int_{s=0}^{r^2} g(s)\, ds \tag{B.0.6}$$

$$= \begin{cases} 0 & r \leq 0 \\ -\frac{1}{16}r^8 + \frac{1}{2}r^6 - \frac{3}{2}r^4 + 2r^2 & 0 < r < \sqrt{2} \\ 1 & r \geq \sqrt{2} \end{cases} \tag{B.0.7}$$

and differentiating gives the p.d.f.:

$$N(r) = \begin{cases} -\frac{1}{2}r^7 + 3r^5 - 6r^3 + 4r & 0 < r < \sqrt{2} \\ 0 & \text{otherwise.} \end{cases} \tag{B.0.8}$$

This p.d.f. is plotted in Figure 4.10, and has shape similar to the Rayleigh distribution. Its expectation is $\frac{128}{315}\sqrt{2} = 0.57466$ pixels. This result is used to demonstrate how close pixel-resolution corner detectors are to achieving the highest localisation accuracy that is possible.

Glossary

AUV	Autonomous Underwater Vehicle; an autonomous robotic submarine.
Baseline	The distance between two camera positions. The baseline is fixed for a stereo camera, but must be estimated relative to previous baselines for a single camera, leading to scale drift.
BoW	Bag-of-Words image representation; a compact representation of an image in term of the set of features it contains.
c.d.f.	Cumulative density/distribution function; given by the integral of the p.d.f.
Data association	The problem of matching landmarks measured by a robot's sensors to previously measured (mapped) landmarks. Data association failure often leads to gross errors.
DOF	'Degrees-of-freedom'; describes the space in which a robot can move. An upright robot moving on a plane has 3 DOF motion (its pose has a 2D position together with a 1D heading orientation), whereas in an AUV may have 6 DOF motion (it may move in three spatial dimensions and rotate about any axis).
FastSLAM	Popular particle filter-based SLAM framework.
FOV	Field-of-view; the angular area viewed by a sensor (such as a camera). For example a panoramic camera has a 360 degree FOV.
GNSS	Global Navigation Satellite System: a system using signals from a satellite network to positioning a receiver, for example GPS.
GPS	Global Positioning System: the most complete and widely-used GNSS system.
Gross errors	Errors which are many times larger than would ever be observed if measurements were approximately normally distributed (e.g. an error ten times the mean error).

Heuristic algorithm An algorithm which in practice often produced a good solution, but which does not always compute a correct solution.

i.i.d. Independent and identically distributed random variables.

IMU Inertial measurement unit; a versatile positioning sensor combining several accelerometers to measure 3D acceleration and rotation.

Localisation Positioning with respect to a map.

Loop closure The event when a robot visits a previously-visited location. Detecting the loop closure event is needed to generate a consistent map, and helps to correct accumulated errors.

LSA Latent Semantic Analysis; a heuristic machine learning algorithm for identifying sets of co-occurring features from training data.

MAV Micro Air Vehicle; a miniature UAV. MAVs are often inspired by flying insects.

MLE Maximum Likelihood Estimate; the estimate of a problem's parameters which are most likely given the data (e.g. the map which is most likely given the observations).

OR Object recognition; the process of identifying that an image contains an instance of a particular object (e.g. a red Subaru Legacy). Object class recognition refers to identifying that an image contains an instance of an object class (e.g. a car).

p.d.f. Probability density/distribution function

Pose The combination of a robot's position and orientation.

SAD "Sum of absolute differences"; a measure of error between two vector quantities (such as a pair of descriptors); sometimes used for comparing vectors instead of the SSD, as it is less sensitive to single outliers.

Scale The robot's estimate of the scale of the world; scale is equivalent to baseline length, which is modelled using a lognormal distribution.

SLAM Simultaneous Localisation and Mapping: A SLAM scheme allows a mobile robot to position itself in an *a priori* unknown environment, by building a map of its environment as it explores, and simultaneously localising itself in this map.

SSD	"Sum of squared differences"; a measure of error between two vector quantities (such as a pair of descriptors); given a vector with normally distributed components with equal variance, the member of a set of vectors which is most likely to be drawn from the same distribution is the one with the lowest SSD from the first vector.
SVD	Singular Value Decomposition; a matrix decomposition used widely for solving linear least-squares optimisation problems; described in Appendix A.1.
TF-IDF	Term Frequency-Inverse Document Frequency, a heuristic measure of the distinctiveness of an image word.
UAV	Unmanned Aerial Vehicle; including fixed-wing aircraft, rotorcraft, airships, and MAVs.

Bibliography

A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen. Interactive digital photomontage. *ACM Transactions on Graphics*, 23(3):292–300, 2004.

M. Agrawal, K. Konolige, and M. R. Blas. Censure: Center surround extremas for realtime feature detection and matching. In *Proceedings of the European Conference on Computer Vision*, 2008.

S. Ahn, M. Choi, J. Choi, and W. K. Chung. Data association using visual object recognition for EKF-SLAM in home environment. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2588–2594, Oct. 2006. doi: 10.1109/IROS.2006.281936.

S. Albrecht, J. Hertzberg, K. Lingemann, A. Nchter, J. Sprickerhof, and S. Stiene. Device level simulation of kurt3d rescue robots. In *Proceedings of the International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster*, 2006.

I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Journal of Discrete and Computational Geometry*, 9(1):81–100, 1993. ISSN 0179-5376. doi: http://dx.doi.org/10.1007/BF02189308.

F. Andert and F. Adolf. Online world modeling and path planning for an unmanned helicopter. *Autonomous Robots*, 27:147–164, 2009.

A. Angeli, D. Filliat, S. Doncieux, and J.-A. Meyer. Real-time visual loop-closure detection. In *Proceedings of the International Conference on Robotics and Automation*, 2008a.

A. Angeli, D. Filliat, S. Doncieux, and J.-A. Meyer. A fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics*, Special issue on Visual SLAM:1–11, 2008b.

D. S. Apostolopoulos, M. D. Wagner, B. N. Shamah, L. Pedersen, K. Shillcutt, and W. L. Whittaker. Technology and field demonstration of robotic search for antarctic meteorites. *International Journal of Robotics Research*, 19:1015–1032, 2000.

T. Bailey and H. Durrant-Whyte. Simultaneous localisation and mapping (SLAM): Part II state of the art. *IEEE Robotics and Automation Magazine*, September:1–9, 2006.

V. K. Balakrishnan. *Schaum's outline of theory and problems of graph theory*. McGraw-Hill, 1997.

S. Barkby, S. Williams, O. Pizarro, and M. Jakuba. Incorporating prior bathymetric maps with distributed particle bathymetric slam for improved auv navigation and mapping. In *Proceedings of IEEE Oceans Conference*, 2009.

K. Barnsdale. Geospatial research centre takes to the air over antarctica. Online at `http://www.grc.canterbury.ac.nz/documents/UAV-Nov-09.pdf`, 2010.

H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision*, Graz, Austria, May 2006.

H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110:346–359, 2008.

J. S. Beis and D. G. Lowe. Indexing without invariants in 3D object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):1000–1015, 1999.

M. Bosse and J. Roberts. Histogram matching and global initialization for laser-only slam in large unstructured environments. In *Proceedings of the International Conference on Robotics and Automation*, 2007.

M. Bosse and R. Zlot. Map matching and data association for large-scale two-dimensional laser scan-based slam. *The International Journal of Robotics Research*, 27:667–691, 2008.

M. Bosse, P. Newman, J. Leonard, and S. Teller. Simultaneous localization and map building in large-scale cyclic environments using the atlas framework. *International Journal of Robotics Research*, 23:1113–1139, 2004.

T. Botterill, S. Mills, and R. Green. Speeded-up Bag-of-Words algorithm for robot localisation through scene recognition. In *Proceedings of Image and Vision Computing New Zealand*, pages 1–6, Nov. 2008. doi: 10.1109/IVCNZ.2008.4762067.

T. Botterill, R. Green, and S. Mills. A Bag-of-Words Speedometer for Single Camera SLAM. In *Proceedings of Image and Vision Computing New Zealand*, pages 1–6, Wellington, NZ, November 2009a.

T. Botterill, S. Mills, and R. Green. New conditional sampling strategies for speeded-up RANSAC. In *Proceedings of the British Machine Vision Conference*, 2009b.

T. Botterill, S. Mills, and R. Green. Bag-of-words-driven single camera SLAM. *To appear in the Journal of Field Robotics*, 2010.

J.-Y. Bouguet. Camera calibration toolbox for Matlab. online, 2010. URL `http://www.vision.caltech.edu/bouguetj/calib_doc`.

M. Brown and D. Lowe. Unsupervised 3d object recognition and reconstruction in unordered datasets. In *Proceedings of the IEEE International Workshop on 3-D Digital Imaging and Modeling*, pages 1–8, 2005.

M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1):59–73, 2007.

M. Brown, R. Szeliski, and S. Winder. Multi-image matching using multi-scale oriented patches. Technical Report MSR-TR-2004-133, Department of Computer Science, University of British Columbia, December 2004.

R. Brunelli and O. Mich. Histogram analysis for image retrieval. *Pattern Recognition*, 34:1121–1134, 2001.

M. Bryson and S. Sukkarieh. Building a robust implementation of bearing-only inertial SLAM for a UAV. *Journal of Field Robotics, Special issue on SLAM in the field*, 24:113–143, 2007.

M. Bryson, A. Reid, F. Ramos, and S. Sukkarieh. Airborne vision-based mapping and classification of large farmland environments. *Journal of Field Robotics*, Special Issue on Visual Mapping and Navigation Outdoors:632–655, 2010.

A. Bur, A. Tapus, N. Ouerhani, R. Siegwar, and H. Hiigli. Robot navigation by panoramic vision and attention guided fetaures. In *Proceedings of the International Conference on Pattern Recognition*, volume 1, pages 695–698, 2006.

J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:679–714, 1986.

D. Capel. An effective bail-out test for RANSAC consensus scoring. In *Proceedings of the British Machine Vision Conference*, pages 1–10, 2005.

R. O. Castle, D. J. Gawley, G. Klein, and D. W. Murray. Video-rate recognition and localization for wearable cameras. In *Proceedings of the British Machine Vision Conference*, pages 1100–1109, 2007.

R. O. Castle, G. Klein, and D. W. Murray. Combining localization with recognition for scene augmentation using a wearable camera. Preprint, 2009.

D. Chekhlov, M. Pupilli, W. Mayol-Cuevas, and A. Calway. Real-time and robust monocular slam using predictive multi-resolution descriptors. pages 276–285, 2006.

M. Chli and A. J. Davison. Active matching. In *Proceedings of the European Computer Vision Conference*, 2008.

O. Chum. *Two-View Geometry Estimation by Random Sample and Consensus*. PhD thesis, Czech Technical University in Prague, 2005.

O. Chum and J. Matas. Matching with PROSAC - progressive sample consensus. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 220–226, Los Alamitos, USA, 2005. ISBN 0-7695-2372-2.

O. Chum, T. Pajdla, and P. Sturm. The geometric error for homographies. *Comput. Vis. Image Underst.*, 97(1):86–102, 2005. ISSN 1077-3142. doi: http://dx.doi.org/10.1016/j.cviu.2004.03.004.

O. Chum, J. Philbin, M. Isard, and A. Zisserman. Scalable near identical image and shot detection. In *Proceedings of the International Conference on Image and Video Retrieval*, 2007.

J. Civera, A. J. Davison, J. A. Magalln, , and J. M. M. Montiel. Drift-free real-time sequential mosaicing. *International Journal of Computer Vision*, 81:128–137, 2009a.

J. Civera, O. G. Grasa, A. J. Davison, and J. M. M. Montiel. 1-Point RANSAC for EKF-Based Structure from Motion. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2009b.

A. Clarke. *OPIRA: The Optical-flow Perspective Invariant Registration Augmentation and other improvements for Natural Feature Registration*. PhD thesis, University of Canterbury, 2009.

L. A. Clemente, A. J. Davison, I. Reid, J. Neira, and J. D. Tardos. Mapping large loops with a single hand-held camera. In *Proceedings of Robotics: Science and Systems*, 2007.

D. Cole and P. Newman. Using laser range data for 3d slam in outdoor environments. In *Proceedings of the international conference on robotics and automation*, 2006.

J. Coughlan and R. Manduchi. Functional assessment of a camera phone-based wayfinding system operated by blind and visually impaired users. *International Journal on Artificial Intelligence Tools*, 18:379–397, 2009.

C. Croarkin and P. Tobias. NIST/SEMATECH e-Handbook of Statistical Methods, March 2010. URL `http://www.itl.nist.gov/div898/handbook/`.

G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Proceedings of the ECCV International Workshop on Statistical Learning in Computer Vision*, 2004.

M. Cummins and P. Newman. Accelerated appearance-only SLAM. In *Proceedings of the International Conference on Robotics and Automation*, pages 1828–1833, May 2008a. doi: 10.1109/ROBOT.2008.4543473.

M. Cummins and P. Newman. FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *International Journal of Robotics Research*, 27(6):647–665, 2008b. doi: 10.1177/0278364908090961.

M. Cummins and P. Newman. Highly scalable appearance-only SLAM - FAB-MAP 2.0. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, 2005.

J. Davis. Mosiacs of scenes with moving objects. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 354–360, 1998.

A. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the IEEE International Conference on Computer Vision*, Oct. 2003.

F. Dellaert and M. Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research*, 25:1181–1203, 2006.

F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proceedings of the International Conference on Robotics and Automation*, 1999.

G. Dissanayake, H. Durrant-Whyte, and T. Bailey. A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem. In *Proceedings of the International Conference on Robotics and Automation*, pages 1009–1014 vol.2, 2000. doi: 10.1109/ROBOT.2000.844732.

M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17:229–241, 2001.

T. Drummond and R. Cipolla. Real-time tracking of complex structures with on-line camera calibration. In *Proceedings of the British Machine Vision Conference*, 1999.

R. O. Duda and P. E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/361237.361242.

H. Durrant-Whyte and T. Bailey. Simultaneous localisation and mapping (SLAM): Part I the essential algorithms. *IEEE Robotics and Automation Magazine*, June:1–9, 2006.

J. Dvorsky, P. Praks, and V. Snasel. Latent semantic indexing for image retrieval systems. In *Linear Algebra Proceedings of the Society for Industrial and Applied Mathematics*, pages 1–8, 2003.

E. Eade. *Monocular Simultaneous Localisation and Mapping*. PhD thesis, University of Cambridge, 2008.

E. Eade and T. Drummond. Scalable monocular SLAM. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 469–476, Los Alamitos, CA, USA, 2006. IEEE Computer Society. doi: http://doi.ieeecomputersociety.org/10.1109/CVPR.2006.263.

E. Eade and T. Drummond. Monocular SLAM as a graph of coalesced observations. In *Proceedings of the IEEE International Conference on Computer Vision*, Rio de Janeiro, Brazil, October 2007.

E. Eade and T. Drummond. Unified loop closing and recovery for real time monocular SLAM. In *Proceedings of the British Machine Vision Conference*, 2008.

A. Elfes, G. Podnar, R. T. Filho, and A. P. Filho. Inference grids for environmental mapping and mission planning of autonomous mobile environmental robots. In *Proceedings of the Workshop on Sensing a Changing World*, 2008.

J. F. Engelberger. Health-care robotics goes commercial: the 'helpmate' experience. *Robotica*, 11(06):517–523, 1993.

Equine vision; Wikipedia, June 2010. URL http://en.wikipedia.org/wiki/Equine_vision.

C. Estrada, J. Neira, and J. D. Tardos. Hierarchical slam: realtime accurate mapping of large environments. *IEEE Transactions on Robotics*, 21:588–596, 2005.

F. Evennou and F. Marx. Advanced integration of wifi and inertial navigation systems for indoor mobile positioning. *EURASIP Journal on Applied Signal Processing*, 2006:1–11, 2006.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results. Online, 2008. URL http://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html.

W. L. Fehlman and M. K. Hinders. Passive infrared thermographic imaging for mobile robot object identification. *Journal of Field Robotics*, 27(3):281–310, 2010.

D. Filliat. A visual bag of words method for interactive qualitative localization and mapping. In *Proceedings of the International Conference on Robotics and Automation*, 2007.

P. Firmin. Satellite navigation technology applications for intelligent transport systems: A european perspective. In *Proceedings of the European Navigation Conference*, 2006.

M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. ISSN 0001-0782.

L. Fletcher. Future challenges for robotic perception. [oral presentation], 2009. SLAM Summer School.

A. Flint, C. Mei, I. Reid, and D. Murray. Growing semantically meaningful models for visual slam. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 467–474, 2010. doi: 10.1109/CVPR.2010.5540176.

J. Folkesson and H. I. Christensen. Graphical slam for outdoor applications. *Journal of Field Robotics*, 24: 51–70, 2007.

E. Foxlin. Pedestrian tracking with shoe-mounted inertial sensors. *Computer Graphics and Applications*, 25: 38–46, November 2005.

F. Fraundorfer, H. Stewénius, and D. Nistér. A binning scheme for fast hard drive based image search. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2007.

U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localization and mapping. *IEEE Transactions on Robotics*, 21(2):196–207, April 2005. ISSN 1552-3098. doi: 10.1109/TRO.2004.839220.

J. Funke and T. Pietzsch. A framework for evaluation visual slam. In *Proceedings of the British Machine Vision Conference*, 2009.

P. Gemeiner, A. J. Davison, and M. Vincze. Improving localization robustness in monocular slam using a high-speed camera. In *Proceedings of Robotics: Science and Systems*, 2008.

A. Gil, O. M. Mozos, M. Ballesta, and O. Reinoso. A comparative evaluation of interest point detectors and local descriptors for visual slam. *Machine Vision and Applications*, 1432-1769:1–16, 2009.

N. Gracias, A. Gleason, and S. Negahdaripour. Fast image blending using watersheds and graph cuts. In *Proceedings of the British Machine Vision Conference*, pages 469–478, 2006.

G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard. Efficient estimation of accurate maximum likelihood maps in 3d. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007a.

G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Proceedings of the Robotics: Science and Systems (RSS)*, 2007b.

G. Grisetti, R. Kuemmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2d and 3d mapping. In *Proceedings of the International Conference on Robotics and Automation*, 2010.

G. Guennebaud and B. Jacob. Eigen 2 matrix library, n.d. URL `http://eigen.tuxfamily.org/`.

J. Guivant. *Efficient Simultaneous Localisation and Mapping in Large Environments*. PhD thesis, The University of Sydney, 2002.

A. Handa, M. Chli, H. Strasdat, and A. J. Davison. Scalable active matching. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010.

C. Harris and M. Stephens. A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988.

R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, second edition, 2003. ISBN 0-521-54051-8.

J. Haverinen and A. Kemppainen. Global indoor self-localization based on the ambient magnetic field. *Robotics and Autonomous Systems*, 57:1028–1035, July 2009. ISSN 09218890. doi: 10.1016/j.robot.2009.07.018. URL `http://dx.doi.org/10.1016/j.robot.2009.07.018`.

J. Hays and A. A. Efros. Im2gps: estimating geographic information from a single image. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, 2008.

R. Hidayat and R. Green. Real-time texture boundary detection from ridges in the standard deviation space. In *Proceedings of the British Machine Vision Conference*, 2009.

C. Hide. *Integration of GPS and low cost INS measurements*. PhD thesis, University of Nottingham, 2003.

C. Hide and T. Botterill. Development of an integrated IMU, image recognition and orientation sensor for pedestrian navigation. In *Proceedings of the International Technical Meeting of the Institute of Navigation*, pages 1–9, San Diego, CA, 2010.

C. Hide, T. Botterill, and M. Andreotti. An integrated IMU, GNSS and image recognition sensor for pedestrian navigation. In *Proceedings of the Institute of Navigation GNSS Conference*, pages 1–10, Fort Worth, Texas, 2009.

C. Hide, T. Botterill, and M. Andreotti. Vision-aided IMU for pedestrian navigation. In *To appear in Proceedings of the Institute of Navigation GNSS Conference*, 2010.

G. Hoffman, H. Huang, S. Waslander, and Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proceedings of the Conference of the American Institute of Aeronautics and Astronautics*, 2007.

D. Hoiem, A. A. Efros, and M. Hebert. Closing the loop on scene interpretation. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, 2008.

E. Hygounenc, I. K. Jung, P. Soueres, and S. Lacroix. The autonomous blimp project of LAAS-CNRS: Achievements in flight control and terrain mapping. *International Journal of Robotics Research*, 23(4-5): 473–511, 2004.

N. Jacobs, B. Bies, and R. Pless. Using cloud shadows to infer scene structure and camera calibration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Proceedings of the European Conference on Computer Vision*, pages 304–317, 2008.

C. Jenkins. Logo tracking technology becomes more sophisticated [omniperception ltd. press release]. online, 2008. URL `http://www.omniperception.com/news/2008/September/29/press_release_logo_tracking_technology_`

P. Jensfelt, S. Ekvall, D. Kragic, and D. Aarno. Integrating slam and object detection for service robot tasks. In *Proceedings of the IROS 2005 Workshop on Mobile Manipulators: Basic Techniques, New Trends and Applications*. IEEE/RSJ, Edmonton, Canada, 2005.

M. Johnson-Roberson, O. Pizarro, S. B. Williams, and I. Mahon. Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys. *Journal of Field Robotics*, 27:21–51, 2010.

T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the eighteenth annual symposium on Computational geometry*, 2002.

L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, March 1990. ISBN 0471735787.

A. Kelly. Precision dilution in triangulation based mobile robot position estimation. In *Proceedings of Intelligent Autonomous Systems*, 2003.

A. Klaptocz and J. Nicoud. Technology and fabrication of ultralight micro-aerial vehicles. *Flying Insects and Robots*, pages 299–316, 2009.

G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007. doi: 10.1109/ISMAR.2007.4538852.

J. Klippenstein and H. Zhang. Quantitative evaluation of feature extractors for visual slam. In *Proceedings of the Canadian Conference on Computer and Robot Vision*, 2007.

V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004.

K. Konolige, J. Bowman, J. D. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua. View-based maps. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

M. Kovac, M. Fuchs, A. Guignard, J.-C. Zufferey, and D. Floreano. A miniature 7g jumping robot. In *Proceedings of the International Conference on Robotics and Automation*, pages 373 – 378, 2008.

N. M. Kwok and G. Dissanayake. Bearing-only SLAM in indoor environments using a modified particle filter. In *Proceedings of the Australasian Conference on Robotics and Automation*, pages 1–3, 2003.

A. J. Lacey, N. Pinitkarn, and N. A. Thacker. An evaluation of the performance of ransac algorithms for stereo camera calibration. In *Proceedings of the British Machine Vision Conference*, 2000.

M. Lampton. Damping-undamping strategies for the levenberg-marquardt nonlinear least-squares method. *Computers in Physics*, 11:110–115, Jan/Feb 1997.

R. Langley. Dilution of precision. *GPS world*, May:53–57, 1999.

C. Laschi, B. Mazzolai, V. Mattoli, M. Cianchetti, and P. Dario. Design and development of a soft actuator for a robot inspired by the octopus arm. In *Proceedings of the International Symposium on Experimental Robotics*, 2009.

T. Lemaire and S. Lacroix. SLAM with panoramic vision. *Journal of Field Robotics*, 24:91 – 111, 2007.

J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10):727–774, 2008.

V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, 2006.

K. Levenberg. A method for the solution of certain non-linear problems in least squares. *The Quarterly of Applied Mathematics*, 2:164–168, 1944.

A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless image stitching in the gradient domain. In *Proceedings of the European Conference on Computer Vision*, pages 377–389, 2004.

F.-F. Li and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

J. Li and N. M. Allinson. A comprehensive review of current local features for computer vision. *Neurocomputing*, 71(10-12):1771 – 1787, 2008. Neurocomputing for Vision Research; Advances in Blind Signal Processing.

E. Limpert, W. A. Stahel, and M. Abbt. Log-normal distributions across the sciences: Keys and clues. *BioScience*, 51(5):341–352, 2001.

D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1150–1157, 1999.

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

I. Mahon, S. Williams, O. Pizarro, and M. Johnson-Roberson. Efficient view-based slam using visual loop closures. *IEEE Transactions on Robotics*, 24(5):1002 –1014, 2008.

M. Maimone, Y. Cheng, and L. Matthies. Two years of visual odometry on the mars exploration rovers. *Journal of Field Robotics*, 24(3):169–186, 2007.

Maple. Waterloo Maple version 7, n.d. URL `http://www.maplesoft.com`.

D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11:431–441, 1963.

J. Martnez-Carranza and A. Calway. Appearance based extraction of planar structure in monocular slam. In *Proceedings of the Scandinavian Conference on Image Analysis*, 2009.

J. Matas and O. Chum. Randomized RANSAC with sequential probability ratio test. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1727–1732, New York, USA, October 2005.

J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *Proceedings of the British Machine Vision Conference*, pages 384–393, 2002.

S. May, S. Fuchs, D. Droeschel, D. Holz, and A. Nüchter. Robust 3d-mapping with time-of-flight cameras. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*, pages 1673–1678, 2009.

C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid. A constant-time efficient stereo SLAM system. In *Proceedings of the British Machine Vision Conference*, 2009.

K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2003.

K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65 (1-2):43–72, 2005. ISSN 0920-5691. doi: http://dx.doi.org/10.1007/s11263-005-3848-x.

M. Milford, G. Wyeth, and D. Prasser. RatSLAM: a hippocampal model for simultaneous localization and mapping. In *Proceedings of the International Conference on Robotics and Automation*, volume 1, pages 403–408, 2004. doi: 10.1109/ROBOT.2004.1307183.

M. Milford, G. Wyeth, and D. Prasser. RatSLAM on the edge: Revealing a coherent representation from an overloaded rat brain. In *Proceedings of IROS*, 2006.

S. Mills and T. Pridmore. Texture unmapping — combining parametric and non-parametric techniques for image reconstruction. In *Proceedings of the Irish Machine Vision and Image Processing Conference*, 2003.

M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002.

M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the 18th international joint conference on Artificial intelligence*, 2003.

J. M. M. Montiel, J. Civera, and A. J. Davison. Unified inverse depth parametrization for monocular SLAM. In *Proceedings of Robotics: Science and Systems, Philadelphia, USA*, 2006.

H. Moravec. Visual mapping by a robot rover. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1979.

E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Generic and real-time structure from motion using local bundle adjustment. *Image and Vision Computing*, 27(8):1178–1193, 2009. ISSN 0262-8856. doi: http://dx.doi.org/10.1016/j.imavis.2008.11.006.

A. Murillo, J. Guerrero, and C. Sagues. Surf features for efficient robot localization with omnidirectional images. In *Proceedings of the International Conference on Robotics and Automation*, pages 3901–3907, 2007. doi: 10.1109/ROBOT.2007.364077.

J. Neira and J. D. Tardos. Data association in stochastic mapping using the joint compatibility test. *IEEE Transactions on Robotics and Automation*, 17(6):890–897, 2001.

R. A. Newcombe and A. J. Davison. Live dense reconstruction with a single moving camera. In *Proceedings of Computer Vision and Pattern Recognition*, 2010.

P. Newman, D. Cole, and K. Ho. Outdoor SLAM using visual appearance and laser ranging. In *Proceedings of the International Conference on Robotics and Automation*, Florida, 2006.

P. Newman, G. Sibley, M. Smith, M. Cummins, A. Harrison, C. Mei, I. Posner, R. Shade, D. Schrter, L. Murphy, W. Churchill, D. Cole, and I. Reid. Navigating, recognising and describing urban spaces with vision and laser. *International Journal of Robotics Research*, 28:1406–1433, 2009.

T. Nicosevici and R. Garca. On-line visual vocabularies for robot navigation and mapping. In *Proceedings of IROS*, pages 205–212, 2009.

F. Nielsen, A. Andre, and S. Tajima. Real-time spherical videos from a fast rotating camera. In *Proceedings of the International Conference on Image Analysis and Recognition*, pages 326–335, 2008.

J. Nieto, J. Guivant, and E. Nebot. Real Time Data Association for FastSLAM. In *Proceedings of the IEEE Conference on Robotics and Autonomation*, pages 412–418, 2003.

J. Nieto, T. Bailey, and E. Nebot. Scan-SLAM: Combining EKF-SLAM and Scan Correlation. In *Proceedings of the Field and Service Robotics conference*, pages 167–178, 2006.

D. Nistér. Frame decimation for structure and motion. In *Revised Papers from Second European Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE)*, pages 17–34, London, UK, 2001.

D. Nistér. Preemptive RANSAC for live structure and motion estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 199–206 vol.1, 2003.

D. Nistér. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–777, 2004.

D. Nistér and H. Stewénius. Linear time maximally stable extremal regions. In *Proceedings of the European Conference on Computer Vision*, pages 183–196, 2008.

D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2161–2168, June 2006.

D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23(1):3–20, 2006.

E. Nowak, F. Jurie, and B. Triggs. Sampling strategies for bag-of-features image classification. In *Proceedings of the European Conference on Computer Vision*, pages 490–503, 2006. doi: 10.1007/11744085_38.

A. Ogale. Google street view from a computer vision perspective. Presentation available online, 2010. URL `http://vision.stanford.edu/teaching/cs223b/lecture/google_streetview_slides.pdf`.

E. Olson, J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2262–2269, 2006.

OpenCV Computer Vision Library, n.d. URL `http://opencv.willowgarage.com/`.

D. Palmer, T. Moore, C. Hill, M. Andreotti, and D. Park. Position estimation using multiple dab single frequency networks. In *Proceedings of the Institute of Navigation GNSS conference*, 2009.

J. Park, B. Charrow, D. Curtis, J. Battat, E. Minkov, J. Hicks, S. Teller, and J. Ledlie. Growing an indoor localization system. In *Proceedings of MobiSys*, 2010.

A. Peddemors, H. Eertink, and I. Niemegeers. Predicting mobility events on personal devices. *Pervasive and Mobile Computing Journal Special Issue on Human Behaviour in Ubiquitous Environments*, 11:1–10, 2009.

T. Peynot, S. Terho, and S. Scheding. Sensor data integrity: Multi-sensor perception for unmanned ground vehicles. Technical Report ACFR-TR-2009-002, Australian Centre for Field Robotics, 2009.

J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007. doi: 10.1109/CVPR.2007.383172.

Point Grey Chamelion Datasheet, 2010. URL `http://www.ptgrey.com/products/chameleon/Chameleon_datasheet.pdf`.

E. Prassler, A. Ritter, C. Schaeffer, and P. Fiorini. A short history of cleaning robots. *Autonomous Robots*, 9(3):211–226, 2000.

Probability density function; Wikipedia, July 2010. URL `http://en.wikipedia.org/wiki/Probability_density_functio`

M. Raibert, K. Blankespoor, G. Nelson, R. Playter, and the BigDog Team. BigDog, the rough-terrain quadruped robot. In *Proceedings of the IFAC World Congress*, 2008.

V. Ramachandran, editor. *Encyclopedia of the Human Brain*, volume 2, chapter Colour Vision, page 15. Elsevier Science Inc., 2002a.

V. Ramachandran, editor. *Encyclopedia of the Human Brain*, volume 3, chapter Motion Processing, page 117. Elsevier Science Inc., 2002b.

V. Ramachandran, editor. *Encyclopedia of the Human Brain*, volume 4, chapter Spatial Vision, page 419. Elsevier Science Inc., 2002c.

A. Ramisa, S. Vasudevan, D. Scharamuzza, R. L. de Mántaras, and R. Siegwart. A tale of two object recognition methods for mobile robots. In *Proceedings of The International Conference on Computer Visions Systems*, volume 5008, pages 353–362, Santorini, Greece, 2008. Springer Verlag, Springer Verlag.

J. Raper, G. Gartner, H. Karimi, and C. Rizos. A critical evaluation of location based services and their potential. *Journal of Location Based Services*, 1(1):5–45, 2007.

Rayleigh distribution; Wikipedia, June 2010. URL `http://en.wikipedia.org/wiki/Rayleigh_distribution`.

P. Robertson, M. Angermann, B. Krach, and M. Khider. Inertial systems based joint mapping and positioning for pedestrian navigation. In *Proceedings of the Institute of Navigation GNSS Conference*, Fort Worth, Texas, 2007.

W. Rong, H. Chen, J. Liu, Y. Xu, and R. Haeusler. Mosaicing of microscope images based on surf. In *Proceedings of Image and Vision Computing New Zealand*, 2009.

E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, October 2005.

E. Rosten and T. Drummond. Machine leaning for high-speed corner detection. In *Proceedings of the European Conference on Computer Vision*, pages 430–443, 2006.

E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:105–119, 2010.

F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce. 3d object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. *International Journal of Computer Vision*, 66:231–259, 2006.

P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection*. John Wiley & Sons, Inc., New York, NY, USA, 1987. ISBN 0-471-85233-3.

D. Scaramuzza, F. Fraundorfer, M. Pollefeys, and R. Siegwart. Absolute scale in structure from motion from a single vehicle mounted camera by exploiting nonholonomic constraints. In *Proceedings of the IEEE International Conference on Computer Vision*, 2009.

C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, 2000.

R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226, June 2007.

S. Se, H.-K. Ng, P. Jasiobedzki, and T.-J. Moyung. Vision based modeling and localization for planetary exploration rovers. In *Proceedings of the International Astronautical Congress*, 2004.

T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):411–426, 2007. URL `http://dx.doi.org/10.1109/TPAMI.2007.56`.

A. Shahrokni. Texture boundary detection by maximizing the distance between two textures. online, June 2004. URL `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/SHAHROKNI1/node5.html`.

J. Shi and C. Tomasi. Good features to track. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

G. Sibley, C. Mei, I. Reid, and P. Newman. Adaptive relative bundle adjustment. In *Proceedings of Robotics Science and Systems (RSS)*, Seattle, USA, June 2009.

G. Sibley, C. Mei, I. Reid, and P. Newman. Vast-scale outdoor navigation using adaptive relative bundle adjustment. *International Journal of Robotics Research*, 29:958–980, June 2010.

SICK LD-MRS-400001 laser scanner technical data, 2010. URL `http://www.sick.com`.

R. Siegwart and I. R. Nourbakhsh. *Introduction to autonomous mobile robots*. MIT Press, 2004.

S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc. GPU-based video feature tracking and matching. Technical Report TR 06-012, University of North Carolina at Chapel Hill, 2006.

J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1470–1477, Oct. 2003.

J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering object categories in image collections. In *Proceedings of the International Conference on Computer Vision*, 2005.

R. Smith, M. Self, and P. Cheeseman. *Autonomous Robot Vehicles*, chapter Estimating Uncertain Spatial Relationships in Robotics, pages 435–461. Springer Verlag, Amsterdam, 1990.

C. Stachniss. TORO - a tree-based network optimiser. online, 2008. URL `http://www.informatik.uni-freiburg.de/~stachnis/toro/`.

C. Stachniss, D. Haehnel, W. Burgard, and G. Grisetti. On actively closing loops in grid-based fastslam. *Advanced Robotics - The Int. Journal of the Robotics Society of Japan*, 19:1059–1080, 2005.

H. Stewénius, C. Engels, and D. Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60:284–294, June 2006.

H. Stewnius. Calibrated fivepoint solver source code. online, 2006. URL `http://www.vis.uky.edu/~stewe/FIVEPOINT/`.

C. Stimming. Lapack++ v2.5.2 matrix library. online, 2004.

H. Strasdat, J. M. M. Montiel, and A. J. Davison. Scale drift-aware large scale monocular slam. In *Proceedings of Robotics: Science and Systems Conference*, 2010.

Summation; Wikipedia, July 2010. URL `http://en.wikipedia.org/wiki/Summation`.

R. Szeliski. *Handbook of Mathematical Models in Computer Vision*, chapter Image Alignment and Stitching, pages 273–292. Springer, December 2005.

R. Szeliski. Image alignment and stitching: A tutorial. Technical Report MSR-TR-2004-92, Microsoft Research, December 2006.

T. A. Tamba, B. Hong, and K.-S. Hong. A path following control of an unmanned autonomous forklift. *International Journal of Control, Automation and Systems*, 7:113–122, 2009.

S. Taylor and T. Drummond. Multiple target localisation at over 100 fps. In *Proceedings of the British Machine Vision Conference*, 2009.

D. Tegolo and C. Valenti. A naïve approach to compose aerial images in a mosaic fashion. In *International Conference on Image Analysis and Processing*, pages 512–516, 2001.

The Robotics Data Set Repository (Radish), 2003. Andrew Howard and Nicholas Roy, from http://radish.sourceforge.net/.

H. C. Thode. *Testing for normality*. CRC Press, 2002.

S. Thrun. Particle filters in robotics. In *Proceedings of the Conference on Uncertainty in AI*, 2002.

S. Thrun and A. Bucken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

S. Thrun and M. Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal of Robotics Research*, 25(5-6):403–429, 2006.

S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *International Journal of Robotics Research*, 23:693–716, 2004.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

N. Tillett. Automatic guidance sensors for agricultural field machines: A review. *Journal of Agricultural Engineering Research*, 50:167 – 187, 1991.

B. J. Tordoff and D. W. Murray. Guided-MLESAC: Faster image transform estimation by using matching priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1523–1535, 2005. ISSN 0162-8828.

P. H. S. Torr and D. W. Murray. The development and comparison of robust methods for estimating the fundamental matrix. *International Journal of Computer Vision*, 24(3):271–300, 1997.

P. H. S. Torr and A. Zisserman. MLESAC: a new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000. ISSN 1077-3142.

A. Torralba, R. Fergus, and W. T. Freeman. Tiny images. Technical Report MIT-CSAIL-TR-2007-024, Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, 2007.

B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment – a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, volume 1883/2000, pages 1–71, Corfu, Greece, 1999.

H. Trivedi. Estimation of stereo and motion parameters using a variational principle. *Image and Vision Computing*, 5:181–183, 1987.

E. Turkbeyler and C. Harris. Building aerial mosaics ii: Metadata, feature matching, loop closure. In *Proceedings of the 6th Annual Technical Conference of the Electromagnetic Remote Sensing Defence Technology Centre*, 2009.

J. Uijlings, A. Smeulders, and R. Scha. Real-time bag-of-words, approximately. In *Proceedings of the Conference On Image And Video Retrieval*, 2009.

C. Valgren and A. J. Lilienthal. SIFT, SURF and seasons: Long-term outdoor localization using local features. In *Proceedings of the European Conference on Mobile Robots*, pages 253–258, September 19–21 2007.

A. Vattani. k-means requires exponentially many iterations even in the plane. In *Proceedings of the 25th annual symposium on Computational geometry*, 2009.

L. Vincent and A. Ulges. Recognizing text in images, 2008. URL `http://www.wipo.int/pctdb/en/ia.jsp?IA=US2007072578`.

D. Wagner, G. Reitmayr, R. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2008.

H. Wang and J. Brady. Corner detection with subpixel accuracy. Robotics Institute Technical Report OUEL 1925/92, Department of Engineering Science, University of Oxford, 1992.

S. Wangsiripitak and D. W. Murray. Avoiding moving outliers in visual SLAM by tracking moving objects. In *Proceedings of the International Conference on Robotics and Automation*, 2009.

S. Weerahandi. *Exact Statistical Methods for Data Analysis*. Springer, 1995.

F. Werner, J. Sitte, and F. Maire. Visual topological mapping and localisation using colour histograms. In *Proceedings of the International Conference on Control, Automation, Robotics and Vision*, 2008.

B. Williams, G. Klein, and I. Reid. Real-time SLAM relocalisation. In *Proceedings of the IEEE International Conference on Computer Vision*, 2007.

B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. Tardos. An image-to-map loop closing method for monocular SLAM. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

S. Winder and M. Brown. Learning local image descriptors. In *Proceedings of the European Conference on Computer Vision*, pages 1–8, June 2007. doi: 10.1109/CVPR.2007.382971.

A. Y. Yang, B. A. MacDonald, and K. A. Stol. Real time simultaneous localisation and mapping for the player project. In *Proceedings of the Australasian Conference on Robotics and Automation*, 2008.

D. Yuen and B. MacDonald. Vision-based localization algorithm based on landmark matching, triangulation, reconstruction, and comparison. *IEEE Transactions on Robotics*, 21(2):217 – 226, 2005.

D. C. Yuen and B. A. MacDonald. Natural landmark based localisation system using panoramic images. In *Proceedings of the International Conference on Robotics and Automation*, 2002.

D. C. Yuen and B. A. MacDonald. Theoretical considerations of multiple particle filters for simultaneous localisation and map-building. In *Proceedings of the International Conference on Knowledge-based Intelligent Information and Engineering Systems*, 2004.

L. Zhang and B. Ghosh. Line segment based map building and localization using 2d laser rangefinder. In *Proceedings of the International Conference on Robotic and Automation*, 2000.

Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Proceedings of the International Conference on Computer Vision*, 1999.

Q. Zhu, B. Wu, and N. Wan. A sub-pixel location method for interest points by means of the harris interest strength. *The Photogrammetric Record*, 22:321–335, 2007.

B. Zitová and J. Flusser. Image registration methods: A survey. *Image and Vision Computing*, 21:977–1000, 2003.

A. Zomet, A. Levin, S. Peleg, and Y. Weiss. Seamless image stitching by minimizing false edges. *IEEE Transactions on Image Processing*, 15(4):969–977, 2006.