Simultaneous Recognition, Localization and Mapping for Wearable Visual Robots

Robert Oliver Castle
Wolfson College

Robotics Research Group
Department of Engineering Science
University of Oxford

Trinity Term 2009

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

Robert Oliver Castle
Wolfson College

Doctor of Philosophy
Trinity Term 2009

Simultaneous Recognition, Localization and Mapping for Wearable Visual Robots

Abstract

With the advent of ever smaller and more powerful portable computing devices, and ever smaller cameras, wearable computing is becoming more feasible. The ever increasing numbers of augmented reality applications are allowing users to view additional data about their world overlaid on their world using portable computing devices.

The main aim of this research is to enable a user of a wearable robot to explore large environments automatically viewing augmented reality at locations and on objects of interest.

To implement this research a wearable visual robotic assistant is designed and constructed. Evaluation of the different technologies results in a final design that combines a shoulder mounted self stabilizing active camera, and a hand held magic lens into a single portable system.

To enable the wearable assistant to locate known objects, a system is designed that combines an established method for appearance-based recognition with one for simultaneous localization and mapping using a single camera. As well as identifying planar objects, the objects are located relative to the camera in 3D by computing the image-to-database homography. The 3D positions of the objects are then used as additional measurements in the SLAM process, which routinely uses other point features to acquire and maintain a map of the surroundings, irrespective of whether objects are present or not.

The monocular SLAM system is then replaced with a new method for building maps and tracking. Instead of tracking and mapping in a linear frame-rate driven manner, this adopted method separates the mapping from the tracking. This allows higher density maps to be constructed, and provides more robust tracking. The flexible framework provided by this method is extended to support multiple independent cameras, and multiple independent maps, allowing the user of the wearable two-camera robot to escape the confines of the desk top and explore arbitrarily sized environments.

The final part of the work brings together the parallel tracking and multiple mapping system with the recognition and localization of planar objects from a database. The method is able to build multiple feature rich maps of the world and simultaneously recognize, reconstruct and localize objects within these maps. The object reconstruction process uses the spatially separated keyframes from the tracking and mapping processes to recognize and localize known objects in the world. These are then used for augmented reality overlays related to the objects.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost I would like to thank my supervisor, Professor David Murray, for his tireless support and endless proof-reading. Thank you to the members of the Active Vision and Mobile Robotics groups for their help and many discussions throughout my research. Thank you to Professor David Lowe for providing SIFT and BBF source code.

My research was funded by the Engineering and Physical Science Research Council (grants GR/S97774 and EP/D037077), and I am grateful to EPSRC for their support.

Throughout this work I have enjoyed the unfailing support of my family, and in particular from Angela who has provided continual patience, support and encouragement.

Publications from this thesis

- R. O. Castle, D. J. Gawley, G. Klein, and D. W. Murray. Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In *Proceedings of the International Conference on Robotics and Automation*, Rome, Italy, April 10-14, 2007, pages 4102-4107.
- R. O. Castle and D. J. Gawley and G. Klein and D. W. Murray. Video-rate recognition and localization for wearable cameras. In *Proceedings of the 18th British Machine Vision Conference*, Warwick, September 11-13, 2007, pages 1100-1109.
- R. O. Castle and G. Klein and D. W. Murray. Video-rate Localization in Multiple Maps for Wearable Augmented Reality. In *Proceedings of the International Symposium on Wearable Computers*, Pittsburgh, Pennsylvania, September 28 - October 1, 2008, pages 15-22. **Best Paper Award**.
- R. O. Castle and D. W Murray. Object recognition and localization while tracking and mapping. In *Proceedings of the 8th IEEE/ACM International Symposium on Mixed and Augmented Reality*, Orlando, Florida, October 19-22, 2009.

Publications under review

- R. O. Castle, G. Klein and D. W Murray. Combining monoSLAM with Object Recognition for Scene Augmentation using a Wearable Camera. *Under review*.
- R. O. Castle, G. Klein and D. W Murray. Wide-area Augmented Reality using Camera Tracking in Multiple Small Maps. *Under review*.

Notation

Throughout this thesis, the following conventions will be used for typesetting mathematics unless otherwise indicated:

- **2D and general vectors** are written in lower case bold: $\boldsymbol{a}$, and a unit vector with a caret $\hat{\boldsymbol{a}}$. Vectors are usually column vectors, with elements specified by subscript index for example $\boldsymbol{x} = (x_1, x_2)^\top$. 3D vectors will be written in uppercase $\boldsymbol{A}$.
- **Matrices** are written in teletype: $\mathtt{A}$, and may have size indicated, $\mathtt{A}_{3\times4}$. The entry in the ith row and jth column of the matrix is $\mathtt{A}_{ij}$.
- **Quaternions** are written as q and their conjugate as $\bar{\mathsf{q}}$.

List of videos

The thesis is accompanied by a DVD of results in video format ordered by their respective chapter. An electronic version of this document is also provided.

Chapter 2

stabilization_not_active.avi	Wearable camera without stabilization
stabilization_active.avi	Wearable camera with stabilization

Chapter 4

exp1_monoslam.avi	Experiment 1: MonoSLAM
exp1_objectslam.avi	Experiment 1: ObjectSLAM, no image doubling
exp1_objectslam_double.avi	Experiment 1: ObjectSLAM, image doubling
exp1_objectslam_allframes.avi	Experiment 1: ObjectSLAM, all frames
exp2_scaling.avi	Experiment 2: Scaling
exp3_occlusion.avi	Experiment 3: Occlusion and depth issues
exp4_gallery_monoslam.avi	Experiment 4: Gallery with monoSLAM
exp4_gallery_objectslam.avi	Experiment 4: Gallery with objectSLAM
exp5_street_monoslam_plate.avi	Experiment 5: Street with monoSLAM and calibration plate
exp5_street_monoslam_noplate.avi	Experiment 5: Street with monoSLAM and no calibration plate
exp5_street_objectslam_plate.avi	Experiment 5: Street with objectSLAM and calibration plate
exp5_street_objectslam_noplate.avi	Experiment 5: Street with objectSLAM and no calibration plate
exp6_oscilloscope_ar_tutorial.avi	Experiment 6: Oscilloscope tutorial
exp6_oscilloscope_feature_tracking.avi	Experiment 6: Oscilloscope tracking

Chapter 5

exp1_multiple_cameras.avi	Experiment 1: Multiple cameras
exp2_map_switching_1cam.avi	Experiment 2: Map switching with a single camera
exp2_map_switching_2cam.avi	Experiment 2: Map switching with two cameras
exp3_overlapping_maps.avi	Experiment 3: Overlapping maps
exp3_mobile_maps.avi	Experiment 3: Mobile maps
exp4_desks.avi	Experiment 4: Desks
exp5_laboratory.avi	Experiment 5: Laboratory
exp6_building.avi	Experiment 6: Building exploration
exp7_01_entrance.avi	Experiment 7: Entrance map creation
exp7_02_triceratops.avi	Experiment 7: Triceratops map creation
exp7_03_table1.avi	Experiment 7: Animal table 1 map creation
exp7_04_table2.avi	Experiment 7: Animal table 2 map creation
exp7_05_dinosaur.avi	Experiment 7: Dinosaur map creation
exp7_06_elephants.avi	Experiment 7: Elephants map creation
exp7_07_tour.avi	Experiment 7: Natural history museum AR tour
exp7_08_footprints.avi	Experiment 7: Example of failure

Chapter 6

exp1_evaluation.avi	Experiment 1: Evaluation
exp2_simple_wall.avi	Experiment 2: Simple wall
exp3_posters.avi	Experiment 3: Poster wall
exp4_cluttered_desk.avi	Experiment 4: Cluttered desk
exp5_gallery.avi	Experiment 5: Gallery
exp6_street.avi	Experiment 6: Street
exp7_oscilloscope.avi	Experiment 7: Oscilloscope tutorial

Abbreviations

AR	Augmented reality
BBF	Best-bin-first
CCD	Charged coupled device
CGI	Computer generated imagery
COTS	Commercial off the shelf
CPU	Central processing unit
DOF	Degrees of freedom
DoG	Difference of Gaussians
EEPROM	Electrically erasable programmable read-only memory
EKF	Extended Kalman filter
GPGPU	General purpose graphics processing unit
GPS	Global positioning system
GPU	Graphics processing unit
GUI	Graphical user interface
HCI	Human computer interaction
HMD	Head mounted display
IMU	Inertial measurement unit
LCD	Liquid crystal display
LED	Light emitting diode
LM	Levenberg-Marquardt
LoG	Laplacian of Gaussian
MID	Mobile internet device
monoSLAM	Monocular (single camera based) simultaneous localization and mapping
OS	Operating system
PC	Personal computer
PCMCIA	Personal computer memory card international association
PDA	Personal data assistant
pdf	Probability density function
PSU	Power supply unit
PTAM	Parallel tracking and mapping
PTAMM	Parallel tracking and multiple mapping
RAM	Random access memory
RANSAC	Randomized sample and consensus
RGB	Red green blue
SfM	Structure from motion
SIFT	Scale invariant feature transform
SLAM	Simultaneous localization and mapping
SURF	Speeded up robust features
UMPC	Ultra mobile portable computer
USB	Universal serial bus
VGA	Video graphics array
XML	Extensible markup language

1

Introduction

In this chapter the motivations for the research in this thesis are laid out, along with the reasoning for the high level decisions taken. The result is the remainder of this thesis, that details the design and construction of a wearable assistive robot, and how to allow a user to explore arbitrary large environments automatically viewing augmented reality at locations and on objects of interest.

1.1 Introduction

The nature of our environment, coupled with aeons of evolution, have made vision the dominant sense in much of the animal kingdom. We use it for a diverse range of tasks, such as navigation, fine closed-loop control of our limbs, recognition of objects, places and people, collision avoidance, and understanding 'events'.

Computer vision applications have been developed which emulate all of these, with various degrees of success. However, much of the work has focussed on 'outside-in' sensing, where the cameras might be static, or in scenarios where the cameras are moved by machine, be it a robot arm or vehicle. For many years, visual sensing was expensive, and the camera and its placement treated with a certain reverence.

Recent years have seen a change in this approach. Advances in miniaturization of digital

camera technology, the availability of prodigious computing power on portable CPUs, the reduction in power consumption, improvements in battery technology, and the availability of high bandwidth wired and wireless communications, have made personal video devices such as mobile phones, hand held computers and game consoles, ubiquitous. With these advances comes the possibilities of wearable computing, and here in particular wearable visual computing, and the adoption of an inside-out, first person, or ego-centric approach to sensing.

1.2 Motivation

One mode of use of wearable visual computing is so-called memory augmentation [122, 57, 88, 71] where information is merely stored, either on the person or by transmission to some base, for later review or analysis. In this thesis however, we are concerned with the more challenging applications made possible by real time analysis of the video stream and the immediate return of information about the current scene to the user. Even here, there are choices. Scene analysis could be achieved by transmitting video to a remote expert, say a medical doctor who could return instructions to a paramedic in the field. From an engineering standpoint, this is a solved problem, and the challenges lie in the human understanding and communications.

However, this does not scale well, in terms of use of expertise with one expert per user, or communications bandwidth. Instead, in many applications, particularly those where the knowledge domain is static and bounded, a computer based assistant, held locally, could provide as much information and in a more timely manner. The provision of such augmented vision reality in a wearable system is the aim of the work presented here.

With simple tasks, a low technology solution such as a paper manual may suffice. As tasks grow in complexity, and require a user to manage data from multiple sources, 'simple' solutions become unwieldy and prone to operator error. For example, imagine an engineer responsible for checking the building of a complex piece of machinery, such as a nuclear submarine. Not only does he need to know how the parts mechanically fit together (connect bulkhead A to

bulkhead B), but how various pipes and ducts connect, what they contain, where the wiring goes, what each wire is for, the types of bolts needed, the orientation of each part, and much more. Then on top of these base data are the various revisions that different parts will have gone through, who has worked on each part and when, what work is left to be done, and how various changes will affect others parts. On paper this kind of information takes up vast volumes and is not easily correlated or kept up to date. Making all this information digital, can help with searching and correlating the data, but various interactions can be missed and interpreting the data can still be difficult. By overlaying the information with the physical world, correspondences and issues could be identified much more quickly.

Augmented reality (AR) systems that integrate all data and seamlessly overlay the required information on images of the real world are not yet a reality, but the work of many in the AR community and the work in this thesis aims to bring this ideal closer to reality.

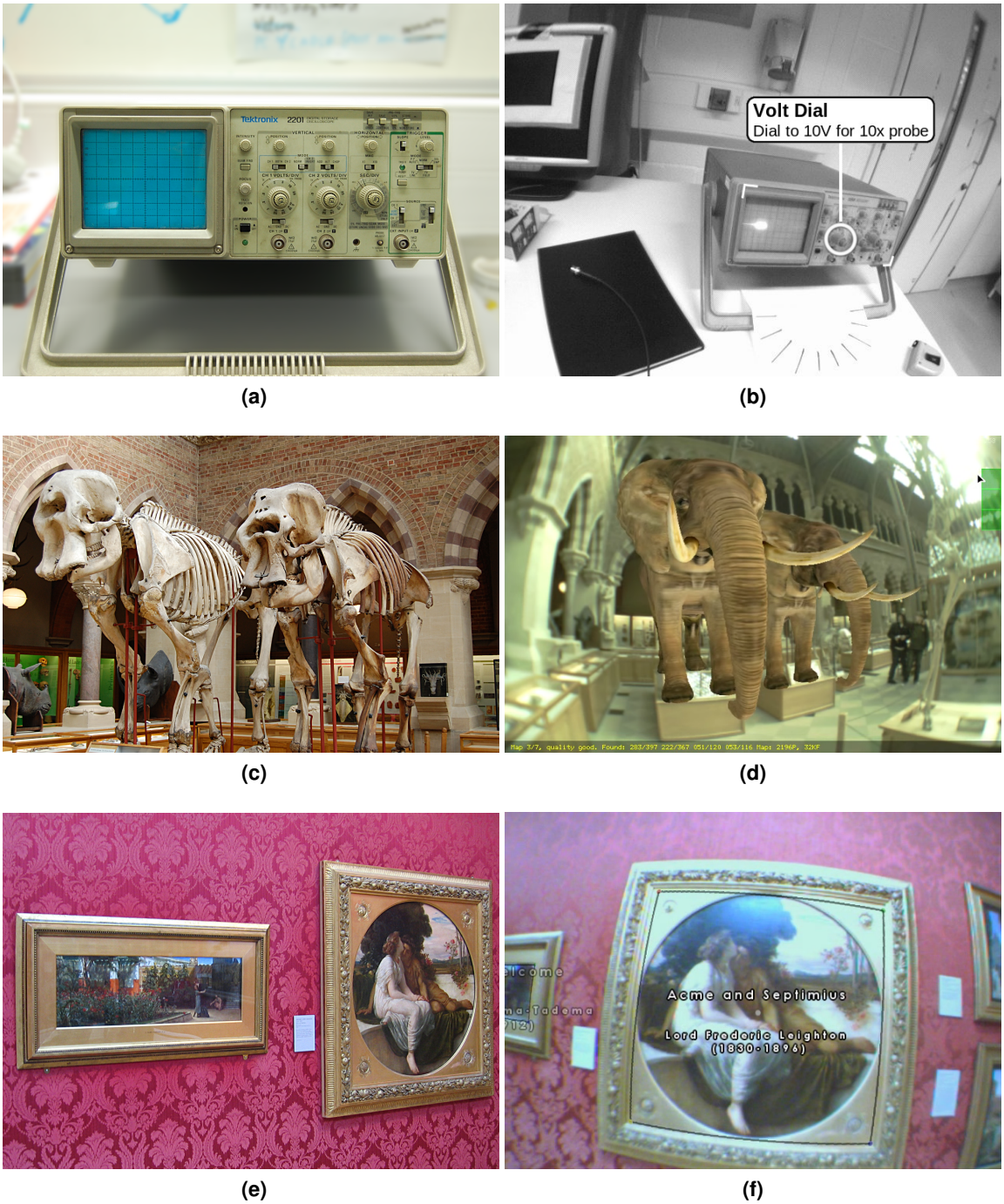
The following recent examples demonstrate AR being applied to solve some of the data overlay problems. Quarles *et al.* [119] demonstrated how students could be taught to use a complex piece of medical machinery and provide the teacher with information on how the students had learnt from their interactions with the machinery all using AR to simplify the process and overlay pertinent data to the students and the teacher. Georgel *et al.* [47] demonstrated a system that could detect discrepancies between CAD models and installed components in industrial environments, allowing incorrect parts or incorrectly oriented parts to be automatically detected, where a human may fail to spot the difference. Schall *et al.* [128] used AR to overlay underground pipe information on a user's hand held display, while the user walked around the street above. This enabled the most suitable location of where needed to be dug up to be found more accurately. Pipe data from various sources was integrated to show all of the known pipes, removing the need to check multiple sources to verify a suitable location. The ability to overlay different layers of information on the world using portable technology provides a new way of interaction and exploration.

These examples solve their particular problem, but they do not generalize well. Quarles is limited to tracking the user's AR viewing device in a small controlled environment. There is no actual link between the physical object of interest and the AR displayed: the AR is simply lined up to match. Georgel is limited to matching models to images, with no knowledge of the world or where objects fit within the world. Schall locates the user within the world using GPS and inertial sensors, which is acceptable for outdoor use, but not indoor. There is no visual registration, leaving the typically large errors from the GPS and inertial sensors to cause misalignments between the AR and the view of the real world.

Fig. 1.1 shows three examples from this thesis. In all these examples the computer based assistant is processing the image feed to determine what information should be displayed for the user. In the first from Chapter 4, Fig. 1.1(a,b), a user is presented with a piece of equipment that has a bewildering array of buttons and dials. Using augmented reality (AR), the user can be guided through different operating procedures with the relevant dial or button highlighted and relevant contextual information shown.

In the second from Chapter 5, Fig. 1.1(c,d), a visit to a natural history museum is brought to life by placing 3D models of elephants over their skeletons. The user is free to walk around the exhibit gaining knowledge of not only what held the elephant up, but also what it looked like. The complexity and content of the AR displayed is here limited largely by the imagination of the designer, with animations, sounds, video, and multiple layers of graphics all being possible.

The final example from Chapter 6, Fig. 1.1(e,f), shows how AR can be used to provide detailed information about paintings in a gallery. Audio guides are common place, but they require the user to locate the right numbers dotted around the gallery and key them into the handset. Using AR the paintings can be automatically labelled when observed and be used to visually highlight items of interest along with any audio commentary that is required.



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

Figure 1.1: Example applications for AR. A bewildering array of buttons and dials in (a) is simplified by using AR in (b) to guide a user through setting up the oscilloscope. The elephant skeletons in a natural history museum (c) are brought to life in (d) by placing 3D models over them. Paintings in a gallery (e) can be recognized and have useful information overlaid for the user (f). These three examples are actual results from this thesis.

1.3 The approach to wearable vision and AR in this thesis

The foretaste of the results presented above suggest an emphasis on visual computing alone. However, there are two parts to generating an AR system based on wearable vision. One is certainly the visual computing for the assistant, and here it is argued for the approach of first building a map of the environment, then locating the camera in it, and recognizing and localizing objects in that environment. But the second part is the design of hardware that makes up the physical device, and they are both dependent on each other. The role of the assistant's software dictates to some degree the hardware design, and what is possible with the physical device dictates what the software can achieve.

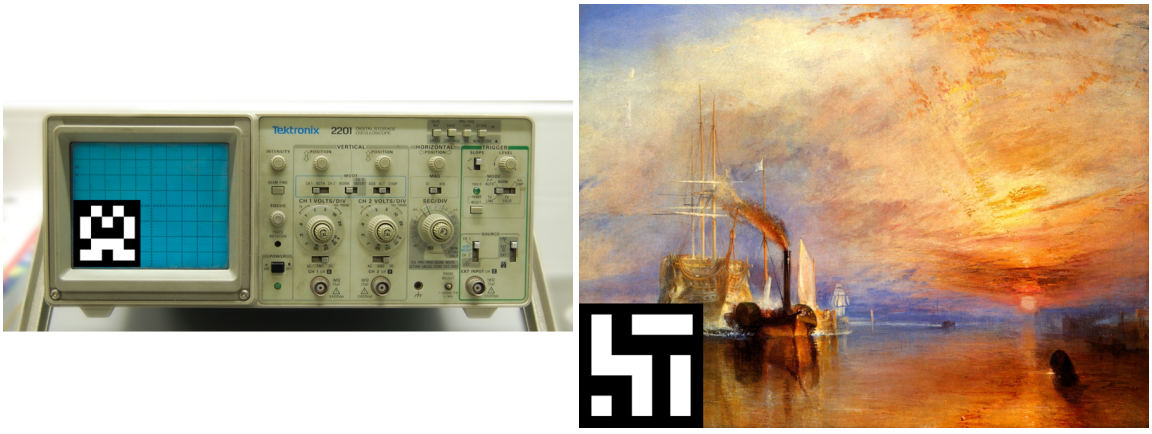
First, the software software side of the assistant is outlined, followed by the physical device.

1.4 Interpreting the world

A user of a wearable AR system needs to be able to explore an environment freely, no matter the size, and be automatically presented with information about the world around them. The user may also need to be able to modify or create new AR experiences at locations of interest, so the ability to do this in situ is also required. To enable this, the system needs to know where the camera (and hence the user) is in the world, and in relation to objects of interest in the world. With this information AR overlays can be placed in the world and in relation to objects.

1.4.1 Object recognition and tracking

A popular way of detecting objects or places of interest and displaying relevant AR has been to place fiducials on objects of interest. The two systems that have come to prominence using this method are ARToolKit [62, 5], and ARTag [41], and there are various derivatives and rivals of these systems. The ARToolKit fiducials and derivatives have been used successfully in applications such as interactive museum tours [129] and commercially available computer games



(a) (b)

Figure 1.2: Placing markers on certain objects is not practical or allowed.

[140]. When these distinctive fiducials are detected, the relevant AR is shown. While this is a cheap and effective way to present AR, it is limited to only showing AR when the marker is in view and only in relation to the fiducials. Knowledge about the structure of the world is not known, which may be acceptable for some applications. The main limitation is the fact that the world has to be plastered with these fiducials, which is not possible in all situations. Fig. 1.2 shows a couple of examples where using a fiducual is not possible. In (a) there is no area large enough to accommodate a marker without obscuring something of importance. In (b) there is no way that anyone is going to be allowed to stick a fiducial on pieces of art, priceless or otherwise. Placing a marker on the wall may be acceptable, but if the AR is to highlight details at precise locations then the fiducial needs to be located exactly at the right location for the AR to match up.

An alternative to using fiducials has been to model 3D objects using hand crafted CAD models [46, 51, 84, 34]. This is effective, as long as the object has clearly defined edges, and the modeller has been able to model the object adequately for detection and tracking. This approach is labour intensive, even for relatively simple objects, and many natural and manufactured objects cannot be tracked well, or at all using this approach. This approach also suffers the same limitations as the fiducials in that the problem being solved is camera tracking relative to the model, and the AR is only shown when the model is in view and has been detected.

Again there is no knowledge of the world, unless the model is the world, such as a room.

The third option is to use natural features that occur on the objects themselves. This removes the need to cover the world in fiducials, or to hand model every object. All that is required is an appropriate number of observations of the desired object. Gordon and Lowe [49] used this approach to create 3D models from a set of images where an object had been observed. This approach removes the need for a modeller. Again, however, the AR is only valid while the object is in view and detected. Using natural features provides the most versatile way to locate and detect objects as no changes to the world are required, and no expert knowledge is required to model all possible objects. There are still issues with this approach, such as objects that are textureless or that have repeating textures, as these either have too few features to track or multiple locations that have the same features causing confusion. The natural feature approach is used in this work as it provides the best way to detect objects that cannot be marked.

1.4.2 Mapping the environment

These three methods for detecting objects or areas of interest all suffered the same problem: AR could only be shown relative to them while in view and being detected. There is no knowledge of the world around the object. What is required is a method for mapping the world around the user to allow AR to also be placed within the world. It could be argued that making a map of the world is no different from modelling an object, just scaled up, and effectively there is no difference. The same objections to fiducials and exhaustive modelling apply to mapping a room or building. Instead a natural feature approach is preferred, and one that can run in 'real-time', *i.e.* at the camera's frame-rate. This way, as a user explores an environment the map is constructed on the fly. In some situations creating a map each time may be preferable, such as in locations where the structure may change over time. In others once the map has been made it can be kept, and from then on the camera only needs to track within it.

There have been two main branches of research for creating maps of an environment, one

is structure from motion (SfM) [83, 116, 43, 104] , and the other is simultaneous localization and mapping (SLAM) [138, 75, 144]. Both of these will be discussed and used later in this thesis. The main difference between the two is that in SfM, all the measurements are required beforehand, and the model of the world is constructed using a batch process. In SLAM the world is recovered in a recursive manner, with each frame received allowing SLAM to add to its model of the world. SfM is typically run as an off-line post production process as it takes a significant amount of time to run, and has found great success in the film industry allowing CGI to be realistically placed in the scene [1]. The recursive nature of SLAM has made it ideal for robotic exploration, though careful map management is required to keep it running in real-time. Various SLAM based hand held single camera systems have been developed that run at frame-rate [28, 36, 18]. The main advantage of SLAM is that it is recursive, and therefore does not need all of the measurements to begin with like SfM. Recently, however, this advantage has been eroded with the appearance of SfM based systems that can add new data as the system continues to explore [106, 102, 65]. In this thesis a SLAM based system is used in Chapters 3 and 4, and then a 'real-time' SfM method is used in Chapters 5 and 6.

With either of these mapping technologies, AR can be placed in the world relative to the mapped area, allowing larger and more involving AR experiences. However, any AR placed on an object that happens to appear in the scene has no actual relationship to that object. The inference is only made by the user. While this may be acceptable in some situations, it is not for all.

Perhaps the ideal combination of object recognition and environment mapping is one which allows objects to be automatically located and AR shown related to them, but within a larger mapped environment where other AR can be shown. The advantage is that once an object has been found in the world its AR can still be shown, whether or not it is in view and has been detected in a particular frame. This thesis will demonstrate the synergy of these two methods.

1.5 The physical device

There are two goals that the physical realisation of the wearable assistant has to meet. The first is to allow the user to complete tasks using AR, and the second is to allow the assistant to seek out relevant information in the surrounding environment. Both of these tasks must be done with minimal interference of the user and of the user's interactions with the environment.

AR systems, such as the ones already noted, typically involve a user exploring using a camera and some form of display (typically head mounted or hand held). These systems are user driven, only displaying the information related to where the user has pointed the camera. This leads to the issue that the user may miss something vital by looking the wrong way. To leverage the power of wearable visual computing the assistant requires an element of autonomy from the user. For example, by allowing the assistant to control the camera it can seek out things in the world of interest to the user. The motorised cameras (active cameras) created by Mayol [96] and Kurata [70] allow this type of exploration. Once can argue however that these systems go too far: now the user is at the mercy of the assistant, and can only go about their task and view AR where and when the assistant decides.

What is required is balance between the assistant's control and the user's control. To enable the assistant and user to explore the environment in a semi-independent manner, the system developed in this thesis will utilise two cameras: one user controlled, and the other assistant controlled.

1.5.1 A semi-autonomous assistant

Kurata *et al.* [70] showed that having a remote human assistant help a user complete a task greatly reduces the time taken. It was also shown that providing the user with visual feedback on what was required of them was superior to audio descriptions or laser pointers. The separation of tasks aided in speeding completion times as the assistant could be looking for the next

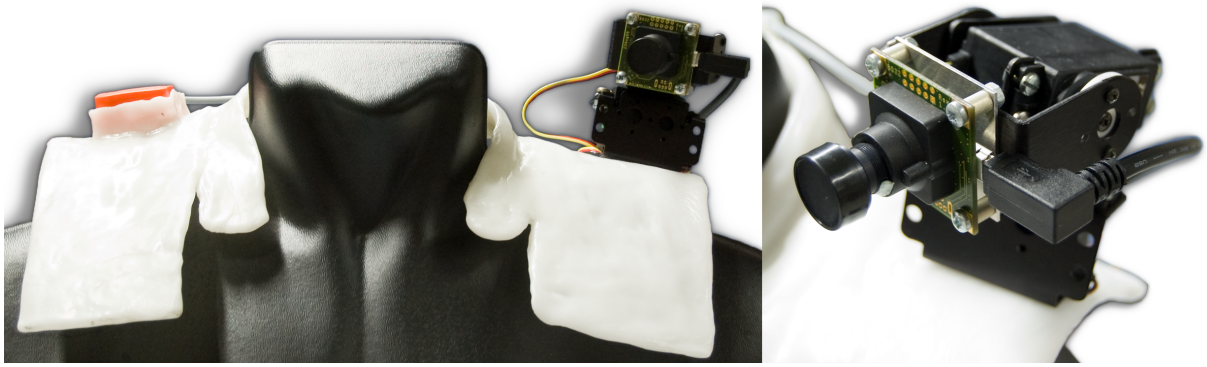


Figure 1.3: The wearable active camera designed in the next chapter.

task to perform while the user worked on the current task. However, the user was not free to explore the world as they wished, essentially becoming puppets for the remote assistant. This was especially true where the camera could not be moved independently of the user, meaning that the assistant had to direct the user to face where he wanted. In some situations this kind of control may be desirable, but in others not.

Mayol's [96] work allowed an operator to tag locations of interest that a shoulder worn active camera could fixate upon, allowing later users to be directed to these locations by the camera fixating upon them. Again, the user is directed solely by the assistant, even though they are not there at the same time. The assistant controlled single camera view restricts the user's freedom to work independently.

To avoid such conflicts the approach used in this thesis is to separate the user from the assistant, allowing them both to have a camera each. The user can then point the camera where they wish to help in task completion, and the assistant (human or computer) can use a second camera that is active to allow the assistant to look around the world semi-independently from the user. In this way, information can be shared collaboratively with minimal interference. This is the approach that will be presented in Chapter 2.

Mayol [96, 94] argued that the best location for an assistive camera is on a user's shoulder, as it provides the largest viewing volume of the world, and provides minimal interference with the user. Kurata also used an active shoulder mounted camera in his work. As detailed in the

next chapter, a shoulder mounted active camera is developed for use in this thesis. Fig. 1.3 shows the active wearable camera developed in this thesis.

1.5.2 A window to an augmented world

Whether a semi-autonomous assistant is required or not, the user needs to see the augmented world, and for this some form of display is required. These fall in three categories: head mounted displays (HMDs), hand held displays (*e.g.* portable LCDs), and projectors.

Projectors are typically heavy, bulky devices. However, a new breed, known as pico projectors, are starting to appear, with an aim to be integrated into mobile phones and digital still cameras. These devices can be used to display contextual information, but it is limited to 2D information only, and requires a suitable surface on which to project. While suitable for some applications, they do not allow the full complexity of AR that can be achieved with regular screen displays. Mistry *et al.* [98] have implemented an AR system using a head worn camera and a pico projector on a pendent around the user's neck. The system allows for novel human-computer interaction (HCI), but lacks the ability to display 3D information. Its main function is to provide novel interfaces and additional contextual information projected onto surfaces surrounding the user. This is also the devices main drawback: private information is displayed in a very public manner; and the system does not scale well to many people using this kind of device together as they would end up competing for suitable projection surfaces.

HMDs could be the ideal solution if the technology was more mature. The current selection of HMDs are rather limited (some are shown in Fig. 1.4), ranging from simple video displays to monocular or binocular 'see-through' displays where a camera is mounted on the reverse of the screen allowing the user to see what is behind the display. The 'video only' displays tend to be the smallest HMDs available, and are ideal for receiving a video feed, for example from a wearable camera. For immersive AR the see-through HMDs are preferred, but they are often large, heavy devices due to the camera or cameras mounted on them. There are several

Figure 1.4: A selection of head mounted displays (HMDs) currently available, ranging from simple video displays to 'see-through' displays in monocular and binocular varieties. In numbered order they are from the following manufacturers [152, 103, 141, 149, 123, 37, 23, 60, 130, 126, 150, 108, 21, 82].

other major problems with HMDs. The displays obscure a user's vision of the world, unless a transparent display technology is used. The transparent technology itself can be questioned: how large is the active area; how transparent is it; and how clearly can graphics be seen. If a see-through device is used then this partially counters the vision problem, but most HMDs have a narrow field of view, owing to the camera lenses used to minimize distortion, and the size of the displays used. This coupled with the inevitable motion lag from the system (the camera image has to go to a computer to be processed and then sent to the display, and the camera and display only have limited refresh rates) can cause nausea and physical injury.

The third option is the cheap and mature technology of flat panel displays (typically using LCD technology), which can be found on many portable devices such as those shown in Fig. 1.5. By placing a camera on the rear of the screen these become 'see-through' displays, or 'magic lenses', allowing the user to view AR on the video feed. The limitation is that the user has to hold the display, stopping the user from using one hand. Advantages of a hand held display over a HMD are: that the user's vision is not obscured; that it can also act as an input device as detailed in the next section; that the image can be viewed by other people, making the device usable by small groups of people, rather than a solitary experience. Wither *et al.* [158] showed that a magic lens allowed users to complete tasks faster or as fast as those with HMDs.

Figure 1.5: A selection of touch screen devices. (a) an 8 inch touchscreen display from Lilliput [80], (b) Mobile phone and media player from Apple [4], (c) a MID from Gigabyte [48], (d) a PDA from Palm [112] (e) a UMPC from Sony [141], (f) a slate PC from Motion Computing [101], and (g) a tablet PC from Lenovo [73].

1.5.3 Interaction

To interact with the wearable, the user must be supplied with a method of data input. Previous wearable computers have utilized portable adaptations of standard desktop computer input hardware, such as small wrist worn or hand held keyboards [20, 91, 95], hand held mice, keypads, and joysticks. Touch screen displays can be used to serve as pointing input devices and keyboards using virtual on-screen keyboards.

An alternative to physical input is to use vision or audio. Voice recognition is still developing, and there is no solution where a computer can understand natural language completely. Instead a system with a limited set of commands could be developed, but even these systems are not perfect, requiring training and patience to use [135, 107, 15]. Speaking to issue commands may not be acceptable depending on the location and task, for example in a quiet or noisy work environment, and there is the significant question of how the computer knows that it is being spoken too.

The vision system can be used to interpret gestures. The shape and motion of the hand can be used to represent different commands. For example, Mayol *et al.* [96, 30] incorporated the hand recognition system of de Campos [29] into a single camera SLAM system to allow 3D

virtual objects to be placed into a scene. The main limitations of gesture recognition, however, are being able to correctly discern the hand from the background, and correctly identify the pose [29, 68, 87]. The camera also needs to be able to see the user's hands, and to know when a gesture is meant for it. Finally, designing a clear and expandable gesture system that has a low false positive rate becomes more challenging as the number of gestures increases.

A final type of input mechanism uses the eye. By having a camera looking at a user's eye, the pupil can be tracked and its gaze used to control an input [35]. Blinking can also be incorporated as a control feature. The problems with eye tracking are similar to those of gesture recognition. Knowing that the user is inputting a command and not just looking in a direction is hard to disambiguate, and setting up a clear and expandable command set is difficult.

1.5.4 The chosen solution

Based on the above arguments the wearable visual robot developed in the next chapter is a two camera wearable system. One camera is an active shoulder mounted camera that is under control of an assistant. The other is attached to the back of a hand held touch screen display, under control of the user for use as a magic lens. The user can see information and video from both the wearable's cameras, but primarily views the hand held video feed. Interaction uses the display itself, with the user touching areas of interest in the image. The user holds the screen in one hand and interacts with it using the other, but there is no reason why the display could not be stowed away when not in use, on a belt holder for example. The system can track using the shoulder mounted camera, removing the need for the user to constantly maintain a good view of the world with the camera.

1.6 Thesis outline

As noted above, the next chapter describes the design, build and test of a wearable assistive robot. It comprises an assistant controlled camera and a user controlled camera. The assistive

camera is a shoulder mounted active camera that automatically stabilizes, and is controlled by the system. The user's camera is part of a hand held magic lens allowing the user to view graphical augmentations of the image. The system is portable and wearable, allowing the user to explore large environments for long periods of time.

In Chapter 3 object recognition is combined with a monocular SLAM system, enabling known objects to be automatically recognized and localized in the world. As well as aiding the mapping process, the detected objects allow AR related to them to be automatically displayed to the user as they explore the world. Chapter 4 details the implementation and evaluation of this system.

In Chapter 5 a new and superior mapping and tracking system based on SfM is used, and extended to allow the user to make multiple maps in an environment. This allows a user to explore environments of any size, and view AR at locations of interest. The system is also extended to allow multiple independent cameras to help build a map simultaneously, thus allowing the dual camera wearable to build maps using both the assistive and user cameras.

Chapter 6 brings the idea of object recognition and localization from Chapter 3 and applies it to the SfM system developed in Chapter 5. The implementation differs greatly and provides a way for moving from planar objects to 3D objects, which is discussed in the conclusions in Chapter 7. The conclusions also look at other enhancements that could be implemented to greatly improve the system.

2

Wearable visual robot design

In this chapter the design of a wearable visual robotic assistant is detailed. The different technology choices are evaluated, and a final design that combines a wearable visual robot, and a hand held augmented reality system into a single portable system is shown.

2.1 Introduction

As outlined in §1.5, a wearable system that is to operate in an assistive manner needs to have access to both imagery that has a degree of independence from the motion and stance of the user, and to imagery whose viewpoint is under the direct and immediate control of the user. The design proposed in this chapter uses two cameras: one is a shoulder mounted active camera under the control of the assistant; the second is attached to the rear of a hand held display under control of the user.

It is self-evident, but bears emphasis, that a wearable system must be, first and foremost, wearable, and for long periods of time. Any extra equipment that a user has to carry around needs to be as unobtrusive as possible, to minimize impact on how they move about and interact with the world. However, the system must be powerful enough and capable enough to perform the tasks required of it. Therefore, throughout the component selection process

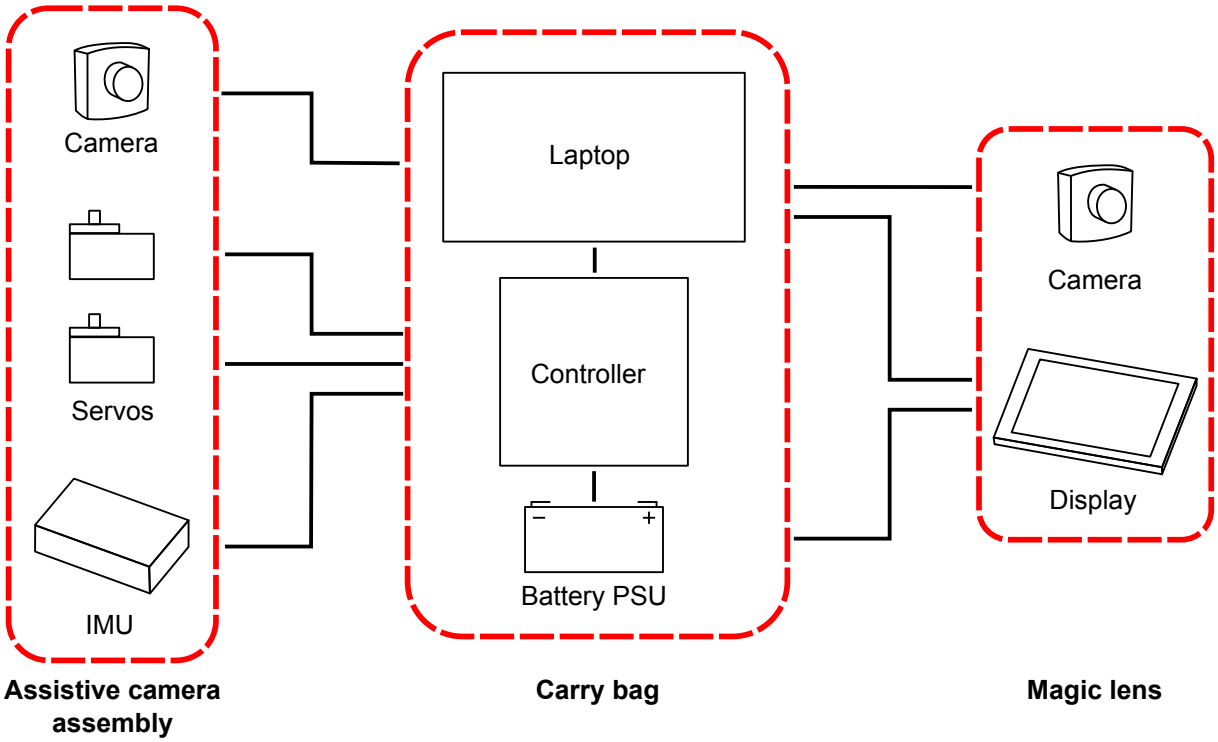


Figure 2.1: The wearable system diagram.

the aim was to minimize weight and size, but balance it against usability. The availability of different components and features also affects the choices made.

The final design, shown in Fig. 2.1, can be broken into three main components: (i) the carry bag holding the processing and power elements; (ii) the assistive camera assembly that is worn around the user's shoulders; (iii) the user's hand held AR device, or magic lens.

The following sections detail each of these main elements, and the selected components within them, along with the factors leading to the particular selections. First, Section 2.2 discusses the carry bag and its contents, and Section 2.3 details the magic lens. Section 2.4 describes the shoulder worn assistive camera assembly, followed by a section which explores the assistive camera's controller, including the stabilization routine. The final section, Section 2.7, describes bringing all the elements together to produce the final wearable.

2.2 System electronics

The principle electronic systems for the wearable are housed in a laptop carry bag, and consist of a portable processor, the assistive camera's control unit, and batteries. With a mass of 4.8 kg, it is somewhat heavier than desirable, a result of the restricted choice of processor and battery technology, as discussed in the following sections. The control electronics are discussed in detail later in Section 2.5. The carry bag is a standard laptop shoulder bag that has been modified with ventilation holes to enable the laptop to operate inside it.

2.2.1 Portable computer

The wearable's computer needs to be sufficiently powerful to meet the demands of video-rate, multi-threaded processing, yet small and light enough to be genuinely portable. It also needs to have appropriate hardware interfaces, which in this case are USB ports and a VGA display port as discussed later.

An ideal, perhaps, would be to include the processor as part of the user's hand held display, reducing the number of components. However, tablet[1] and slate[2] computers are currently both too heavy and bulky for single-handed use, and also underpowered. This leads to the need to separate the computer from the hand held display. The choice now becomes one of either a portable computer or building a custom computer that can be worn, such as Piekarski's [114]. For size, weight, ease of deployment, and the ratio of features and computation power to size and weight, a portable computer is always a better option: portable components are difficult to source singly, and custom built machines tend to be made from larger, desktop components.

At the time of selection (2006) Intel Dual Core CPUs had the best performance, but in laptops these were often paired with Intel GPUs, such as the GMA950. These are unsuited to real time vision, owing to their appallingly slow 3D image rendering. The preferred manu-

[1] Tablet computers are standard laptops with a touch screen that rotates to cover the keyboard.

[2] Slate computers are hand held displays with an embedded computer, *i.e.* a tablet without a keyboard and a fixed display.

facturer of GPU was Nvidia, because of their superior graphics performance and support for Linux drivers. At this point, choice was severely restricted as Nvidia chipsets were most often included in laptops with large displays making them "moveable desktops" rather than true portables[3]. Another issue was that of hardware interfaces, namely Firewire ports. The larger gaming and professional laptops generally have a better port selection, but none at the time had a full size powered Firewire port, restricting the use of Firewire cameras as detailed later. Apple Mac laptops were discounted, despite having a powered Firewire port, due to having AMD/ATI GPUs, and the unknown element of running Linux on them.

The portable computer selected for the wearable is an IBM/Lenovo T61p Thinkpad [73] with a 2.20 GHz Intel Dual Core CPU, 2 GB RAM, and an Nvidia Quadro FX 570M with 256 MB RAM. It has three USB ports, two PCMCIA card slots, mini-Firewire and external VGA port, this last used by the external display. The total mass of the laptop is 2.86 kg, of which 0.54 kg is from an additional battery replacing the DVD drive. Part of this bulk, the 15.4 inch widescreen display, is unfortunately redundant during operation.

2.2.2 Battery PSU

The laptop, as noted above, contains two batteries, providing it with a run time of around 2 – 2.5 hours when under load. The other components that require their own power are the hand held display, the servos, and the control electronics. The batteries for these were selected so as to last at least as long as laptop. The hand held display uses a 1 kg 12 V battery, lasting between 1 – 2 hours depending on the screen's brightness level. The servos share a 0.3 kg 6 V battery, which enables them to run for many more hours than the laptop and display. They require a separate battery from the rest of the system, because of the large fluctuating currents drawn when moving that can reset other electronic devices. The control electronics box contains two standard PP3 9 V batteries, one for the electronics and one for the IMU. These also last for much longer than the laptop and display batteries.

[3] 19 inch wide-screen displays and larger are not suitable for portable computing.

Figure 2.2: An example of a magic lens. The AR model of a bunch of sunflowers can be seen placed on the table in the display.

2.3 The magic lens

For the user to view AR constructs in the world a magic lens is used. This is a display with a camera mounted on the rear, and placing AR elements on the images from the camera, the user is able to view an augmented scene 'through' the display. Fig. 2.2 shows an example of this, with the scene on the display showing a bunch of sunflowers placed on the table.

2.3.1 The touch screen display

To remove the need for extra input devices, the display has a touch screen, allowing the user to interact with computer by using their finger as a pointing device. Text entry is accomplished through a virtual on-screen keyboard.

The touch screen display used is an 8 inch display with 800×480 pixels and a mass of 0.83kg from Lilliput [80]. The screen connects to the laptop via a VGA and USB ports, the latter providing the touch inputs. The wide-screen format was chosen, both, to match the output from the selected cameras (see below) and the laptop's display.

2.3.2 The magic lens camera

The second component of the magic lens is its camera. To ease cooperative use of this camera with the shoulder worn assistive camera, it was decided that both should be identical. The selection of both involves power and interfacing considerations, but physical constraints arise primarily from the latter.

A first decision is between the use of a Firewire (IEEE-1394) camera or a USB camera. For routine laboratory use Firewire is preferred over USB as it supplies video frames at a guaranteed sustained rate, and can support multiple cameras on the same bus. Unfortunately, powered Firewire ports are rarely supplied on portable computers, and are not on the model chosen earlier to satisfy the processing demands. The small Firewire cameras used in laboratory work will drain a 9 V PP3 battery in only 30 minutes. USB, on the other hand, provides the necessary power on the bus, but each *bus* (not port) will support just one camera's video output and, even then, not at a guaranteed rate. The number of USB ports does not correlate with the number of buses, and on the selected laptop the three USB ports are connected to two independent busses. Three busses are required, one for each camera, and one for the touch screen input and servo controller. It has been found that using a USB PCMCIA card acts as a separate bus, allowing the selected laptop to support the wearable's requirements. A further advantage of USB over Firewire for the wearable, are the cables: they are thinner and therefore more flexible; the camera end connectors come in smaller sizes; cables with angled connectors exist, reducing the amount the cable protruding from the camera. Because of these factors, USB cameras have been adopted.

The second requirement is that the camera must be globally shuttered. Most cheap webcams are fitted with rolling shutters, by which successive columns in the image are captured at successive times, introducing distortion when the camera is moving relative to the scene.

The third requirement is that the camera must be small enough for mounting on motors on

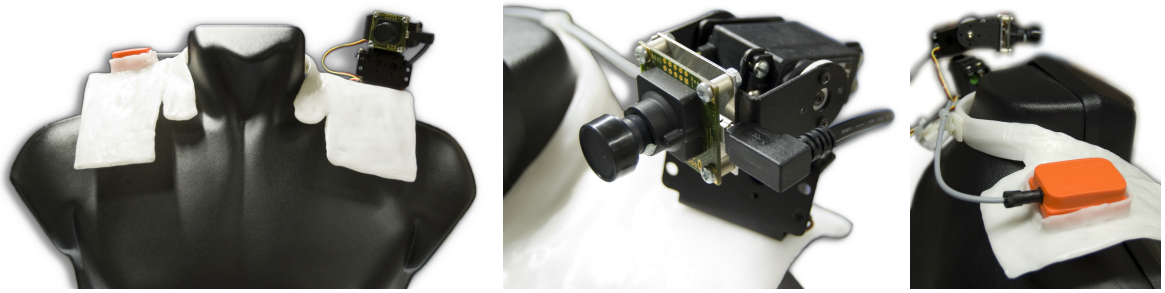
the user's shoulder. Mayol's solution [96] was to desolder the CCD imager from a Firewire camera, and to remount on a smaller board at the end of an extension cable. The resulting image was somewhat noisy, because of capacitive coupling. A later commercially produced remote head based on the same CCD suffered similarly, and the supplied cable was too short for the required use, with no other lengths available.

The final requirement is that the cameras must have drivers for the Linux OS. All Firewire cameras do by default, as there are well developed libraries for the IEEE-1394 standard. USB cameras have no unified protocol. Professional-grade cameras are most often supplied with closed source libraries. Many cheaper webcams do use one of several common protocols (due to the same internal components), but very few adhere to the standards, and do not supply Linux drivers. The lack of Linux support, their rolling shutters, and poor image quality rule webcams out of consideration.

The camera selected was the IDS µEye UI-1226LE USB camera [61]. Each camera is fitted with 2.1 mm wide angle lens, and provides 752×480 bayer images at 30 Hz. Each has a mass of 19 g (including the lens), and has a width and height of 36 mm and a depth of 20 mm. The USB cables used are angled to reduce strain upon the connectors, and to reduce the chances of snagging. The camera can be seen in Fig. 2.3(b) mounted on the assistive camera assembly discussed below. For the magic lens the USB cable is combined with the cables from the display in a single loom that runs to the carry bag.

2.4 The assistive camera assembly

The wearable camera assembly, shown in Fig 2.3, consists of a moulded plastic collar that sits on the user's shoulders, an inertial measurement unit (IMU) on one shoulder, and a pan-tilt camera assembly carrying the µEye USB camera. The total mass of the assembly is 0.6 kg.



(a) (b) (c)

Figure 2.3: The assistive camera assembly. (a) shows the front view of the wearable, with the IMU on the left and the active camera on the right. A close-up of the camera and the IMU can be seen in (b) and (c) respectively.

2.4.1 The collar mounting for the assistive camera

Mayol showed that an active camera mounted on a user's shoulder provided the best viewing location and also minimal user interference. His collar design, however, mounted the camera very close to the user's head, and it was possible for the user to come into contact with the camera if they turned their head too far. The collar itself was also unstable. If they leaned over at an angle too quickly it would become dislodged, in turn causing the user to become overly rigid in their motions and to move their head unnaturally in case the camera was knocked.

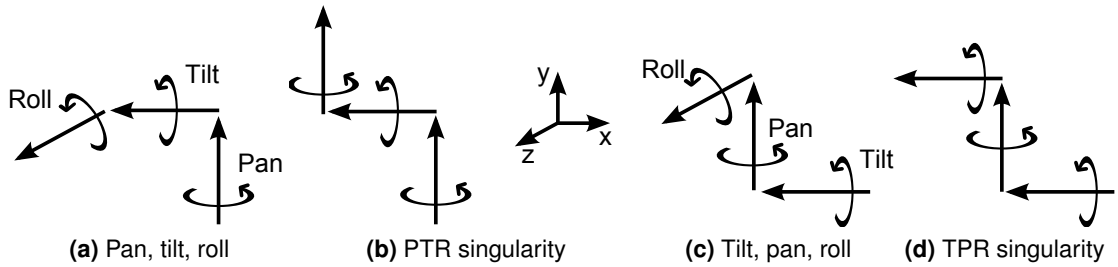
The solution is to move the camera away from the user's neck and out onto their shoulder. This requires a platform to be built out from the collar for the active camera to sit upon. It was found that merely extending Mayol's design with an added platform made the collar even more unstable. To stabilize the collar a counteracting force is required along with a better grip on the user. The solution, shown in Fig. 2.3, involves balancing out the load over both shoulders and spreading the area of contact with the user's body. The collar does not rotate forwards, as the sections on the user's upper chest pin it to the user, and it does not fall backwards due to this pinning, and due to all the weight being on the top and front. It is also designed so that it folds inwards, gripping the user slightly. The camera is mounted on one shoulder and the IMU on the other. Compared with Mayol's device, this design allows much greater freedom of movement, and improved stability.

The prototype collar is made from a modelling plastic [3] that softens in warm water. This allows it to be moulded into any shape desired and, if modifications are required, warming softens the plastic again. It was moulded by forming the basic shapes as flat pieces, then shaping on a model and allowing to cool and set. The neck piece and shoulder pads were joined by warming the plastic until sticky and smoothing the parts together. Fine adjustments were made by using a hot air gun to warm the desired area. A housing with a push fit was made for the IMU, removing the need for fastening. The active camera assembly is screwed into the base, and the screws are recessed on the underside, so as not to irritate the user. Several cable tidies were also placed on the collar to keep the cables from getting tangled.

2.4.2 Assistive camera motors

The choice of the motors to move the assistive camera is limited by the camera selection, and the user's shoulder. They have to be powerful enough to move the camera around smoothly and hold it at any position, while being small and light enough for placing on a user's shoulder. Miniature servo motors were chosen for their power to size and weight ratios, and their ease of control.

The motors also need to be arranged so that the camera can view all of the workspace it requires, regardless of the user's orientation. Mayol argued for a tilt-pan-roll camera arrangement, as shown in Fig. 2.4(c). This arrangement allows the camera to look fully up at the ceiling, but limited it to panning to only $90°$ to look sideways. At this angle a singularity occurs between the tilt and roll motors, as shown in (d). While singularities can be overcome, Mayol's camera could not look behind the user, effectively limiting the camera to look at the workspace in front of the user only. Instead a pan-tilt-roll arrangement is preferred here, as it has a singularity when facing directly up, and in the expected applications this condition will not be required. This configuration is shown in Fig. 2.4(a). With this configuration the camera is free to observe the entire viewing volume. Some way into the design, it was decided that the roll



(a) Pan, tilt, roll **(b)** PTR singularity **(c)** Tilt, pan, roll **(d)** TPR singularity

Figure 2.4: Servo motor configurations. (a) shows the pan, tilt, roll layout, along with its singularity when looking up (b). (c) shows Mayol's tilt, pan, roll layout with its singularity when looking sideways (d).

motor should be discarded. This was due to the total size and weight of the motor assembly and the camera, causing it to be unstable. Without the roll there is no possibility of a singularity, allowing the camera to look directly upwards if required. However, the lack of a roll motor affects the ability to stabilize the camera, and this shall be dealt with in Section 2.5.3.

The servo motors selected for the pan-tilt assembly are Hitec HS-422 servo motors [56]. The motors are held together using commercial servo brackets [86] designed for motors of this size. The cable loom running from the carry bag to the assistive camera assembly consists of: a power and data cable for the IMU; two 3 wire cables for the servos providing power and the control signal; a USB cable for the camera. The assembly has a mass of 0.13 kg, and can be seen in Fig. 2.3(b).

2.4.3 Inertial measurement unit

To enable the active camera to stabilize itself an inertial measurement unit (IMU) is required. These devices vary in complexity and accuracy, from simple 3-axis accelerometers which can measure their angles of roll and pitch (but not yaw) to 3D orientation units that can measure roll, pitch and yaw, allowing an active camera to know its current pose. These devices require power, and need to be connected to a PC or embedded electronics to interpret and act on their data, so the motors can be moved appropriately. If stabilization is all that is required, an accelerometer solution would provide the cheapest and smallest ($< 1\text{cm}^2$ footprint) solution. However, a full 3D orientation sensor can be used to feed back orientation data to the computer,

data which can be fused with a visual tracking system to enable saccades to, and fixation on, areas of interest, and to improve the camera tracking.

The IMU selected is a XSens MTx [159] that provides computed roll, pitch and yaw angles, and raw data if required. This particular model is small and lightweight as it was designed for human motion tracking, and provides data in an open format making it possible to process data on a non-PC CPU. It has a mass of 30 g, and is mounted on the opposite shoulder from the active camera, shown in Fig. 2.3(c). This helps balance the weight, but more importantly reduces the amount of electromagnetic interference from the motors which affects the IMU's measurements.

2.5 Assistive camera controller

A separate control CPU is used to control and stabilize the active camera. The active camera is not controlled directly by the laptop for two reasons. First, it removes an unnecessary burden from the laptop which would otherwise have to continually monitor the IMU data and update the motor positions. Secondly, and more crucially, in the absence of serial and parallel interfaces on modern portable computers the only alternative is to use a RS232 serial to USB converter. These introduce a significant delay of around 500 ms into data transmission, which is wholly unacceptable for real time control. Instead, a custom control circuit is used to monitor data from the IMU and the laptop, and to move the servos accordingly.

The selected processor is an Atmel ATMega128 [7]. It has: two serial ports, one for the computer and one for the IMU; signal generators that are independent of the CPU allowing the servos to be held with zero CPU usage; a 16 MHz clock; low power consumption. Furthermore, it is programmable in C, and has well supported development tools.

The stock 16 MHz crystal was replaced with one at 14.7456 MHz to allow the serial ports to run at 115,200 baud without data corruption or loss. The baud rate is required for the IMU to provide data at 100 Hz. The processor is mounted on an ETT development board [39]

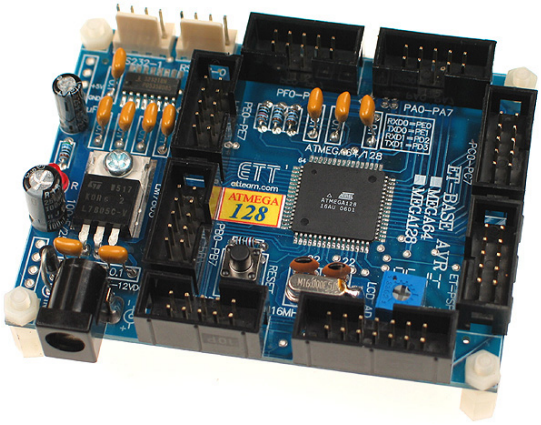
(Fig. 2.5(a)), and housed in a box containing an interface board between the external connections and the development board and two 9 V batteries, one for the electronics and one for the IMU (Fig. 2.5(b,c)).

2.5.1 Program design

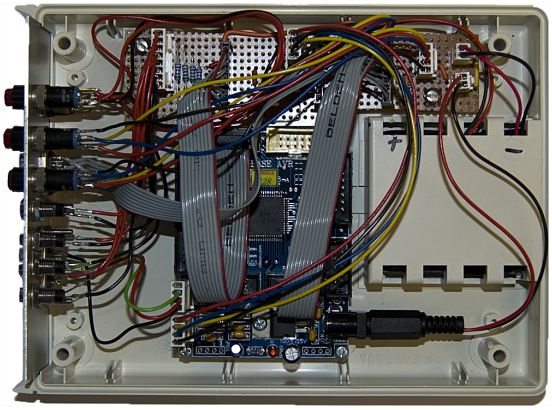
The controller software runs as shown in Fig. 2.6. Upon starting it initializes itself, the servos, and the IMU, and notifies the PC that it is now awake. The program then enters its main loop from which it does not return. The program can be switched between measurement and configuration modes by toggling a switch on the front panel. In configuration mode a program on the PC is used to set up the servo limits and home positions. It can also be used to view data from the IMU. In measurement mode the controller checks to see if the IMU connected, and if not, endeavours for a set number of attempts to connect. A failure to connect places the controller in a fail state. Once connected, the controller waits until the IMU has settled and records the IMU's settled angle values. It begins to process the IMU data. The IMU uses a built-in Kalman filter[4] to fuse the accelerometer, gyroscope and magnetic compass data to provide angle data for all three axes. The angle data are returned as Euler angles, and if the full raw data have been requested these are returned as well. The angle data are extracted from the IMU message, and used to calculate the servo movements required to stabilize the camera. Any received messages from the PC are acted upon and, finally, if raw data has been requested, this is sent to the PC.

During this main program, various interrupts are triggered. These interrupts are kept to a minimum in terms of number and time, as they momentarily stop the main process. The main pair of interrupts are the two serial interfaces, which interrupt to copy data received on the line to a buffer for later processing. The IMU sends data at a rate of 100 Hz, and the PC only when it requires data or desires the camera to move. The third interrupt is for a system clock which interrupts every millisecond. The clock is used to check for time-outs on the serial ports.

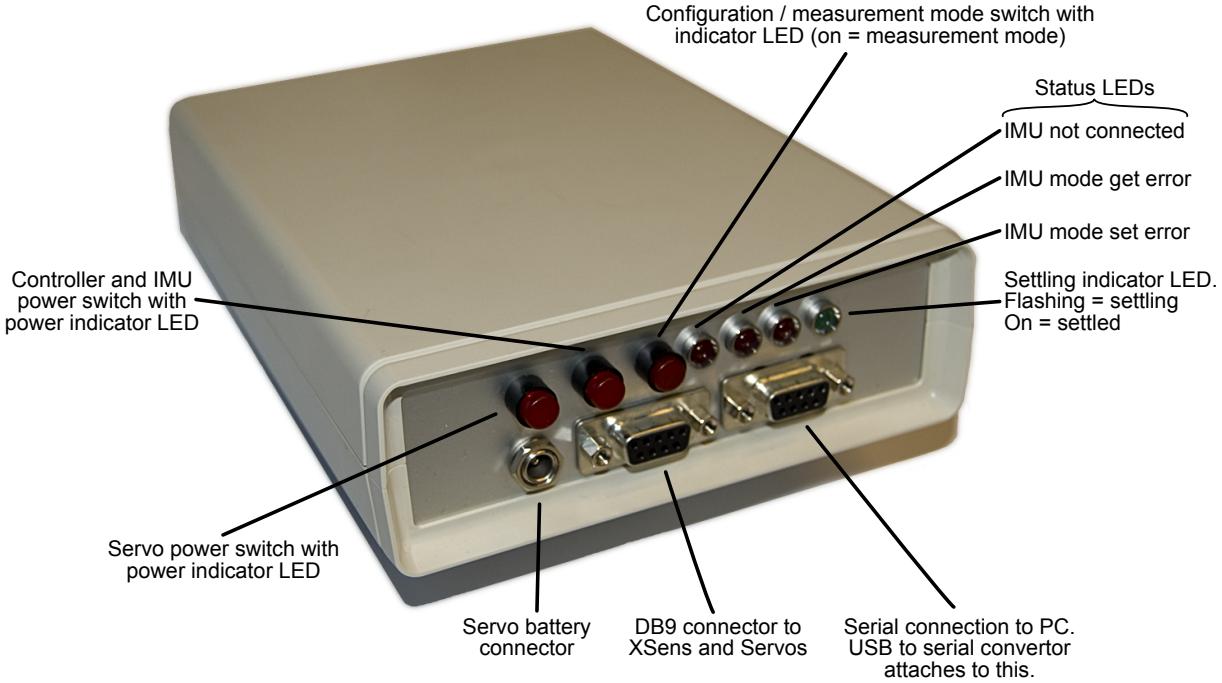
[4] see §3.3 for details on the Kalman filter



(a) ETT ATMega128 development board.



(b) Controller box internal.



(c) Controller box.

Figure 2.5: The assistive camera controller.

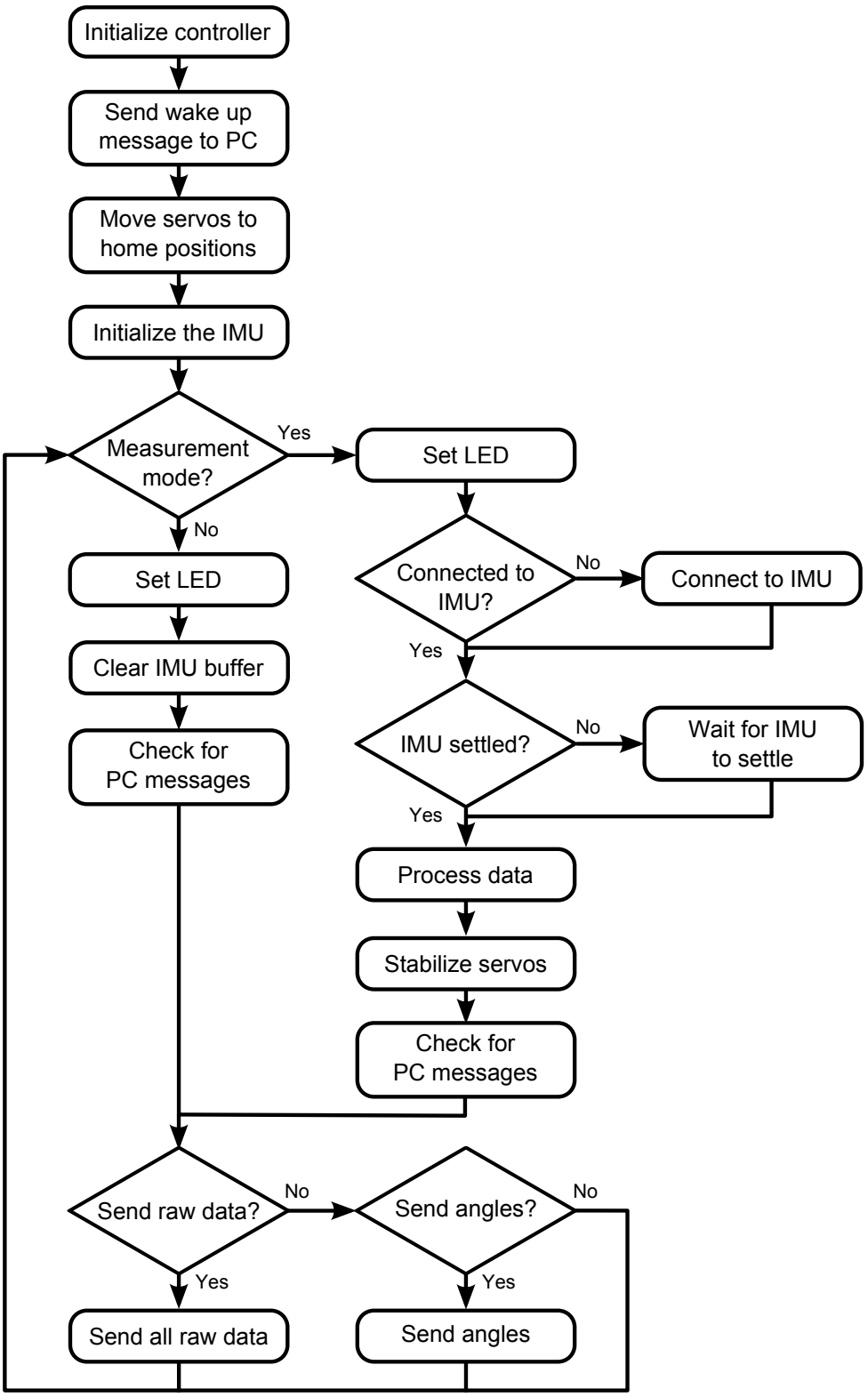


Figure 2.6: The assistive camera controller program flow.

Sender	Message	Data
Controller	Wake up message	No data
PC	Request for servo limits	Servo number
PC	Set new servo limits	Servo number, min, max and home positions
Controller	Servo data	Servo number, min, max and home positions
PC	Move servo	Servo number and position
Controller	Servo position	Servo number and position
PC	Enter config mode	No data
Controller	Acknowledge config mode	No data
PC	Enter measurement mode	No data
Controller	Acknowledge measurement mode	No data
PC	Request raw data	Type of data
Controller	Acknowledge raw data request	No data
Controller	Raw data message	The specified raw data

Table 2.1: Controller and PC messages.

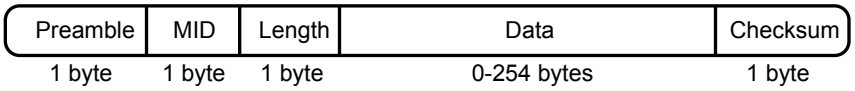


Figure 2.7: The controller to PC message structure for sending and receiving messages.

2.5.2 Message structure

A message structure allows different types of data to be sent to and received from the controller. The messages are summarized in Table 2.1, and fall into two main categories: those pertaining to servo control, and those pertaining to IMU data. The XSens IMU message structure was adopted for the controller allowing IMU data to be easily passed back to the PC with minimal change, and allowing considerable code reuse, which is essential on such a memory limited device.

All messages are formatted as shown in Fig. 2.7. The Preamble defines the start of a message; the MID is the message identifier; the Length specifies the length of the data packet; all of the data is held in the Data section; and the Checksum provides a method of validating the message integrity. All of the PC messages are acknowledged by the controller, and in the case where values are set, the values that the controller actually set are returned. This ensures that the PC knows what values are actually in use, and is especially important if the PC tries to set a value outside of the limits.

2.5.3 Stabilization

As discussed previously, the active camera has no roll axis. This affects the ability to stabilize, but is not a significant problem under normal use. Stabilization will be first looked at for a pan-tilt-roll camera, and then modified for a pan-tilt device.

To stabilize a pan-tilt-roll camera and enable it to fixate, angle data from the IMU is required, along with knowledge of the servo positions. The change in angle of each axis $\boldsymbol{\Delta} = (p_\Delta, t_\Delta, r_\Delta)^\top$ from its initial position $\boldsymbol{x}_0 = (p_0, t_0, r_0)$ is calculated using the current measurement $\boldsymbol{x}$ from the IMU as

$$\boldsymbol{\Delta} = \boldsymbol{x} - \boldsymbol{x}_0. \tag{2.1}$$

Each axis delta is checked for angle wrap, and corrected as needed. Then the new pan p, tilt t, and roll r positions can be calculated from

$$p = p_0 + p_\Delta \tag{2.2}$$

$$t = t_0 + t_\Delta \cos(p) + r_\Delta \sin(p) \tag{2.3}$$

$$r = r_0 + r_\Delta \cos(p) - t_\Delta \sin(p). \tag{2.4}$$

The pan calculation is performed only when fixating. For pure stabilization (*i.e.* maintaining orientation) the current pan value is used instead. When the camera is facing forwards, changes in the tilt and roll only affect the tilt and roll motors respectively, but when the camera is facing sideways (90°) the tilt now only affects the roll motor and vice versa. When at an angle in between, the tilt and roll changes affect both motors.

The approach to stabilization without a roll motor is the pragmatic one of using p and t as calculated, but ignoring r. The camera points in the correct direction, but the resulting image is of course not stabilized against rotations about the optic axis. Although inconvenient for viewing, all the image pixels and hence image features are available, and visual computation (say, for matching or recognition) is unaffected. Although not implemented, it would be straightforward to apply the cyclorotation in graphics hardware.

The effect of the stabilization routine is demonstrated in the following experiment. The assistive camera is horizontal and facing forwards when the user sits down. In Fig. 2.8 there is no stabilization, and it can be clearly seen that the AR elephant goes almost completely out of view as the user bends down. In Fig. 2.9 the stabilization is enabled and the AR elephant remains in view at all times.

2.6 Initial calibration

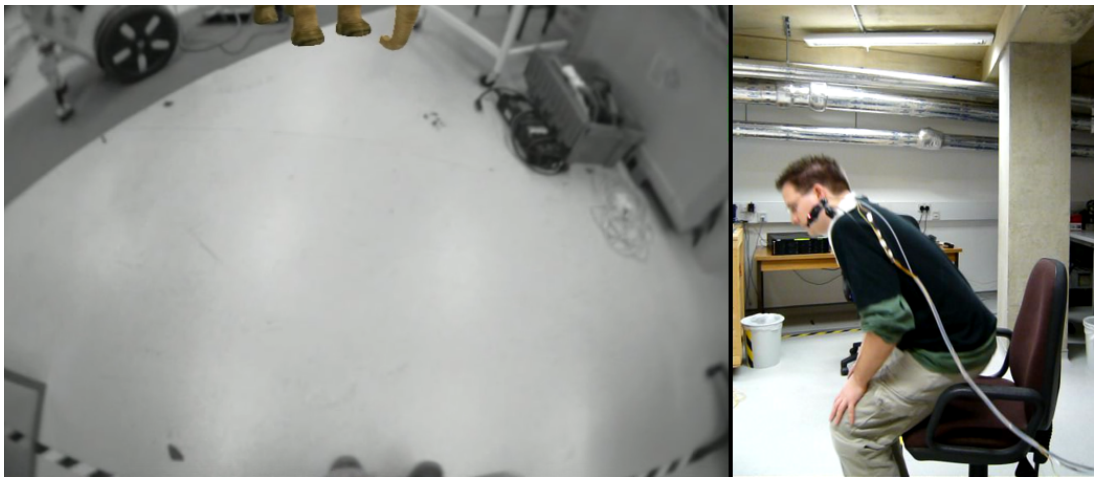
Before the system can be used initially, the servos need to be set to a default home position. This is a one off procedure, unless the position needs to be changed. To do this the servo controller is connected to the PC and the servos. An application is run on the PC to adjust the position of the servos. Once a desired location has been found, typically horizontal and facing forwards, the settings are saved to the controller's EEPROM. Now when the system is started the controller moves the motors to their home position.

Each time the system is turned on, the IMU's values need to be determined as a reference to the servos' home position. To do this the user must stand still while the IMU settles. To indicate this an LED on the front of the controller flashes when unstable, and then stays on once the system is ready. The process only takes a couple of seconds, provided the user stays still.

2.7 The complete wearable system

The wearable system is wired up as shown in Fig. 2.10. The system can be used with either one or both cameras, and it is a simple matter to add or remove the assistive camera assembly or the magic lens. Under typical operation the system can run for around 1.5 – 2 hours, with the display battery typically draining first, particularly during outdoor use which requires a brightened screen. Once the system has been powered up the assistive camera automatically begins stabilization and faces forwards. The assistive camera will remain facing forwards,

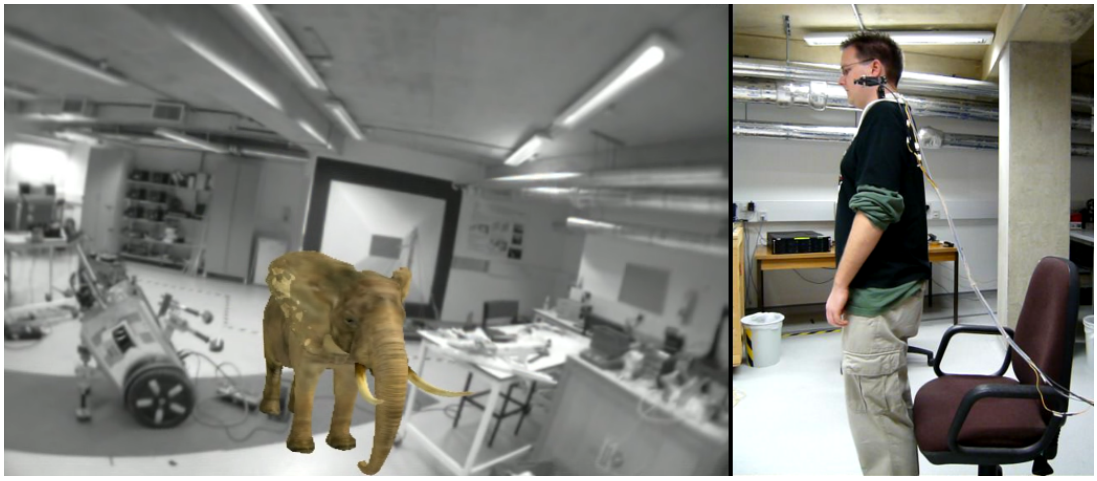
(a) AR in view.



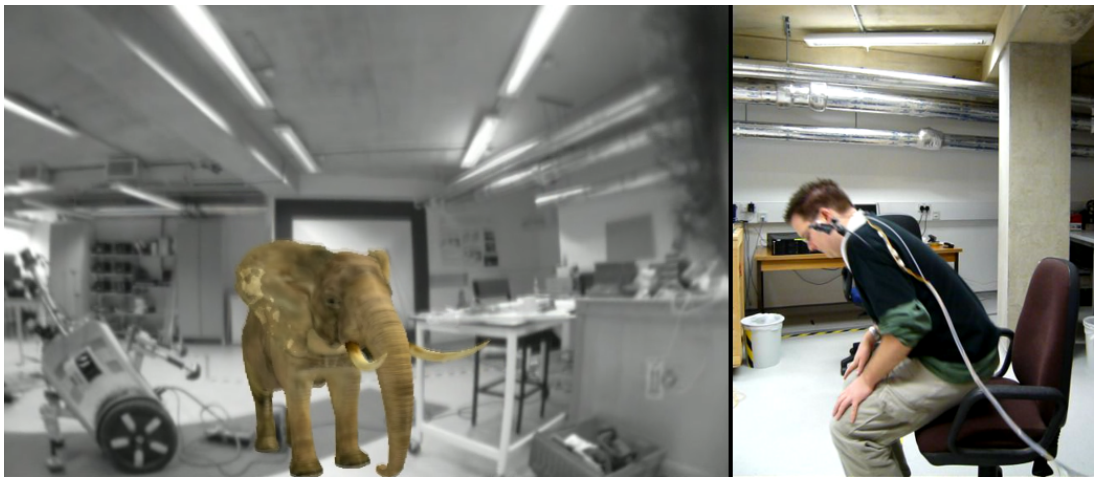
(b) AR goes out of view replaced by featureless floor.

(c) AR back in view.

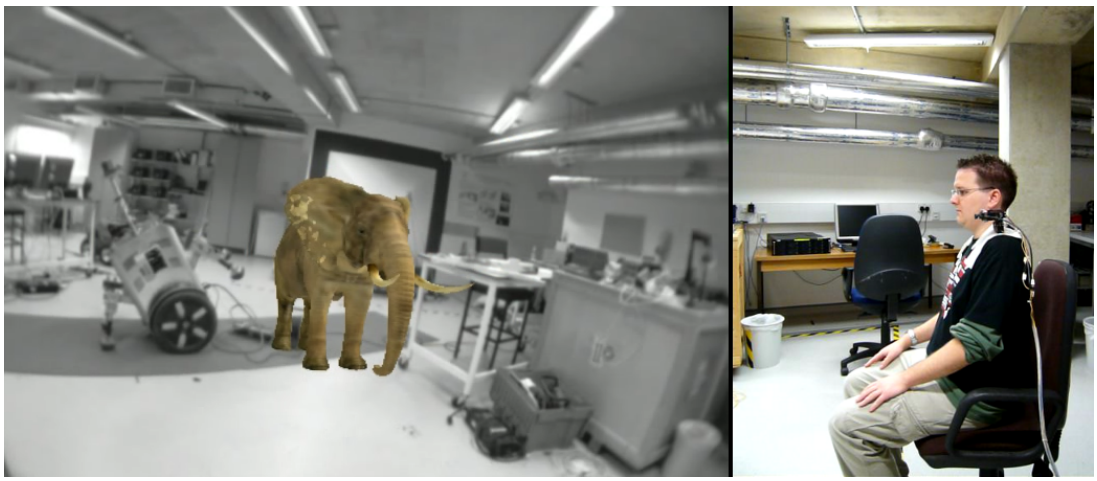
Figure 2.8: The assistive camera being used without stabilization (see video stabilization_not_active.avi).



(a) AR in view.



(b) AR remains in view.



(c) AR still in view.

Figure 2.9: The assistive camera being used with stabilization (see video stabilization_active.avi).

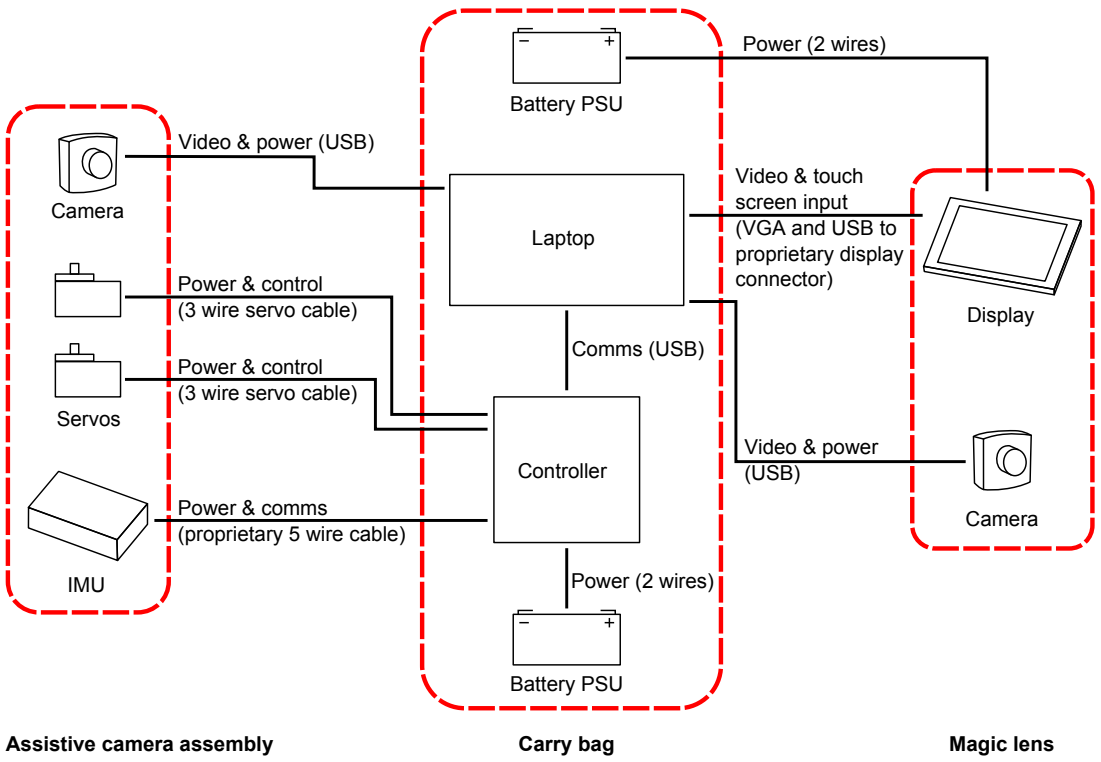


Figure 2.10: How the assistive wearable is wired together.

until it is instructed otherwise from the laptop via the controller. Fig. 2.11 shows the completed wearable in use.

2.8 Limitations and future work

The developed wearable robot provides a well featured and ergonomic vision platform for performing research. However, there are a variety of enhancements still required for the wearable robot before it can be considered complete, and as technology has progressed there are some elements that could now be updated.

The two main enhancements that are required are (i) a roll motor to enable full stabilization, and (ii) fixation and saccade control. Locating a small, flat, and light roll motor is the main hardware change required. The addition of the motor is a relatively simple task, as space has been left for it to be wired into the controller. The main software change is to fully enable the fixation and saccade control. The majority of the work required will be on the computer side

(a) Side view. **(b)** Over the shoulder view.

Figure 2.11: The wearable in use.

as part of the tracking and mapping software. The changes required on the controller will be to record the desired bearing and move the camera to point to it as best as possible, effectively setting a temporary new home position. During this operation the pan updates would be used. In addition to this, scan modes may need to be implemented to enable the camera to view all of the environment when the computer requires it.

Another software change that may be required is to decouple the camera's rotation from the user, when not under fixation. Currently, if the user turns too fast the camera image may blur or the camera may end up looking at an unmapped area becoming lost. To stop this the camera needs to be able to turn at a slower rate. This can be achieved by using the pan data to counteract turns, and then catch up at a slower speed. Further testing is required to see if the tracking system fails under rapid user rotation, or whether the system can keep up with such motions.

There are a variety of limitations with the wearable, with most stemming from available

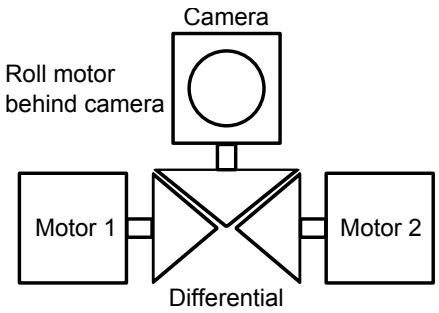


Figure 2.12: Differential based assistive camera design.

hardware and software. As always, a custom built solution designed from the ground up will always provide a better solution then one assembled from available hardware. Putting that aside, as it is only a practical solution for a commercial product, and focussing on COTS components, there a several improvements that can be made. First, the laptop should be replaced with the latest, multi-core laptop with an Nvidia GPU. Since purchasing the original laptop, there has been a (minor) shift away from Intel only GPUs, and smaller, lighter laptops now contain Nvidia GPUs. There has also been progression in GPUs themselves, with the latest chipsets supporting general purpose programming (GPGPUS) allowing applications to be processed on the GPU. This can provide substantial speed-up in execution, and will be briefly discussed in later chapters. Second, the batteries for the display and other electrical components should be swapped with higher density devices, such as lithium-ion (though this would possibly require additional circuitry to manage the batteries). The third improvement would be to find a physically smaller and lighter display. The final improvement would be to change the carry bag design for a backpack design, to aid in weight distribution for the user, and modified to enable the individual components to be swapped in and out with ease.

The most significant improvement that could be made given complete control over the design and build of all aspects would be to reduce the laptop to a mere processing unit. An alternative motor layout that would yield a compact, and enclosed design is a differential based design, as illustrated in Fig. 2.12. Carefully design of the motor controllers and the physical layout is required, as freely rotating motors would be required, along with encoders for monitoring the motor positions.

2.9 Conclusion

The wearable robot presented in this chapter provides the foundation of the research in the following chapters. The final design is portable, wearable, and runs for around two hours, allowing experiments of a significant length to be conducted. The separation of the assistive camera from the AR view allows the user to explore the world freely viewing AR constructs as they wish. The assistive camera is then free to view the world as the (human or computer) assistant requires. The data from both cameras and the IMU is provided to the portable computer for use in the tracking and mapping applications.

3

Integrating object recognition with single camera SLAM: (I) system design and review

This chapter describes the design of a system which combines an established method for appearance-based recognition with one for simultaneous localization and mapping using a single camera. As well as identifying planar objects, the objects are located relative to the camera in 3D by computing the image-to-database homography. The 3D positions of the objects are then used as additional measurements in the SLAM process, which routinely uses other point features to acquire and maintain a map of the surroundings, irrespective of whether objects are present or not. The chapter provides a detailed review of existing processes, and describes how to combine the information, which is collected at different rates, using a delayed-state update.

3.1 Introduction

Broadly, there is much that can be shared between an assistive vision system for a robotic camera to be worn or held by a human user, and a visual navigation system by an autonomous mobile vehicle. Two key requirements of both are (i) that they need to be able to localize themselves within the environment as they traverse it, and (ii) that they need to build and maintain an understanding of the environment, both at the level of basic geometric structure and at the higher level of distinct objects and their associated semantic labels.

An important difference, however, is that while a mobile robot needs fine-grained structural

information about the environment before it can move at all, a human being can usually be relied upon to avoid obstacles and undertake intelligent exploration. So for a wearable assistive system, rather more important than the answer to "where am I?" is the answer to "what is around me?". The answer must be provided in a manner that is both timely and readily interpreted by the wearer.

The aim of the work presented in this chapter and the next is to do just that, by combining appearance-based object recognition and localization with video-rate monocular simultaneous localization and mapping.

The problems of determining what is where relative to the camera, and of augmenting the visual "experience", have been subjects of sustained interest in machine vision from the late 1970s. Indeed, in controlled environments, such as spacecraft docking and remote handling, where detailed and well-carpentered models are available and strong prior expectations apply to appearance and motion, the problem is all but solved using top-down methods [46, 51, 84, 34]. Objects may be detected initially by seeking clusters of primitive geometrical features, and a coarse initial pose set from them. Then, in a typical top-down process, model features (usually edges) are projected into the image using the prior estimated pose, and the posterior estimate found by minimizing displacements between these projected features and those actually observed in the image. There may be no need to establish a world coordinate system, but if there is, objects known to be stationary can be used as a basis. More recent systems allow users to build the models on-line [14] or from captured image sequences [55] by hand-selecting features such as edges or corners in a image that belong to the desired object. Whatever the method of building, a significant amount of effort and time is required to produce geometric models. This leaves a difficult issue to resolve, that of second-guessing which geometric elements of an object will actually provide reliable image features to track.

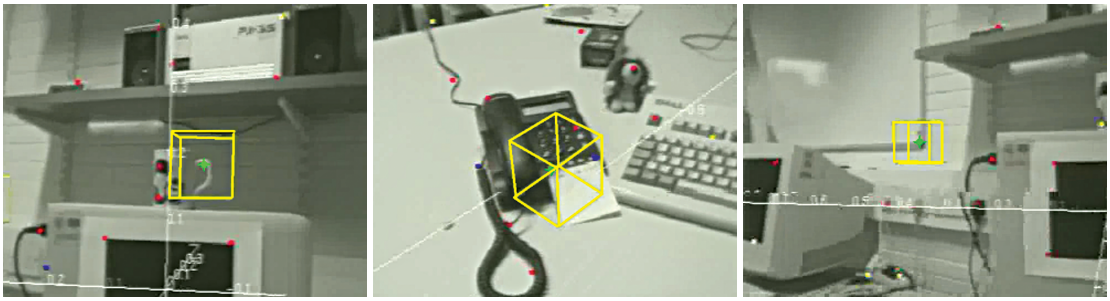
By contrast, rather than adopting a purely geometric approach, the system developed here uses the appearance of features to recognize objects. Object detection and recognition using

learned appearance models has been a story of remarkable success in computer vision in recent years. It has been made possible by the observation that the variety of feature appearances is limited, and that the co-occurrence of features is indicative not only of object identity, but also of object-class identity [85, 40, 109].

It is also argued that to establish a more persistent sense of AR, the camera should not rely on its viewing objects alone, but should also grow and maintain an independent representation of the environment, and establish both its position and the position of objects within that environment. Those working in the area of wearable systems have often used GPS for localization: examples (in the context of activity analysis) are found in [6, 50, 79]. Visual sensing in established world frames is also common: in [113] external cameras provide a fix on the wearer's location, and in [69] and [44] inertial sensing combined with visual measurement of pre-mapped fiducial targets does the same. But if measurement in established frames of reference is unimportant (at least at the outset), a more general approach is to use techniques drawn from either structure from motion (SfM) or simultaneous localization and mapping (SLAM).

Here, EKF-based monoSLAM is used for localization and map-building, as developed in the Active Vision Laboratory by Davison and colleagues [26] [28]. This method (as do other monocular formulations, such as Eade and Drummond's FastSLAM method [36], and the Unscented Kalman filter method of Chekhlov *et al.* [18]) builds a time-evolving map of 3D features and their uncertainties, locating the features and the camera with reference to some coordinate frame. The methods are designed to be used with hand held cameras, and to operate at video-rate, and they allow the placing of AR elements within the map (*e.g.*, [92] [121] [17] [72]).

Indeed, monoSLAM has been used previously with wearable cameras in this laboratory. In his thesis, Mayol used it to build a 3D map around the user, and allowed the user to designate points of interest in the environment for the camera to fixate on [96]. Fig. 3.1 shows examples extracted from a video sequence, where the wearable has been told to fixate on points on a cup, telephone, and whiteboard. However, these 3D points of interest are nothing more than



(a) Cup. **(b)** Telephone. **(c)** Whiteboard.

Figure 3.1: Hand selected points on objects of interest using Mayol's system.

that. The object associations were entered by the user, and not generated automatically. There was no memory of objects, and so the points, and their associations, were ephemeral, with subsequent runs having no knowledge of them.

Here, the aim is to achieve association automatically, by combining recognition with SLAM. The system that is developed can detect and recognize planar objects using appearance models, then localize the objects in 3D and add them to a map of the environment that the wearable robot is building as the user explores. The recognized objects can then be used in augmented reality applications, where for example, they are labelled or used in an interactive tutorial. The detected objects also provide an additional measurement source for the mapping system, improving the accuracy of the maps.

3.1.1 System and chapter overview

The outline of the system to be developed is shown in Fig. 3.2. Its parallel architecture allows object recognition and localization to occur independently of video-rate 3D mapping and camera tracking. The later sections of this chapter discuss the functioning of its constituent parts and their interconnection, and the following chapter describes its implementation and evaluation.

In Fig. 3.2, the lower processing path handles the frame-to-frame map building and camera tracking. As mentioned above, it is based on monoSLAM and is described in detail in

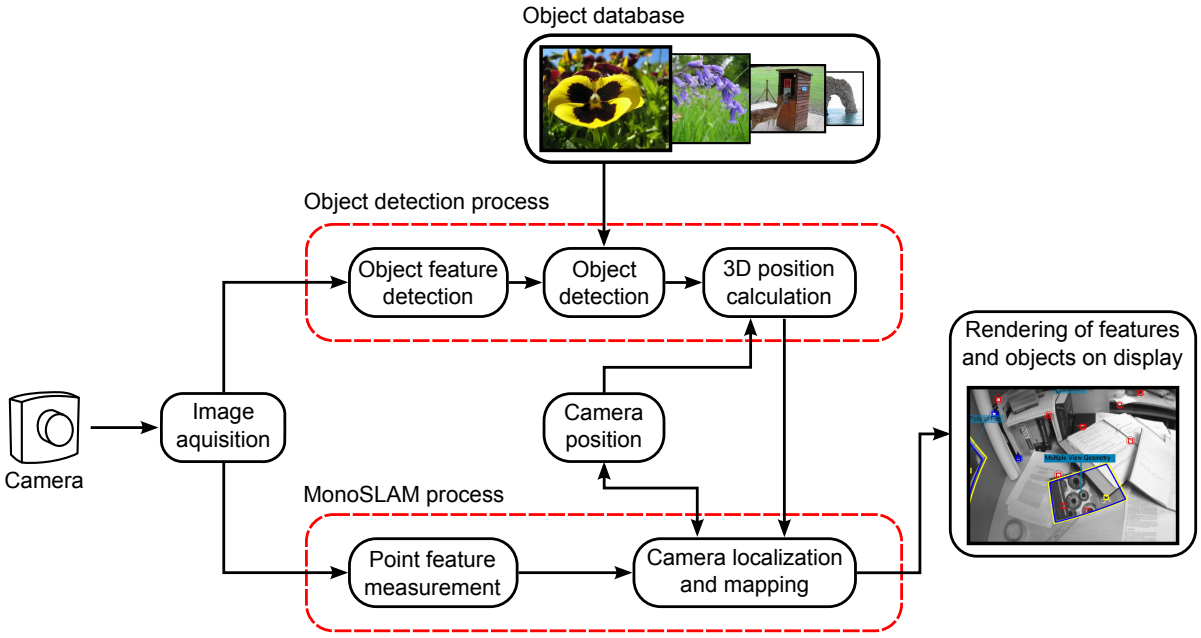


Figure 3.2: An overview of the system flow for detecting objects and integrating their measurements into the monoSLAM system.

Section 3.3. The monoSLAM thread uses the image to detect and measure the location of previously detected features, and adds new features as required. These measurements, along with the previous camera and feature positions, are used to estimate the new camera position.

The upper processing path handles the object recognition and localization. It is based upon Lowe's approach, and is discussed in Section 3.4. It does not run on every frame, but only as often as time permits, always deferring to the more pressing computing needs of SLAM. Once features are extracted, they are compared with those held in a database to find any known objects present in the scene.

The novel work in this chapter lies in the interaction of these processes, and in Section 3.5, the 3D pose of recognized objects relative to the camera is calculated, and these poses are then passed to monoSLAM as measurements. In turn monoSLAM adds the objects to the map, using them as additional landmarks and helping to locate the camera.

Once objects have been identified and located in the map, the knowledge of the camera position and orientation relative to the map allows them to be exploited in augmented reality

applications. For example, their outline can be viewed on a display overlaid on the camera image, and object specific information added. The rendering of augmented reality overlays is described in Section 3.6.

Section 3.7 describes a relocation module which proves essential for robust operation of monoSLAM, and Section 3.8 looks at how the developed system can be used with non planar objects. Matters of implementation, the experimental evaluation of the system, are discussed in the next chapter. However, as clear from Fig. 3.2, the first step in both the localization and recognition paths is that of feature detection and description. Section 3.2 addresses this area, and explains why different methods are used in the two paths.

3.2 Feature detection and description for localization and recognition

After an image has been acquired, the first substantive step in both localization and recognition paths is that of feature detection and description. At their roots, both SLAM and appearance-based recognition involve determining the similarities between one image with another. On first consideration, it might seem ideal therefore to use the same method of feature detection and description for both. However, localization and mapping involves matching between the current image and one taken by the same camera only a few tens of milliseconds earlier; whereas recognition involves the current image and one taken a considerable time earlier, most likely by a different camera, from a different viewpoint at a different range, and under different lighting conditions. Matching across such large spans of time *etc*. requires much greater care to be taken with feature description than when matching from frame-to-frame, and, in turn, computational efficiency and effectiveness demand that different methods be used. On the one hand, localization demands a method that is computationally modest and hence descriptively modest, and, on the other, object recognition calls for a method that is descriptively rich and hence computationally costly.

And so, in the core work of this chapter, two detectors and descriptors with different costs and descriptive powers are used. In chapter 5 yet another method is required, one that is computationally very cheap, and consequently weakly descriptive. It is convenient to review the operations of all three here.

3.2.1 Computationally fast, descriptively weak features

One of the general approaches to feature detection, and in particular that of corner detection, is to define how the pixels underlying a feature should appear, and then scan the image for locations that come close to that ideal. This idea was proposed by Smith and Brady [139] in the SUSAN edge and corner detector. This used a circular mask and compared the pixels in the mask to the central one, the nucleus, and, if similar, the "USAN" count was incremented. The USAN is the area of the mask with the same brightness as the nucleus. This leads to 1D (edge) features having low USAN values, and 2D (corners) even lower ones, allowing the features to be discriminated. The principal advantage is that of speed: convolution is avoided, and detection is some three times faster than Harris and Shi-Tomasi methods used for SLAM here (see later).

The aptly-named FAST detector of Rosten and Drummond [124, 125] makes even more sparing use of pixels. A ring of pixels at a certain radius (typically 16) around each pixel is considered, and the location flagged as a feature if at least n (typically $9 \leq n \leq 12$) contiguous pixels are either all brighter, or all darker, than the centre pixel. The key advantage is again speed: convolution is avoided again, but now many pixels can be rejected as putative features with only two or three comparisons. It is around 20 times faster than Harris, and is also invariant to in-plane rotation. The main limitations of FAST are that it fails under motion blur, and that it detects only at a single scale. However, FAST is so cheap that it provides a useful pre-filter to select sites at which to apply more expensive methods, such as the Shi-Tomasi detector (*e.g.* Smith *et al.* [136] and Williams *et al.* [156, 155]).

In this thesis, FAST is used in chapter 5, where several thousand features per image are recovered for localization and mapping using structure from motion. However, the method of localization and mapping used in this chapter, monoSLAM, can only maintain 30 Hz operation using around 100 features. Moreover, feature detection is not the computational bottleneck, and avoiding mis-matching is of great concern. It therefore makes sense here to use a somewhat more robust and descriptive detector from the outset, drawn from the class discussed next.

3.2.2 Computationally modest, descriptively modest features

Perhaps the most widely used corner detector in geometrical applications has been that of Harris and Stephens [52]. The Harris detector approximates the auto-correlation surface of an $(2n+1)^2$ image patch centred on (x, y) by evaluating a Gaussian-weighted sum-of-squared differences between it and a shifted patch as

$$\begin{aligned} E_{xy} &= \sum_{u=-n}^{+n}\sum_{v=-n}^{+n} w(u,v)(I(x+u,y+v)-I(u,v))^2 \\ &= \sum_{u=-n}^{+n}\sum_{v=-n}^{+n} w(u,v)(xI_x+yI_y+O(x^2,y^2))^2 \\ &\approx \begin{pmatrix} x & y \end{pmatrix} \mathtt{M} \begin{pmatrix} x \\ y \end{pmatrix} \end{aligned} \tag{3.1}$$

where I_x denotes $(\partial I/\partial x)$, and so on, w is a Gaussian mask, and where the symmetric matrix's elements are $m_{11} = w * I_x^2$, $m_{22} = w * I_y^2$, $m_{12} = m_{21} = w * (I_x I_y)$. The eigenvalues $\lambda_{1,2}$ of $\mathtt{M}$ are proportional to the principal curvatures of the auto-correlation, and a corner is declared where both are large. In practice, Harris proposed an edge response function $R = \lambda_1\lambda_2 - \kappa(\lambda_1+\lambda_2)^2$, where R is positive for a corner and negative for an edge, avoiding explicit determination of the eigenvalues: their product is $\det(\mathtt{M})$ and sum is $\mathrm{trace}(\mathtt{M})$. While the location is good, and the detector is rotationally invariant (although pixelation issues always confound this to an extent), the descriptor, *i.e.* R, is weak as it is just a single value, and one derived non-linearly. In most applications, Harris corners are augmented for matching with a descriptor comprising the pixel patch around the corner's location. It is a relatively slow method, requiring convolutions with

a Gaussian smoothing mask and derivative masks, but the principal disadvantage is that it is not scale invariant.

While this makes it unsuitable for object detection (because objects may appear anywhere in a scene and hence at different scale in the image), the drawback is of little importance during frame-to-frame tracking of features, where there are usually tight search bounds available from prediction. Indeed, it is the detector used in monoSLAM, though with one small change. Strictly, the detector used is that of Shi and Tomasi [132]. The only difference of note is that Shi-Tomasi computes the eigenvalues explicitly, which may be of value as it has been found that $\min(\lambda_1, \lambda_2)$ is a more stable descriptor than R under affine deformations of the image. However, as matching here always relies on the accompanying image patch (typically, 11×11 pixels), there is no functional difference between it and Harris. The continued use of Shi-Tomasi owes something to inertia: it was used by Davison and Murray for stereo EKF-SLAM in [27], and its inclusion in the monoSLAM system followed from that.

3.2.3 Computationally expensive, descriptively strong features

It has been noted that Harris (and hence Shi-Tomasi) is particularly sensitive to changes in scale. Recent years have seen much investment in augmenting feature detectors with descriptors that provide substantial invariance to changes not just in scale, but also in viewpoint and illumination.

Mikolajczyk and Schmid [97] recently compared the ability of ten different descriptors to cope with rotation (up to 50°), scale change (by factors up to 2.5), viewpoint change, image blur and illumination (effected by changing focus and aperture), and JPEG compression. A strong performance in all save compression and blur is essential to object detection with a wearable camera running at frame rate, and a good result for image blur is desirable. They tested using both structured scenes (containing distinctive boundaries from buildings) and textured scenes. Importantly, to remove bias in the detection stage, they tested descriptors at locations derived

using a variety of detectors. Their evaluation criterion was based on the number of correct and incorrect matches for between an image pair.

Remarkably, in all of the tests, the descriptors based on and including the scale invariant feature transform (SIFT) proposed by Lowe [85] performed best. Mikolajczyk and Schmid's modified SIFT descriptor, called GLOH (gradient location and orientation histogram), obtained the best results in most tests, always closely followed by the original SIFT descriptor. SIFT also performed well on both textured and structured scenes.

3.2.4 Scale invariant feature transform

Because SIFT and its derivatives performed so well (and because details of its performance were better documented than GLOH), it was selected for use as the object detector in this chapter, and it is described in more detail now. The reader familiar with its operation might omit the next sub-sections.

3.2.4.1 SIFT detector

The detector in SIFT uses the Difference of Gaussians (DoG) approximation to the Laplacian of Gaussians (LoG). In the late 1970s, and on the basis of a connection with human visual response, Marr and Hildreth proposed using zero-crossings of the LoG signal as an edge detector [90]. More relevantly here, Lindeberg [81] detected image blobs at extrema in both scale and space of the "scale-normalized" LoG signal

$$L_{\text{norm}} = \sigma^2 \nabla^2 L = \sigma^2 \nabla^2 (G * I) \,, \tag{3.2}$$

where I is the image, and G is the 2D spatial Gaussian with scale σ given in Eq. 3.3 below. Computation is much reduced by approximating L_{norm} using a Difference of Gaussians. The relationship is shown as follows. Given a 2D Gaussian G, its derivative with respect to scale is found:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{3.3}$$

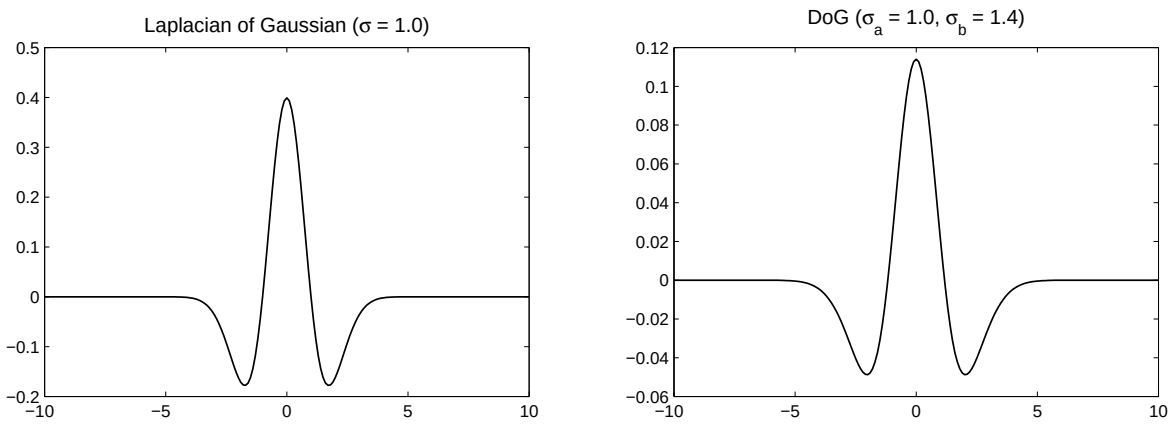


Figure 3.3: The difference of two Gaussians with $\sigma = 1.0$ and $\sigma = 1.4$, compared with the Laplacian of Gaussian.

$$\frac{\partial G}{\partial \sigma} = \frac{1}{2\pi\sigma^3}\left(-\frac{x^2+y^2}{\sigma^2} - 2\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) . \tag{3.4}$$

The Laplacian of the Gaussian turns out to be proportional to the derivative, as

$$\nabla^2 G = \frac{1}{2\pi\sigma^4}\left(-\frac{x^2+y^2}{\sigma^2} - 2\right)\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) = \frac{1}{\sigma}\frac{\partial G}{\partial \sigma}. \tag{3.5}$$

The interest is not in G, but in the smoothed image $L = (G * I)$. However the linearity of differentiation and convolution operators tell us immediately that

$$\nabla^2 L = \frac{1}{\sigma}\frac{\partial L}{\partial \sigma} , \tag{3.6}$$

and using a forward difference approximation

$$\sigma\nabla^2 L = \frac{\partial L}{\partial \sigma} \approx \frac{L(x,y,k\sigma) - L(x,y,\sigma)}{k\sigma - \sigma} . \tag{3.7}$$

Thus the difference of Gaussians is proportional to the desired normalized LoG signal

$$\begin{aligned} D(x,y,k,\sigma) &= L(x,y,k\sigma) - L(x,y,\sigma) \\ &\approx (k-1)\,\sigma^2\,\nabla^2 L . \end{aligned} \tag{3.8}$$

Lowe notes that the approximation has negligible effect on the stability of detection and localization of extrema, even for substantial differences in scale. Fig. 3.3 compares the Laplacian of a 1D Gaussian with its approximation.

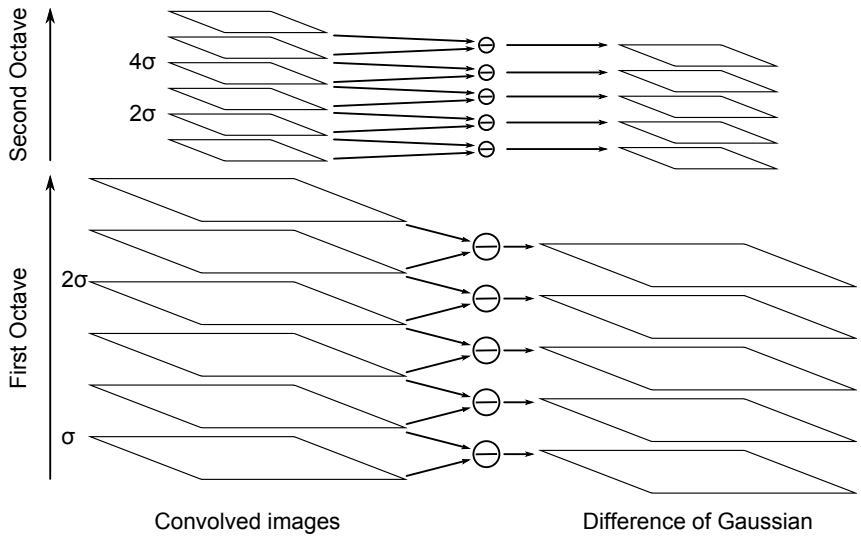


Figure 3.4: The Gaussian pyramid on the left is created by successive convolutions with a Gaussian. At each octave (doubling of σ), the image is halved in size. The difference of Gaussians pyramid is shown on the right.

Adjusting the Gaussian's scale moves the convolved image through scale-space. In SIFT a DoG pyramid is a set of discrete samples of the DoG response through scale space, and is generated from a Gaussian pyramid in the following way. A value of $k = 2^{1/3}$ is used in Eq. 3.8, giving three layers per octave in the Gaussian pyramid. Subtraction yields the Difference of Gaussians, as shown in Fig. 3.4. However, because it is efficient to halve the image's linear dimension at each octave, the practical algorithm requires five DoG images per octave to allow search for extrema (three, plus two "bookends" for forward and backward differencing), which in turn requires six Gaussian images. Images from a particular octave of the Gaussian and DoG pyramids are shown in Fig. 3.5. Lowe does not use the input image for the first octave, instead the image is doubled, and this is then used for the first octave. This is done because it results in a greater number stable features.

The search for extrema compares each pixel in a DoG image with its spatial and scale-space neighbours. Once the extrema are found, they are filtered by contrast and edge-response, and located to sub-pixel accuracy by fitting a 3D quadratic function [12]. The extrema are then filtered for contrast to ensure that the difference between them and the surrounding pixels is significant. Finally they are filtered on their edge response: features on edges are unstable and

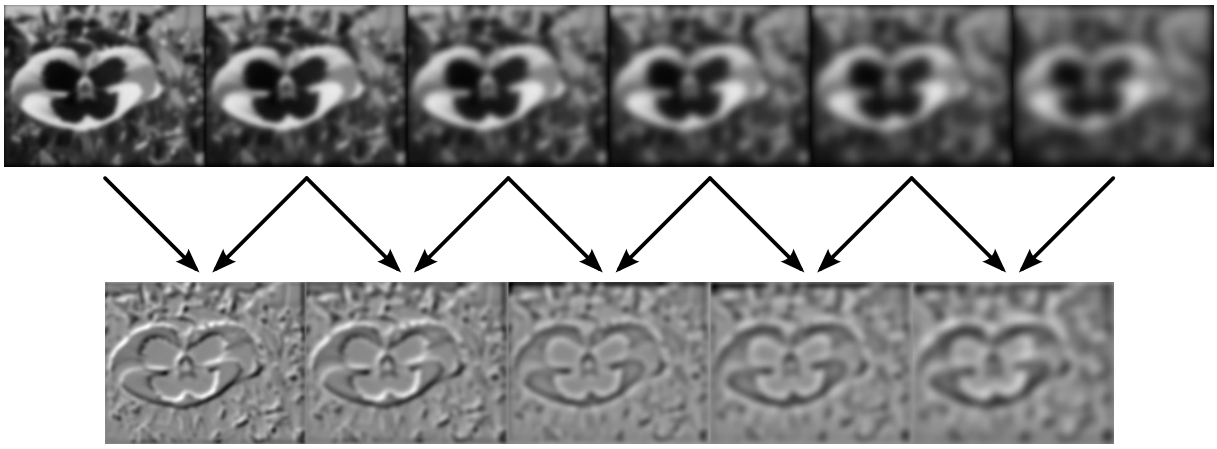


Figure 3.5: An example of the images required to compute one octave. The top row are the six images in the Gaussian pyramid, and below are the five DoG signals, three covering the octave and two at each end to allow forward and backward checking for extrema.

hard to relocalize, as all positions along an edge appear similar.

3.2.4.2 SIFT descriptor

For each extremum or "keypoint" detected, an orientation is assigned by forming a histogram of the orientations of the pixels surrounding the extremum at the extremum's scale, and selecting the dominant orientation. A pixel's orientation is found by calculating the angle of the gradient across the pixel, and its magnitude is the magnitude of the gradient.

The descriptor, illustrated in Fig. 3.6, is formed from the orientations and magnitudes of the pixels around the keypoint by sampling over a 16×16 array aligned with the keypoint's orientation. This array is broken down into 4×4 subregions, and within each subregion the orientations are used to fill a histogram of eight bins, weighted by the magnitudes of the pixels and a Gaussian to give pixels nearer the extremum greater weight than those further away. The combination of these orientation histograms from these subregions gives a 128-valued vector. Lastly, the vector is normalised to provide invariance to illumination change. An illumination change is assumed to multiply all gradients by a constant factor.

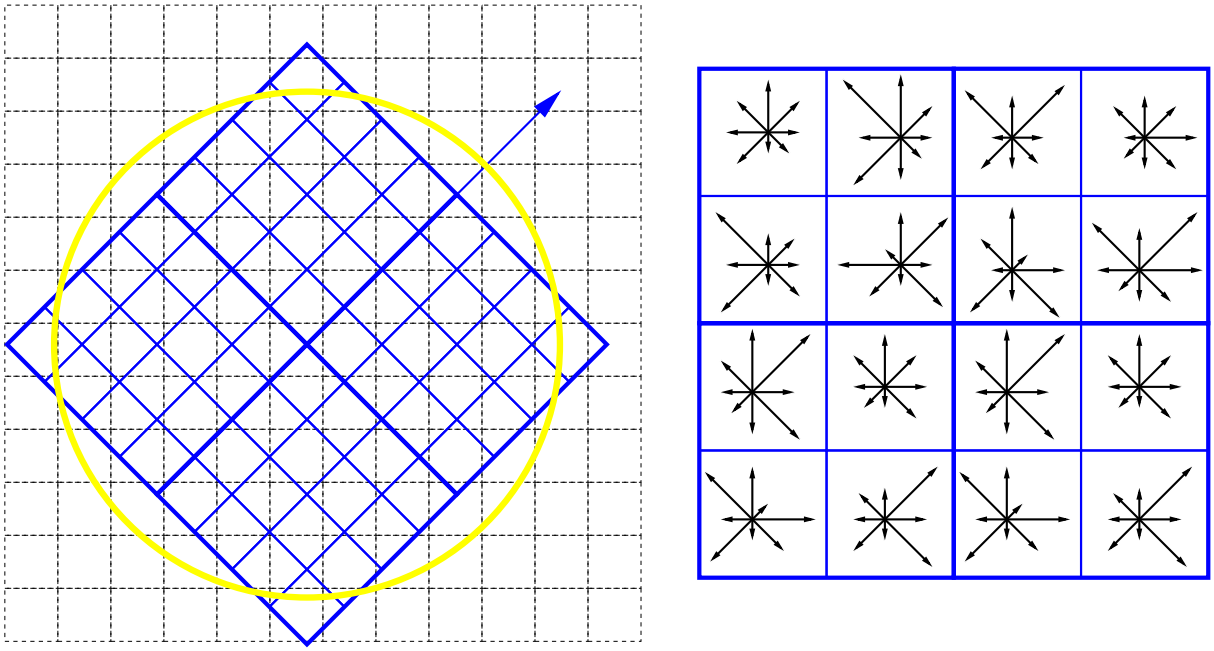


Figure 3.6: The left image shows an 8×8 array oriented around the keypoint overlaid on the image pixels. (In reality, a 16×16 array is used.) The circle represents the Gaussian that is applied and the arrow is the orientation of the keypoint. The image on the right shows the full descriptor with the orientations of the pixels for the 4×4 subregions.

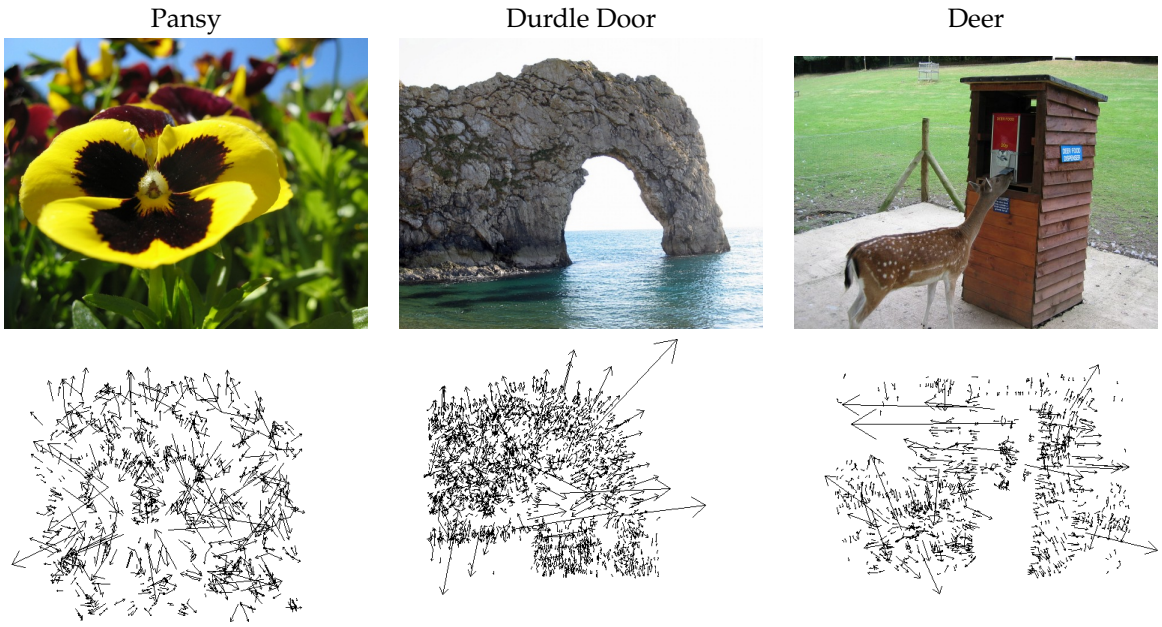


Figure 3.7: Example outputs from the SIFT detector. 940 keypoints were found on the Pansy image, 3 026 on Durdle Door, and 1 215 on the deer image. The base of the arrows are the feature locations on the original image. The orientation of the arrow is the orientation of the descriptor. The length of the arrow represents the scale the feature was found on, with longer arrows coming from higher scales.

3.2.4.3 Timing

Examples of the output from SIFT are given in Fig. 3.7. SIFT detected 940 features on the 600×480 Pansy image, and took some 2 000 ms to compute. With the initial image doubling disabled the processing still takes around 1 000 ms. This is far too long for 30 Hz frame-rate operations. Hence the need for faster feature detectors for the monoSLAM tracking, and an independent object recognition process.

3.2.4.4 Limitations

SIFT performs well in many situations, however, there are some where it either does not work or gives the incorrect answer. SIFT requires textured surfaces to find features on, making many man-made objects and structures essentially invisible to it. It is also not very discriminating between objects that look similar, especially if there is only a colour difference, as it does not use colour information. A good example of this type of failure is with the Multiple View Geometry book by Hartley and Zisserman. The first edition had an orange background, and a slightly squatter shape compared to the blue second edition. However, SIFT only finds features on the image of the eyes and the title, which are the same between versions except slight variations in size and location. This causes one to be recognised as the other. Depending on the application a different feature detector may be required, or a combination of different types. Here however, the objects of interest are generally well textured and distinctive, although as will be seen in the experiments in Chapter 4 not always enough for SIFT to detect them.

3.2.5 A postscript

After the work in this chapter was completed, several interesting developments in feature detection have occurred. A notable improvement in the computation speed of SIFT has been achieved by parallelization on GPUs (graphics processing units), with 640×480 images processed at around 27 Hz [134]. The SURF descriptor (Speeded Up Robust Features) [8] has been

proposed as a faster competitor to SIFT, and it is also available in a GPU version, allowing it to run on 640×480 images at 100 Hz [19].

It should be noted that these GPU implementations require GPGPU graphics chipsets, ones capable of general purpose computation. At present these are only available for desktop computers, not for genuinely portable computers, placing these accelerated versions of SIFT and SURF just beyond the scope of wearable computing.

3.3 Simultaneous localization and mapping

This section details the monoSLAM process (the lower half of the system diagram in Fig. 3.2), which is concerned with map-building and camera localization.

As noted in the introduction, general, passive, approaches to locating a wearable camera are adopted in this thesis, using techniques drawn from structure from motion and simultaneous localization and mapping. The emphasis in SfM has tended to be placed on obtaining dense structure for model generation. Image feature tracks are built up incrementally over a sequence, and, from initial estimates given by multi-focal geometry, the camera poses and 3D scene positions are optimized in a batch using off-line bundle adjustment (*e.g.* [116, 115, 43, 104, 1]). Optimality requires all observations that project from a particular scene point to be in proper correspondence, which is difficult to assure using a method which, in its basic form, does not supply frame-by-frame estimates of structure and camera position. By contrast, SLAM [138, 75, 144] places emphasis on the continual recovery of the state of the camera and scene structure, and on maintaining information about the correlation between state members. This makes re-matching after neglect easier, and also allows uncertainty to be reduced immediately throughout the camera and map state vector upon loop closure, again assisting feature matching.

Recently, SLAM's clear advantage over SfM for real-time operation has been significantly eroded, if not eliminated, first by the recovery of high quality visual odometry which allows

tracking of the camera pose between more widely-spaced keyframes, and by the ability of modern processors to run bundle adjustments on this reduced number of keyframes at a pace that keeps up with incoming video [106, 102, 65].

However, in this chapter, recovery of location is achieved by the monocular SLAM method devised in 2003 in the Active Vision Laboratory by Davison. Like the original methods based on sonar sensing by Smith and Cheeseman, and Leonard and Durrant-Whyte [137] [74] [75], it is based on the extended Kalman filter (EKF), and uses much of the formalism developed for stereo visual SLAM in [27]. The $O(n^2)$ computational complexity of EKF-SLAM (quadratic in the map size n, rather than cubic, provided the environment is modelled as stationary) has made finding other methods to handle large-scale maps a major concern (*e.g.* [67], [76], [145]).

Of the methods with lower "headline" complexity, the Information filter has been applied to SLAM by Thrun *et al.* [145] [144] and Walter *et al.* [153]. The Information filter is an inverse formulation of the Kalman filter which recursively updates the inverse covariance, or information matrix, $\mathtt{P}^{-1}$ and the information state $\mathtt{P}^{-1}\boldsymbol{\chi}$ rather than the covariance $\mathtt{P}$ and state $\boldsymbol{\chi}$. In SLAM, the information matrix tends to be highly sparse and mostly diagonal, because observations tend to be highly self-correlated. The update complexity is close to $O(n)$, allowing larger maps to be handled than with the EKF. However, the map is not immediately interpretable as the information matrix needs to be inverted to recover the covariance matrix. This is computationally expensive and, unfortunately, makes it unworkable in monoSLAM as the process of active search for features requires the covariance matrix to be available at each frame for projection into the image space.

FastSLAM [99, 146] is based on a particle filter. A set of weighted particles, where each particle is a distinct location, represents the pdf describing the robot's location. Each particle contains its own Kalman filter and its own map which is updated at each measurement. The advantage is that the map positions are uncorrelated, allowing the updates to be performed faster than the EKF at around $O(m\log(n))$, where m is the number of particles and n is the

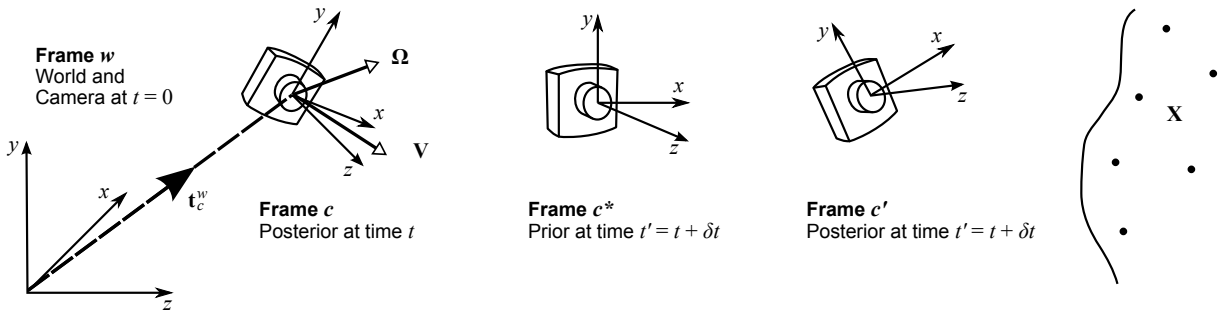


Figure 3.8: World and camera frames of reference.

number of features in the map. Its disadvantages are that, typically, a large number of particles are required, each of which requires a complete update, and that storage costs are large.

Notwithstanding its quadratic complexity, it has been argued by Davison *et al.* that the EKF remains well-suited to vision using sparse landmark points [28]. Mayol *et al.* [93] have suggested that this is particularly the case for vision for wearables. They point out, first, that sparseness of representation is no hindrance to navigation because, as noted earlier, one can usually rely on the wearer to move around without mishap; and secondly, that the need to impose a limit on the growth of the feature map in order to maintain video-rate performance is quite compatible with the notion of a local workspace of fixed volume around the wearer. It is the method used here.

3.3.1 Monocular SLAM

The problem to be solved in monoSLAM is illustrated in Fig. 3.8. A single camera with *a priori* unknown pose moves with unknown rectilinear and angular velocities in front of scene points $\boldsymbol{X}$ which are also unknown *a priori*. Although apparently straightforward, estimating the unknowns is challenging. First, because neither single nor multiple views of a single point yield depth when the motion is unknown, information needs to be combined from detected features collectively and over time. Second, the evolution of the state and the measurement process is non-linear.

3.3.1.1 State evolution

The extended Kalman filter is used to estimate the state $\boldsymbol{\chi}$ and covariance $\mathtt{P}$ of a discrete-time non-linear dynamical process, whose evolution is modelled from time step k to $k+1$ as

$$\boldsymbol{\chi}_{k+1} = \boldsymbol{f}(\boldsymbol{\chi}_k, \boldsymbol{u}_k, \boldsymbol{v}_k)\,, \tag{3.9}$$

using observations which are related to the state through a vector of non-linear functions

$$\boldsymbol{x}_k = \boldsymbol{h}(\boldsymbol{\chi}_k, \boldsymbol{n}_k)\,. \tag{3.10}$$

The control input $\boldsymbol{u}_k$ is actually zero here, and $\boldsymbol{v}_k$ and $\boldsymbol{n}_k$ are process and measurement noise, respectively. The state vector

$$\boldsymbol{\chi}_k = \left[\boldsymbol{c}_t^\top, \boldsymbol{X}_1^\top, \ldots, \boldsymbol{X}_n^\top\right]_k^\top, \tag{3.11}$$

comprises two parts. The $\boldsymbol{X}_i$ are the 3D locations of map features, which are fixed in the world, so that the evolution of the posterior to the prior at the next time step is idempotent and noiseless

$$\left\{\boldsymbol{X}^i\right\}_{k+1|k} = \left\{\boldsymbol{X}^i\right\}_{k|k}\,. \tag{3.12}$$

Element $\boldsymbol{c}_t$ consists of the time-dependent camera pose and velocity: the orientation is represented by unit quaternion q^{wc}, the position is that of the camera's optic centre in the world frame, $\boldsymbol{t}_c^w$, and the instantaneous angular and rectilinear velocities vectors $\boldsymbol{\Omega}^c$ and $\boldsymbol{V}^w$ are referred to the camera and world frames, respectively. The camera state is assumed to evolve between the posterior estimate at time k and prior at $k+1$, an interval δt later, as

$$\begin{bmatrix} \mathsf{q}^{wc^*} \\ \boldsymbol{t}_{c^*}^w \\ \boldsymbol{\Omega}^c \\ \boldsymbol{V}^w \end{bmatrix}_{k+1} = \begin{bmatrix} \mathsf{q}^{wc} \times \mathsf{q}\left((\boldsymbol{\Omega}^c + \boldsymbol{a}_\Omega^c \delta t)\delta t\right) \\ \boldsymbol{t}_c^w + (\boldsymbol{V}^w + \boldsymbol{a}_V^w \delta t)\,\delta t \\ \boldsymbol{\Omega}^c + \boldsymbol{a}_\Omega^c \delta t \\ \boldsymbol{V}^w + \boldsymbol{a}_V^w \delta t \end{bmatrix}_k, \tag{3.13}$$

where it assumed that the accelerations $\boldsymbol{a}_\Omega^c$ and $\boldsymbol{a}_V^w$ are drawn from zero-mean Gaussian sequences acting throughout the duration δt. The state covariance is fully populated:

$$\mathtt{P} = \begin{bmatrix} \mathtt{P}_{cc} & \mathtt{P}_{cX_1} & \cdots & \mathtt{P}_{cX_n} \\ \mathtt{P}_{X_1c} & \mathtt{P}_{X_1X_1} & \cdots & \mathtt{P}_{X_1X_n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathtt{P}_{X_nc} & \mathtt{P}_{X_nX_1} & \cdots & \mathtt{P}_{X_nX_n} \end{bmatrix}. \tag{3.14}$$

3.3.1.2 The extended Kalman filter

The EKF first linearizes the plant and measurement processes about the current state estimate, and then deploys the Kalman filter mechanism to solve. The EKF's prior estimates for state and covariance are

$$\boldsymbol{\chi}_{k+1|k} = \boldsymbol{f}(\boldsymbol{\chi}_{k|k}) \tag{3.15}$$

$$\mathtt{P}_{k+1|k} = \nabla\mathtt{F}\mathtt{P}_{k|k}\nabla\mathtt{F}^{\top} + \nabla\mathtt{G}\mathtt{Q}\nabla\mathtt{G}^{\top}\,, \tag{3.16}$$

where $\mathtt{Q}$ is the plant noise covariance $\mathtt{Q} = \delta_{ij}E[\boldsymbol{v}_i\boldsymbol{v}_j^{\top}]$, and $\nabla\mathtt{G}$ is defined below. The innovation covariance and Kalman gain matrices are derived as

$$\mathtt{P}_{xx} = \nabla\mathtt{H}\mathtt{P}_{k+1|k}\nabla\mathtt{H}^{\top} + \mathtt{R} \tag{3.17}$$

$$\mathtt{W} = \mathtt{P}_{k+1|k}\nabla\mathtt{H}^{\top}\mathtt{P}_{xx}^{-1}\,, \tag{3.18}$$

where $\mathtt{R}$ is the noise covariance. The predicted measurements are $\boldsymbol{x}_{k+1|k} = \boldsymbol{h}(\boldsymbol{\chi}_{k+1|k})$ and, once the actual measurements $\boldsymbol{x}_{k+1}$ at the timestep $k+1$ are available, the state and covariance updates are made using

$$\boldsymbol{\chi}_{k+1|k+1} = \boldsymbol{\chi}_{k+1|k} + \mathtt{W}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_{k+1|k}) \tag{3.19}$$

$$\mathtt{P}_{k+1|k+1} = \mathtt{P}_{k+1|k} - \mathtt{W}\mathtt{P}_{xx}\mathtt{W}^{\top}. \tag{3.20}$$

The three Jacobians have elements

$$\nabla\mathtt{F}_{ij} = \frac{\partial\chi_{i(k+1)}}{\partial\chi_{j(k)}};\ \nabla\mathtt{H}_{ij} = \frac{\partial x_{i(k+1)}}{\partial\chi_{j(k+1)}};\ \nabla\mathtt{G}_{ij} = \frac{\partial\chi_{i(k+1)}}{\partial v_{j(k)}}. \tag{3.21}$$

3.3.1.3 Measurement process

The features are assumed to arise from a perspective projection of the scene points into the camera

$$\lambda_i\begin{bmatrix}\boldsymbol{x}_k^i\\1\end{bmatrix} = \mathtt{K}\,[\mathtt{I}|\mathbf{0}]\begin{bmatrix}\mathtt{R}^{cw} & \boldsymbol{t}_w^c\\ \mathbf{0}^{\top} & 1\end{bmatrix}_k\begin{bmatrix}\boldsymbol{X}_k^i\\1\end{bmatrix} \tag{3.22}$$

where $\mathtt{R}^{cw}$ and $\boldsymbol{t}_w^c$ transform map points in the world frame into the camera frame, $\mathtt{K}$ is the camera's intrinsic calibration which may change but is always known, and λ_i expresses the unknown scale introduced by homogeneous coordinates. The $\boldsymbol{x}_k^i$ are obtained from actual image measurements using a correction for radial distortion (discussed later). Also projected into the image is the associated uncertainty region derived at the 3σ limit from the prior covariance, and it is within this region that a matching feature is sought using Shi-Tomasi (as described in §3.2.2). Features that are added to the 3D map are stored with an 11×11 pixel appearance template. Searches are made within the region for correspondence using normalised sum-of-squared difference correlation.

3.3.2 Initialization and subsequent map management

To break the depth/speed scaling ambiguity and establish a metric scale, monoSLAM is initialized using a calibration plate which delivers 3D features at known positions. These are inserted into the map with zero uncertainty. The standard plate is a rectangle, and can be seen in Fig. 3.9.

To satisfy the hard constraint of completing processing within 33 ms, a set of map management criteria is used to restrict the number of features observable in any one view. A feature is predicted to be observable from some particular viewpoint if its projection into a model of the camera lies well within the image, and if both angular and range differences between the current viewpoint and the feature's initial viewpoint are not so great as to make the image template valueless. New features are added if the number of those actually observable falls to five or fewer, and a feature is deleted if its long-term ratio of actual observability after search to predicted observability falls below 1:2. In this implementation, points are initialized in the map using the inverse depth method of Montiel *et al.* [100].

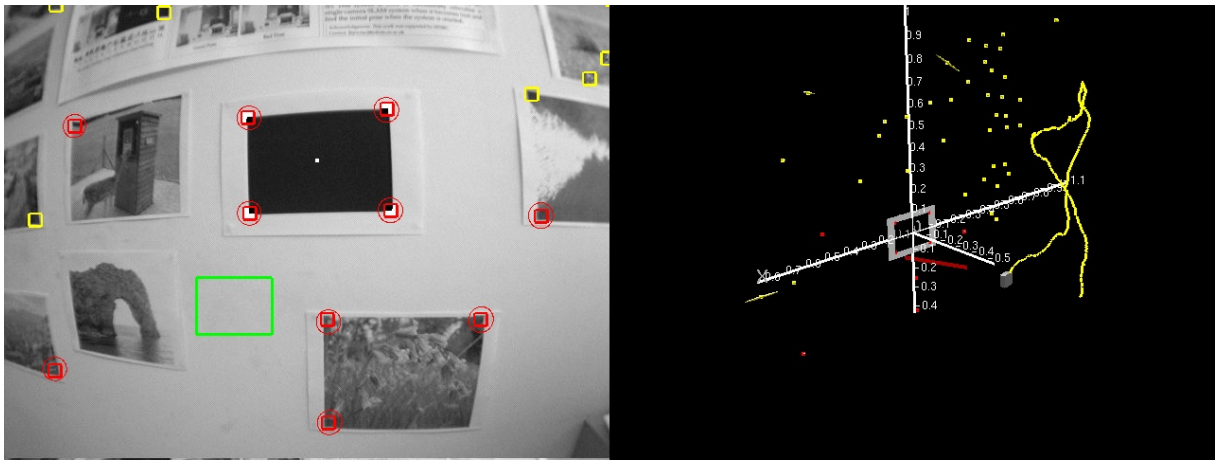


Figure 3.9: The monoSLAM system running. The image on the left is the camera view with features that have been detected highlighted. The image on the right shows the 3D view of the map with the points and their uncertainty ellipsoids.

3.3.3 Typical performance of monoSLAM

Typical output from a stand-alone monoSLAM system is shown in Fig. 3.9. In the left-hand image the ellipses represent the uncertainties in the detected features, and the rectangles are the outlines of the patches around the features that are used for matching. The right-hand image of Fig. 3.9 shows a 3D representation of the map; here the camera can be seen near the centre of the image and the detected features around the camera in 3D. The features are surrounded by their covariance ellipsoids.

3.4 Appearance-based recognition

Although monoSLAM gives a complete, dense, account of the camera's track (or pose) over time, the 3D map it produces is extremely sparse. The immediately interpretable spatial information available to the user is minimal, as comparison of the image of a scene and its maps in Fig. 3.9 shows. The comparison serves both (i) to reinforce the earlier statement that Mayol's demonstration of fixation with an active wearable (Fig. 3.1) is compelling largely because the *human viewer* augments the fixation point with significance belonging to the underlying object, and (ii) to motivate the combination of automatic object recognition with SLAM for assistive

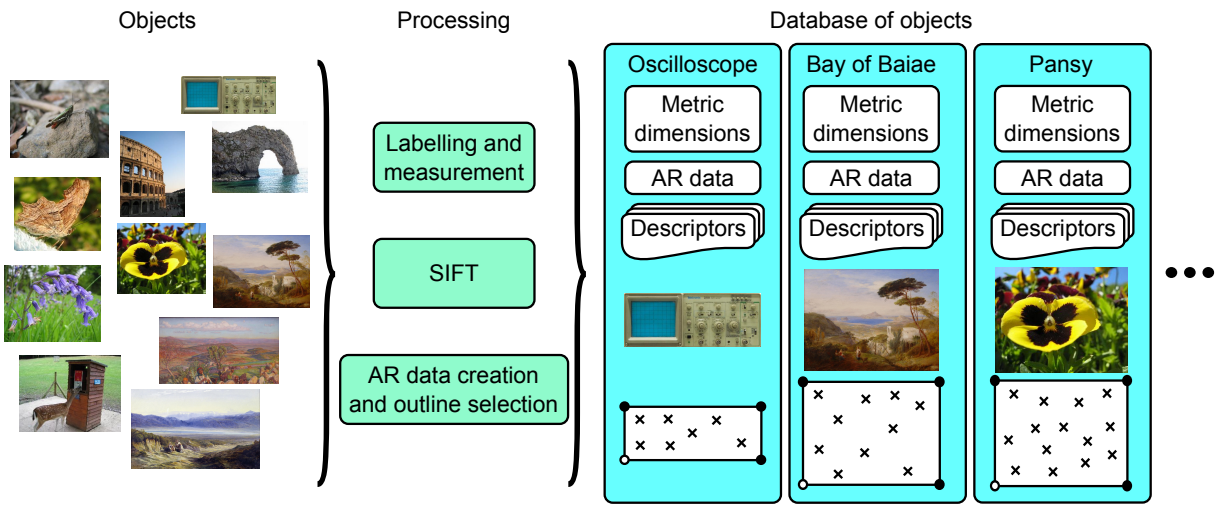


Figure 3.10: The object database contains planar objects (here pictures), the list of SIFT descriptors and their locations $\boldsymbol{x}^i$ (crosses), and the locations of boundary points, three of which are marked for use in the SLAM map.

wearable vision.

This section details the method of recognition and object localization used. This is the upper processing path on the system diagram given earlier in Fig. 3.2.

3.4.1 Object database

To be able to recognize particular objects, the system needs to have learnt their characteristics. This is done by creating a database of objects that can be queried when matching. The database consists of planar objects, a planar constraint was imposed from the outset so that objects could be located from a single view, allowing a greater degree of independence between the recognition and SLAM threads (the removal of this constraint is discussed in Chapter 7). Objects are added off-line to the database, with a degree of manual intervention. The process could be achieved on live imagery, though considerable effort would be required to produce a polished, intuitive, interface.

As illustrated in Fig. 3.10, to construct an entry an image of the object is captured and, after correcting for radial distortion, SIFT descriptors $\boldsymbol{\sigma}^i$ and their positions $\boldsymbol{x}^i$, $i = 1...I$ are computed. The image does not need to be fronto-parallel, and so the homography $\mathtt{H}$ between the

scene and image is found by choosing $n \geq 4$ image points whose corresponding scene points $\boldsymbol{X} = [X, Y, 1]^\top$ can be located in a object-based coordinate system. The database entry

$$\begin{aligned} \mathcal{O}_j \quad = \quad & \{\, \mathcal{I}_{\mathrm{R}}, \quad \{\boldsymbol{\sigma}^i, \boldsymbol{X}^i = \mathtt{H}^{-1}\boldsymbol{x}^i\}_{i=1\ldots I}, \\ & \{\boldsymbol{X}^m_{\mathrm{B}}\}_{m=1\ldots M}, \quad m_1, m_2, m_3 \in \{1\ldots M\} \\ & \{\boldsymbol{X}^n_{\mathrm{AR}}, \text{“AR-markup”}\}_{n=1\ldots N}\,\}_j \end{aligned} \tag{3.23}$$

contains (i) the image $\mathcal{I}_{\mathrm{R}}$ of the object rectified by the homography so that it appears as a fronto-parallel view, (ii) the list of SIFT descriptors $\boldsymbol{\sigma}^i$ and their locations $\boldsymbol{X}^i$ within the object plane, (iii) the locations of several boundary points $\boldsymbol{X}^m_{\mathrm{B}}$ to define the object extent, (iv) the indices of three boundary points flagged for use in the SLAM map, and finally (v) a number of positions $\boldsymbol{X}^n$ tagged with annotations for AR.

Each object entry typically contains around 2 000 keypoints, and each database holds tens of objects. To permit efficient search, all database keypoints are structured in a kd-tree as explained below.

3.4.2 Object detection

While SLAM takes every image to build and maintain the map and camera track, the object detection thread selects images at a lower rate of around 1.5 Hz in this implementation. SIFT keypoints are extracted (the most time consuming part of the object detection) and their locations corrected for radial distortion. This is found to be faster than undistorting the whole image prior to running SIFT, and the distortion is found not to significantly affect the SIFT descriptors themselves.

To detect the presence of objects, each keypoint in the current image needs to be matched to at most one keypoint in the database, and a geometric consensus between all the keypoints matched to a particular object established. This is achieved using Beis and Lowe's best-bin-first modification of kd-trees [10], followed by a verification stage using robust homography fitting.

The matching criterion is so-called second-nearest-neighbour matching, where a keypoint $\boldsymbol{\sigma}^n$ from the current image is matched to the i-th keypoint from the j-th object $\boldsymbol{\sigma}^i_j$ provided both that they are nearest-neighbours,

$$i, j = \arg\min_{i',j'} \left(||\boldsymbol{\sigma}^n - \boldsymbol{\sigma}^{i'}_{j'}||_2 \right)^{1/2}, \tag{3.24}$$

and that the distance to the next nearest neighbour is significantly larger. But to search the large database of features in faster than linear time, a modification of k-dimensional trees [45] is used.

In a kd-tree, each node represents a datum $\boldsymbol{\sigma}$ from the database. Creating the root node involves choosing a dimension of $\boldsymbol{\sigma}$ and then setting the node to represent the datum with some chosen "pivotal" value along that dimension. The recipe of choice depends on the problem domain. Here at the root the first dimension of $\boldsymbol{\sigma}$ is used, and the pivot is the datum with the median value along that first dimension. This value defines a splitting hyperplane, and all remaining data with values of that dimension less that the pivot value must lie in the left sub-tree of the node, and, conversely, all data with values greater or equal, in the right sub-tree. The two child nodes of the root are selected using the second dimension of $\boldsymbol{\sigma}$, each from amongst the nodes in the left and right sub-trees, respectively. The grandchildren are selected using the third dimension, *etc.*, and after the last dimension, one cycles through again from the first until until all the data is partitioned. Once built, to perform a search for the nearest-neighbour of a $\boldsymbol{\sigma}^n$ from the current image, the query descends the tree from the root, selecting the left or right branch by comparison of value of the relevant dimension of $\boldsymbol{\sigma}^n$ with the value at the node (*e.g.*, at the root if $\boldsymbol{\sigma}^n(1) < \boldsymbol{\sigma}^{\mathrm{root}}(1)$, the query descend leftwards). Once it reaches a leaf node, this node becomes the current best match. The query then traverses back up the tree checking if any node is closer than the current best, but now using the full Euclidean distance. If it is, it becomes the current best, and the entries in the node's other tree have to be checked. If not, the query ascends a level. Once the root node is reached the current best match is confirmed as the nearest neighbour.

A kd-tree is efficient only when the number of data $N \gg 2^k$, where k is the dimensionality. SIFT descriptors have 128 dimensions, and this requirement is grossly violated (by a mere 10^{33}) with a database of, say, 10^5 keypoints. Instead, Lowe [85] suggests a best-bin-first (BBF) algorithm [10] that is based on kd-trees, but returns the nearest neighbour with high, but not unit, probability.

Because an exhaustive search is not made, the BBF method uses multiple trees to find a pair of nearest neighbours. Four trees are formed in the following manner. First the set of keypoints is randomized to allow for unbiased sampling. The keypoints are then partitioned by selecting the first hundred keypoints and calculating the mean and variance for each dimension. The top five dimensions with the highest variance are found and one is selected at random. This becomes a node, and the splitting occurs at the mean, with keypoints where this dimension is less than the mean forming the left branch and the rest forming the right. This process then repeats for each branch until each keypoint is assigned to a leaf node. Each tree then has a different path to a particular keypoint. The trees are created prior to the system running, and only need to be recomputed when features are added or removed from the database.

To compare a keypoint to the database keypoints, the two nearest neighbours need to be found to ascertain a match. The keypoint is processed through each tree with its dimensions checked against each node. Each branch that is not selected is recorded for later distance comparison. Once a leaf is reached, the distance between the keypoint and the database keypoint is calculated, and it is added as a nearest neighbour. Results from the other trees may provide a different keypoint. If a different keypoint is found it is also stored as a nearest neighbour. If more than two different keypoints are found as nearest neighbours they replace the keypoint that is furthest away of the pair. Then the branches that were not taken in all of the trees are checked. They are checked in order of minimum distance to the splitting dimension and only until the maximum number of checks (128) has been done. If any of these result in a nearer neighbour then this replaces the current furthest of the best pair. The distance between the pair

of nearest neighbours is then checked and if the nearest is closer than 0.6 times the distance of the second then it is accepted as a match. This allows the difficult cases where the distances are similar to be ignored.

3.4.3 Match filtering through homography estimation

If the number of matched points from the current image to any given object is greater than a threshold (here, 8 is used), the object is regarded as a candidate for further testing.

As the database objects are known to be planar, the positions of the database objects' keypoints $\boldsymbol{X}$ and the positions of their observed and matched image points $\boldsymbol{x}$ are (or should be) related by a plane-to-plane homography $\boldsymbol{x} = \mathtt{H}'\boldsymbol{X}$. However, because BBF is not certain to find the closest matches, and closest matches found may in any case be mis-matches, RANSAC is used to compute a robust homography between the image and database keypoints [42, 148].

A random sample is selected, containing the minimum number of data, s, required to generate a hypothesis (*i.e.*, here $s = 4$), and the number of data supporting the resulting hypothesis are counted as inliers. This is repeated N times and the final fit made using the largest set of inliers. The number of trials is determined from the desired probability p that at least one sample set is free from outliers (here $p = 0.99$), and ϵ is the estimated fraction of outliers in the data. Then

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)} \,. \tag{3.25}$$

The fraction ϵ has been found adaptively, by setting ϵ to a worst case estimate and, as samples are taken and the consensus shows fewer outliers, reducing ϵ. Typically ϵ is less than 0.3, with a mean value around 0.05.

Objects that still retain eight or more keypoint matches in the fitting of the final homography are declared recognized and visible, and the homography itself, $\mathtt{H}'$, used for object localization as described in the next section.

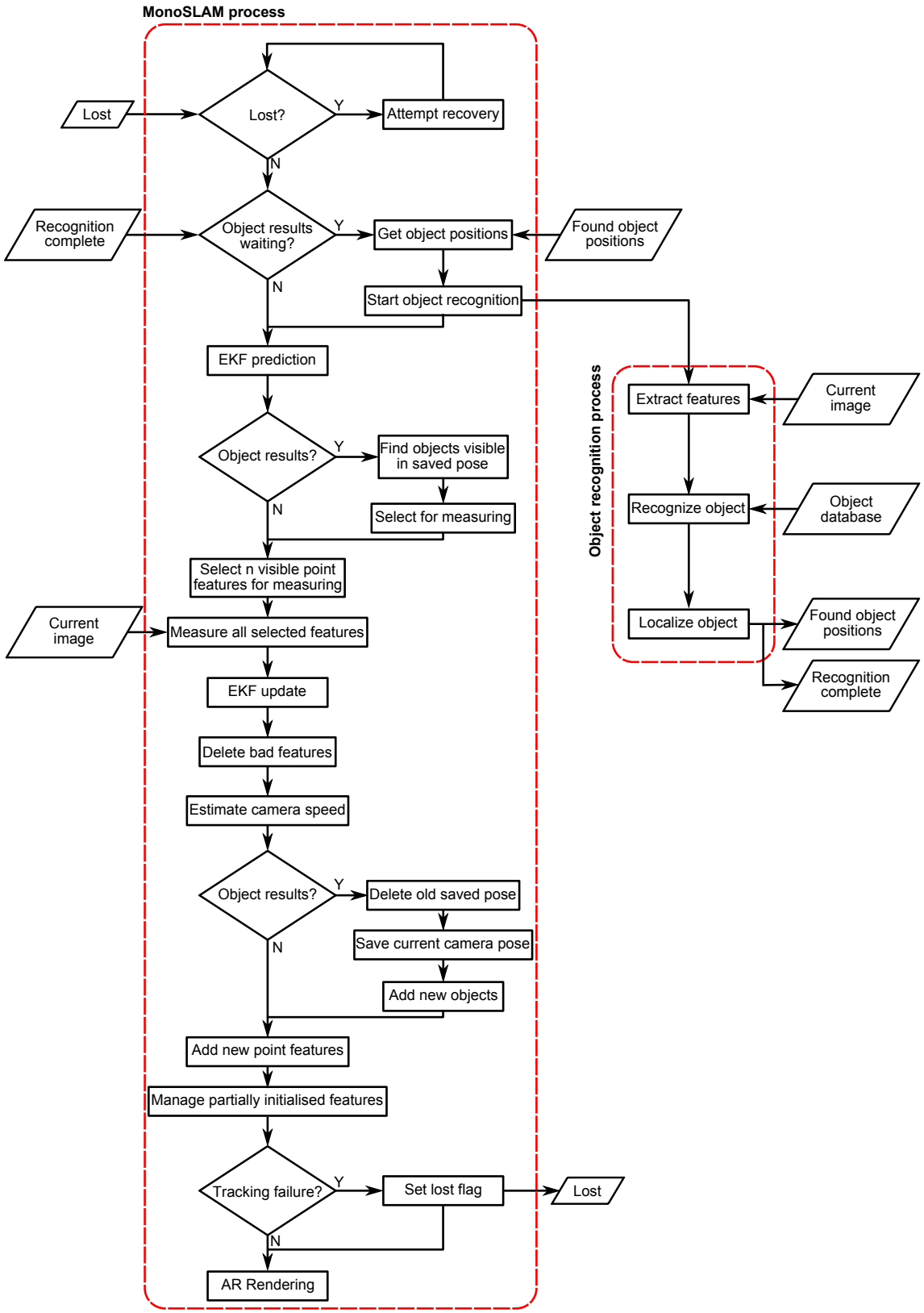


Figure 3.11: A detailed view of the monoSLAM and object recognition processes and their interactions.

3.5 The interaction between recognition and SLAM

Up to now the object recognition has remained independent from the tracking and mapping, but once the detected objects are localized they are either added to the map or update object measurements already in the map. This involves the two processes interacting. This section details how the objects are localized in 3D and how they are added to the map. Fig. 3.11 provides a more detailed view of the system, showing how the monoSLAM step integrates the object results with the standard monoSLAM procedure.

The next section deals with the last stage of the object recognition process, the localization of the detected objects. Then Section 3.5.2 details how objects are added to the map, which covers the interaction of the objects with the monoSLAM process as shown in Fig. 3.11.

3.5.1 Object localization

Having determined an object is visible, its location is recovered by decomposing the homography between the object in the database and the current image. In the Cartesian object-centred coordinate frame, the object lies in the plane $Z = 0$, and 3D homogeneous points on the object are $\boldsymbol{X}^{(4\times1)} = [X, Y, 0, 1]^\top$. In any view, therefore, the projection can be written up to scale in terms of extrinsic and intrinsic parameters as

$$\boldsymbol{x} \sim \mathtt{K}[\mathtt{R}|\boldsymbol{t}]\boldsymbol{X}^{(4\times1)} = \mathtt{K}\,[\boldsymbol{r}_1\ \boldsymbol{r}_2\ \boldsymbol{t}] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}, \tag{3.26}$$

where $\mathtt{K}$ is the camera calibration, $[\mathtt{R}|\boldsymbol{t}]$ the rotation and translation matrix. The first two columns of the rotation matrix $\mathtt{R}$ and the translation relative to the camera are found modulo a positive scaling factor using the homography already computed from,

$$\begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t} \end{bmatrix} \sim \text{sign}([\mathtt{K}^{-1}\mathtt{H}']_{33})[\mathtt{K}^{-1}\mathtt{H}']\,. \tag{3.27}$$

Because the estimate $\mathtt{H}'$ is noisy, there is no guarantee that $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ found as above will be orthogonal. The closest rotation matrix and overall scale for the translation are determined

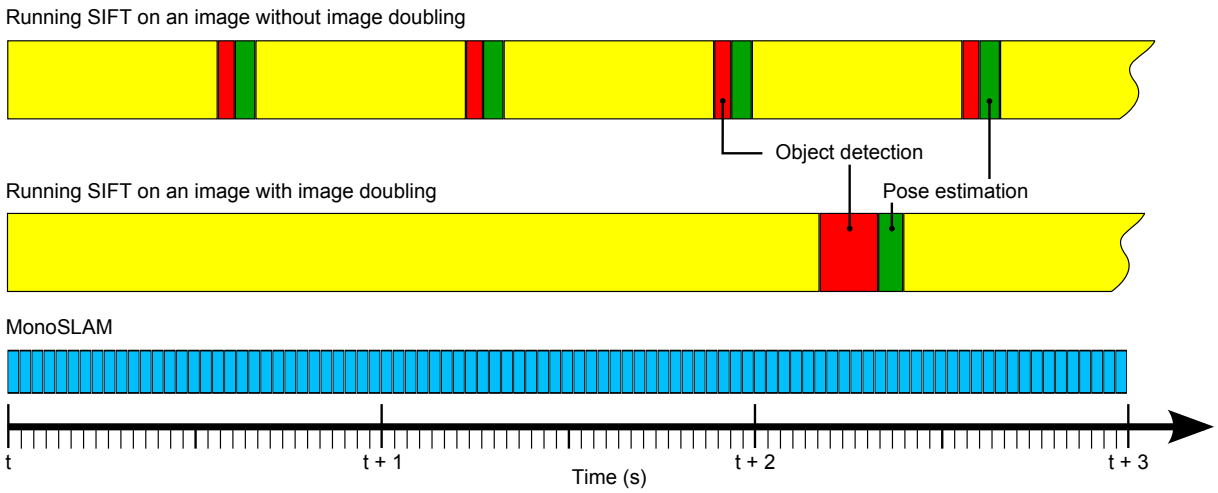


Figure 3.12: ObjectSLAM processing time line. The monoSLAM thread runs at 30 Hz, whereas the object detection thread only runs at around 1.5 Hz without image doubling and around 0.43 Hz with image doubling. The large processing time disparity can be clearly seen. Times are for a 640×480 input image and a database containing 5 objects with a total of 9 433 keypoints.

using singular value decomposition

$$\mathtt{U}\Sigma\mathtt{V}^\top \leftarrow [\boldsymbol{r}_1\ \boldsymbol{r}_2\ \boldsymbol{r}_1 \times \boldsymbol{r}_2] \tag{3.28}$$

$$\mathtt{R} \leftarrow \mathtt{U}\mathtt{V}^\top \tag{3.29}$$

$$\boldsymbol{t} \leftarrow \boldsymbol{t}/\sqrt{\Sigma_{11}\Sigma_{22}} \tag{3.30}$$

where $\Sigma_{11,22}$ are the first and second singular values.

The rotation matrix and translation vector calculated in this way specify the transformation of the camera from the frame of reference of an object's canonical database image. This transformation is applied in reverse to place the object in the frame of reference of the camera at the time the image was selected. A further transformation, determined by the camera's pose at the time of capture relative to the world coordinate frame defined by the SLAM map, is then applied to derive the position of the object in world coordinates.

3.5.2 Map insertion

It was noted in the introduction that the object recognition process does not run at the same rate as the monoSLAM process. MonoSLAM must run at frame-rate (here, 30 Hz) to satisfy

the principle that even the fanciest motion model cannot improve on reality, and hence making measurements is always better for tracking than prediction. The object recognition thread only runs as fast as it can process an image and recognize objects within it. Fig. 3.12 shows the difference in processing times between the monoSLAM thread and the object detection thread. SIFT's processing frequency is some 1.5 Hz,[1] but is dependent on the size of the database and the number of objects found in a frame. This slower and potentially asynchronous performance means that once object recognition and localization has completed, the camera pose and uncertainty will have been updated some twenty times. A mechanism is required to permit measurement updates using recognized objects at whatever time the detection and recognition processes manage to complete processing a frame.

The method that has been adopted to achieve this is that of delayed decision-making proposed by Leonard and Rikoski [77]. Suppose monoSLAM is running as normal, and that at a timestep k the object recognition and localization modules commence. The current state vector is augmented by the camera pose, $\boldsymbol{s} = [\boldsymbol{t}, \mathsf{q}]$,

$$\mathcal{X}_k^{\mathrm{A}} = [\boldsymbol{c} \quad \boldsymbol{s} \quad \boldsymbol{X}_1 \quad \cdots \quad \boldsymbol{X}_n]_k^\top \ , \tag{3.31}$$

initialized to the current pose value $\boldsymbol{c}_k$. The covariance matrix is similarly augmented

$$\mathsf{P}_k^{\mathrm{A}} = \begin{bmatrix} \mathsf{P}_{\mathbf{cc}} & \mathsf{P}_{\mathbf{cs}} & \mathsf{P}_{\mathbf{c}\mathbf{x}_1} & \cdots & \mathsf{P}_{\mathbf{c}\mathbf{x}_n} \\ \mathsf{P}_{\mathbf{sc}} & \mathsf{P}_{\mathbf{ss}} & \mathsf{P}_{\mathbf{s}\mathbf{x}_1} & \cdots & \mathsf{P}_{\mathbf{s}\mathbf{x}_n} \\ \mathsf{P}_{\mathbf{x}_1\mathbf{c}} & \mathsf{P}_{\mathbf{x}_1\mathbf{s}} & \mathsf{P}_{\mathbf{x}_1\mathbf{x}_1} & \cdots & \mathsf{P}_{\mathbf{x}_1\mathbf{x}_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathsf{P}_{\mathbf{x}_n\mathbf{c}} & \mathsf{P}_{\mathbf{x}_n\mathbf{s}} & \mathsf{P}_{\mathbf{x}_n\mathbf{x}_1} & \cdots & \mathsf{P}_{\mathbf{x}_n\mathbf{x}_n} \end{bmatrix}_k \ , \tag{3.32}$$

where $\mathsf{P}_{\mathbf{sc}} = \mathsf{P}_{\mathbf{cc}}[\partial \boldsymbol{s}/\partial \boldsymbol{c}]^\top$, and so on. After the saved camera pose has been added to the state, its value can no longer be directly measured. However, the correlation values contained in P^{A} between this saved pose and other elements of the state enable its value to be updated as the EKF's cycles continue. Therefore, as the state continues to be updated, the saved pose will be refined such that it remains consistent with newer state estimates. Once the object detection and localization completes, say N frames later, the updated saved camera pose $\boldsymbol{s}_{k+N}$ and associated

[1] This frequency assumes no image doubling, as mentioned later in §4.3.

covariance is used to determine the position of any recognized objects in the world, rather than $\boldsymbol{s}_k$. Then the saved pose is deleted from the state vector and covariance matrix. Although only one saved state is used here, the mechanism allows for multiple detection processes to start and finish at different times, were further processing power available.

The delayed insertion method provides a faster and less complex update compared with the alternative of rolling back the EKF, inserting the measurement, and then rolling forward by recalculating all measurements from the frame the object detection was performed on. However, using a saved camera pose to calculate the location of objects relies on the monoSLAM part of the system maintaining a good estimate of the camera pose and trajectory during the intervening frames. This a reasonable assumption, because if monoSLAM fails to do this then the system will become unrecoverably lost anyway.

A number of methods for representing objects in the 3D map can be envisaged. The approach used here is to allow the recovered 3D position of the planar object to define 3D point measurements. The feature positions themselves are not entered, but instead the three points $\boldsymbol{X}_{\mathrm{B}}^{m}$, $m = m_1, m_2, m_3$ from the object's boundary designated in the object database entry are used. For example, for a rectangular picture, three of the four corners are inserted into the map. The benefits in this approach are, first, no additional mechanism is required in the SLAM process. Provided reasonable values are supplied for the (typically much lower) 3D error in these points, constraints on the scene will propagate properly through the covariance matrix. Secondly, there is no reliance on any particular SIFT features being re-measured over time. Thirdly, the boundary points provide a convenient representation of the extent of the object for graphical augmentation, as discussed now.

3.6 Rendering

Once an object has been located in 3D and inserted into the map, it can be presented to the user of the wearable system for augmented reality applications. The experimental evaluation

of Chapter 4 shows not just the outline of recognized objects but also areas of interest from within the object's surface rendered onto the image stream from the camera.

The cameras used in this work have wide angle lenses, generating considerable barrel distortion. Before a camera can be used it needs to be calibrated not just for the intrinsic parameters K, but also for a single distortion parameter κ_1. Von Seidel's series expression for radial distortion (a power series in r^2, where r is the distance from the centre of distortion) is approximated using Harris' invertible model [53] (described in theses by Beardsley [9] and Tordoff [147], and rediscovered by Swaminathan and Nayar in [143])

$$\begin{aligned} \boldsymbol{x}_d - \boldsymbol{c}_\mathrm{d} &= \frac{\boldsymbol{x}_u - \boldsymbol{c}_d}{\sqrt{1 - 2\kappa_1 r_u^2}} \\ \boldsymbol{x}_u - \boldsymbol{c}_\mathrm{d} &= \frac{\boldsymbol{x}_d - \boldsymbol{c}_d}{\sqrt{1 + 2\kappa_1 r_d^2}} \end{aligned}, \tag{3.33}$$

where the first distortion coefficient, κ_1 is negative for barrel distortion, $\boldsymbol{c}_d$ is the centre of distortion, $\boldsymbol{x}_u$ is the ideal undistorted image point, $\boldsymbol{x}_d$ is the distorted point, and $r_{u,d} = |\boldsymbol{x}_{u,d} - \boldsymbol{c}_d|$. The centre of distortion is not the principal point, but, following Willson's work [157], is taken as the image centre.

3D points are assumed projected from the world frame into the undistorted image using Eq. (3.22) and, after de-homogenization, distorted. The following steps display the outline of the object:

1. Transform the 3D boundary points from their world coordinate frame into the camera coordinate frame.
2. Check if the 3D boundary points are in front of the camera.
3. Calculate the 3D locations of the remaining corners and object centre.
4. Project the 3D boundary and centre points using the camera model on to the camera image.
5. Draw a line between each corner, accounting for the lens distortion.

6. Draw the name of the object an appropriate distance above the object outline, but within the image boundary.

7. Draw a line connecting the object centre to the name.

Entities such as lines and arcs are parameterized, and drawn as multiple short segments between distorted end points.

AR elements at other locations on the object are expressed in the database as coefficients of a pair of 2D basis formed by the boundary points. Their 3D positions are found by applying the coefficients to the 3D vectors formed by the basis, and are then projected into the ideal image (Eq. 3.22), and distorted (Eq. 3.33).

3.7 Relocalization on tracking failure

All the components in Fig. 3.11 involved in "smooth running" have now been described. However, when monoSLAM is used with a hand held camera, the camera can easily become "lost", and frequently does. Tracking loss occurs because of failure to match features using active search: cross-correlation between the 11×11 patches which accompany each Shi-Tomasi feature and the 3σ region where the feature is predicted in the next frame. There are several causes, the principal ones being erratic camera motion which confounds the constant velocity model, motion blur, occlusion, cumulative measurement error, and straying into a featureless area.

To avoid rebuilding the map from scratch, the relocalization method of Williams *et al.* [155] is invoked when one of the following symptoms is detected: (i) less than three features matched; (ii) camera uncertainty is too large (> 0.05); (iii) no visible features; or (iv) too few features matched since a previous recovery. Part of the relocalizer runs continually in the background, using a modified randomized fern classifier to learn the appearance of the monoSLAM map points. When tracking is lost, matches are sought between feature patches from the current view and those in the map, and RANSAC is used with randomly selected sets of three matches

to solve for the camera pose. When a valid pose is found (*i.e.*, one with support from many inliers) monoSLAM attempts to continue tracking from this location. The relocalizer has a limited range of distance and angle displacement over which it can function, but it is found highly compatible with the camera being under human control.

Randomized ferns are a derivative of randomized trees. Randomized trees were used by Fua and co-workers [78] [111] for real-time tracking over large camera displacements, and can cope with image noise, changes in scale, rotations, aspect ratio, and illumination changes. The method treats feature recognition as a classification problem, where a class, C, is all possible views of a particular feature. The multiple views can either be harvested from actual views of a feature, or synthetically generated. The set of randomized trees, $\{t_j\}$, $j = 1, \ldots, T$, are binary decision trees of depth D which map an input patch to one of 2^D posterior distributions at a particular tree's leaf nodes. Fig. 3.13(a) shows an example tree with $D = 3$ and $C = 12$, but actual implementations typically have $T = 40$, $D = 15$, and $C = 100 - 1\,000$. At each node, $\{Q_i\}$, a question is asked of two randomly chosen pixels in the feature patch p_n,

$$Q_i = \begin{cases} 0 & p_n(u_i, v_i) \geq p_n(u_i', v_i') \\ 1 & \text{otherwise} \end{cases} . \tag{3.34}$$

Although randomly chosen, once the random pair is chosen for a particular node, they are fixed for that particular tree.

The posteriors are filled during a training phase by filtering all of the views of each feature down all of the trees. When a patch for a particular class arrives at a leaf node the class in the posterior is incremented. Then when the system is running, features to be matched have their patch filtered through all of the trees. The resulting posteriors from each tree are summed and the class with the largest sum over a threshold is considered a match.

As noted earlier, randomized ferns [110] are a modification to the randomized tree algorithm that simplifies the tree structure. Instead of asking different random questions at each node of the tree, all nodes at the same level ask the same question. The histogram is now formed from sum the answers to each binary question from all ferns. This is much faster to compute and

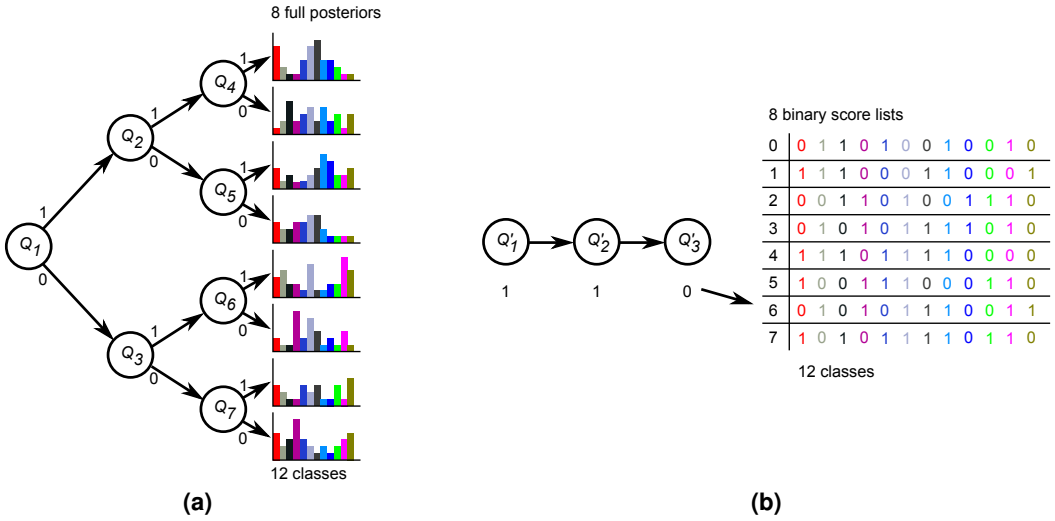


Figure 3.13: (a) An example of a randomized tree with a depth of 3, and (b) a randomized list as used by Williams *et al.* for the monoSLAM relocalizer.

less memory intensive as the decision trees are now very simple straight lines.

Williams *et al.* use their own independently developed version of randomized ferns that is adapted for the relocalizer used here. An example is shown in Fig. 3.13(b). The two key differences to note are (i) that the classifier for the relocalizer is trained continually on-line, whereas the systems by Lepetit and Fua are trained once off-line and (ii) whereas the Lepetit and Fua systems operate with hundreds of features visible per frame, the relocalizer uses a far sparser set with around 10 to 20 visible per frame. This led Williams *et al.* to tune the classifier for recall rather than precision, and develop four other notable differences. First, some features will inevitably resemble each other. Lepetit and Fua's method penalise these cases, leading to the possibility that none are recognized. Here though they are treated independently, meaning that features added later do not affect the current ones. Secondly, to reduce memory requirements the full normalised posterior distributions are not stored. Instead a binary score string of one bit per class is stored, with a 0 indicating that a training example was never found in that leaf, and a 1 if at least one training example was. This leads to a storage requirement of $2^D \times C \times T$ bits for all scores. Thirdly, explicit noise handling is done during training, instead of adding random noise. Finally, instead of testing if a pixel is brighter than another, which only mea-

sures image noise in areas of uniform intensity (such as man-made environments), a modified test is done. The difference in pixel intensity is compared against a randomly chosen offset, z_i,

$$Q'_i = \begin{cases} 0 & p_n(u_i, v_i) - p_n(u'_i, v'_i) \geq z_i \\ 1 & \text{otherwise} \end{cases} . \tag{3.35}$$

The binary score lists are constantly updated while the monoSLAM system is running. When a point is first observed the classifier generates 400 synthetically warped versions of the 11×11 patch, so it can be recognized under different views. This warping is performed on the GPU to minimize CPU loading, and the rest of the classification is run in a background thread taking around 30 ms to train a new feature. Then features in the map that are successfully matched have their new view harvested for the relocalizer to reinforce the initial training stage.

A final modification affects the way new features are selected by monoSLAM. To improve the distinctiveness of the points in the map, the classifier aids the selection of new map points. During the process of initializing a new point the monoSLAM system usually has a choice of potential candidates. Rather than taking the one with the strongest corner response, each candidate corner is first classified with the current classifier. The feature selected for initialization is now the one that scores lowest against all other points already in the map. This leads to a map of points which are more easily discriminated.

3.8 Near planar objects

This work has focussed on the recognition of planar objects, but not all objects that may want to be tracked are planar. The restriction to planar objects is not a fundamental one: 3D objects could be inserted into the SLAM map almost as easily as planar objects. Indeed, Gordon and Lowe [49] demonstrate this, albeit in the context of structure from motion rather than SLAM, and Wangsiripitak and Murray [154] have recently inserted mobile 3D objects into monoSLAM's EKF maps. However, with any model there remains a question of how well-

carpentered does the object have to be? Obviously this depends on tolerances in SIFT and SLAM matching, and in the typical overall errors in the map. There is a variety of research on locating and tracking 3D objects using 3D CAD models and edge tracking [33, 117, 120, 64]. Brown and Lowe [13] showed how objects 3D objects can be reconstructed from enough views using SIFT and the correspondence between all of the SIFT descriptors for that object can be found. Both of these methods require significant time to either construct the model or capture all of the views required. For near planar objects, such as an instrument panel, they can still be recognised and tracked if they are treated as planar. The limitation is that some correctly matched features on the object will fail the geometric consistency check during the homography estimation when the camera is not directly in front of the object. These features will typically be on protruding and recessed parts of the object as these will move around the most as the camera changes position. In practice this leads to near planar objects having a smaller viewing angle than for planar objects. The maximum viewing angle for different objects depends on the non-planar parts of the object. Near-planar objects are explored in the next chapter.

3.9 Conclusion

This chapter has developed a method for combining planar object recognition with an EKF based single camera tracking and mapping algorithm. It has also reviewed the background operation of each component part. The object recognition and localization runs independently from the monoSLAM tracking and mapping. Then, once objects have been localized, the monoSLAM process integrates these measurements with the map. This enables the objects to act as measurements, as well as allowing AR related to each object to be displayed automatically on the objects in the world frame.

The next chapter provides implementation details of the method presented here, and evaluates its performance in an indoor setting and three AR applications.

4

Integrating object recognition with single camera SLAM: (II) implementation and evaluation

4.1 Introduction

The first part of this chapter details the software developed to realize the objectSLAM system developed and described in the previous chapter. The second part evaluates the system in a variety of scenarios. The chapter concludes with a look at the system's limitations and possible improvements.

4.2 Implementation

Two substantial blocks of software have been implemented: one is the objectSLAM system itself, and the other is an application to aid in database creation and management.

4.2.1 Object database manager

Before the objectSLAM software can be used a database of known objects has to be created. As detailed in the previous chapter this involves running SIFT on an image to extract features,

```
<objectDB>
  <Object name="Deer" >
    <description text="One hungry deer" />
    <dimensions width="0.264" height="0.193" unit="m" />
    <image location="deer.jpg" />
    <keys location="deer.key" />
  </Object>
  <Object name="Oscilloscope" >
    <description text="" />
    <dimensions width="0.326" height="0.13" unit="m" />
    <image location="oscilloscope.jpg" />
    <keys location="oscilloscope.key" />
    <overlays>
      <overlay id="0" type="abs" x="320" y="240" image="welcome.png" />
      <overlay id="1" type="float" x="500" y="320" label1="Power Button"
       label2="Press to turn on oscilloscope" />
      <overlay id="2" type="float" x="586" y="100" label1="Vertical Position"
       label2="Dial until line on display is centred" />
    </overlays>
  </Object>
</objectDB>
```

Listing 4.1: Example XML object database with two entries. Note the multiple overlay entries for the oscilloscope. Their use will become evident in Fig. 4.24.

measuring and labelling the object. The database is written in XML (Extensible Markup Language), and holds the details of each object. Listing 4.1 shows an example of a typical database. Each object has as a minimum: a name; a description; an image file name; a key file name; and its metric dimensions. If the object is to also have further AR overlays these can be listed in an overlays section. There are two types of overlay, one associated with the object's location, and one with the detection of the object. The latter is an image that is displayed on the camera image, at a fixed position, regardless of the object's location. This is useful for displaying an information screen when the object is first detected. The second type is for placing labels that are associated with particular points of interest on the object. These entries define a location on the original image, and a pair of labels associated with it. Each of the overlays has an id tag to determine the order that they should be shown.

To aid the creation of databases, software was developed for reading and writing the database XML files. The software can process an object's image for features and allows the user to add the basic information needed. Fig. 4.1 shows the software with a database loaded, and the detected SIFT features shown on an object. The AR labelling was added at a later date to the

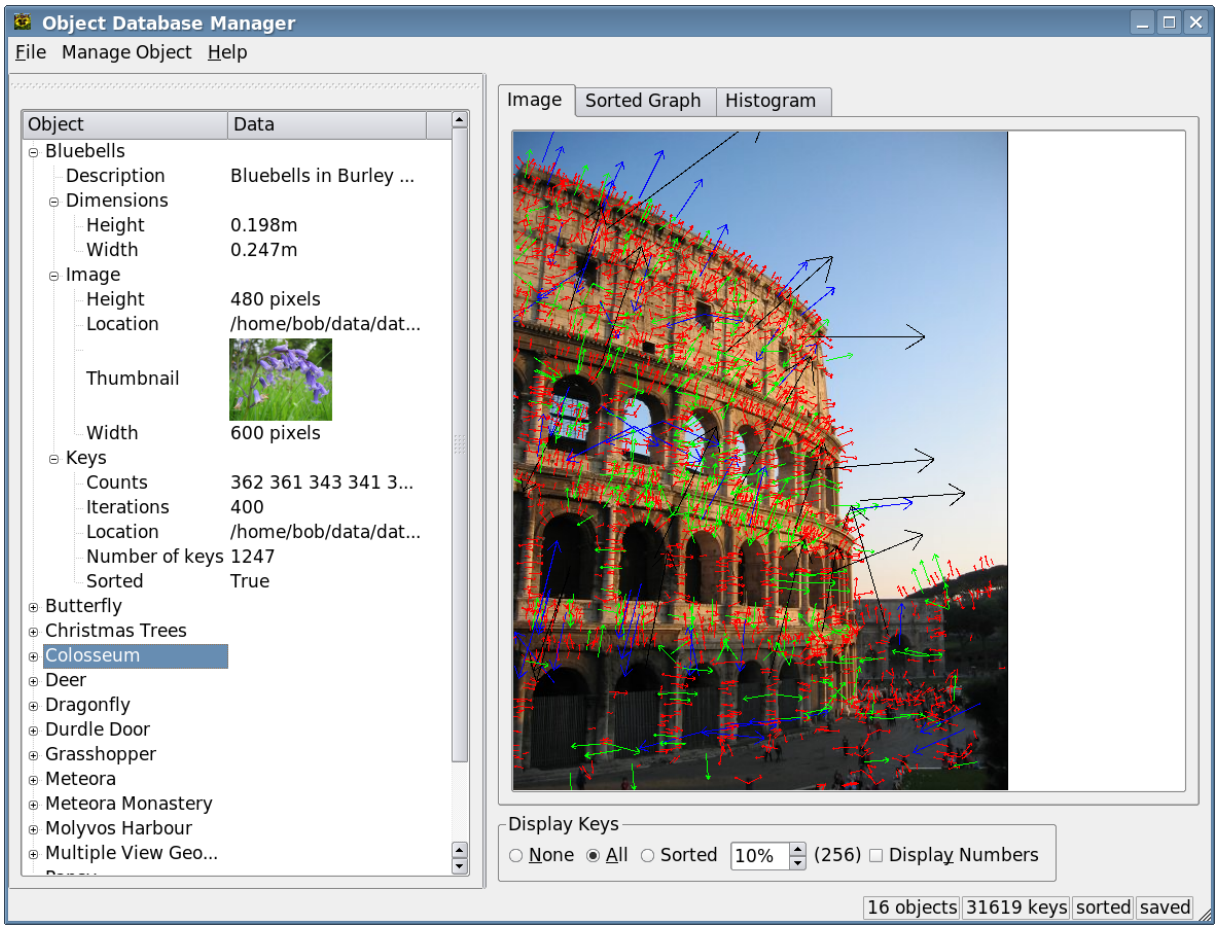
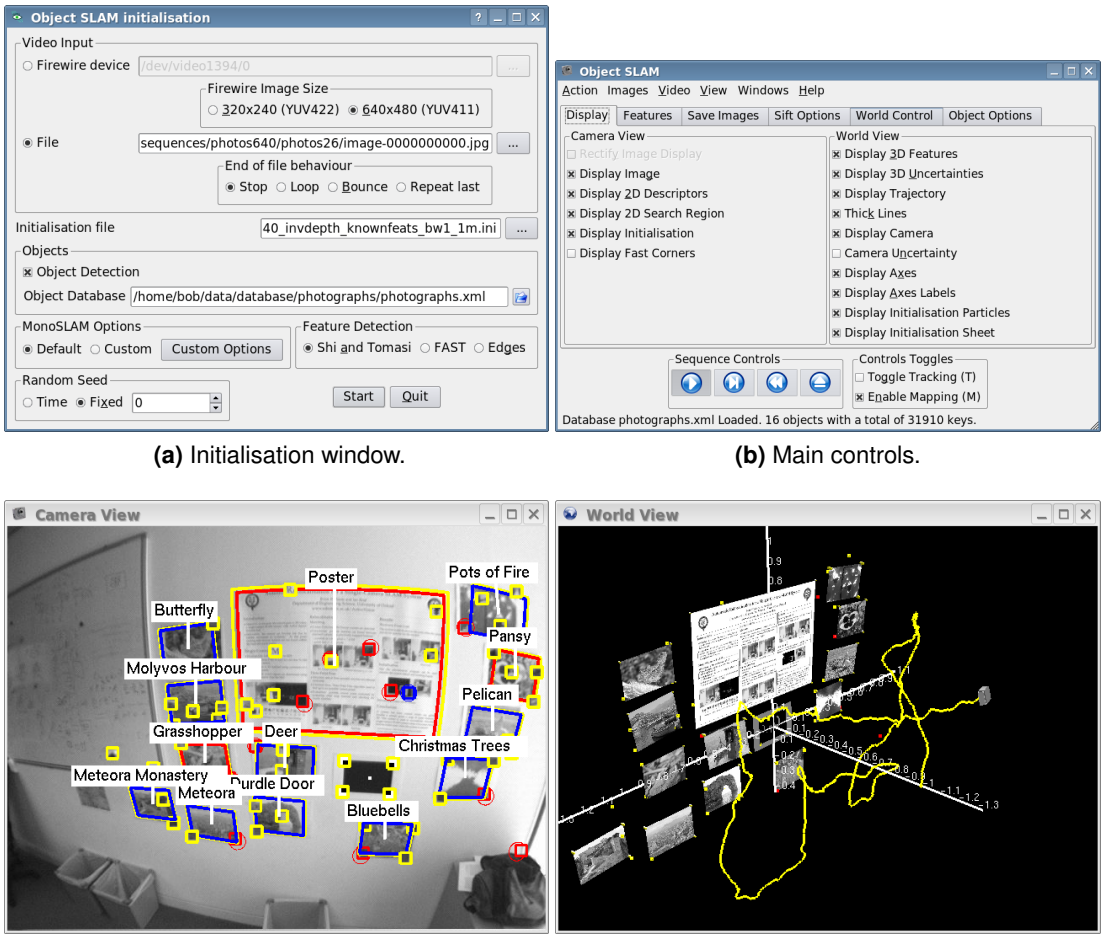


Figure 4.1: A screenshot of the object database manager software with a database loaded.

database format, and is not currently integrated into this software, instead it is done by hand. The software is written in C++ and uses the Qt3 toolkit [118] for the GUI. The SIFT feature extraction and database management is performed by a library that this application and object-SLAM application share. The SIFT code is largely based on Lowe's original, with modifications to fit it into the multiple object database framework.

4.2.2 ObjectSLAM

This is the main application, and is written in C++ and developed and operated under Linux on a dual core processor (2.20 GHz Intel Pentium Core 2 Duo). The application uses a variety of libraries, the main one being SceneLib [25] that provides the base monoSLAM framework as developed in the Active Vision Laboratory by Davison. It is a modular C++ framework that is



(a) Initialisation window. **(b)** Main controls.

(c) Camera view. **(d)** 3D map view.

Figure 4.2: Various windows from the objectSLAM software.

designed to allow new derivatives to be developed easily. Also used are VW, the Active Vision Laboratory's maths and vision library [2]; Qt3 for the GUI interfaces; OpenGL for 3D graphics visualisation [131]; and the database and SIFT libraries as described above.

Fig. 4.2 shows the various windows from the application. The initial configuration screen in (a) allows the user to select from either a live camera feed from a Firewire camera or from an image sequence on disk. They can also choose different camera calibration files; whether to run with or without object detection; and adjust a variety of monoSLAM parameters. Once the main application is running, the user is presented with (b) a controls dialog, (c) a camera view, and (d) a 3D map view. The main controls allow the user to adjust what is displayed on the output windows, record sequences, adjust various SIFT settings, and adjust various 3D

viewing positions. Other windows can also be toggled to display a variety of statistics about the running system.

4.3 Experimental evaluation

Reading Chapter 3, the version of monoSLAM used here has the following configuration: Shi-Tomasi is used as the feature detector; inverse depth features are used to represent the features in 3D; and the randomized fern method of Williams *et al.* is used to recover a lost camera. Including operating system overheads, monoSLAM takes approximately 10 ms for a 640×480 image when executing on one core with around 20 point features, leaving some 20 ms per frame to perform any further computation. Object detection and object localization is run in a separate thread on the second CPU core, continuously grabbing and processing frames.

For a typical frame, SIFT detects around 1 000 keypoints and takes on average 2 200 ms to complete when image doubling is enabled. Without image doubling SIFT detects around 500 features in 700 ms on average. Matching against a database of 5 objects containing 9 433 keypoints takes around 166 ms with doubling and 42 ms without. Hence, while the point based SLAM runs at 30 Hz, the object detection runs, at best, at around 0.43 Hz with doubling and 1.5 Hz without. These timings of course vary with the size of the database, the number of features found in a frame, and the number of objects found in the scene, making the delayed update method essential.

The first set of experiments show how the system works, and highlight the advantages and the limitations of the system. The final set of experiments show the system being used in real world situations, and the successes and failures there-in.

To aid understanding of the images shown in the results section Fig. 4.3 highlights items of interest and explains their meaning. It is also strongly suggested that the accompanying videos for each experiment are viewed to fully appreciated how the system works and runs in real time.

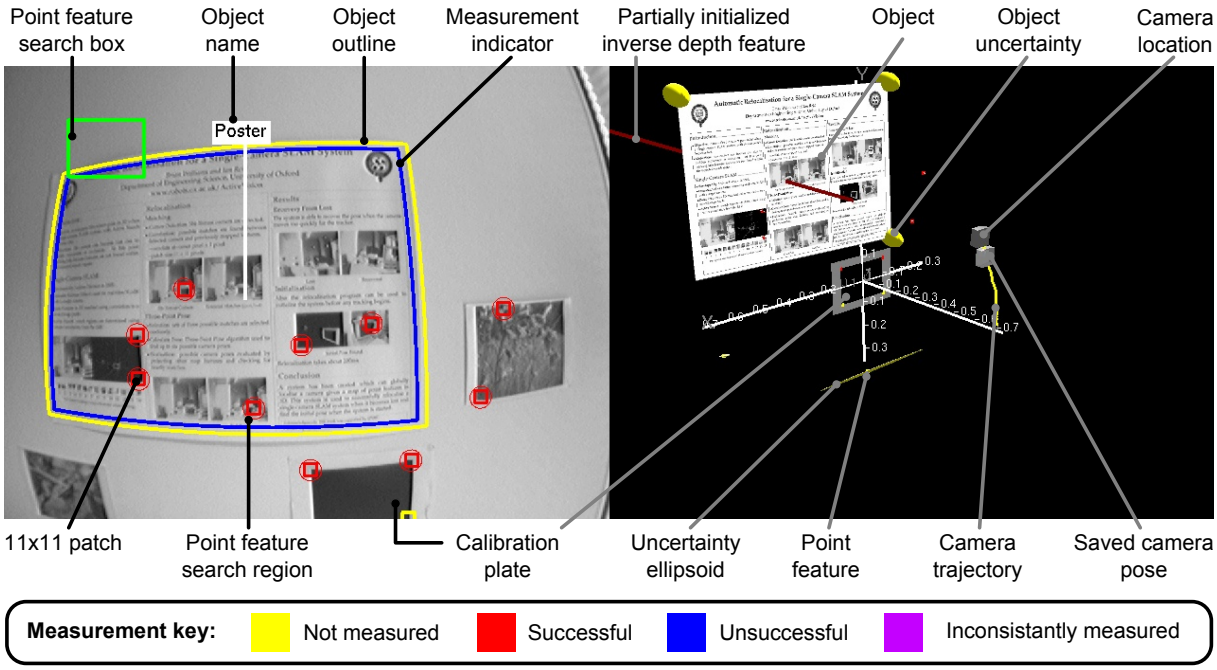


Figure 4.3: This is a typical output frame from objectSLAM. The image on the left is the camera view with graphical overlays for the user. The right hand image is the 3D map view. This shows were the features and objects are in the world relative to each other and the camera. The camera's location and trajectory can also be seen. The detected objects in the camera display have two outlines. The outermost is the actual object outline, and its colour shows its measurement status for that frame. The inner outline is a measurement indicator. Its colour shows whether the last measurement was successful or not, the boundary it describes is meaningless. As measurements occur infrequently on a single frame, this inner rectangle is useful for seeing what that measurement was.

4.3.1 Planarity and measurement error

As a first illustration of the method, a small database of five planar objects, containing 9 433 keypoints, is used. SIFT is run on each object and the keypoints generated. The information that is stored in the database for each of the objects is given in Table 4.1.

The video sequence used in this experiment will be used with the system running under four different modes. The first is with the original monoSLAM system. The second is with object detection, but with SIFT not doubling images. The third is object detection with image doubling, and the final mode is with object detection running on every frame with image doubling. The monoSLAM results will provide a reference frame for the remaining results. Modes two and three with and without image doubling allow a comparison between the two to see if image doubling is required or beneficial. The final mode, shows what the system would be capable

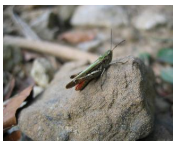
Object label	Image	Image size	No. of keypoints	Object size (mm)
Bluebells		600 × 480	1 247	246 × 198
Durdle Door		600 × 480	3 026	246 × 198
Grasshopper		600 × 480	1 362	246 × 198
Pansy		600 × 480	940	246 × 198
Poster		640 × 453	2 858	841 × 594
		Total:	9 433	

Table 4.1: Characteristics of the objects in the five object database.

of if the detection could run every frame, and represents the "gold standard". Figs. 4.4, 4.5, 4.6, and 4.7 show the evolution of processing for each of the four modes, from initial calibration of the SLAM system to the same final frame, 970 frames later.

In Fig. 4.4 it can be seen that monoSLAM tracks the scene successfully, but the information about the scene is sparse and limited to 38 point features that were detected and tracked.

Fig. 4.5 shows the same sequence, but now with object recognition enabled. As time proceeds, objects are recognized, and are added to the map as measurements. With each remeasurement of an object, its uncertainty is reduced, improving the quality of the map. This can be observed in the 3D representation with the poster where its uncertainty ellipsoids have been greatly reduced between the second and third frames in Fig. 4.5. It has also been better located, improving its vertical alignment. Without image doubling SIFT was able to run 56 times and Table 4.2 shows how many times each object was successfully detected and the angle to

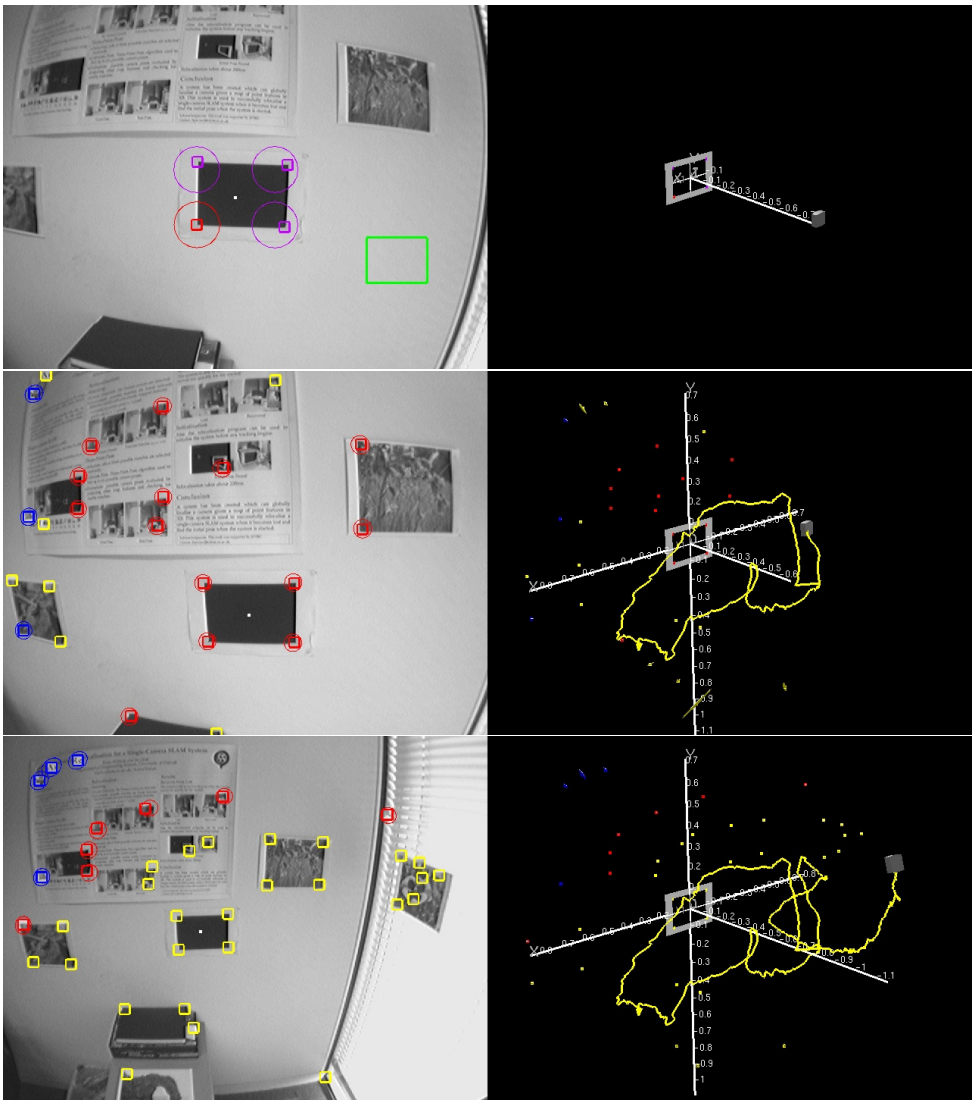


Figure 4.4: Frames from monoSLAM running on the video sequence. MonoSLAM tracks successfully, and makes a sparse map containing 38 point features (see video exp1_monoslam.avi).

the plane that the object was observed in. Along with the five detected objects the underlying monoSLAM system was also tracking 34 point features.

Repeating the experiment again, but this time with image doubling enabled for SIFT, similar results were obtained. All five objects were recognized and added to the map, but the key difference is that the image recognition took longer to process each frame resulting in fewer measurements overall. This can be seen clearly in Fig. 4.6, where the poster is detected at a much later time (frame 147 rather than frame 54). The effect that is most noticeable to the user is that objects in this sequence tend to be recognized just as they go out of view. This is hardly

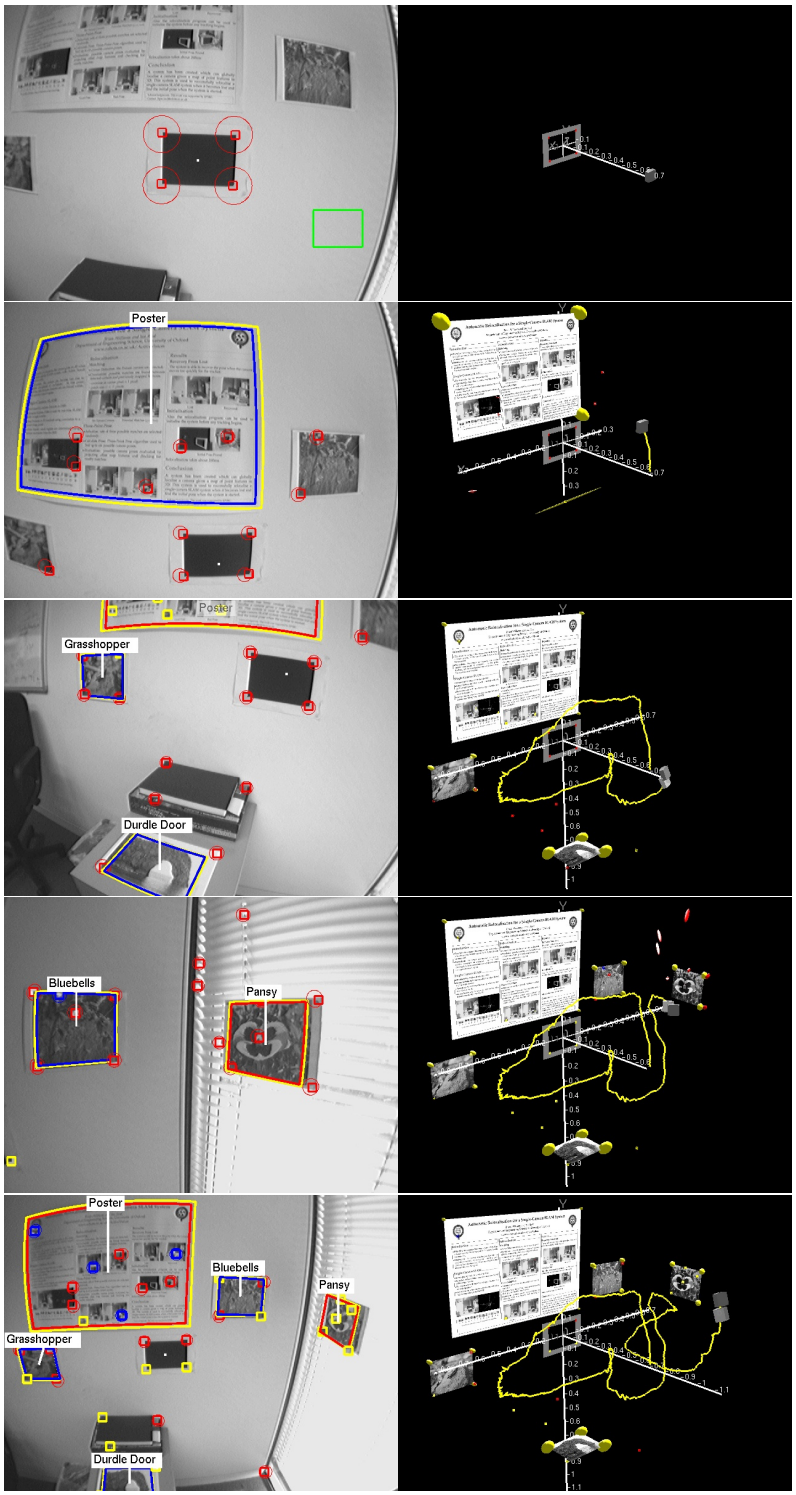


Figure 4.5: Frames from objectSLAM running on the video sequence without image doubling. From top to bottom: the initial image with calibration plate visible in the map; the first object is detected; more objects are detected and added to the map; all objects have been detected and successfully localized (see video exp1_objectslam.avi).

Object	No. of detections			Angle to plate (°)			
	No doubling	Doubled	All frames	Actual	No doubling	Doubled	All frames
Bluebells	5	2	152	0	2.8 ± 6.2	6.4 ± 9.0	2.0 ± 1.2
Durdle Door	1	1	48	90	89.9 ± 14.6	87.4 ± 12.5	84.8 ± 2.4
Grasshopper	7	2	142	0	4.7 ± 8.1	6.1 ± 13.8	4.6 ± 2.2
Pansy	6	1	108	90	93.7 ± 6.2	93 ± 4.3	91.1 ± 1.2
Poster	24	5	515	0	2.8 ± 1.2	6.2 ± 6.6	0.7 ± 0.5
No. frames processed	56	15	970				

Table 4.2: The number of times each object was successfully measured (enough matches and a valid homography found using RANSAC), and the final angle to calibration plate for running object SLAM with and without SIFT image doubling and running SIFT on every frame.

desirable as it means that the user would have to either move the camera at a slower speed, or keep going back to view what has just been detected. With image doubling SIFT was able to run 15 times and Table 4.2 shows how many times each object was successfully detected and the angle to the plane that the object was observed in. Along with the five detected objects the underlying monoSLAM system was also tracking 35 point features.

The final mode has objectSLAM running every frame with image doubling: the "gold standard", as every frame is checked for objects using the best possible detection. This mode is currently impossible to run in real time, but provides a useful comparison with the other two object detection modes. Fig. 4.7 shows a selection of frames from the results of this mode. The poster object was detected much sooner (frame 21), and by the time the poster had been initially detected in the no doubling mode (frame 54) a second object has also be detected, as can be seen in the second image shown in Fig. 4.7. SIFT ran on all 970 frames of the sequence and Table 4.2 shows how many times each object was successfully detected and the angle to the plane that the object was observed in. 33 point features were also tracked. One anomaly that can be seen in the third image in Fig. 4.7 is what appears to be a measurement error with the object detection of the Durdle Door poster. It is shifted to the right by a considerable amount. Study of the sequence in full shows that it is located correctly initially and measured for a few frames, and then not observed again. While it is not being observed, monoSLAM incorrectly

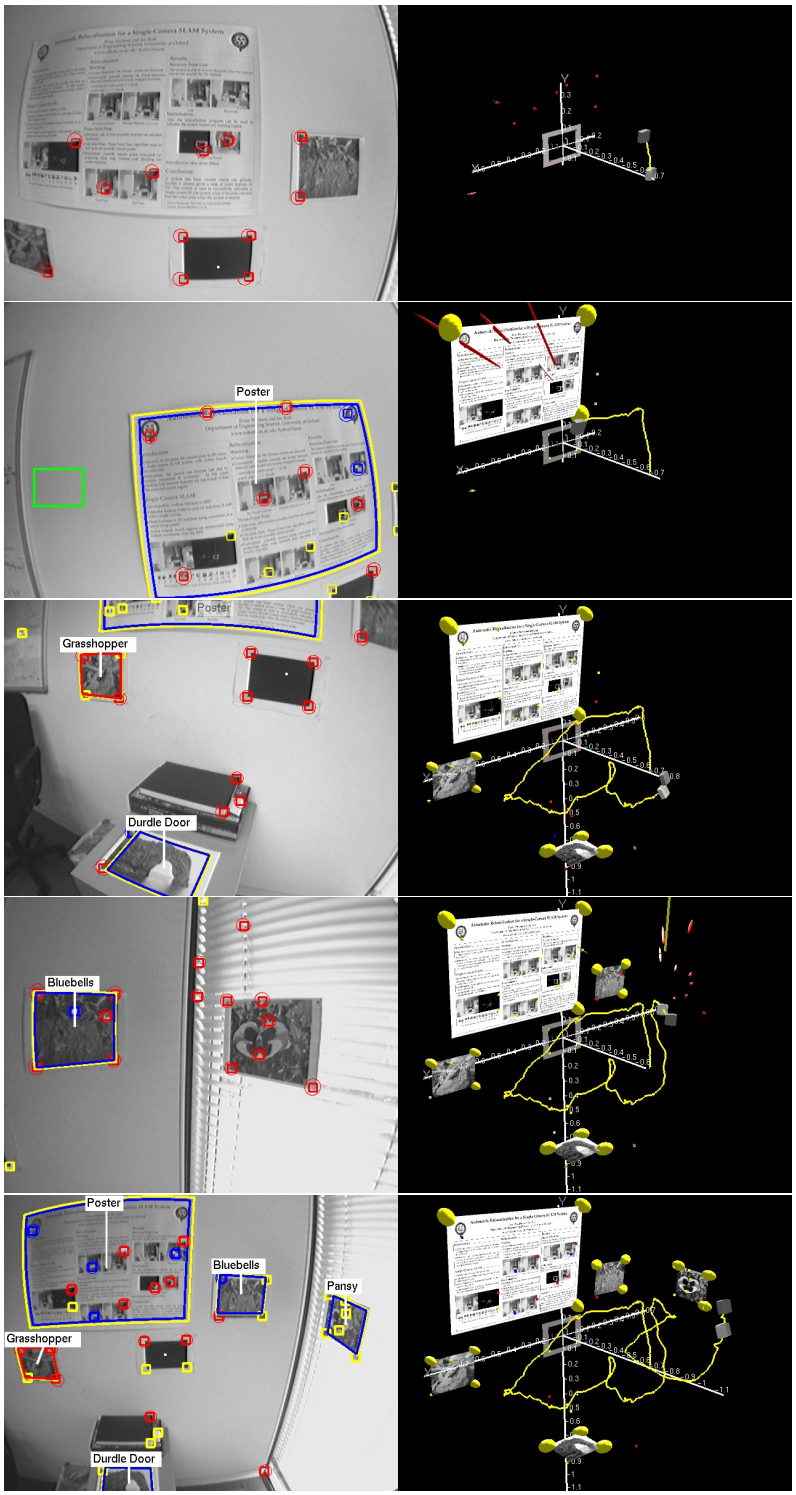


Figure 4.6: Frames from objectSLAM running on the video sequence with image doubling. From top to bottom: frame where the first object was detected without image doubling; the first object is detected; more objects are detected and added to the map; all objects have been detected and successfully localized (see video exp1_objectslam_double.avi).

locates the feature on the top left of the Durdle Door image, locating it further to the right that it should be in frame 455. The movement of just this single feature moves the estimate of the camera's location slightly left, misaligning the object outline. This happens again on the next frame (no. 457) shifting the outline further to the right. As the camera pulls up to observe more of the scene, the feature continues to slide around the top of the Durdle Door poster. Eventually it settles down in the correct location, and the outline once again appears in the correct location. Because no objects were observed in these frames the error goes uncorrected. This highlights how even with object detection running on every frame, if objects are not observed and monoSLAM makes a mistake, the map can be corrupted.

Doubling the input image to SIFT yields more features and better feature detection results [85]. However, for a real time application there is always a trade off between quality and speed. With image doubling enabled, the object detection runs around three times slower, which in this sequence meant that the Durdle Door and Pansy posters were only measured once and the Bluebells and Grasshopper were only measured twice. By the time the object recognition had completed one frame the camera had moved a significant distance away, so any objects that had been visible were no longer in view. The main benefit of image doubling is for objects that are smaller or further away than can be detected without the camera having to move closer to them. However, if an object only appears in frame while the object recognition is running on a previous frame, the object will not be detected and the benefit is lost.

In contrast, without image doubling all of the objects are detected five or more times, except Durdle Door, and all of the objects are localized with more certainty. Durdle Door was only measured successfully once: most likely due to the too brief clear view of the object.

Table 4.2 compares this matching data and also shows how well aligned the objects are to the plane that they are supposed to lie in. All of the objects for all modes are in their respective planes within experimental error. It can be seen clearly that the extra observations permitted by not doubling the image result in better localized objects.

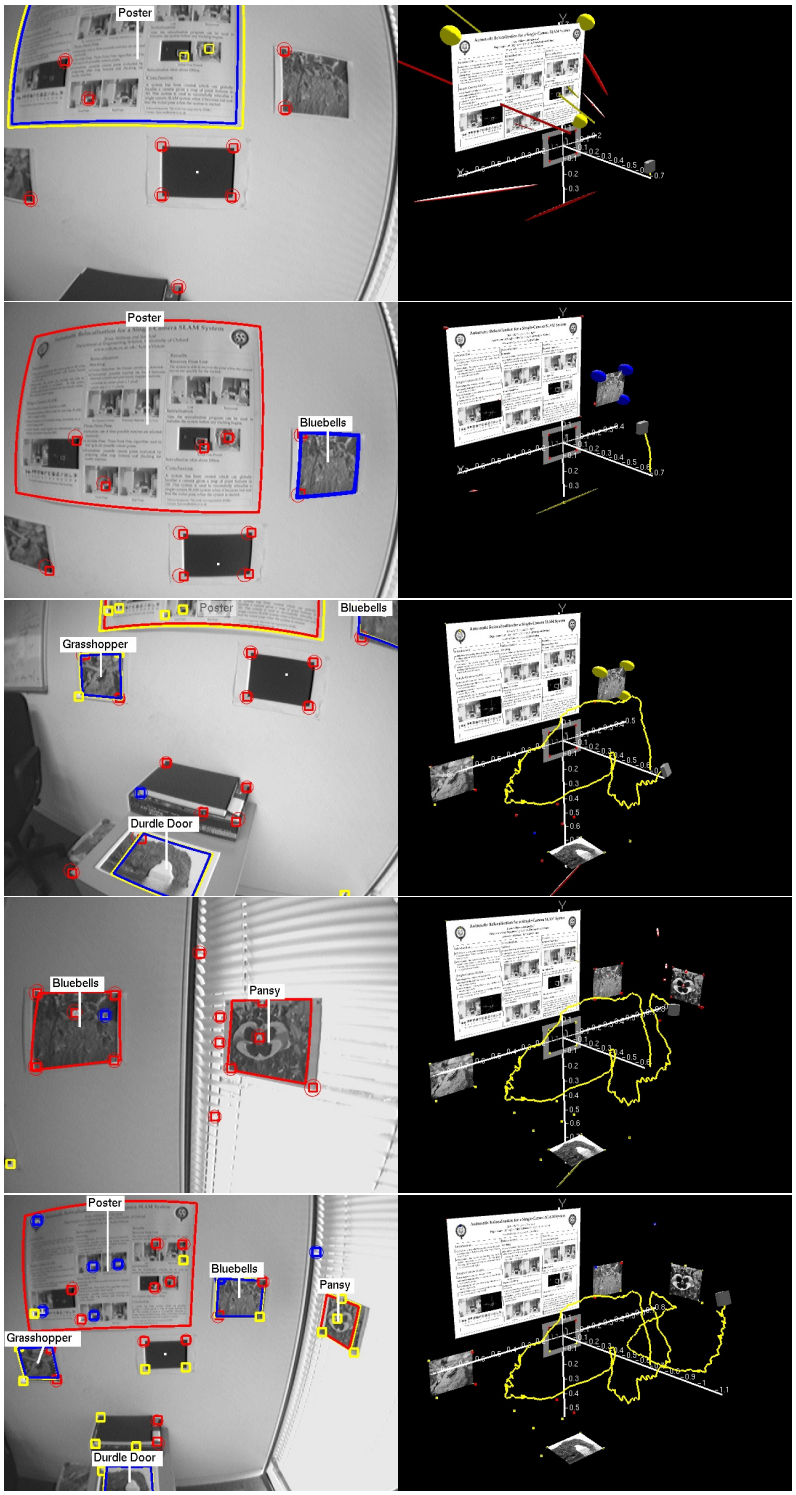
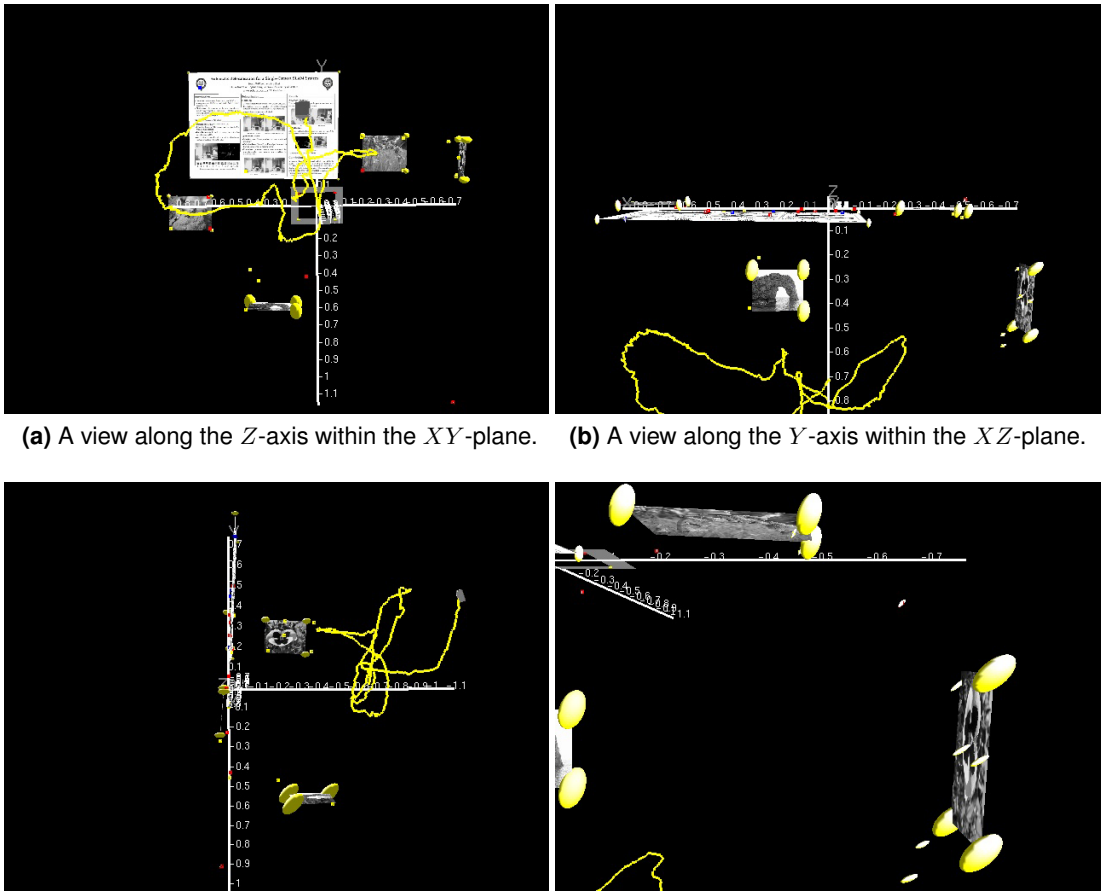


Figure 4.7: Frames from objectSLAM running SIFT on every frame from the video sequence. From top to bottom: the first object is detected; more objects are detected and added to the map; monoSLAM measurement error, causes the outline of Durdle Door to apparently shift location; all objects have been detected and successfully localized (see video exp1_objectslam_allframes.avi).



(a) A view along the Z-axis within the XY-plane. **(b)** A view along the Y-axis within the XZ-plane.

(c) A view along the X-axis within the YZ-plane. **(d)** A view showing the corner detail of two objects.

Figure 4.8: Various 3D views around the map obtained for the object recognition without image doubling.

Fig. 4.8 shows three graphics views around the recovered 3D map from the no doubling mode. Three of the objects (Poster, Bluebells, Grasshopper) should be coplanar with the calibration plate (and hence in the XY-plane), and the other two (Pansy, Durdle Door) should by mutually orthogonal and in the ZY and XZ planes respectively.

For the rest of the experiments object recognition is used without image doubling.

4.3.2 Scaling with number of objects

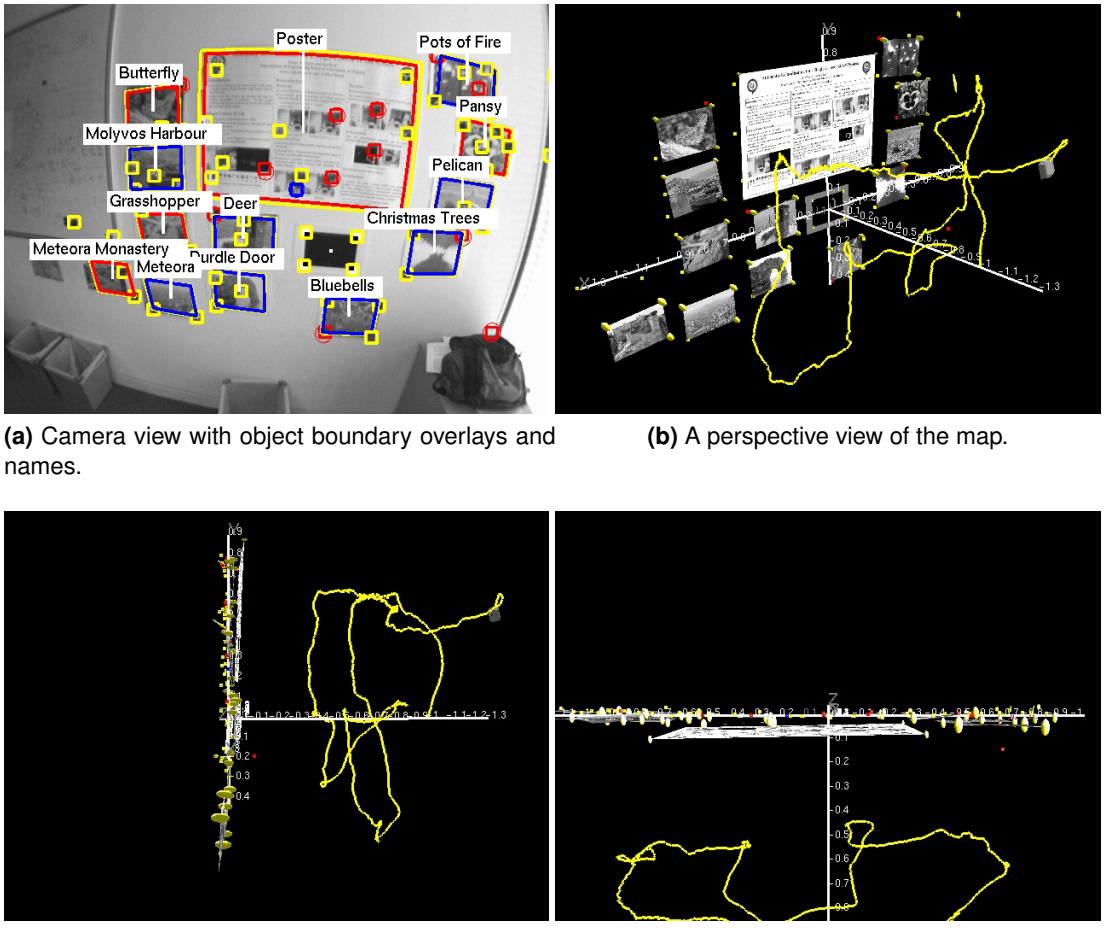
In this experiment the size of the database is increased to 16 objects with a total of 31 910 keypoints, increasing the average database search time to around 75 ms. The wall in the sequence contains 13 of these objects, and the object recognition was able to run 103 times, and to detect

Object	No. of measurements	Angle to plate (°)
Bluebells	9	3.7 ± 5.9
Butterfly	12	0.9 ± 5.3
Christmas Trees	6	6.0 ± 5.2
Deer	6	2.6 ± 8.3
Durdle Door	7	2.2 ± 6.5
Grasshopper	12	7.5 ± 5.8
Meteora	5	6.6 ± 7.3
Meteora Monastery	4	3.4 ± 8.8
Molyvos Harbour	12	7.4 ± 0.5
Pansy	14	2.4 ± 5.1
Pelican	13	2.7 ± 6.8
Poster	42	2.8 ± 1.4
Pots of Fire	6	2.2 ± 7.4

Table 4.3: It can be seen from this table that each object was successfully measured multiple times and all are localized within the plane of the wall within experimental error.

and localize all of them correctly. Each of the 13 objects in the scene was found multiple times as shown in Table 4.3. Fig. 4.9 shows a selection of views around the map and the camera view with the object AR overlays. By having this many objects in the map, the number of point features that can be tracked while maintaining 30 Hz operation reduces. MonoSLAM can perform the EKF updates at 30 Hz with a maximum of around 100 point features. Each object is equivalent to 3 point features, because of the 3 point representation used. This means that the 13 objects are equivalent to 39 point features. By then end of the sequence 68 point features are in the map along with the 13 objects. This represents the practical limit for the number of objects that can be tracked simultaneously. The more objects that are tracked, the sparser the map has to become, and enough detail needs to be maintained for the camera to track successfully between object measurements. This effectively means that either a small map can be made around a very dense cluster of objects, like the wall of posters shown here, or a larger map with only a few objects present.

As the size of the database increases the search time will continue to increase. The rate will be less than linear, because of the kd-tree based method being employed. Fig. 4.10 shows how the processing time varies as the size of the database increases. The processing time is for an average frame containing 483 features. The search times are for databases varying in size from



(a) Camera view with object boundary overlays and names.

(b) A perspective view of the map.

(c) A view along the X-axis within the YZ-plane.

(d) A view along the Y-axis within the XZ-plane.

Figure 4.9: (a) Final AR view and (b-d) various 3D views around the map (see video exp2_scaling.avi).

1 object with 1 247 keypoints to 53 objects with a total of 106 071 keypoints. The processing time, even for a large database is still small compared to the time taken to run SIFT on a frame.

The limiting factor, therefore, that will restrict the number of objects that can be detected is not the size of the database, but the number of objects that can be in the map at any one time.

4.3.3 Occlusion and depth issues

In this third experiment, the sequence shows the camera exploring a more visually cluttered desk top environment. A database of 16 planar objects with a total of 31 910 features is used, but only a subset of these objects appears in the scene. Fig. 4.11, shows frames from the video and Fig. 4.12 shows various views around the recovered 3D map in which there are four picture

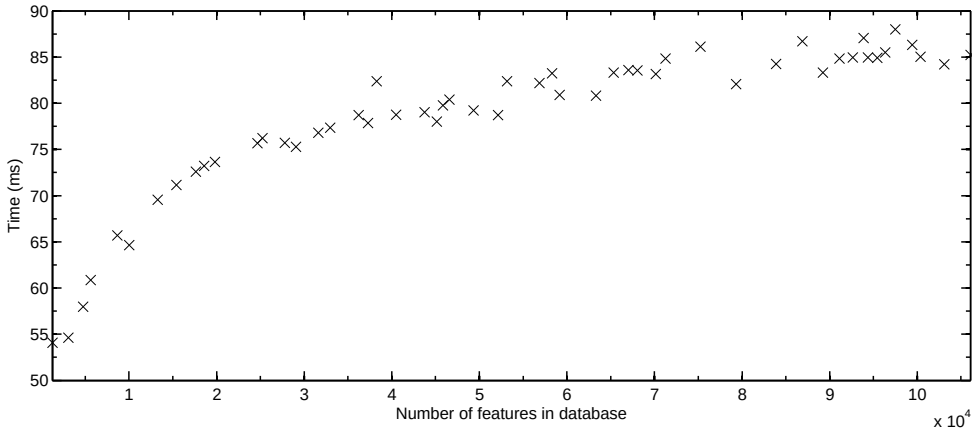
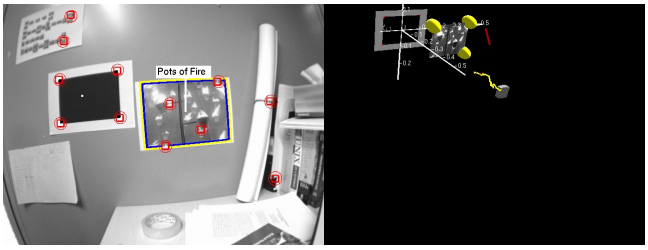


Figure 4.10: As the size of the database increases, the matching time also increases. The graph shows the average time to match the keypoints in an average frame containing 483 keypoints to databases of varying numbers of objects, ranging from 1 object (1 247 keypoints) to 53 objects (106 071 keypoints).

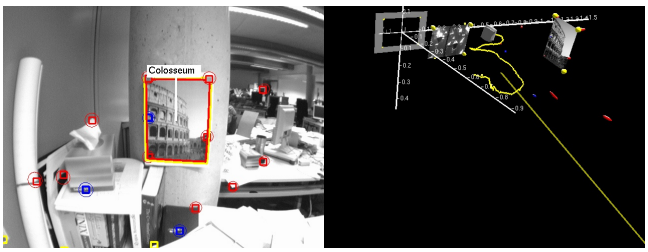
objects, one of which (Pots of Fire) should be coplanar with the calibration plate (and hence in the XY-plane), two of which (Multiple View Geometry, Grasshopper) are in the XZ plane and the final object (Colosseum) is in the ZY plane. All the objects were recovered within their respective planes to within experimental error (Colosseum 92.1$\pm$6.1°, Grasshopper 87.2$\pm$2.7°, Multiple View Geometry 84.7$\pm$2.5°, Pots of Fire 6.5$\pm$7.5°). Tuning the performance to the size of the covariance suggest that the lateral and depth errors are of order 10 mm and 20 mm respectively.

SIFT is still able to detect objects that are partially occluded, and if enough features are correctly detected then the pose of the object can be recovered. In this example, the Multiple View Geometry book is successfully detected and its position recovered despite it being partially occluded.

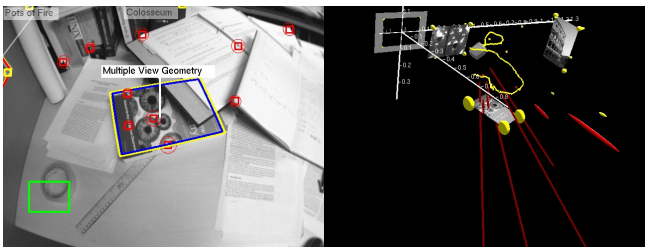
The first three objects are located successfully, and their overlays displayed around the object borders in the camera view. The overlay for the final object (Grasshopper), however, appears to be incorrectly placed, and moves around as the camera moves. While it has been correctly located in terms of its plane, its depth appears to be incorrect (see Fig. 4.12d, and the accompanying video). This problem is not due to the object detection and localization algorithm, but to the underlying monoSLAM algorithm. With a single camera there is a speed/scaling am-



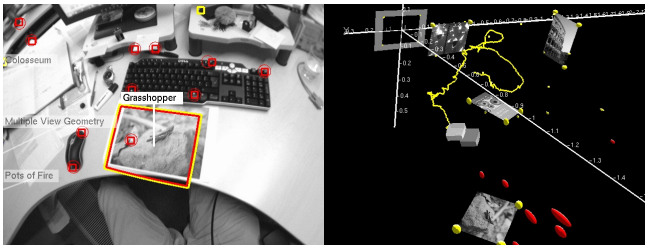
(a) First object, Pots of fire, is added to the map.



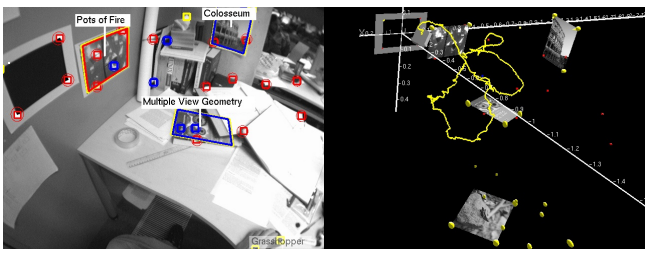
(b) Second object, Colosseum, added to the map.



(c) Third object, Multiple View Geometry, has been added to the map.

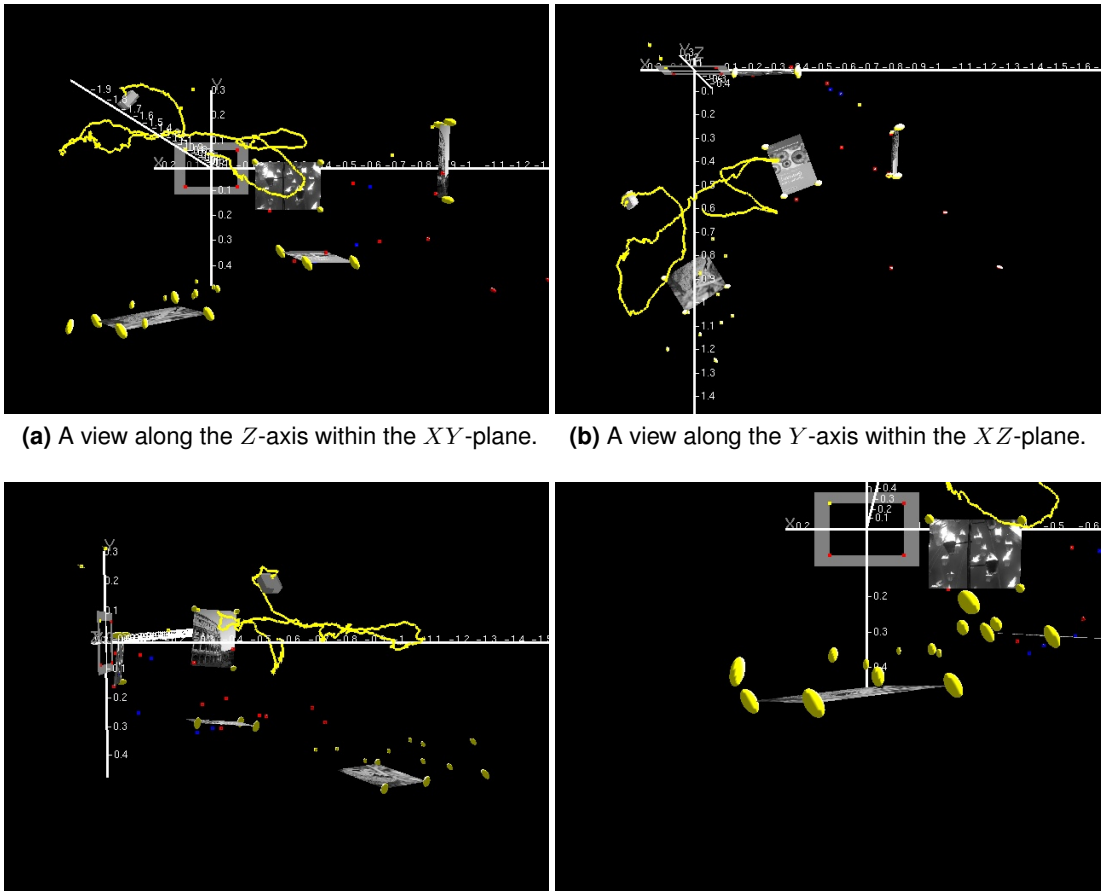


(d) Final object, Grasshopper, located. Overly appears incorrect due to depth issues with the point features.



(e) All objects have been detected and localized.

Figure 4.11: Frames from the cluttered desk top sequence running from top to bottom with the camera view shown on the left and the map on the right (see video exp3_occlusion.avi).



(a) A view along the Z-axis within the XY-plane. **(b)** A view along the Y-axis within the XZ-plane.

(c) A view along the X-axis within the YZ-plane. **(d)** A close up view of the Grasshopper object.

Figure 4.12: Various 3D views of the map of the cluttered desk top.

biguity that needs to be resolved, the points in the scene may be close to the camera and the camera moving slowly or they may be far away and the camera moving fast. Also, the near pure rotation that the camera undergoes prior to observing the object results in poorly located features, as no depth information is recoverable. These poorly localized features continue to be measured as being closer to the camera than the previous measurement, and creep towards it. By the end of the sequence features that should be at the same depth as the Grasshopper object end up being above it. Without object recognition, monoSLAM can survive these measurement errors, but the map does become distorted. With object recognition, objects either end up appearing to be at the wrong depth, or if their measurement is timely, *i.e.* when point features around the object have not been fully initialized, they help enforce the correct depth upon the features. Owing to the small number of features that monoSLAM can measure per

frame, a poor measurement of depth has occurred for the new features around the Grasshopper poster. This error becomes 'baked in' with each measurement of the features, which will be measured almost every frame while in view. Whereas the object will only be measured a few times, meaning that its more accurate measurement has a low effect of the map due to these errors. If the object was measured with a similar frequency to the features then its effect upon the map would be greater. This issue cannot be resolved by a global rescaling of the map, as the problem is only local to a few features. The calibration plate used at the start ensures that the scale is correct, but as the camera moves further away from it errors begin to accumulate.

4.4 Ashmolean gallery - real world scenario I

This experiment takes the objectSLAM system out of the laboratory environment and into a real gallery in Oxford's Ashmolean Museum of Art and Archaeology. The aim of this experiment is to explore as much of the gallery as possible and detect and label the observed paintings correctly. The gallery database has 37 paintings with a total of 74 452 features. The size of the area that can be explored will be severely limited by the size of map that objectSLAM can handle in real time, but this is a known limitation with the underlying monoSLAM system. The main goal is to detect the paintings and locate them in a map accurately.

4.4.1 Practical problems

Three practical problems were found with this sequence. The first problem encountered was in measuring the paintings to obtain their dimensions for the database. Understandably the gallery does not allow people to touch the paintings, let alone put a ruler up against them. This meant that each painting needed to be photographed with an object of known size in the scene. The inaccuracies in such measurement can be large, as each painting is not a set depth away from the wall, and each leans forwards by different amounts.

A second difficulty in the gallery is that of repeating textures, predominantly (to the eye at

least) on the wallpaper, but also on the frames of the paintings causing mismatching. The best case is that features found on repeating texture will have difficulty in being matched and end up being deleted. The worst case is that features will be incorrectly matched, and corrupt the map. In practice, and unexpectedly, the texturing on the wallpaper gave rise to very few features and it was those on the frames that caused corruption, as can be seen in the results. This is a limiting factor in monoSLAM: with few features it is hard to tell when a feature measurement is inconsistent. With many more features these inconsistent measurements could be detected.

The third problem is monoSLAM's $O(n^2)$ scaling. Because the processing has to be done before the next frame arrives, the state vector can only become so large. For the machine used in these experiments around 100 features can be managed in the state. This limits how far the camera can travel, and in this sequence monoSLAM is able to track only one wall of the gallery. When objects are added to this, the maximum size decreases by three features per object as discussed before.

4.4.2 Running without calibration

With a single camera there is a depth/speed scaling ambiguity, and using a calibration plate sets scale, resolving this ambiguity. When a calibration plate is not used, the scale becomes set by the choice of process noise for acceleration in the constant velocity EKF. If the depths and speeds are under-estimated in the EKF's state, the filter will place too much weight on current measurements. The scales therefore becomes essentially arbitrary. The calibration plate could not be used in the gallery, because modification of the galley was not permitted. Fig. 4.13 shows frames from the sequence with just monoSLAM running, and Fig. 4.14 shows views around the map that was created. It can be seen that as the camera moves further away from the start, the scale, and the camera acceleration begins to be incorrectly estimated and increases. The uncertainty in the features also increases the further they are away from the start. This is an issue with EKF based SLAM because the uncertainty of the features and camera are all coupled

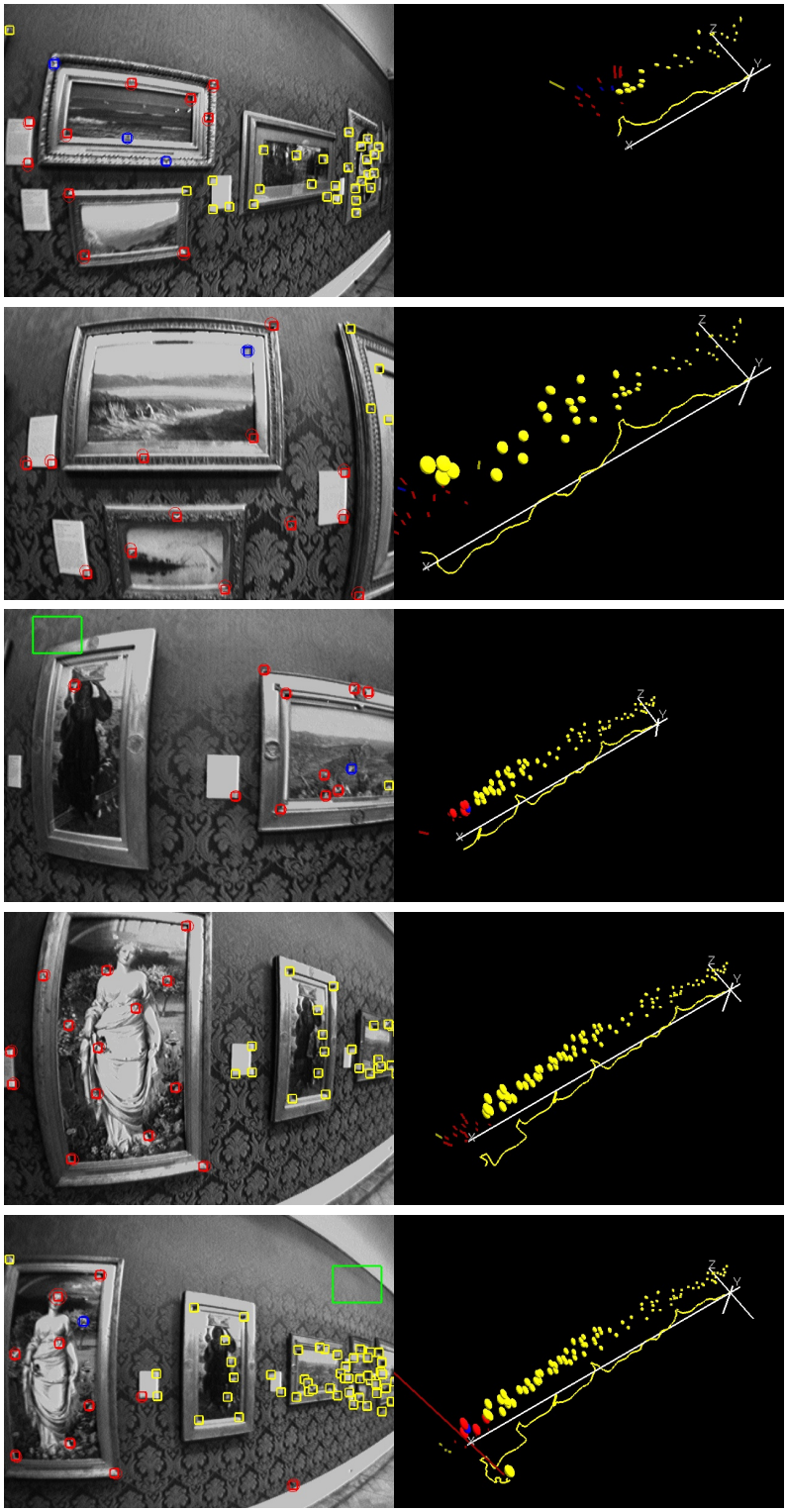
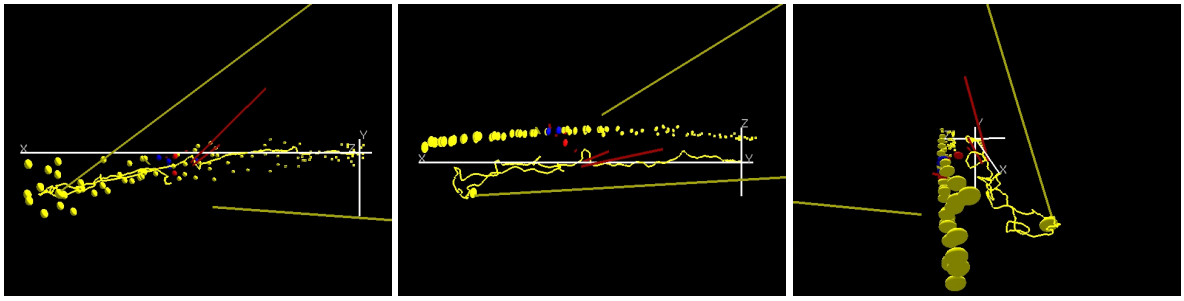


Figure 4.13: Frames from the Ashmolean gallery sequence with monoSLAM (see video exp4_gallery_monoslam.avi).



(a) A view along the Z-axis within the XY-plane.

(b) A view along the Y-axis within the XZ-plane.

(c) A view along the X-axis within the YZ-plane.

Figure 4.14: Various 3D views around the monoSLAM gallery map.

and add to each other. The system is able to track the length of the wall, but eventually becomes lost when the camera starts to return down the wall. By the end of the sequence the system was tracking 104 features.

When the sequence is run with objectSLAM the detection of the first painting helps set a scale for the partially initialized features, placing them at the correct depth when they become fully initialized. The second painting helps enforce this scale. This can be observed in the frames from the video shown in Fig. 4.15. The subsequent features that are added to the map as the camera explores away from the first painting, also become initialized at the correct depth. However, as the camera moves further away, the scale begins to drift, and features are incorrectly measured causing the camera's estimated position to be incorrect. By the time the third painting is detected the maps is becoming distorted and curving away from where it should be. As the system continues, further incorrect feature measurements are made and the curving and twisting of the map occurs. Paintings located during this are localized well locally to the point features, but like the point features, their global position is inaccurate. Fig. 4.16 shows a variety of views around the map that was created.

During the sequence ten paintings were clearly observed, seven of them were successfully recognized. The three that were not detected were either too small for SIFT to match enough features to, or too featureless with repeating texture to be matched correctly.

Comparing the trajectories of the monoSLAM and the objectSLAM (Fig. 4.17) it can be seen

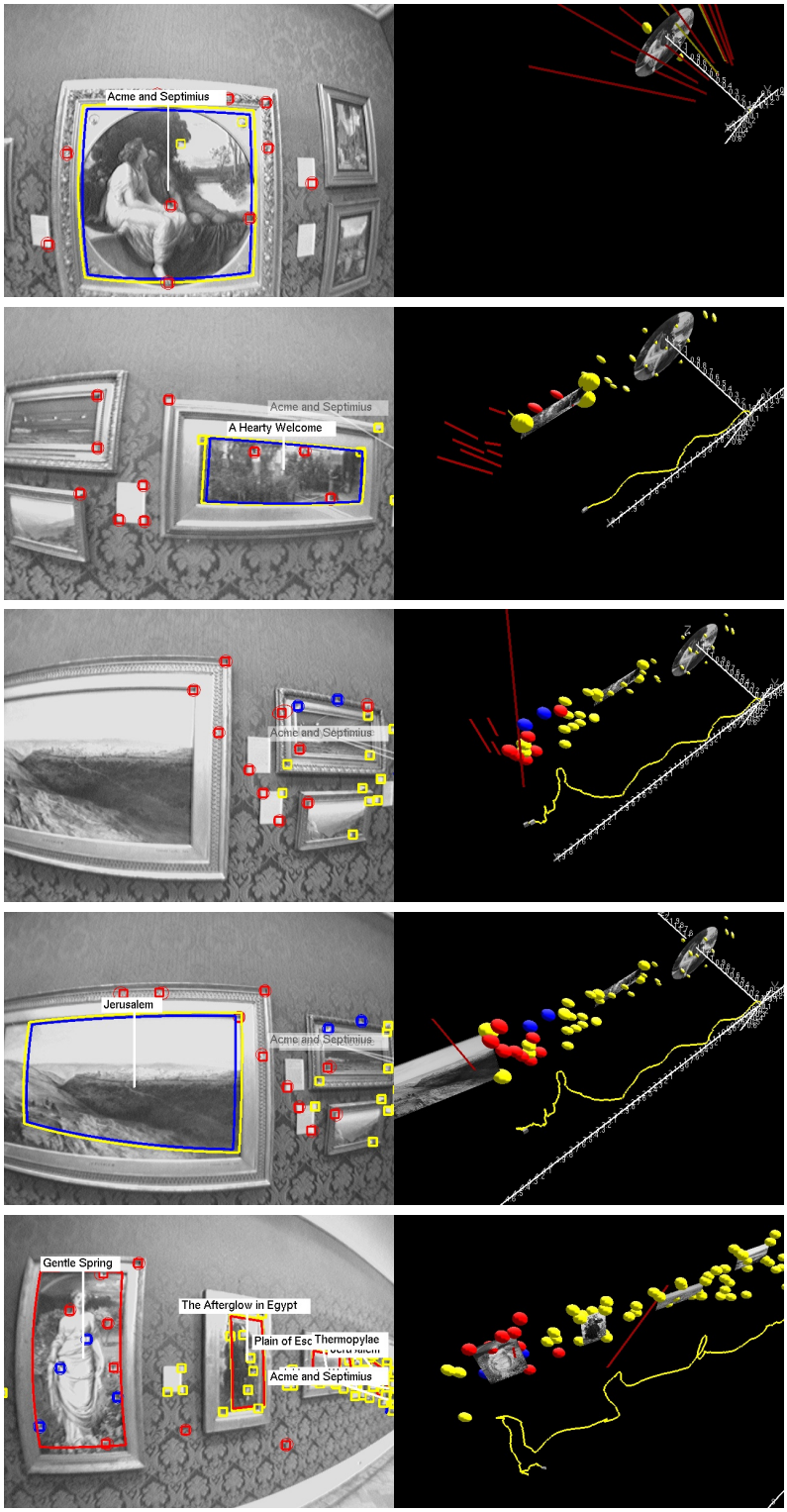
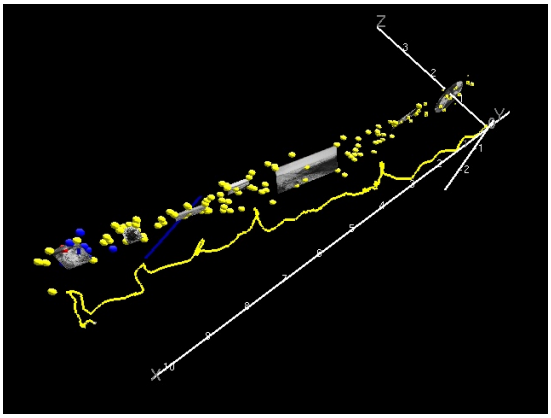
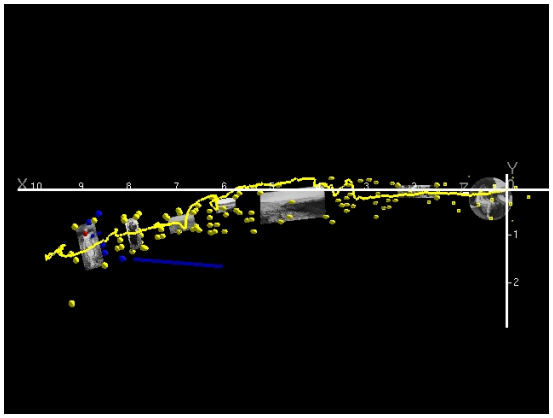


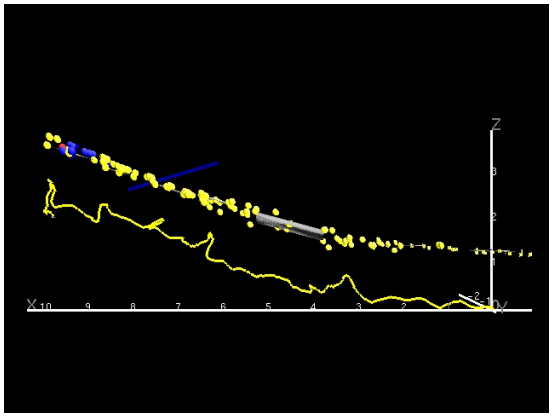
Figure 4.15: Frames from the Ashmolean gallery sequence with object recognition (see video exp4_gallery_objectslam.avi).



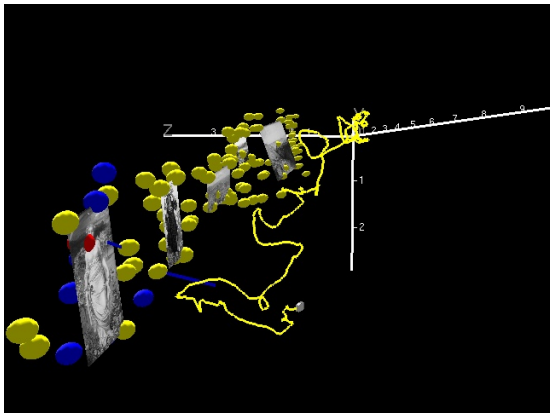
(a) A perspective view of the map.



(b) A view along the Z-axis within the XY-plane.



(c) A view along the Y-axis within the XZ-plane.



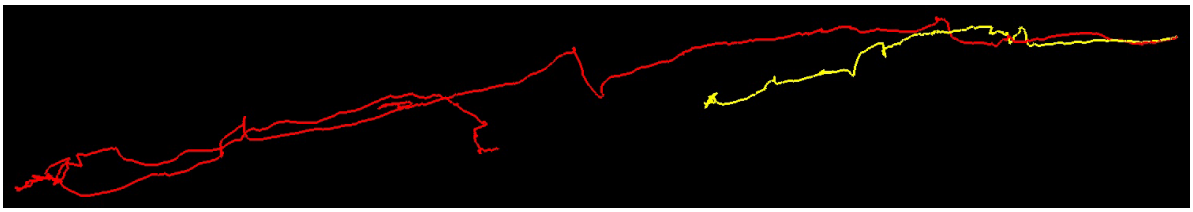
(d) A view along the X-axis within the YZ-plane.

Figure 4.16: Various 3D views around the objectSLAM gallery map.

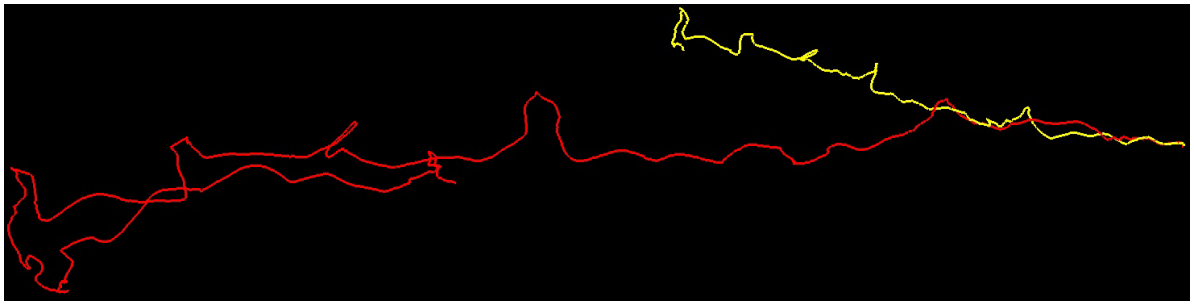
how detecting the paintings constrains the unbounded scaling that occurred with monoSLAM. The same general shape can be seen in both trajectories, and both curve downwards. The monoSLAM trajectory is straighter than the objectSLAM one, this is due to the bad measurements during the objectSLAM run that distorted the map.

4.4.3 Summary

Taking the system out of the laboratory and into a real gallery highlighted a significant limiting factor with the developed method. Requiring an accurate metric measurement of all objects beforehand is not always practical. For some scenarios it is, but for this one it was not. It also highlighted how object recognition can help enforce scale into an uncalibrated system.



(a) A view along the Z-axis within the XY-plane.



(b) A view along the Y-axis within the XZ-plane.

Figure 4.17: Two 3D views of the camera trajectories. Red is the monoSLAM trajectory, and yellow is the objectSLAM trajectory.

4.5 Street scene - real world scenario II

This second real world scenario takes the system outdoors to recognize shop fronts. Eight shop fronts with a total of 11 359 keypoints were measured and added to a database. This is a particularly challenging scenario, for several reasons. First, vehicles and pedestrians passing in front of the camera obscure the scene causing features to either be not measured or measured incorrectly, which can either break the tracking, causing the system to fail, or corrupt the map. Secondly, the varying lighting also changes the appearance of features and objects significantly making tracking and object recognition more challenging. Thirdly, shops change their fronts regularly, which means that over time the images in the database may not match what is seen in the real world. Two examples of this are shown in Fig. 4.18.

The speed/scaling ambiguity discussed earlier became very prevalent here, due to the large depths involved. A significant translation of the camera yielded a small change visually, which the underlying monoSLAM system can interpret incorrectly as a small translation with the points close to the camera. This can cause the features in the map to tend to creep towards the

(a) Shop successfully matched.

(b) Shop unsuccessfully matched.

(c) Restaurant successfully matched.

(d) Restaurant unsuccessfully matched.

Figure 4.18: The left hand images are the database images, and the right hand images are frames from captured sequences. The building fronts were successfully matched in sequences captured on the same day that the database images were taken, but not a couple of months later when fronts had changed. However, even on the successful matches the match counts are low, and not all correct.

camera until the map effectively collapses. The failure would have been prevented if a shop front had been detected early on, effectively pinning the point features to the same depth, as occurred in the gallery sequence.

Starting with and without a calibration plate affected the results dramatically. Beginning with monoSLAM and using the calibration plate the results in Fig. 4.19 were obtained. The scale is fixed at the start, and features detected around the same depth are localized correctly. However, the features across the street do not become localized correctly, due to the speed/scaling ambiguity. Instead they creep forwards until the map collapses. When the calibration plate is not used an arbitrary depth is set and the features across the street dominate the map, enforcing the depth that supports their measurements. Fig. 4.20 shows the results obtained.

When objectSLAM is run on the same sequence, similar results occur when the calibration plate is used. The point features are measured much more regularly than objects, so the map collapses before the objects make any significant impact. This can be clearly observed in Fig. 4.21. However, when the calibration plate is not used the detected shop front enforces a scale upon the partially initialized features. Fig. 4.22 shows the evolution of the map and

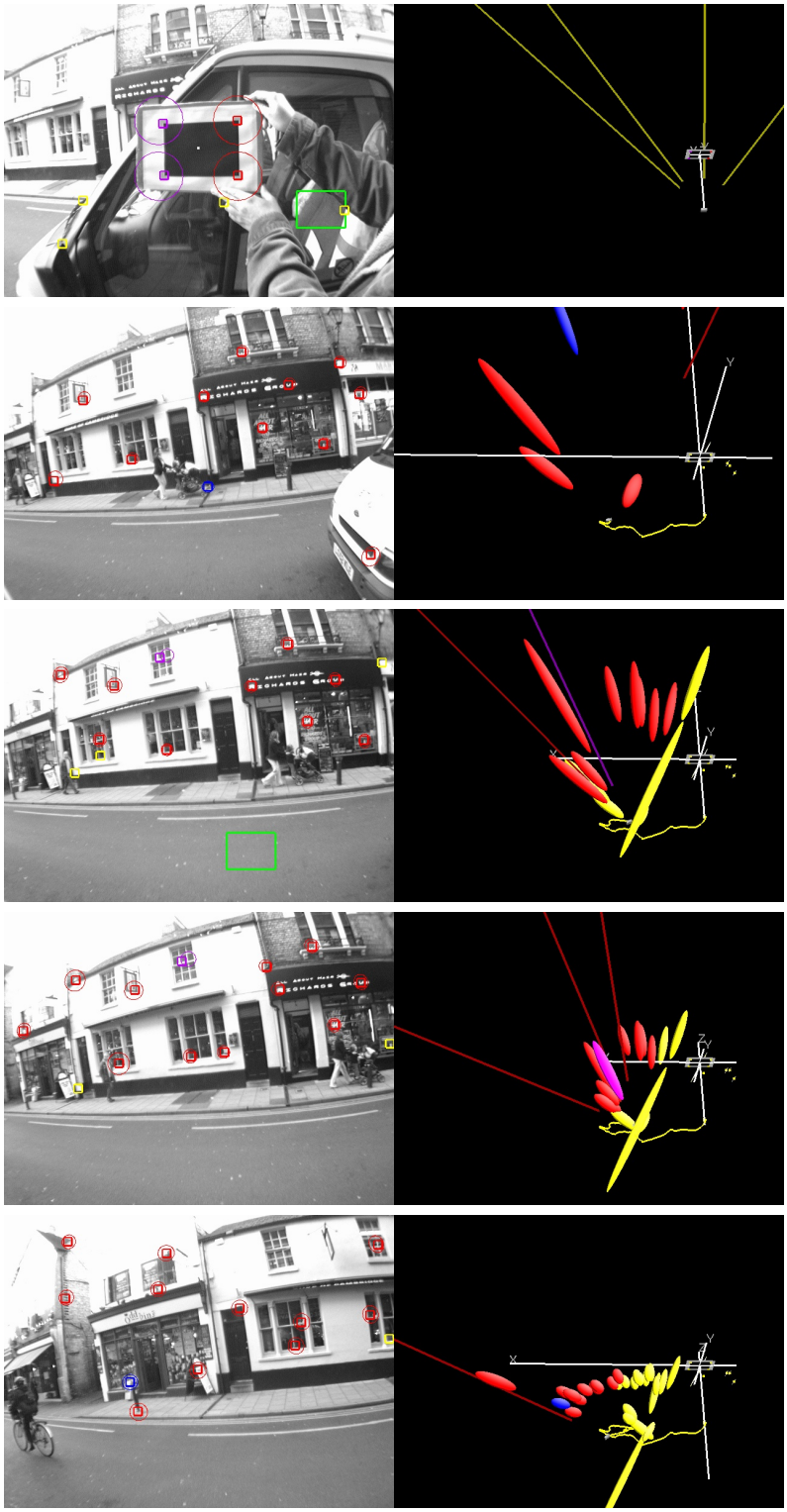


Figure 4.19: Street sequence with monoSLAM and calibration plate (see video exp5_street_monoslam_plate.avi).

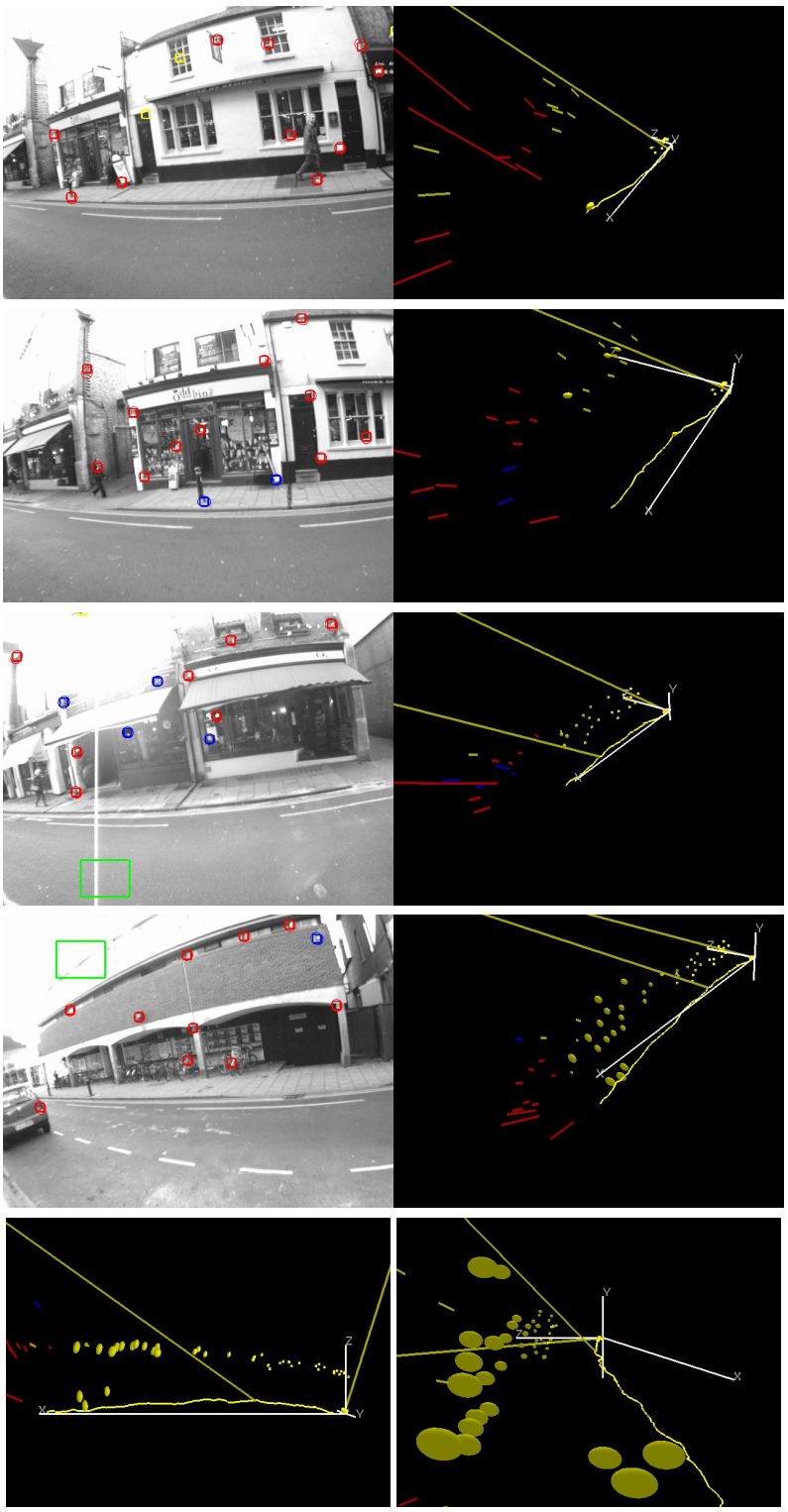


Figure 4.20: Street sequence with monoSLAM and no calibration plate (see video exp5_street_monoslam_noplate.avi).

Fig. 4.23 shows views around the final map. The scale remains reasonably well enforced, however, successive bad measurements corrupt the map, and then objects being measured at locations that differ with the features measurements end up causing the camera to become lost.

This outdoor sequence is particularly hard for monoSLAM and objectSLAM, and is pushing the limits of what monoSLAM was designed for, which was desk top and small room environments.

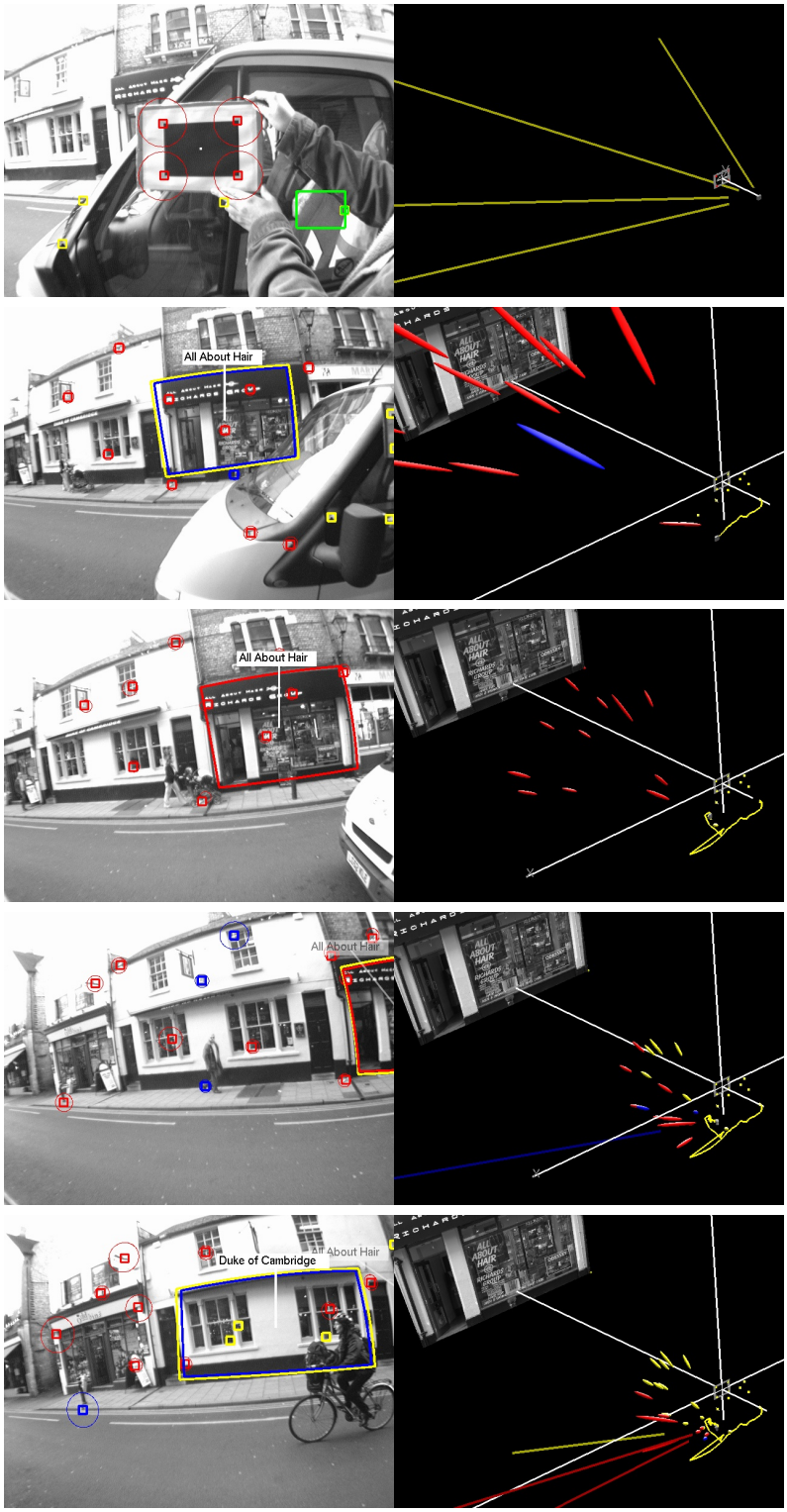


Figure 4.21: Street sequence with object recognition and calibration plate (see video exp5_street_objectslam_plate.avi).

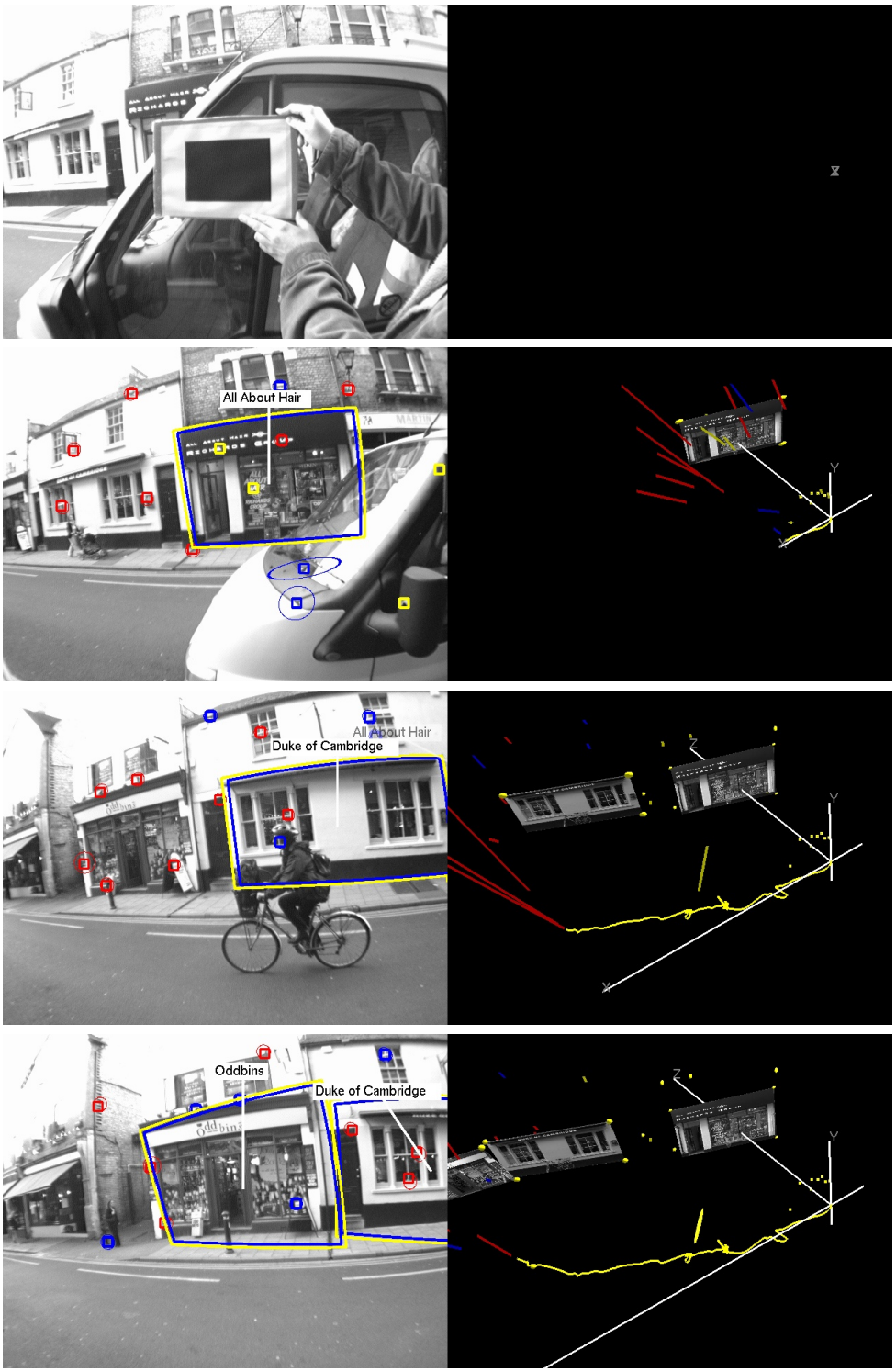
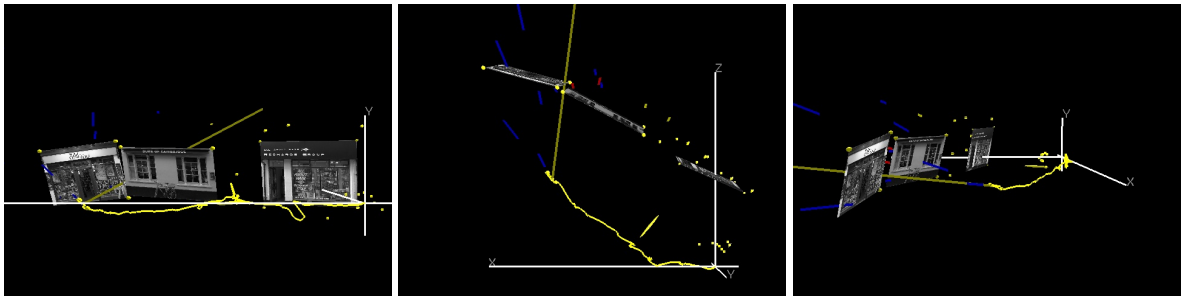


Figure 4.22: Street sequence with object recognition and no calibration plate (see video exp5_street_objectslam_noplate.avi).



(a) A view along the Z-axis within the XY-plane.

(b) A view along the Y-axis within the XZ-plane.

(c) A view along the X-axis within the YZ-plane.

Figure 4.23: Various 3D views around the objectSLAM street map where no calibration plate was used.

4.6 Oscilloscope tutorial - real world scenario III

The previous real world experiments showed that there are limitations with the system when used in environments where objects cannot be measured accurately, or where their appearance changes. They also highlighted some shortcomings with the underlying monoSLAM system that limit how and where the system can be used in its current form.

This final real world experiment, however, shows how well the system can work when in a small environment with objects that can be measured. This experiment also shows how near planar objects can also be recognized and tracked with sufficient accuracy to position detailed overlays.

An augmented reality training program has been created to guide a user through the basics of setting up an oscilloscope. Fig. 4.24 shows frames from the video of this tutorial. The user starts the system pointing at the calibration plate to give them the metric scale, and then the camera is moved around the desk and the map is built. When the oscilloscope front panel comes into view the system detects it and begins the tutorial. The tutorial guides the wearer through powering up the oscilloscope, setting the dials to the correct positions, and connecting a probe to the correct socket. As the user completes each task they press a button on the computer to advance to the next instruction. If the oscilloscope goes out of view the AR overlay is unaffected, because the location of the oscilloscope is in the map. If the oscilloscope was

moved however, then there would be a misalignment between the AR overlay and the oscilloscope. Measurements of the oscilloscope only occur when the camera is in front of it, otherwise, due to the non-planarity the measurements fail. Once it is in the map constant remeasuring is not necessary, only that the camera maintains a good track of the map features.

On the face of the oscilloscope the screen surround and various knobs protrude to a maximum of 20 mm from the panel front. The model of it in the object database is just a planar fronto-parallel picture. The recognition process performs satisfactorily to a maximum angle of some 20° on this object. Beyond this the view of the object is sufficiently changed to cause recognition to fail. Fig. 4.25 shows the same frames as Fig. 4.24, but with the underlying tracking system data shown instead.

This experiment has shown that the object recognition can be used accurately enough to place detailed AR within objects, and that the system can cope with a degree of non-planarity in the objects of interest.

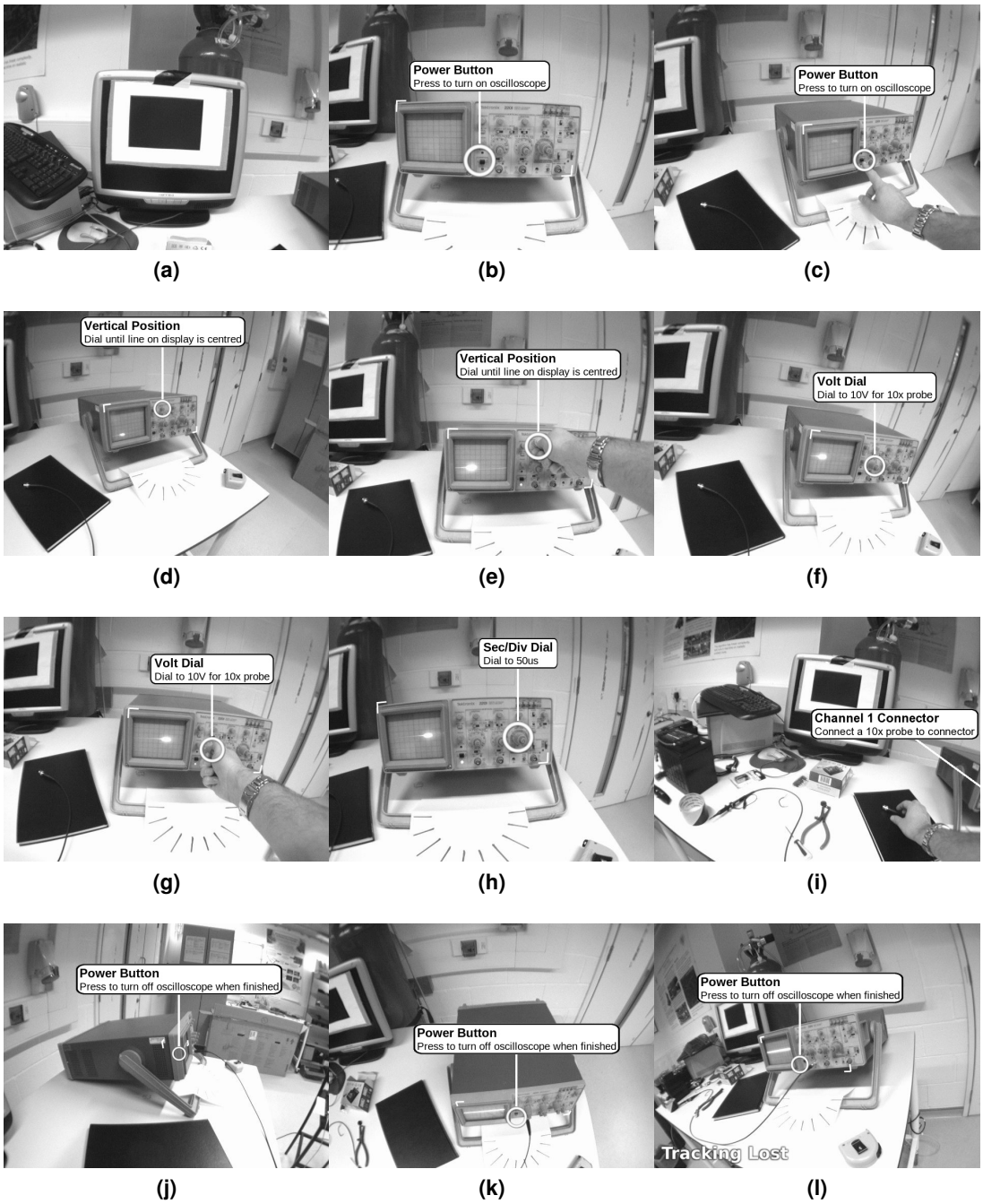
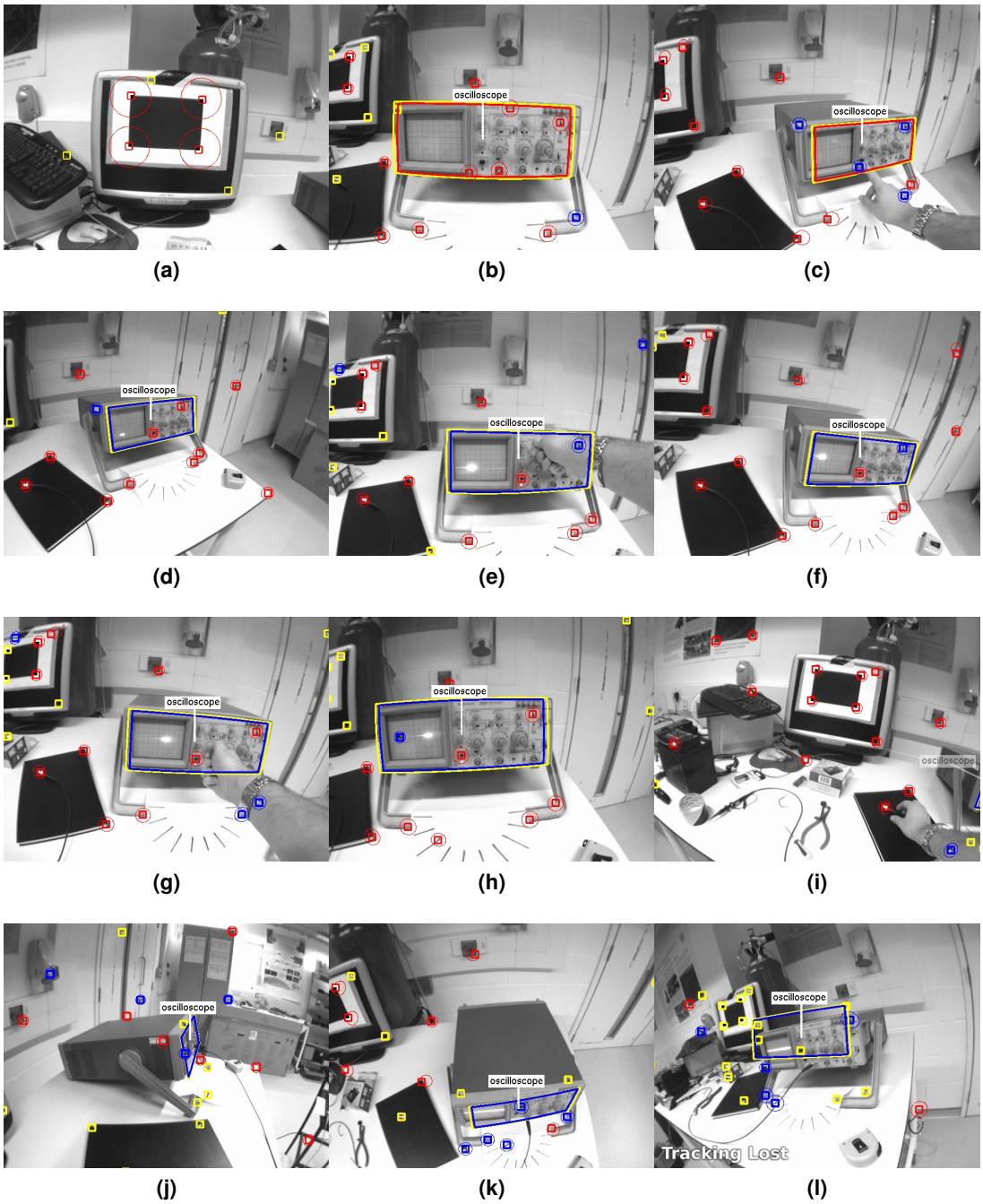


Figure 4.24: A demonstration both of recognition and localization continuing despite a degree of non-planarity, and of the automatic positioning of overlays (see video exp6_oscilloscope_ar_tutorial.avi).



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figure 4.25: Frames from the oscilloscope tutorial showing the underlying tracking system's feature measurements (see video exp6_oscilloscope_feature_tracking.avi).

4.7 Limitations and improvements

The system as implemented has limitations, as revealed by the experiments. They can be split into two categories, those relating to the underlying monoSLAM system and those relating to the object detection system.

4.7.1 MonoSLAM

The monoSLAM system has a variety of limitations that restrict development in several directions. To maintain frame-rate operation at 30 Hz all processing needs to be completed before the next frame arrives 33 ms later. To ensure this several constraints had to be placed on the system during its design. The EKF, and consequently monoSLAM, is an algorithm that executes in a linear fashion, and does not lend itself to being parallelized. The size of the map needs to be carefully managed to ensure the EKF updates do not take too long. On an Intel 2.20 GHz Dual Core Pentium a maximum map size is around 100 point features, and for each frame only a few measurements can be made. Exploration is effectively limited to the desk top or small rooms. Adding an object to the map adds three point features, so the more objects that are observed the smaller the overall scene coverage.

In the previous chapter the wearable design consisted of two cameras, one active shoulder mounted camera and one hand-held camera. To use both cameras together in a monoSLAM based system, with both cameras running at 30 Hz unsynchronized in the same map, the processing for each frame would have to be done in half the time and shrinking the map to $1/\sqrt{2}$ of its size.

MonoSLAM typically uses a calibration plate to start off the map, to give it metric scale. Without a calibration plate, detected features are initialized at an arbitrary depth. If sufficient features become localized relatively quickly, the map tends to settle to a consistant scale, and vice versa. In practice more often than not this ambiguity causes features to creep towards

the camera until the map collapses. When object detection is added to this unscaled situation either good or bad things can happen. If an object is detected at the start and measured a few times to become well localized, this certainty in scale propagates through the covariance matrix, causing the other features to become localized at the correct depth. This was demonstrated in the gallery and street sequences. If the map has become well localized at a scale that is different from the metric scale, and then an object is detected, the object will be inserted at the wrong depth in comparison with the features. There are typically two outcomes to this conflict and neither are good. The first is that the map becomes so corrupted that too many matches fail, causing complete tracking failure. The second outcome is that the certainty in the standard point features dominate the system, as they are measured every frame, whereas the object is only measured infrequently. When this occurs, a measurement from the object recognition thread differs so much from where the object is expected to be (within the 3σ uncertainty ellipsoids for each corner) it is discarded as being incorrect.

In short, object detection using this method works best when a calibration plate is used to give the metric scale and when objects can be measured accurately beforehand.

4.7.2 Object detection

Currently the object detection thread grabs the current frame, processes, then repeats. This is fine when exploring, but inefficient if the camera is still or hovering around the same area, looking in the same general direction. A more effective use of the processing time would be to drop a pose into the map when the camera is in a 'unique' location and save the associated frame for later processing. A unique location would be one where the camera's location and orientation is significantly different enough to warrant running object detection again. This would have some overlap with a previously processed frame, so if an object was detected in both then it could be localized more accurately. The object detection thread, could then process the 'dropped' frame when it had time. The limiting factor is the size of the map, due to the

real time processing constraints, and adding more poses would reduce the effective size of the map. A limited number could be kept however, and if the processing fell behind the addition of new poses, then previous ones could be thinned out by some criteria, such as removing the old ones or increasing the distance between dropped frames.

In the next chapter a new tracking system will explored that has many benefits over the monoSLAM system for wearable AR systems.

4.8 Conclusion

This chapter has presented the combination of recent methods of, first, recognition using appearance models and, second, localization and scene reconstruction using point-based monocular visual SLAM to identify and recover the 3D geometry of objects, and then insert them as 3D measurements into the map for updating by an extended Kalman filter. The method has been demonstrated using planar objects, and shown to work with near planar objects as well. By fitting a higher geometrical entity to visual data, the measurements entered into the SLAM map are sparse, robust to partial occlusion, and less likely to be incorrect through mismatching.

The relatively slow and variable rate of delivery of geometry from the recognition process has been properly accommodated in SLAM's statistical framework using Leonard and Rikoski's method of delayed decision-making, which inserts a temporary place-holder location in the state and covariance. This is updated during the time the recognition takes to complete, and is then deleted once it has been used to calculate the geometry of recognized objects.

On the semantic side, the wearable robot was able to present the user with information about an object and highlight its location in the world. This was demonstrated by displaying simple boundary outlines and the names of the objects. A more detailed demonstration was also shown in the form of an interactive tutorial which had multiple labels at precise locations on an object.

5

Parallel tracking and multiple mapping

In this chapter a new method for building maps and tracking is used. Instead of tracking and mapping in a linear frame-rate driven manner, this new method separates the mapping from the tracking. This allows much denser maps to be constructed, and provides more robust tracking. It also provides a flexible framework which this chapter and the next leverage to enable a rich AR experience. This chapter shows how the system can be extended to support multiple cameras, and multiple maps, allowing the user of a wearable two camera robot to explore large sprawling environments.

5.1 Introduction

The previous chapter demonstrated how camera localization and 3D map-building can be combined with object recognition and localization to provide a sense of augmented reality to the user of a wearable vision system.

This chapter is concerned with shortcomings in the localization and mapping process. Chief amongst these is the modest number of points (~100) that can be managed at video rate when the fully populated covariance matrix in monoSLAM has to be maintained. This number is such that not only is the map sparse, but also its spatial extent is quite limited. Allowing recognized objects to provide further measurements may have added to the integrity and richness of the map, but also had the unintended consequence of further reducing its reach.

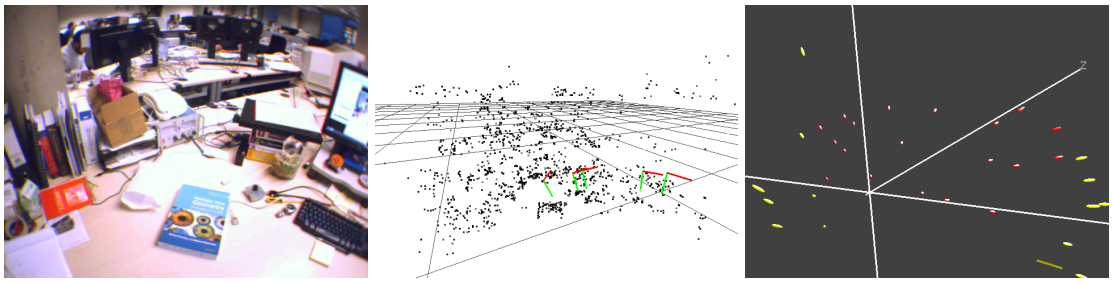
The principal aim of the work reported in this chapter is to extend the size of the mapped region for wearable computing by constructing and managing *multiple* maps.

It is argued here that in wearable computing there is no imperative to stitch the separate maps together geometrically, unlike in robot vehicle SLAM where sub-mapping is common enough, and where care is taken to recover the Euclidean transformation between maps by solving the absolute orientation problem using shared points. Here instead maps can be linked in two ways: for the user by augmenting them with a visual "signpost", and for the vision system by a mechanism which detects when the camera has left one map and when it has entered another. Uninteresting regions between mapped areas need not be mapped at all.

Multiple maps might be built readily enough using monoSLAM, but here, and indeed for the remainder of the work in this thesis, the entire camera localization and mapping method is replaced.

Much of the motivation for this change is, of course, to find a method that can handle at video-rate a greater number of points in each *single* map. Of nearly equal concern is the wish to avoid a method with a strong motion model. Modelling the evolution of the state is fundamental to a recursive filter, as it allows the accumulated wisdom of past measurements to be combined meaningfully with the current measurements via the state estimate and covariance. Less fundamentally, but rather importantly in vision, modelling and prediction make data association more reliable. In recursive processing this is the more important, because errors due to mismatching become locked into the state: there is no later opportunity for review of earlier data. So important is the need to avoid mismatches that a user of monoSLAM tends to move the camera cautiously rather than freely, to ensure that the motion model is satisfied.

Instead, the recent method of Parallel Tracking and Mapping (PTAM) [65] to localize the camera and build a 3D map is adopted and built upon. This is not a recursive method, but instead uses repeated batch-mode bundle adjustments, using image measurements only from well-separated keyframes. The key advantage is that each map may contain thousands (up to



(a) (b) (c)

Figure 5.1: A desk top scene (a), and a comparison of the 3D maps obtained after 20 seconds acquisition at 30 Hz frame rate using (b) the parallel tracking and mapping method of [65] and (c) monoSLAM [28]. The far greater feature density in (b) is evident.

~20 000) points, rather than the hundred or so points in monoSLAM. Moreover, the problem of refining the map and camera poses is treated separately from the more urgent task of maintaining knowledge of the pose of the camera (camera tracking). They run in parallel, in separate threads and, where hardware allows, on separate CPU cores. The difference in the map feature density between PTAM and monoSLAM is shown in Fig. 5.1, and is such that the 3D structure of objects starts to be become evident. PTAM maps typically contain some $2\,000 - 10\,000$ points and $40 - 120$ keyframes.

In this chapter PTAM is used for wearable computing, and extended by proposing a mechanism which allows multiple cameras to build and utilize a single map or many maps, with switching between them enabled using "lost camera" recovery.

Experiments are undertaken in small environments to demonstrate the viability of the method, dubbed Parallel Tracking and *Multiple* Mapping (PTAMM), and to show that, if maps are sufficiently restricted spatially, the maps alone can be treated as objects or places. PTAMM is then demonstrated in two larger environments, the second being a potential application in which visual AR is delivered to user walking through exhibits in a natural history museum.

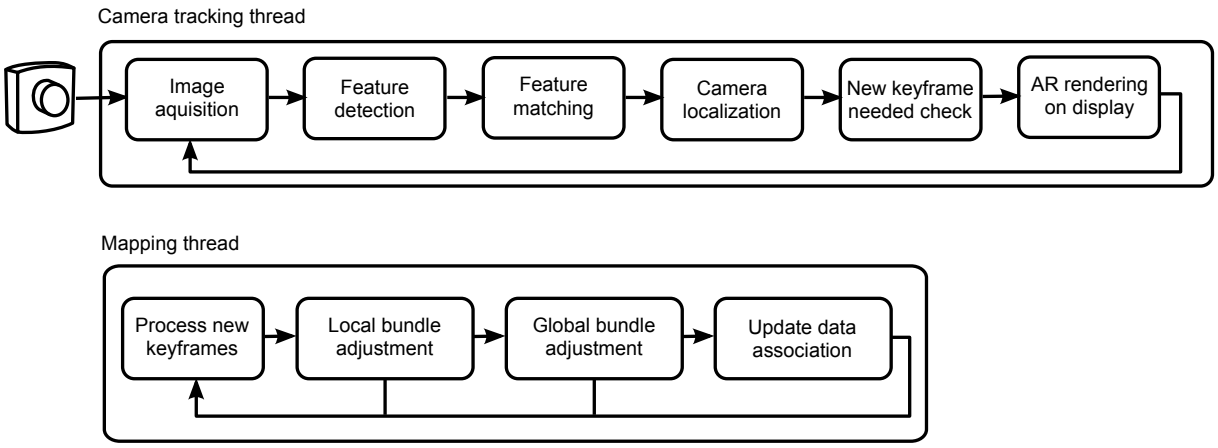


Figure 5.2: The process flow for PTAM under steady state operation. The tracking and mapping processing threads are independent.

5.2 Parallel tracking and mapping

The parallel tracking and mapping method of Klein and Murray [65] is designed to achieve robust camera tracking for small AR workspaces while maintaining 30 Hz operation. These workspaces are expected to be predominantly rigid and desk or small room sized. Within these constraints the method is able to handle large changes in scale, rotations, rapid saccades, and recovery from tracking failure. This enables the system to provide a robust AR experience with minimal jitter or tracking failure while the user explores the environment.

PTAM splits the tracking from the mapping, allowing each to run independently from the other. Fig. 5.2 summarizes the process flow of the tracking and the mapping. The motivation for separating them comes from the observation that visual SLAM methods, both EKF- and FastSLAM-based, spend too great a time per frame referencing the 3D map and its covariance. The root of their difficulties might be viewed as under-exploitation of the epipolar geometry which allows camera motion to be found without addressing scene structure explicitly.

The following sections cover the tracking, keyframe selection, and mapping of PTAM.

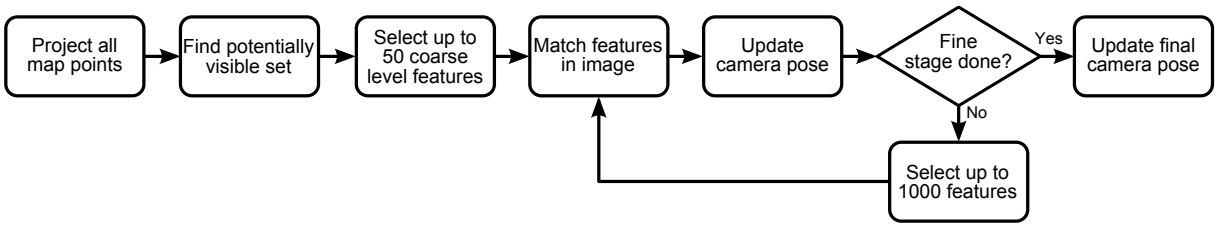


Figure 5.3: The tracking process for the parallel tracking and mapping system.

5.2.1 Camera tracking in PTAM

The upper process flow in Fig. 5.2 summarizes the camera tracking when running normally, using an already initialised 3D map.

When a frame is captured from the camera, the tracking thread first generates a luminance or grey-level image from it, along with a RGB image for later use with the augmented display. The grey-level image is sub-sampled into a four-level pyramid, and corner features are computed at each scale using the FAST-10 detector [125] (reviewed in §3.2.1).

A prior pose for the camera is obtained from a camera motion model that assumes constant (or decaying) velocity in the presence (or absence) of measurements. This model plays no part in the tracking and mapping processes other than to assist data association. The pose is refined and matching features are found in a two stage process shown in Fig. 5.3. Initially all of the map points are projected from 3D into the image space to find a potentially visible set. Then using this subset, up to 50 map points at the coarsest scales are selected, and those that are matched to the current image feature list are used to correct the prior pose using robust minimization. A further iteration is made, adding more map points (up to 1 000) from the fine to coarse scales. A pose for the frame is then estimated from all of the matches found.

The camera pose $\hat{\boldsymbol{\mu}}$ is found iteratively from a set of successful observations by minimizing the reprojection error,

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} \sum_j \rho\left(\frac{e_j}{\sigma_j}, \sigma_T\right) \ , \tag{5.1}$$

where ρ is a robust measure of the reprojection error

$$\boldsymbol{e}_j = | \, \boldsymbol{x}_j - \boldsymbol{x}(\boldsymbol{K}, \boldsymbol{\mu}, \boldsymbol{X}_j) \, | \ , \tag{5.2}$$

and where $\boldsymbol{x}_j$ is a measurement, $\boldsymbol{x}(\boldsymbol{K}, \boldsymbol{\mu}, \boldsymbol{X}_j)$ is the projection of the corresponding map point $\boldsymbol{X}_j$, and $\boldsymbol{K}$ is the camera's intrinsics and distortion parameters. The measurement variance is assumed to be isotropic in the image, and to scale as 2^{2l} where l is the pyramid level. Rather than a standard weighted least squares formulation with

$$\rho(e_j/\sigma_j) = (e_j/\sigma_j)^2 \ , \tag{5.3}$$

a robust Huber M-estimator [59, 160] with threshold σ_T is used in which, writing $\beta = (e_j/\sigma_j)$,

$$\rho(\beta) = \begin{cases} \beta^2/2 & \text{if } |\beta| \leq \sigma_T \\ \sigma_T(|\beta| - \sigma_T/2) & \text{otherwise.} \end{cases} \tag{5.4}$$

The distribution threshold σ_T is set as the estimate of the error distribution's standard deviation. The minimization is performed for ten Gauss-Newton iterations, and the camera's current pose is set as the result. As described above this is first done in the coarse stage and then again for the fine stage.

The fraction of features successfully matched is monitored to provide a measure of tracking quality, and two thresholds are chosen. If the fraction falls below the higher, tracking continues as above, but the frames involved are prevented from becoming keyframes for the mapping process described below, as the image quality is probably poor. If the fraction falls below the lower threshold, it is assumed that tracking has failed terminally, and a method of relocalization is used to recover the lost camera (see §5.3.3).

5.2.2 Selecting frames as keyframes

Some of the frames captured are designated as keyframes and play an important role in the mapping process. As the camera moves around the environment, the current frame is selected as a keyframe whenever (i) tracking is deemed good, (ii) the camera has translated by a mini-

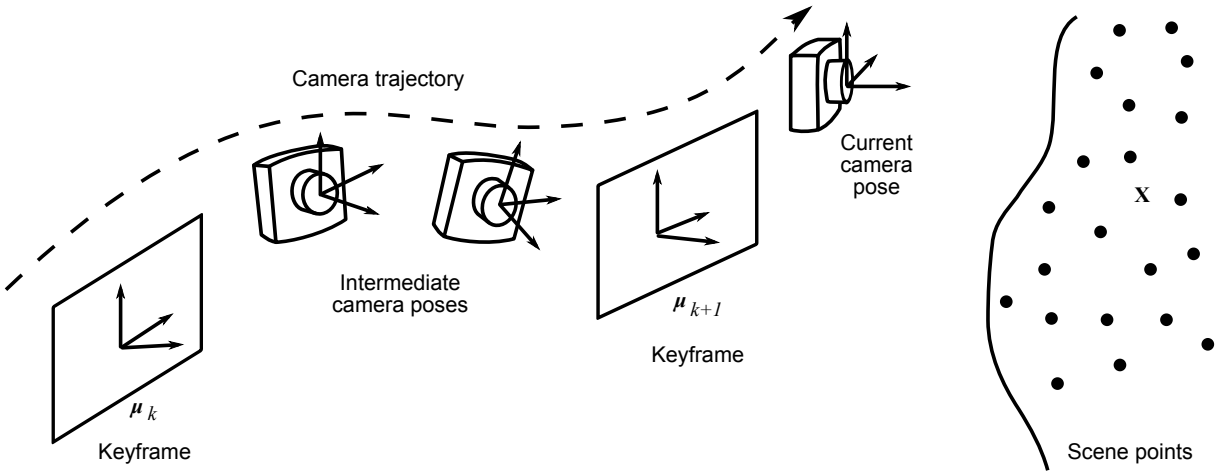


Figure 5.4: Evolution of PTAM tracking and keyframe addition.

mum distance from any previous keyframe, (iii) around 20 frames have passed since the previous keyframe, and (iv) the current queue of new keyframes is less than three. These last conditions restrict the volume of processing. The minimum distance is based on the Euclidean distance between the frames and mean depth of points in the scene, causing keyframes near surfaces to be closer together. The new keyframe is added to a queue for the mapping thread to process. The k-th keyframe includes the features $\boldsymbol{x}_{ik}$ computed in it, the match list between those features and the map points $\boldsymbol{x}_{ik} \leftrightarrow \boldsymbol{X}_j$ (although not all features will be matched), and the best estimate of the camera pose from the tracker, $\boldsymbol{\mu}_k$. Fig. 5.4 illustrates the process as the camera moves through a scene with some of the camera frames and poses being selected and turned into keyframes.

5.2.3 Mapping in PTAM

Fig. 5.5 shows the mapping process in PTAM. It runs separately from the tracking, allowing it to take (far) longer than the inter-frame period to compute maps.

To initialize a map at the outset, a five-point stereo algorithm is used [142], one similar to those in [24], [38], [102]. The first camera viewpoint is selected by the user and its image and feature list becomes the first keyframe, with pose $\boldsymbol{\mu}_1 \equiv \{\mathtt{R}, \boldsymbol{t}\}_1 = \{\mathtt{I}, \mathbf{0}\}$. The camera is then moved to a new position, with sufficient care to allow features to be tracked using the image

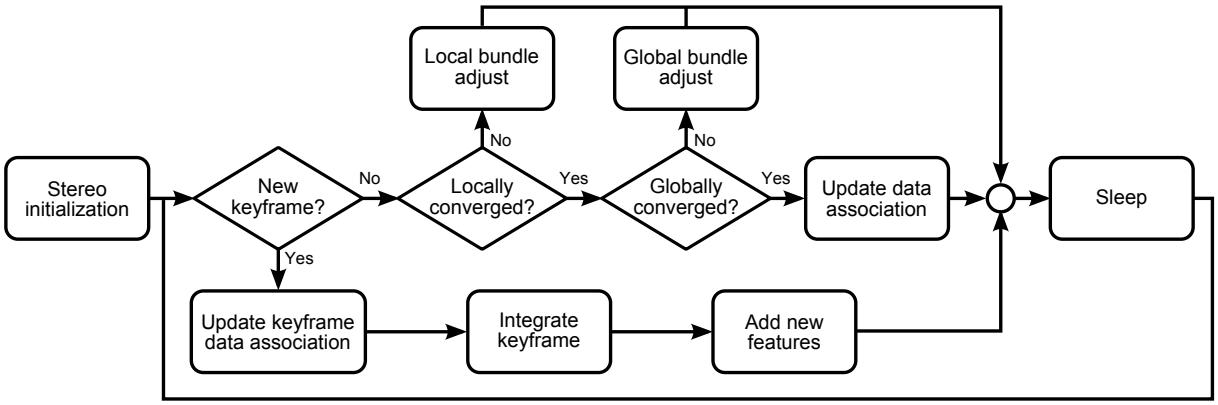


Figure 5.5: The mapping process for the parallel mapping and tracking system.

alone. The image is captured as the second keyframe, and Nistér's relative pose algorithm [105] is used with RANSAC to compute the new camera pose $\boldsymbol{\mu}_2 = \{\mathtt{R}, \boldsymbol{t}\}_2$. Pairs of matching FAST corners from the two images are then used to triangulate the initial set of 3D scene points $\boldsymbol{X}_j$ for the map. It is assumed that $|\boldsymbol{t}_2|$ is some 100 mm so that a reasonable scale can be applied to the map. The scale is, of course, arbitrary and does not affect later processing.

Often, many of the initial 3D features will lie on a desk top or ground plane so, as a convenience, a dominant plane is found in the structure, and the map reoriented so that this defines the scene's $Z = 0$ surface. This plane plays no special role in the ensuing structure from motion calculations, and is merely a convenience for AR applications. The time taken to initialize the map is dominated by the time the user takes to translate the camera carefully and select the initial keyframes: in practice it takes a couple of seconds.

With the map initialized, camera tracking begins as described above, occasionally adding a new keyframe to the queue for processing. As mentioned, the keyframe comes with corner features in the image pyramid, of which a subset is matched to the map, and a camera pose estimate. The new keyframe and its features are integrated in to the map in the following manner. New 3D points are added to the map by finding image features which are unmatched, particularly salient, and are distant in the image from any matched feature. To find their 3D positions, a search is made for image matches in the spatially closest keyframe, and the position triangulated using the estimated keyframe poses.

At this stage, the estimated 3D positions $\{\boldsymbol{X}_j\}$, $j = 1 \dots J$ of all map points and keyframe camera poses $\{\boldsymbol{\mu}_k\}$, $k = 2 \dots K$ (that is, all except the first, which is fixed) are optimized in a bundle adjustment, using Levenberg-Marquardt [89]. Let the estimated position of the projection into keyframe k of the estimated map point $\boldsymbol{X}_j$ be $\boldsymbol{x}(\boldsymbol{K}_k, \boldsymbol{\mu}_k, \boldsymbol{X}_j)$, where $\boldsymbol{K}_k$ are the intrinsics and distortion parameters of camera k (assumed known and fixed). The magnitude of the image error is then

$$e_{jk} = | \, \boldsymbol{x}_{jk} - \boldsymbol{x}(\boldsymbol{K}_k, \boldsymbol{\mu}_k, \boldsymbol{X}_j) \, | \ , \tag{5.5}$$

where $\boldsymbol{x}_{jk}$ is the measurement that arose from $\boldsymbol{X}_j$. The adjustment is

$$\{\hat{\boldsymbol{X}}\}, \{\hat{\boldsymbol{\mu}}\} = \arg\min_{\{\boldsymbol{X}\},\{\boldsymbol{\mu}\}} \sum_j \sum_k v_{jk} \rho \left(\frac{e_{jk}}{\sigma_{jk}}, \sigma_T \right) \ , \tag{5.6}$$

where the visibility flag

$$v_{jk} = \begin{cases} 1 & \text{if point } j \text{ is visible and matched in } k \\ 0 & \text{otherwise,} \end{cases} \tag{5.7}$$

σ_{jk} is the likely error, and σ_T is the threshold for ρ which is now a Tukey M-estimator [151],

$$\rho(\beta) = \begin{cases} \frac{\sigma_T^2}{6} \left(1 - [1 - (\beta/\sigma_T)^2]^3\right) & \text{if } |\beta| \leq \sigma_T \\ \sigma_T^2/6 & \text{otherwise.} \end{cases} \tag{5.8}$$

The time taken to execute bundle adjustment scales as $O(K^3)$. This limits the map to modest workspaces, of small room sized scenes with perhaps 150 keyframes and $20\,000$ points in the map before the scope to explore is curtailed. However, because this is a batch process and the data retained throughout, it is open to the mapping process to delay performing the complete adjustment until time and processor loading permits. Thus when the camera is exploring, and new keyframes and measurements need to be added to the map frequently, local bundle adjustments on a limited number of keyframes are performed.

In recent work on visual odometry [102, 106], using temporally local bundle adjustments (a sliding window) has been found to be remarkably accurate even without a global adjustment. Here the locale is spatial, which will be partially correlated with time, but will include old keyframes if the camera revisits a location after a period of neglect. This ensures that the pose of the most recent keyframe and its neighbours are optimized.

5.3 PTAMM: multiple trackers and maps

The main proposal of this chapter is the use of multiple maps for wearables, but there is also the need to allow multiple independent cameras to build and work in a single map. This is because the wearable system of Chapter 2 has two independent cameras. Although two or more sensors might readily provide measurements for EKF-based SLAM, care would have to be taken to synchronize them to avoid having to run the update cycle more frequently. By contrast, the degree of independence of the tracking and mapping processes in PTAM make it highly suited to multiple cameras.

5.3.1 Multiple camera trackers and the map maker

Given sufficient processing hardware, it would be routine to replicate the tracking and mapping processes of PTAM so that multiple cameras build multiple maps in a one-to-one fashion. Much more interesting is that, because the tracking process (image capture, feature detection, and camera pose estimation) is largely independent of the map making process, it proves comparatively straightforward to allow multiple cameras to add information into a *single* map.

A map is initialized in the same manner as described above, using just one camera so that the coordinate system is defined uniquely. Thereafter, any number of cameras can use those 3D points to determine their poses from frame to frame. Additional cameras join the map using existing functionality, that of relocalization when a camera becomes lost. Relocalization is discussed later in §5.3.3.

The more interesting question is when and how to allow different cameras to insert keyframes into the map? This is also answered straightforwardly, because in single camera PTAM new keyframes are determined by *spatial separation* of poses, not temporal separation. In the multiple camera version, each camera tracker provides frames that individually satisfy the earlier criteria to the mapping process (the map maker), which decides when to insert the keyframes

into the map and run local or global adjustments. In effect, any additional views can be utilized at will, and multiple trackers can work on a map simultaneously or at different times.

The number of cameras that can be attached are limited by bus and processor bandwidth. On the dual core processor used on the wearable system, one core can handle two cameras comfortably, with another core required for the map maker. With three cameras, degradation in the tracking and map building is evident due to the processors' inability to handle all of the required processing.

5.3.2 Multiple maps

As noted earlier, the reach of PTAM maps is limited by $O(K^3)$ complexity in the number of keyframes. Interpreting this as a hard numerical limit is not straightforward, because there are several layers of optimization involved. Visual odometry from the trackers alone gives good estimates of pose, and these and the map are part-refined using local adjustments (which are $O(1)$ as they use a fixed number of keyframes). For a given processor, a map's reach then depends on the degree of exploration, and the extent to which local adjustments suffice until accumulated error would cause the global adjustment to fail.

Although one can monitor CPU utilization, there is no clear way of monitor to what extent the global adjustment can be pushed into the background. This thesis takes the pragmatic view that the wearer must *always* be able to explore their environment freely, no matter the size or the distance between areas of interest. For this reason the use of multiple maps is proposed, where each is sized conservatively, and is anyway under the control of the wearer.

The ideal for managing multiple maps is to use a separate map maker for each map, and to run each as an independent process, as shown in Fig. 5.6. In this way the optimality of all maps is assured (or at least being worked upon). Aspiring to this on modest hardware requires compromise, not least to fall within the limits on independent threads that can be running. On the wearable system, for example, one map maker is used with both cameras adding to the

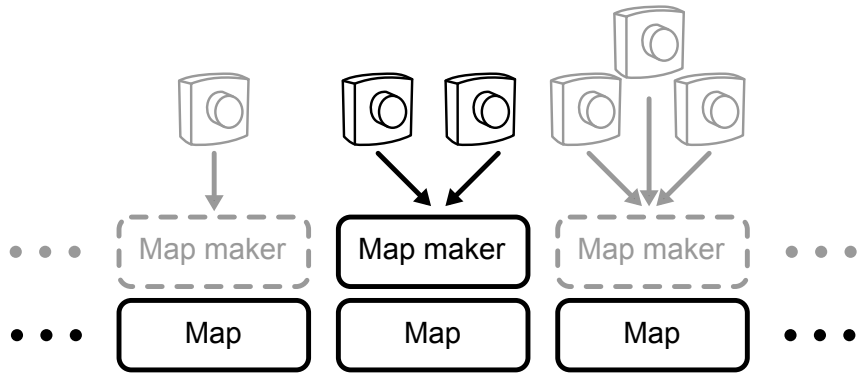


Figure 5.6: An array of maps, each with their associate map maker fed by single or multiple cameras. The two highlighted cameras and map maker show the system configuration used.

same map, and if a different map is to be used, both cameras and the map maker need to move to it. The configuration used is highlighted in Fig. 5.6.

Were more computing resource available, other approximations to the ideal can be considered, always with the aim of ensuring that when a camera wants to use a map, it is already fully adjusted. For example, a "roving" map maker that never has a camera associated with it might be used to optimize maps that are currently not in use. Or a "paging" mechanism might be used to operate upon only the most recently used, or the most closely associated, maps.

5.3.3 Map switching via relocalization

The final problem to address with multiple maps is how to switch from one map to another. The answer lies with the relocalizer that is used to recover a "lost" camera in a single map.

5.3.3.1 The relocalizer

The earliest version of the PTAM system relocalized a lost camera in its single map using the method of Williams *et al.* [155], the method previously used for monoSLAM and reviewed in §3.7. However, randomized trees occupy around 1.3 MB of memory per map point, which is tolerable for a map with a hundred features, but not for maps containing thousands of points. A far cheaper relocalization method was introduced in [66] and [16].

Unlike monoSLAM, PTAM uses keyframes, and these are spread quite uniformly and densely

throughout a map. The new relocalizer creates a descriptor to describe each keyframe and each incoming camera image, by sub-sampling the image eight-fold (in practice $640\times480 \rightarrow 80\times60$), then applying Gaussian blur with scale $\sigma = 5$.

Because of the greater density of points and the method of camera tracking, PTAM's camera rarely becomes lost during normal use. When it does (most often because of a very fast change in direction) the current camera image descriptor is compared to those from keyframes using zero mean sum of square differences:

$$D = \sum((I_{ki} - \mu_k) - (I_{ci} - \mu_c))^2 \,, \tag{5.9}$$

where I_k and I_c are the intensity values of the i^{th} pixel for the keyframe and camera image descriptors respectively, and μ are the means. The keyframe that has the least difference is accepted and the camera position is set to that of the keyframe, and the rotation of the camera estimated using a direct second order minimization [11] of the sum of squared differences. If tracking fails to restart, the relocalizer is invoked again.

5.3.3.2 Map switching

The mechanism just described can be used without significant modification to switch between maps. However, in multiple map PTAMM, when tracking fails it is uncertain whether the camera remains looking at an area belonging to the same map, or at one of another map, or indeed at an unmapped area. To resolve this, the relocalization mechanism is modified to cross-correlate with all of the keyframes from all of the maps. The cost of searching grows linearly with the map size, which would eventually lead to a system that responds too slowly. However each comparison calculation only takes 0.016 ms on average[1]. This means that $62\,500$ keyframes can be searched each second, equivalent to over 500 large maps (*i.e.*, each with around 120 keyframes). This limit is not likely to be approached in the near term, as the number of active large maps that can be held in 2 GB RAM is around 30.

[1] As ever, on an Intel 2.20 GHz Dual Core processor.

A recovered camera will lie either in the current map or in a different map. If the former, then tracking resumes as normal. If the latter, the tracker and map maker must switch maps, and then resume tracking from the estimated pose.

An exception to this relocalization strategy is required for the map usage scheme adopted for the wearable system. Recall that owing to processing limitations, both trackers and the map maker have to use the same map. Now if one of the trackers becomes lost but the other is not, the search will only occur in the current map. If all trackers are lost the complete search is performed by all trackers, and the first tracker to recover causes the other to abandon their complete search in favour of attempting to find their positions in the recovered tracker's map. The map maker also switches to the map designated by the trackers: it has no role in the map selection.

The map switching process is automatic, and transparent to the user as they explore the world. While they are viewing a map they can observe the AR constructs, if the camera becomes lost this is indicated until the system recovers. If the camera is still viewing the same map the AR constructs for that map will reappear. When the user moves away from the map the system will eventually become lost and remain lost until it enters a mapped area. At this point it will recover and display the relevant AR constructs for that map.

5.4 Implementation and usage

PTAMM is written in C++ and runs under Linux on an Intel dual core 2.20 GHz processor. It is built on top of Klein's PTAM code. For this reason, a different set of tools and libraries are used from that in Chapter 4, namely: the TooN library [32] for maths operations; the CVD library [31] for image manipulation and video capture from camera and disk; and OpenGL for 3D graphics. To enable the use of the USB cameras used in the wearable system, an additional library was written to allow video capture using the CVD framework. Unlike the software implemented in the previous chapter, there is no external GUI to the camera and world views. Instead all com-

mands are either issued via a command line interface or via menus in the OpenGL windows themselves. These are implemented using OpenGL using Klein's GKTools library [63].

The configuration of the system is sketched in Fig. 5.6. The maps are considered as a central resource, built by multiple cameras performing tracking and mapping. Those "builders" can also use the maps for AR at the same time. However, a set of users can be considered who use the maps solely for tracking or who have limited capacity to build the maps further.

In practical applications for AR outside the laboratory the following procedure is adopted. A super-user (or super-builder) creates multiple maps of the environment at predominantly non-overlapping sites of special interest. During this phase, the builder can explore as freely as necessary to build up a map of the size and detail required. If maps overlap, no attempt is made to merge structural information. This causes indeterminate behaviour, dependent on how the relocalizer recovers. However, with care, overlapping maps can be used effectively. Examples of both these behaviours are shown in the experiments.

Once all of the maps have been made, the system can be given to the users. Access to the maps could be restricted in several ways. For example, a map might be made "read-only", so no new measurements can be added (as is done in this work); or the map might be made writeable within its original bounds to avoid interference with other maps.

5.5 Experiments I: basic operation

The experiments show first the basic operation of the system, followed by increasingly larger scale experiments which exploit all the features.

The experiments were performed using the wearable system in either a two camera mode or a single camera mode. In the two camera mode, the hand held camera and the shoulder mounted active camera are used, and in the single camera mode just the hand held camera is used. The different operating modes are noted in the experiments. All the results presented

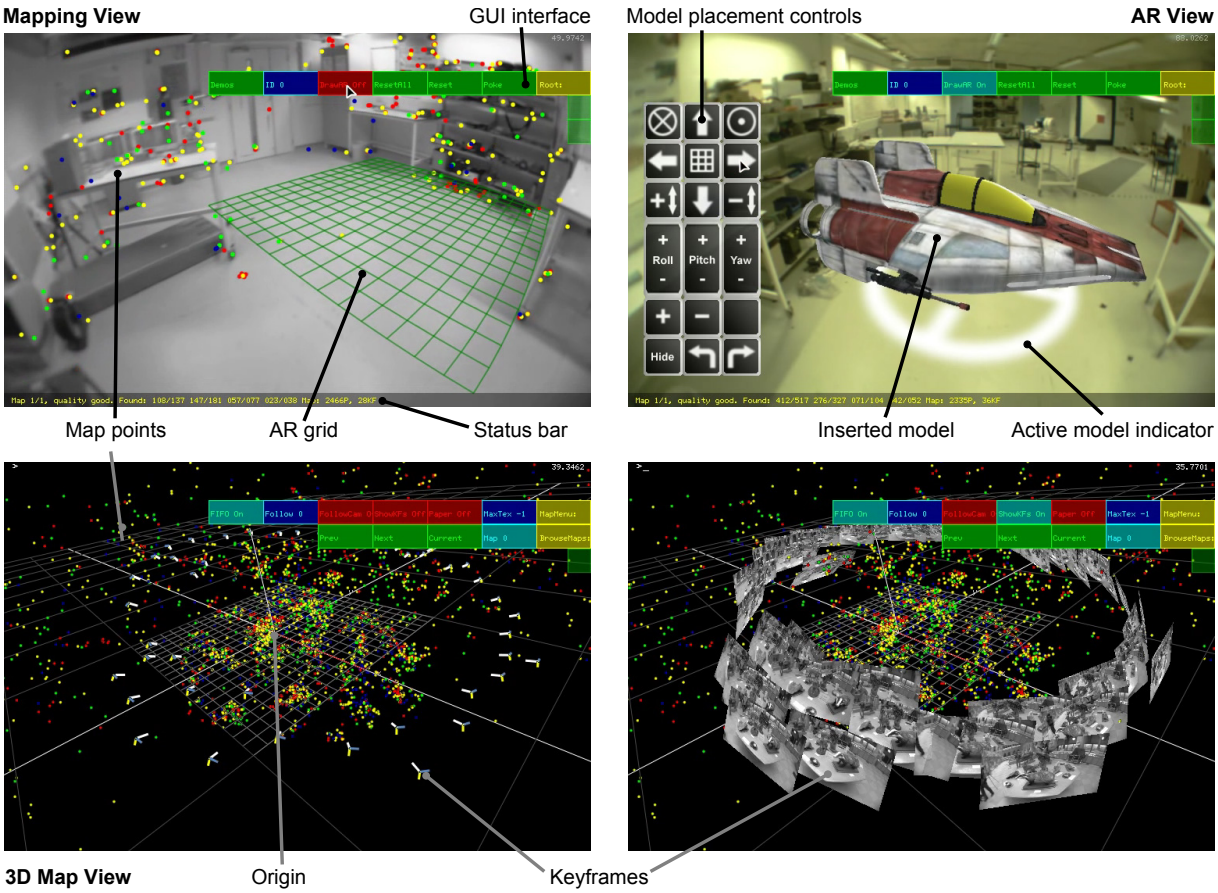
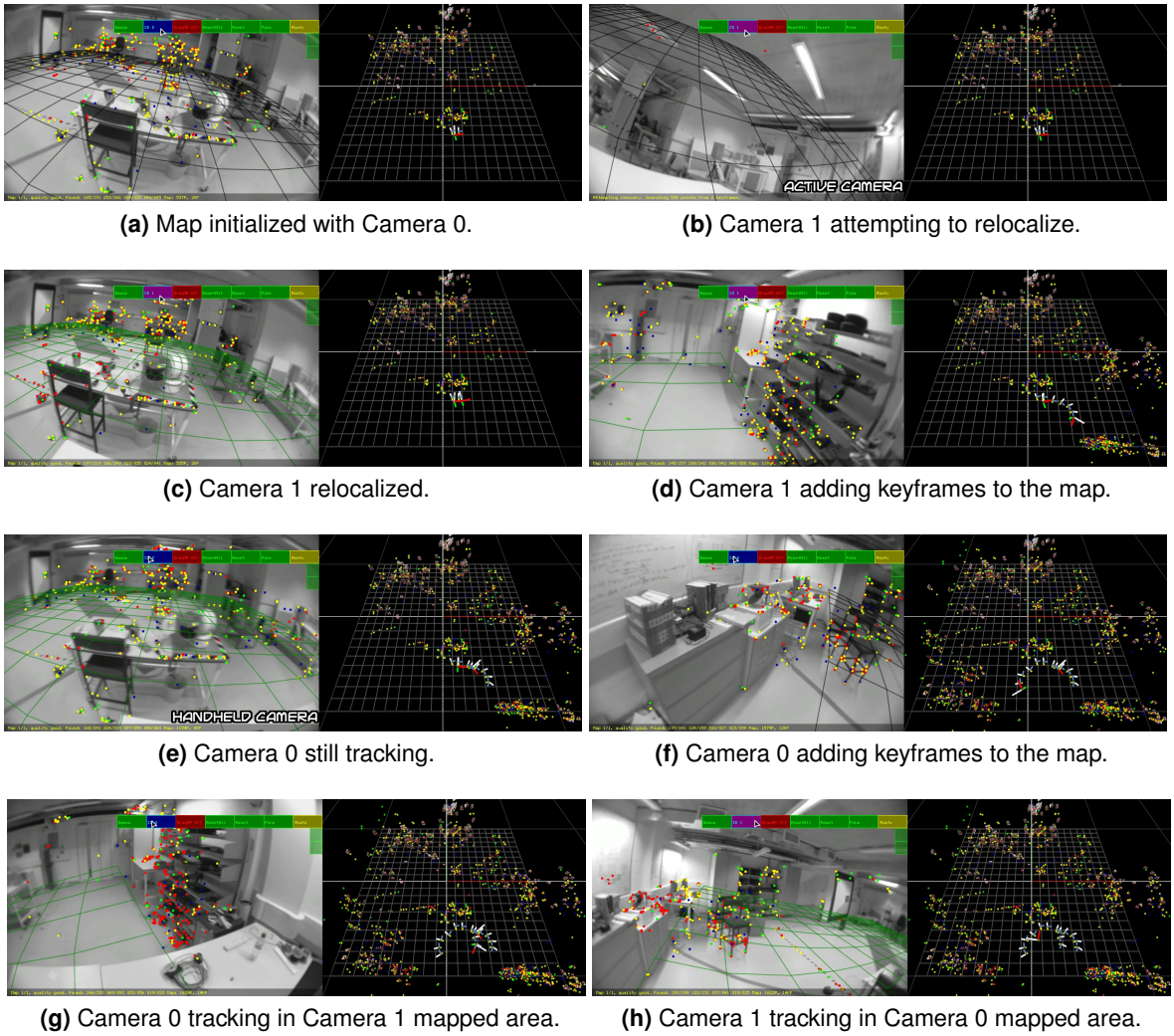


Figure 5.7: Typical output frames from PTAMM. The top row shows the user's camera view. On the left is system running in its default mapping view mode, with basic map data overlaid on the image. On the right is one of the AR data addition modes, where the user can add and manipulate models by simply touching the screen. The bottom row shows a typical 3D map view output, with the keyframes shown as simple axes on the left and as their image on the right. The map can be moved around to view the world and various options can be toggled to show different data, such as the keyframe views.

here were processed live at 30 Hz and stored directly to video.

To aid understanding of the images shown in the following experimental sections Fig. 5.7 highlights items of interest and explains their meaning. The accompanying video material is also an important supplement to aid the reader's understanding of how the system works and runs in real time.

The first experiment in this section shows how multiple cameras can add to the same map. The second shows how map switching works in practice, and the third how overlapping maps affect the system.



(a) Map initialized with Camera 0.

(b) Camera 1 attempting to relocalize.

(c) Camera 1 relocalized.

(d) Camera 1 adding keyframes to the map.

(e) Camera 0 still tracking.

(f) Camera 0 adding keyframes to the map.

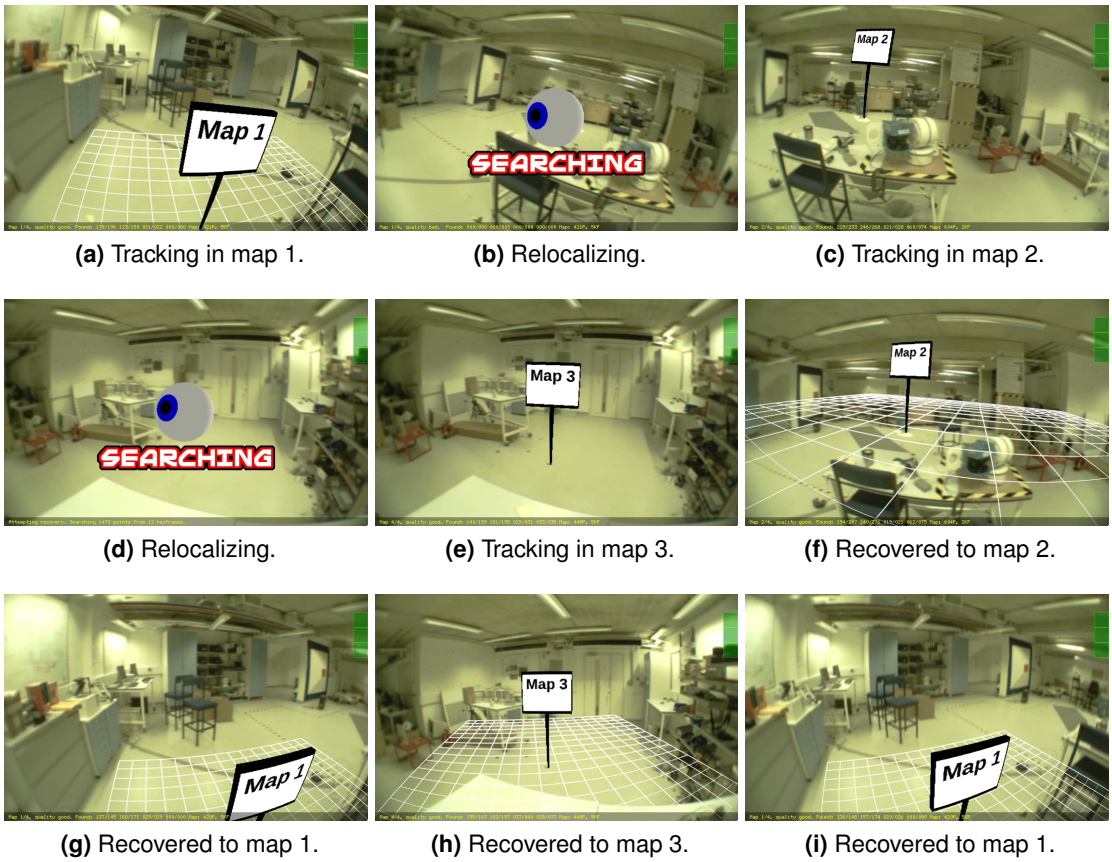
(g) Camera 0 tracking in Camera 1 mapped area.

(h) Camera 1 tracking in Camera 0 mapped area.

Figure 5.8: *Expt 1.* Example showing how multiple cameras can use the same map (see video exp1_multiple_cameras.avi).

5.5.1 Multiple cameras

Expt 1. This initial experiment shows how the system can utilize two cameras to build a map together, and Fig. 5.8 shows frames from the sequence. The map is initialized using the first camera, and this camera begins to track and supply keyframes to the map maker. Once the map has been created, the second camera localizes itself within the map using the relocalization mechanism. Once successful, it too tracks and adds keyframes to the map. Either camera can track through areas mapped by the other, and add keyframes when they are spatially required.



(a) Tracking in map 1. **(b)** Relocalizing. **(c)** Tracking in map 2.

(d) Relocalizing. **(e)** Tracking in map 3. **(f)** Recovered to map 2.

(g) Recovered to map 1. **(h)** Recovered to map 3. **(i)** Recovered to map 1.

Figure 5.9: *Expt 2a.* Example showing how a camera can switch between multiple maps (see video exp2_map_switching_1cam.avi).

5.5.2 Multiple maps

Here the map switching behaviour is considered. In the first part only one camera is used, and in the second both cameras are used.

Expt 2a. Three maps have been created and the tracker placed in a read only mode, so no further measurements can be made or keyframes added. Fig. 5.9 shows frames from the sequence. When the camera leaves the observable area of the map it becomes lost. Then, when it enters the second map it recovers and begins tracking again. This process repeats when moving to the third map. The order in which the camera observes the maps is also unimportant, because no prior assumptions are made as to the location of the lost camera: all maps are searched exhaustively.

Expt 2b. Recall that when two cameras are in the same map, but one becomes lost, it can only recover into the the current map. This is the case even if it is by chance observing a different map. If both cameras become lost, the first one to successfully recover causes the other to abandon its global search and only attempt to recover to the same map. Fig. 5.10 shows frames from the sequence.

5.5.3 Overlapping maps and mobile maps

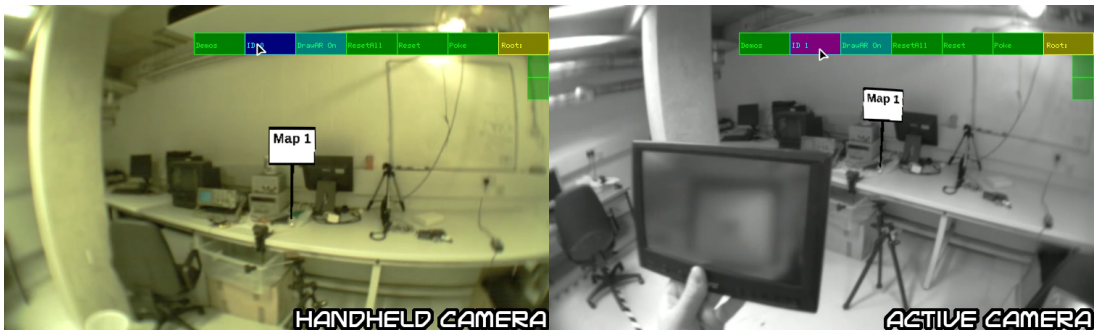
The next pair of experiments highlight a limitation of the method, but go on to show how it can be used to advantage.

Expt 3a. Because there are no explicit links between maps there is no way to detect overlaps. If two maps overlap they will both possess keyframes of similar views. When the camera becomes lost the relocalization algorithm will select the best matching keyframe, which may or may not be in the map that was expected. The worst case is that the system will never recover to a particular map, because other overlapping maps always have keyframes that obtain a higher score. Fig. 5.11 shows frames from a sequence with this occurring.

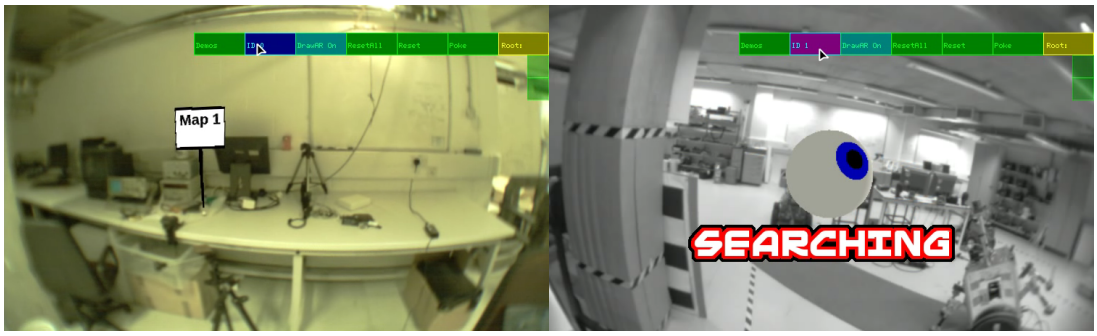
Expt 3b. With careful manipulation however, this can be used to build detailed maps "inside" other maps. In the example shown in Fig. 5.12, a map is made of a portion of a room, and a second detailed map is made on a keyboard. The keyboard exists in both maps, however it was not observed in detail in the room map.

5.6 Experiments II: medium to large scale

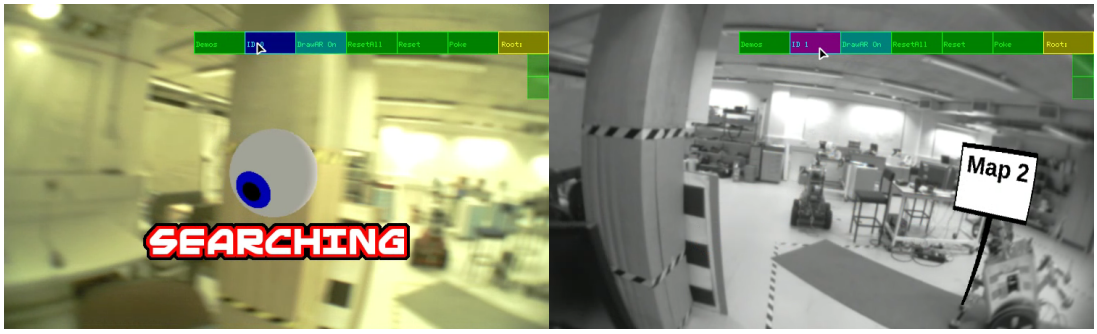
The next set of three experiments, show the system working on a larger scale. The first shows the robustness, and some limitations of the system, the second shows a practical use for overlapping maps, and the final one shows how the system can be used to explore large, sprawling environments.



(a) Both cameras in map 1.



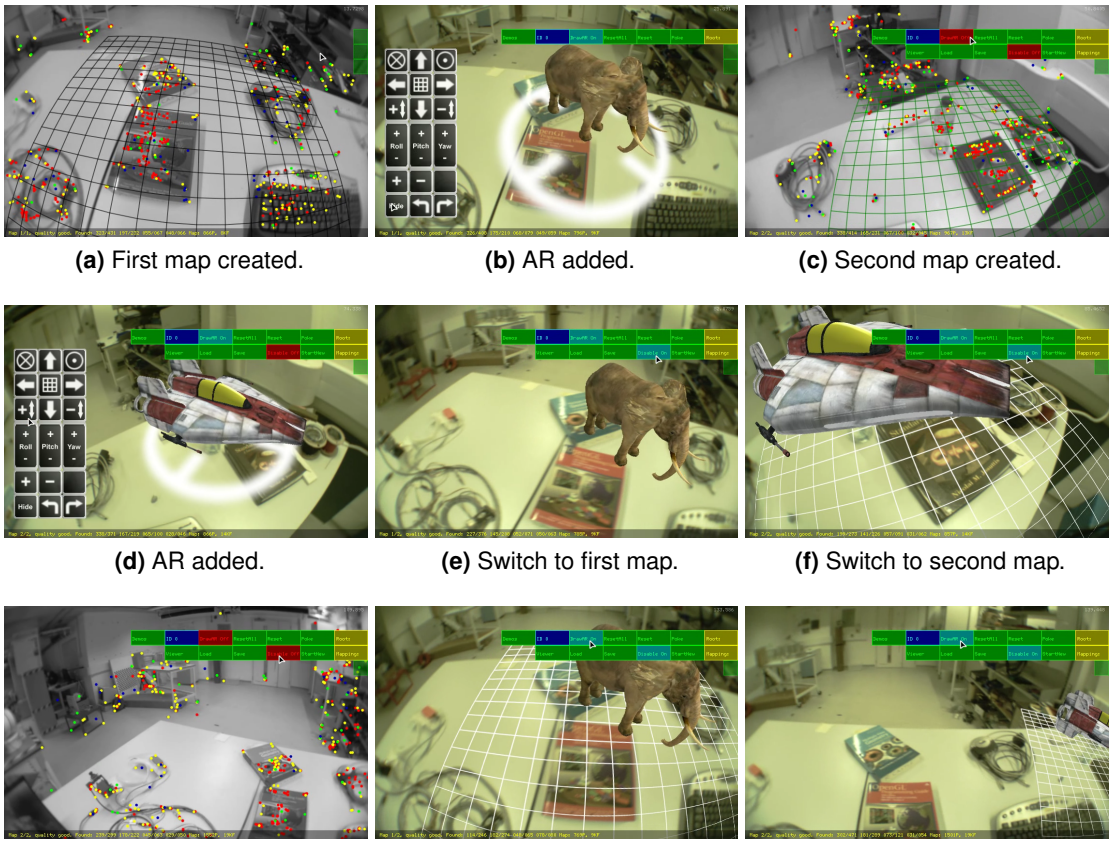
(b) Wearable camera lost, looking at map 2.



(c) Hand held camera lost, wearable camera recovers to map 2.

(d) Hand held camera also recovers to map 2.

Figure 5.10: *Expt 2b.* Example showing how multiple cameras can switch between multiple maps (see video exp2_map_switching_2cam.avi).



(a) First map created. **(b)** AR added. **(c)** Second map created.

(d) AR added. **(e)** Switch to first map. **(f)** Switch to second map.

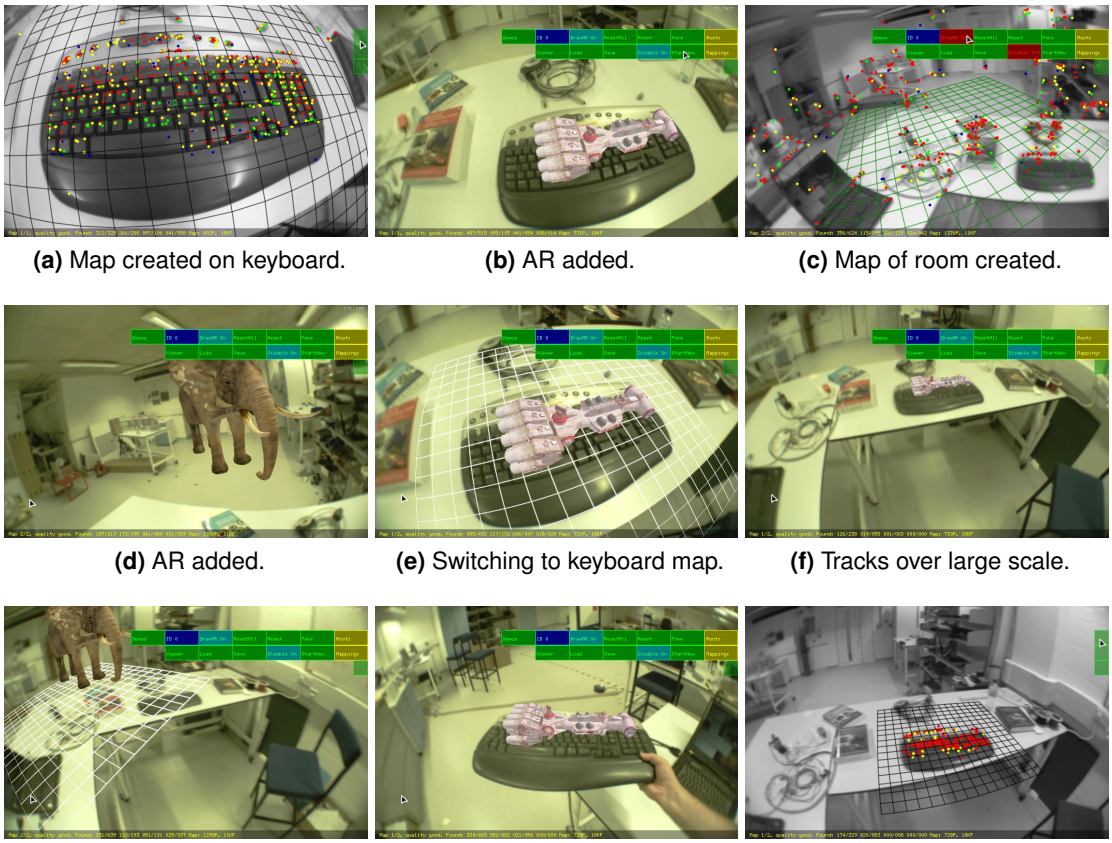
(g) Second map extended over first. **(h)** Indeterminate recovery. **(i)** Indeterminate recovery.

Figure 5.11: *Expt 3a.* Example showing how overlapping maps can cause indeterminate switching behaviour (see video exp3_overlapping_maps.avi).

5.6.1 Desk sequence

Expt 4. The purpose of this experiment is to demonstrate the multiple mapping capability of the system, its robustness to similar maps, and how it compares to the original single map system. In this experiment 12 maps were made on 15 desks. Each desk is similar (curved shape, with a computer), but has varying amounts of clutter depending on its occupant. The user creates the maps and adds some AR to show to whom the desk belongs, and which group they are a member of.

The system was able successfully, and correctly, to relocalize onto all of the mapped desks. Frames f–h of Fig. 5.13 show the relocalization of three of the maps. The system was even able to relocalize correctly onto two desks (frames k and l) that were sparse in features (frames i and



(a) Map created on keyboard. **(b)** AR added. **(c)** Map of room created.

(d) AR added. **(e)** Switching to keyboard map. **(f)** Tracks over large scale.

(g) When lost, recovers to room map. **(h)** Maps on objects move with them. **(i)** Keyboard map features.

Figure 5.12: *Expt 3b.* Example showing how maps can be embedded within each other (see video exp3_mobile_maps.avi).

j). When the user is traversing the areas between maps the system becomes lost and attempts to relocalize (frame e). Once the user arrives at a mapped location the system is able to recover and tracking resumes.

At one point during the labelling (frame g), the map of one user's desk is allowed to grow and cover neighbouring previously mapped desks. This causes the two maps to overlap. Because no geometric links exist between the maps, the system has no way of handling the overlap in a defined manner. Instead the system will recover to the map that provides the best pose from the relocalizer, and in this case the system only recovers to the latter map, resulting in the label on the other desk not being seen.

Each map contained between 3 and 37 keyframes and 290 to 1 900 map points, resulting in a

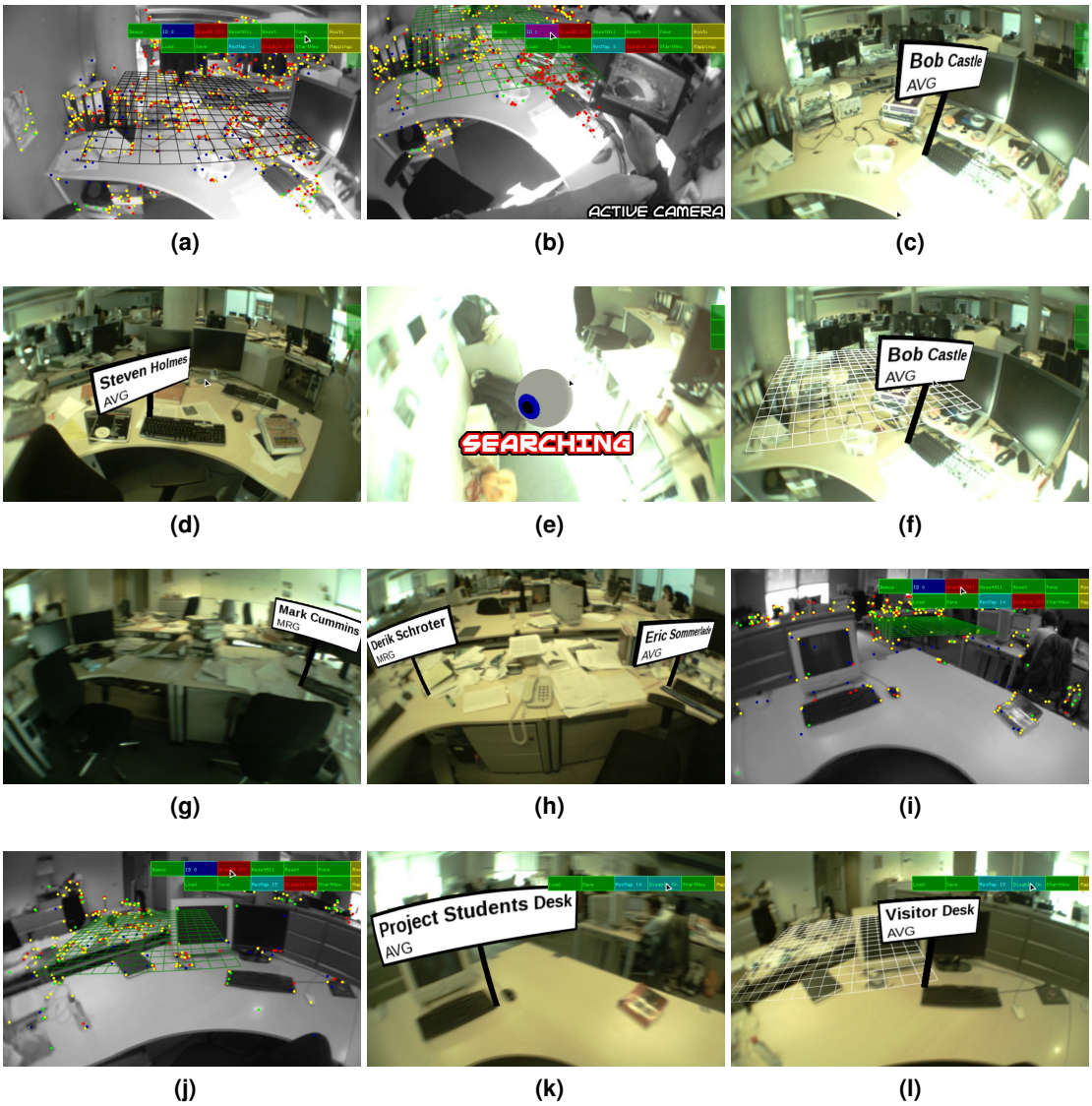


Figure 5.13: *Expt 4.* A demonstration both of multiple maps and robustness to self similarity of maps. 12 maps were made of 15 desks, and each desk was augmented with the user's name and research group. (a,b) Hand held camera and active camera view working in the same map, and (c) AR added to map. (d) Another map created and labelled. (e) Attempting relocalization, and (f–h) successful relocalization on different maps. (i–l) Creation of maps on two sparsely featured desks, and subsequent successful relocalization (see video exp4_desks.avi).

total of 177 keyframes and 12 327 map points. As a comparison, the original single camera and single map PTAM system was used to try and create one large map of the same environment. The system was able to effectively map the room, creating a map containing 163 keyframes and 10 144 map points. However, PTAM has no occlusion reasoning, and 3D map points that lay behind various objects in the room (pillars, monitors, *etc.*) were projected into the image

as potentially visible, but were consequently unmatched. Although tracking continued, it was deemed poor, preventing new views being tagged as keyframes, bringing further building of the large map to a halt. This leads to the system becoming more fragile to use, and the user having to retrace their steps when the system becomes lost more and more frequently, until a suitable relocalization position is found.

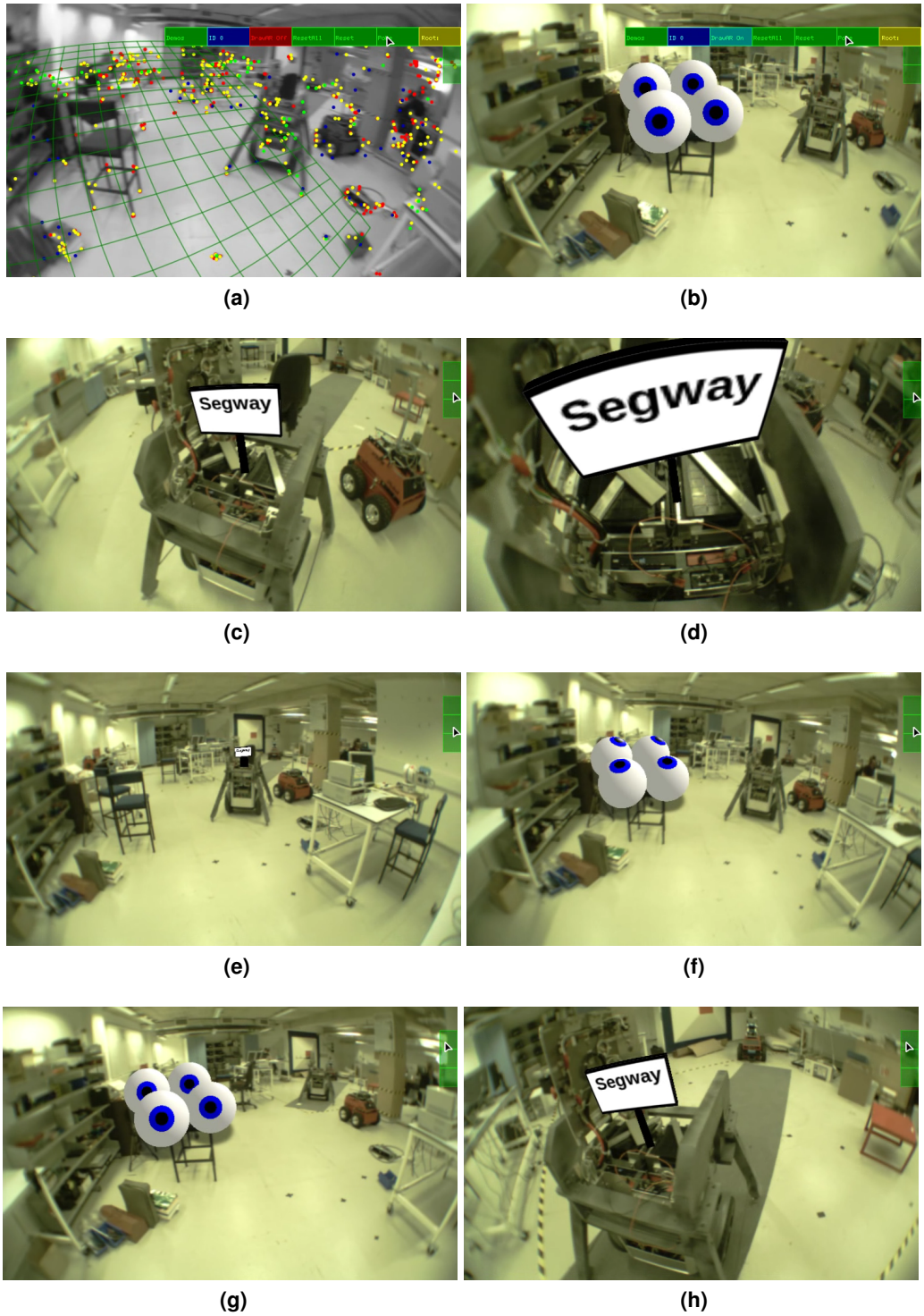
5.6.2 Laboratory sequence

Expt 5. This experiment shows a practical use for overlapping maps. Two maps are created: one is a large map of a laboratory, the other is a small map located on the top of a Segway mobile robot. In Fig. 5.14 frames a and b show the map of the laboratory being created and augmented. Frame c shows the robot map with its AR. The smaller robot map is located inside the first, larger, laboratory map, and they are therefore fully overlapping. The system will track the room map, until the user is looking at the robot close-up. At this point the system is unable to maintain tracking on the room map and enters relocalization. It relocalizes onto the robot map, and resumes tracking (frame d). It remains tracking the robot map as the user backs away (frame e), and only relocalizes to the room map when tracking is lost (frame f).

The main advantage of having a smaller map on the robot, as opposed to having one large map containing all the AR, is that this allows the robot to move to different locations and still maintain its own AR. In frame g the robot has moved, but the room map is still able to be tracked as the majority of the scene structure is fixed. When the user approaches the robot in its new location the relocalizer switches the system to the robot map (frame h).

5.6.3 Building sequence

Expt 6. This penultimate experiment shows how the system can be used in large sprawling environments, providing AR around the environment. Twelve maps were made around the University of Oxford Engineering Science buildings on multiple floors. Fig. 5.15 shows frames



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 5.14: *Expt 5.* A demonstration of the ability to move between overlapping maps, and relocate maps that have moved. (a,b) Map of room created, and AR overlay added. (c) Map on mobile robot created, and labelled. (d) Switching to the robot map. (e) Robot map tracked over a large scale change. (f) Switching back to the room map. (g,h) After robot has moved relocalization is successful on the room and robot map (see video exp5_laboratory.avi).

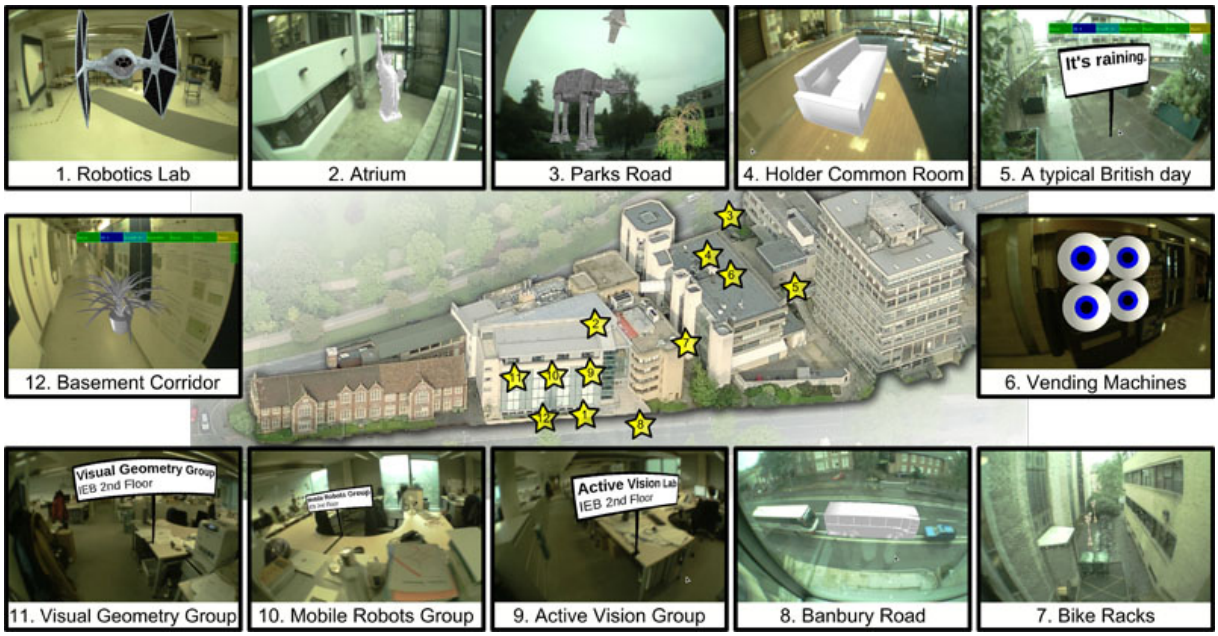


Figure 5.15: *Expt 6.* A demonstration of exploring a large scale environment containing twelve maps, with augmented reality overlays, over multiple buildings and floors (see video exp6_building.avi).

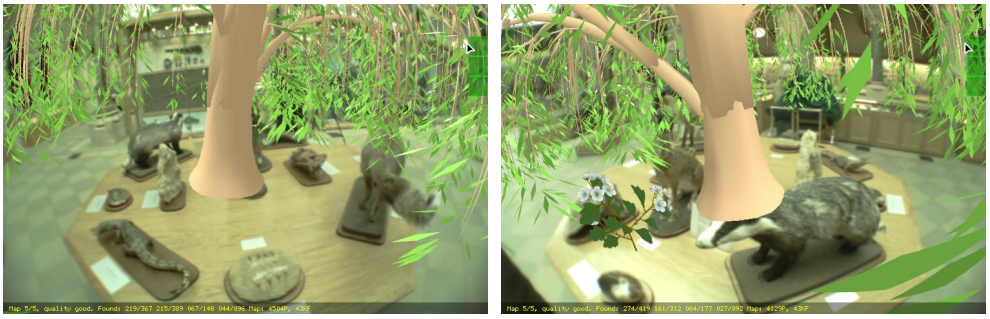
from the sequence at each of the map locations, along with the locations of each of the maps in the buildings. Once created the user was able to explore the building in an unstructured manner, with the system recovering the maps as the user entered each of them.

5.7 Experiments III: a prototype AR application

Expt 7. In this final experiment a real world application for multiple maps is shown. The system is used in a single camera mode, using the camera on the back of the hand held display. An AR tour is created in the Oxford University Museum of Natural History by creating maps around a variety of exhibits and adding AR models to them. The maps are then made read only and saved. The maps along with their associated AR can be loaded at a later date when a user wants to explore the museum using the system. As the user approaches an exhibit the system relocalizes into the appropriate map, and the relevant AR overlays are displayed. The users are free to explore the museum in an unstructured manner. Figs. 5.16 and 5.17 show frames from the video of the user's exploration, and Fig. 5.18 shows the system being used to build the maps and augment them. A selection of 3D views of the maps created are shown in Fig. 5.19.

(a) Museum entrance. **(b)** Tracking over a large scale.

(c) Triceratops. **(d)** Real world objects do not occlude AR.

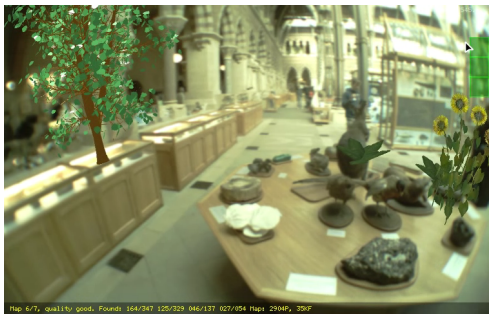


(e) Stuffed woodland animals. **(f)** No real world occlusions.

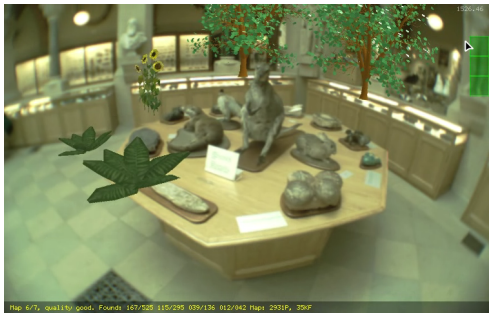
(g) Plateosaur AR. **(h)** Plateosaur AR.

Figure 5.16: *Expt 7.* Frames from the video showing a user exploring the natural history museum. Frames show the museum entrance, the triceratops, the first stuffed animals table and the plateosaur (see videos exp7_01_entrance.avi, exp7_02_triceratops.avi, exp7_03_table1.avi, exp7_05_dinosaur.avi, and exp7_07_tour.avi).

(a) More stuffed animals.



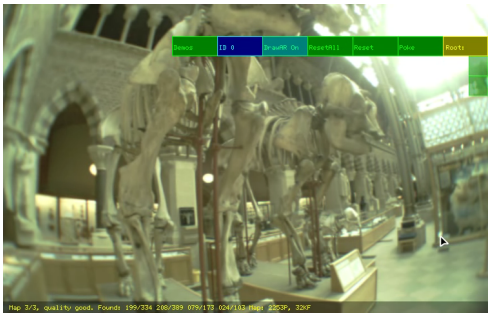
(b) User can view exhibit from all angles.



(c) Distorted map causes AR misalignment.

(d) Elephants.

(e) Close up view with AR.



(f) Close up view without AR.

Figure 5.17: *Expt 7.* Frames from the video showing a user exploring the natural history museum. Frames show the second stuffed animals table and the elephants (see videos exp7_04_table2.avi, exp7_06_elephants.avi, and exp7_07_tour.avi).

The tracker was able to relocalize into all of the maps as the user explored the museum, however there were problems with the system that became apparent. They fall into two main categories: (i) user interface problems; and (ii) mapping and tracking problems.

5.7.1 User interface problems

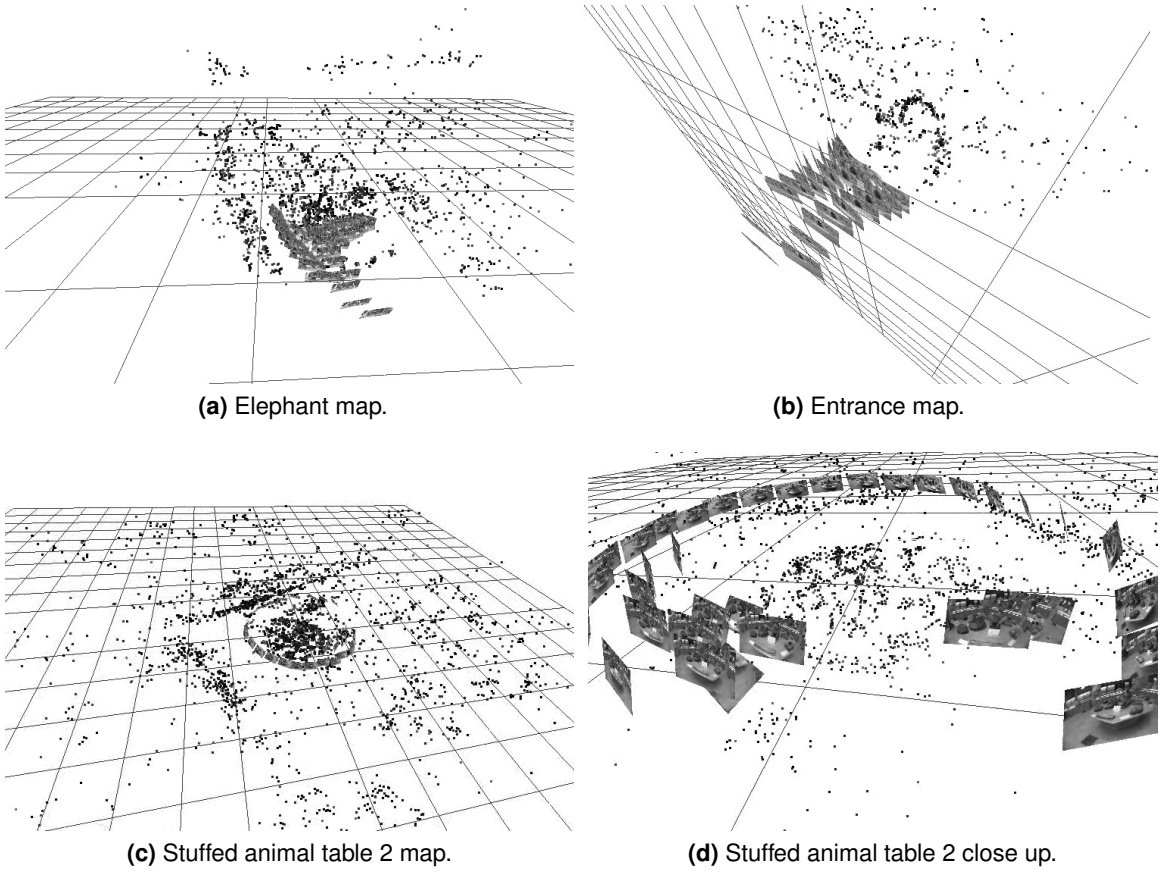
Well placed AR models can look very convincing, the elephant models in particular show how the AR can enhance the exhibit. This can be seen in Figs. 5.17e and 5.17f where the user switches

(a) Triceratops. **(b)** Stuffed animals. **(c)** Elephants.

Figure 5.18: *Expt 7.* The AR system being used to create maps and augment the museum exhibits.

from viewing the 3D model to the actual skeleton. However, the 3D models used where not tailor made to the exhibit, which meant that the overall shape of the 3D model never fitted the skeletons exactly. From the front the elephants looked correct, but as the user moved to the back, it could be seen that the back legs did not match up. For a commissioned AR exhibition custom models would be needed.

A second problem was with the placement of models. This user interface problem became apparent during prior testing, where an object placed in the scene looks correct from one view, but moving to another shows that the object is at a completely different location and depth. To help alleviate this issue the system was modified to allow models to be placed where a map point was. Once a user has found a well localized map point, clicking on the point moved the model to this location. The models in this experiment were placed in this manner, however the triceratops model proved problematic. Initially the model was to be placed directly over the skull in the same way as the elephant models. The problem was that the skull was close to the floor, requiring the model to be standing below floor level. Selecting a point that was at floor level resulted in a model that was located correctly at that point, but then subsequently moving the model and scaling it to get it at the correct depth resulted in the prior issue of not knowing what depth the model was at. This meant that the model could not be placed as desired and instead was placed next to the skull. To rectify this problem, further user interface controls are needed, such as being able to place arbitrary points on a model at map point locations. Being able to define an arbitrary planes to align a model with would also help.



(a) Elephant map. **(b)** Entrance map.

(c) Stuffed animal table 2 map. **(d)** Stuffed animal table 2 close up.

Figure 5.19: *Expt 7.* Selection of maps from the natural history museum experiment.

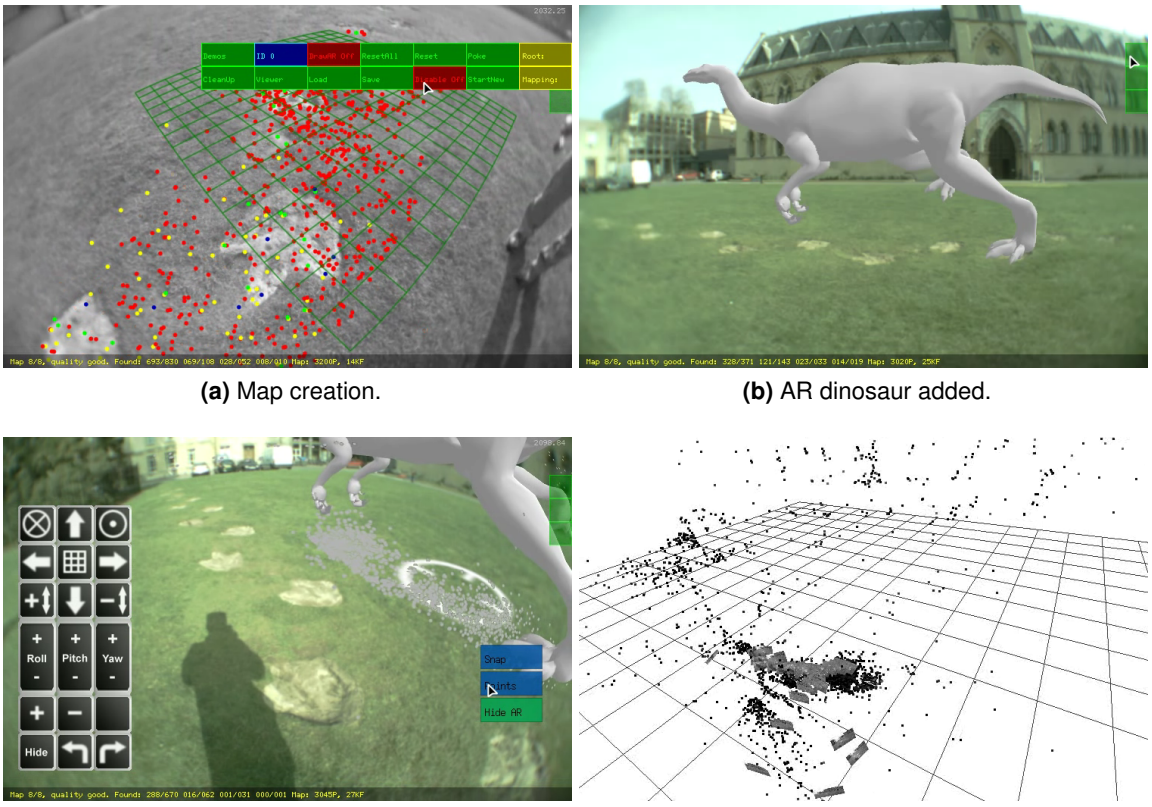
5.7.2 Mapping and tracking problems

The most obvious problem with PTAMM to a user exploring the museum is the lack of occlusion, which breaks the AR illusion. A well placed model in a reasonably open area could be observed from many positions convincingly, however if a real world object came in front of where the AR model should be the model would still appear in front, as shown in Figs. 5.16d and 5.16f. This happens because the system does not know about real world objects, and therefore cannot work out occlusion boundaries. The system could be extended to handle 3D objects and surfaces to allow realistic occlusions. The lack of occlusion information also affects the map points, with occluded points possibly getting deleted when found to be unobservable.

The second most noticeable problem is to do with loop closure, and the lack of it. When the camera completes a loop, such are circling a table, the first and final keyframes both observe

the same 3D features, which should be measured as the same 3D map points by the system. However, if there has been measurement drift due to tracking error, then the final keyframes will be far enough apart for the system to observe these 3D points as different points. Bundle adjustment usually stops these misalignments through its minimization. However, the minimization process can end up in a local minima which satisfies the observations, and the loop will remain unclosed. To this end separate algorithms are needed to detect loop closure and enforce these constraints. There is no mechanism in the system currently to detect loop closures, so instead the maps can have open loops as shown in Figs. 5.19c and 5.19d. It can be clearly seen from the visible keyframes that there is an overlap, but the measured geometry has them separated significantly. The tracker is still able to track in this map, but when it reaches one end of the loop it will keep tracking using the 3D map points from that end of the loop until enough are observed from the other end of the loop. At this point the camera's location will jump to the other end. When AR elements are added to the map the discontinuous nature of the map becomes very apparent. AR elements added using features from one side of the loop will look correctly located but as the user traverses the loop the AR elements will become misaligned as shown in Fig. 5.17c. The AR becomes aligned again when the tracker jumps from the end of the loop to the beginning. The system is usually able to close most small loops through bundle adjustment alone and accurate tracking, but this example shows that an additional process is required to catch the exceptional cases.

The final problem is to do with similar textures, which affects how the system tracks and relocalizes. Fig. 5.20 demonstrates how feature patches can become misaligned when the surrounding texture is similar to that of the feature patch. The map is created successfully, mainly due to steady camera motion and good tracking, and AR elements are added. However, as the camera moves back towards the map origin, the features on the grass become mismatched and the map drifts along the grass, eventually resulting in tracking failure. This is a common issue with patch matching and was also observed in the monoSLAM system in the previous chapter.



(a) Map creation.

(b) AR dinosaur added.

(c) Tracking failure.

(d) 3D map view.

Figure 5.20: *Expt 7.* PTAMM failure mode (see video exp7_08_footprints.avi).

One way around this would be to discard points that have patches that are too similar to each other and their surroundings. This would be fine for maps where the areas of repeating texture are small compared to the surrounding areas as these areas would become featureless regions, but there would be enough remaining features to track. However, when the repeating texture makes up a large proportion of the environment, such as with the grass here, the number of viable features is greatly reduced, making the tracking less robust to failure.

The small blurry image relocalizer works well for maps that have keyframes that are significantly different, providing a correct answer most of the time. The relocalizer stops working as effectively, returning more false positves when more keyframes are added that have similar descriptors, and when the camera traverses areas where the system is lost, and the frames being compared are similar to those in the map. The system handles these false positives by first trying to track from the puntative location for a few frames. If it is unable to track, then

(a) Incorrect relocalization. **(b)** Incorrect tracking. **(c)** Eventual failure.

Figure 5.21: *Expt 7.* Relocalization and recovery failure.

the tracking fails and the relocalizer starts again. In Fig. 5.21 the user is walking from one exhibit to another and the system incorrectly recovers to the entrance map. The tracking system then tries to track, and normally would fail after a few frames, but here the system tracks successfully for a few seconds, incorrectly matching features in the world to those in the entrance map. Eventually the tracking does fail, but it highlights the weakness in the relocalizer. It relies too much on the system's ability to track correctly or fail, instead of ensuring a more accurate answer is provided. If the map was not in a read only mode, then new keyframes would have been added, corrupting the map.

5.7.3 Summary

Using PTAMM an AR tour of the natural history museum was able to be created, mapping and placing AR only at locations of interest. The system relocalized correctly for the majority of the time allowing the user to freely explore the museum as they desired. The ability to build maps and augment them in real time allowed different layouts to be experimented with providing instant feedback. Improving the user interface to have more features like those found in a CAD modelling program would improve the user's ability to place models where they desired. Using multiple maps allows unique AR to be placed on each exhibit, and if an exhibit is moved the map would still be found on it. Also, if an exhibit is modified, only the map for that exhibit needs to be updated or redone. If a single large map had been used, any changes to the locations or content of the exhibits would mean either recapturing the whole map or carefully editing the

map to remove the invalid exhibit features, and then updating the map with the new changes and reimplementing the AR. Loop closure also becomes a larger issue as map sizes increase, allowing more opportunity for measurement drift.

5.8 Limitations and improvements

The limitations of the system have been covered largely in the experimental section, but the most important are highlighted here.

Relocalization. The relocalizer was designed to work when the camera was displaced slightly within its current (single) map. Being appearance-based, but somewhat weakly at the pixel level, it is easily confused when in an unmapped region with broadly similar appearance to one that has been mapped, and reliance is then placed on the tracking to fail to show that this is an unknown area. It would be better both the make the appearance element stronger, and to incorporate geometrical tests as part of the relocalizer. The FAB-MAP system by Cummins and Newman [22] provides useful pointers. Also, adding in higher order features for tracking such as edges could also help with tracking robustness as Klein and Murray demonstrated in [66] for the PTAM system.

Loop closing. Although relocalization was mentioned as a problem between different maps, it will become one for single maps when considering loop-closure, an area that has not yet been worked on in PTAM.

Occlusion. One issue that affects both the user experience and the underlying system is that of occlusion. Because PTAM is based on points and has no notion of surfaces, the AR illusion is broken when a virtual object moves behind a surface, but is still rendered. Also when a object occludes previously visible map points the system attempts to measure them, fails, and eventually deletes them if they fail to be observed enough, corrupting the map.

While there are methods to recover objects from point based structures, *e.g.*, recently, [55],

these methods are hardly automatic or fast. Segmentation using, say, superpixels [127], could be combined with edge and depth data to recover surfaces, but this remains a difficult area. Recovering crude surface structure would certainly help with occlusion reasoning for the tracker, but it is unclear how users would react to AR rendering which is sometimes occlusion-aware and sometimes not.

Global positioning. A last area of interest, particularly from the perspective of pervasive computing, HCI and applications, is that of adding absolute location information to each map and the user, using GPS or alternative technologies for indoor operation. Being able to guide a user to other maps is one advantage, another is that of downloading relevant maps, *i.e.* within a certain distance around the user, to the wearable system while on the move.

5.9 Conclusion

This chapter has shown that the ability to build multiple modestly-sized maps within an environment is useful in wearable applications in which the wearer needs to, and can be trusted to, move around the world and between mapped regions without detailed guidance.

It has been shown that the method of parallel tracking and mapping provides the required geometric underpinning, but also conveniently splits the processing in a way that can be exploited to allow multiple cameras to build and work in multiple maps. In turn this allows much larger areas to be mapped.

The resulting system, PTAMM, has been demonstrated in a range of experiments of increasing scale, showing that multiple maps allow for augmentation of individual objects and large sprawling environments. No particular impediment has been found to scaling the system upwards as further computing resources become available. Although, a more intelligent relocalizer is required to allow the use of appearance and geometry, and to allow hierarchical accessing of the maps.

6

Parallel object recognition, tracking and multiple mapping

In this chapter the parallel tracking and multiple mapping system from the previous chapter is combined with the recognition of planar objects from a database. The system is able to build multiple feature rich maps of the world and simultaneously recognize, reconstruct, and localize objects within these maps. The object reconstruction process uses the spatially separated keyframes from the tracking and mapping processes to recognize, reconstruct and localize known objects in the world, allowing augmented reality overlays to be placed relative to the objects. The process is independent from the mapping and tracking processes.

6.1 Introduction

In Chapter 3 a method was developed which combined object recognition and localization with simultaneous localization of the camera and mapping of the environment provided by monoSLAM. The experimentation in Chapter 4 provided evidence that including objects in the maps improved the user experience, and including them as measurements improved the quality of the map. However, the performance of the whole was compromised by the sparseness and small extent of the map. Chapter 5 addressed this by replacing monoSLAM with PTAM, increasing the number of points in an individual map by two orders of magnitude, and the linear dimension by approximately one order of magnitude. Furthermore, multiple maps were introduced as a method for freeing the user from the confines of a small workspace.

This chapter will re-introduce objects, but now in the context of PTAMM, and with significant changes in the way they are located and managed.

In Chapters 3 and 4 planar objects were recognized in an image by running SIFT, and each located by deriving the homography between image and database entry of known size. An object was added to the SLAM map as three point measurements. This added significant information to the map, but simply because the map was so sparse.

In PTAMM however, the number of points in the map is up to two orders of magnitude higher and there would be little benefit to the geometrical integrity of the map from so small a number of additional points. Although, of course there remains significant benefits to the user from augmenting the map with recognized objects. Because of this there are four significant changes in the object recognition and localization process.

The first change is that instead of grabbing the current frame whenever the object detection process is ready, only keyframes are used as the input images for the object detection process. Keyframes are, by design, spatially well separated and densely cover the mapped area, allowing a thorough search of the space. This is a vast improvement over the original method were only the latest frame was used, which was acceptable during new exploration, but not when the camera was still or looking around the same area.

The second change is that instead of requiring all objects to be physically measured beforehand, which in real world situations is not always practical, only a fronto-parallel image of the object is required. The rectification of the database image is not required.

The third change is related, and results from the removal of the metric information. Without this information an object cannot be localized in 3D from a single image. Instead points on the object are reconstructed using triangulation from keyframes. The keyframes facilitate this because the pose of each keyframe is known from the mapping process, and the density of the keyframes ensures that an object will be seen several times from different viewpoints. Removing the metric scale from objects also removes the need for the maps to have a metric

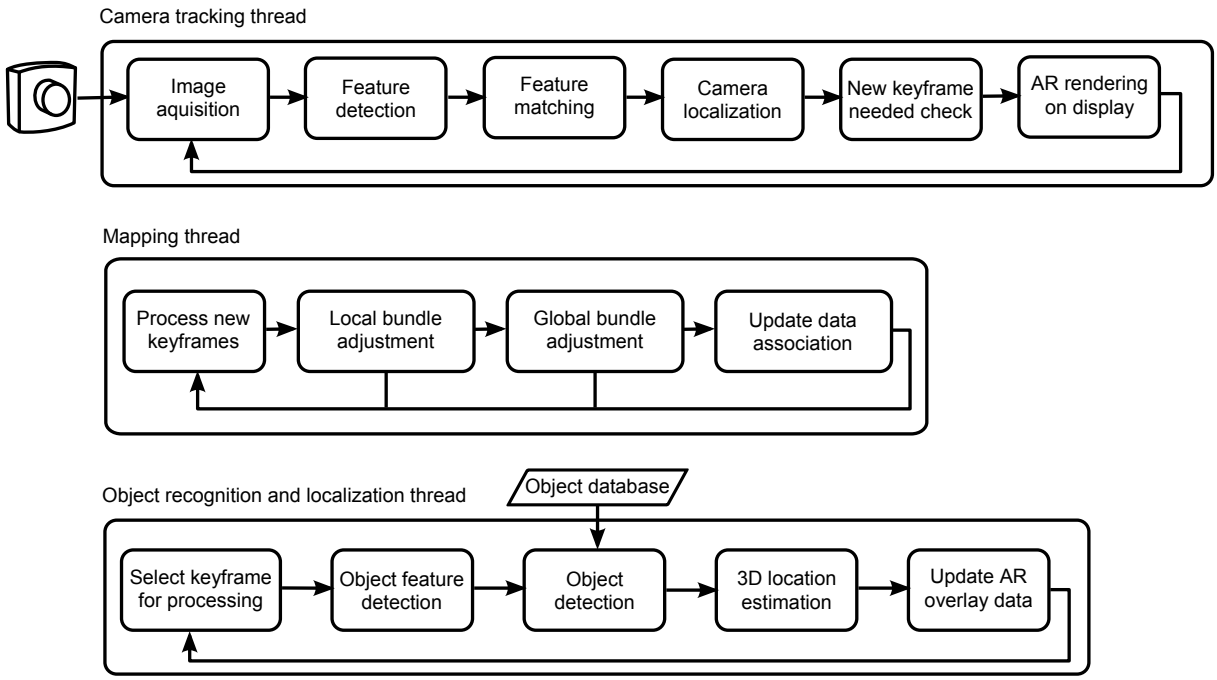


Figure 6.1: The process flow for PTAMM with object recognition and localization. The processing threads are all independent.

scale.

The fourth and final principle change is that the detected objects are not added to the map as measurements. Their locations are held separately, and only used for AR applications.

Fig. 6.1 shows the process flow for object recognition within the PTAMM framework. The first two rows summarize PTAMM's camera tracking and mapping threads as they run in steady state. The third row is the focus of this chapter and summarizes the handling of objects: their recognition, reconstruction, and localization. It can be seen that the object recognition process is added as a new thread that is all but independent of PTAMM, and that it supplies results to the AR engine. The underlying PTAMM system is unchanged. This separation would also allow multiple or indeed remote object detection processes. These possibilities are discussed in Section 6.12 at the end of the chapter.

The following sections describe each of the processing blocks shown in the object recognition and localization thread in Fig. 6.1. Section 6.3 details how a keyframe is selected for processing, and Section 6.4 describes how objects are recognized from the features. Section 6.5 details how

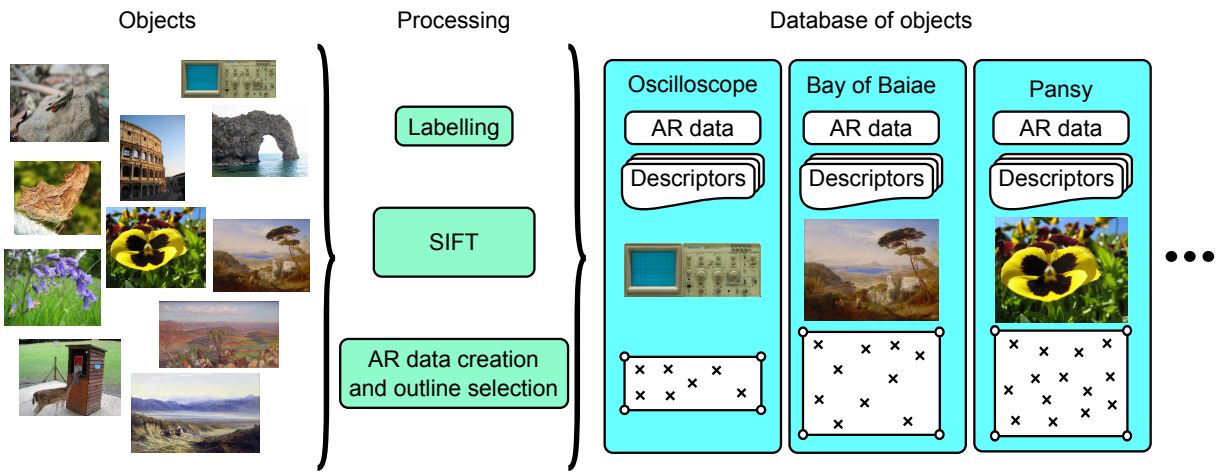


Figure 6.2: The object database contains planar objects (here pictures), the list of SIFT descriptors and their locations $\boldsymbol{x}^i$ (crosses), and the locations of boundary points.

recognized objects are reconstructed and localized in the world, and Section 6.6 describes how the AR is displayed so that it is related to the detected objects. An experimental evaluation of the system is carried out in Section 6.8. The chapter concludes with a look at some of the limitations and improvements. First, the next section describes the changes made to the object database.

6.2 Object database

The object database is similar to that used in Chapter 3, though now there is no need to store the object dimensions, or to select three boundary points.

To construct a database entry, an image of the object is captured and, after correcting for radial distortion, SIFT descriptors $\boldsymbol{\sigma}^i$ and their positions $\boldsymbol{x}^i$ are computed. The database entry

$$\mathcal{O}_j = \left\{ \mathcal{I}_j,\ \{\boldsymbol{\sigma}^i_j,\ \boldsymbol{x}^i_j\}_{i=1\ldots I_j},\ \{\boldsymbol{x}^m_{\mathrm{B}j}\}_{m=1\ldots M_j},\ \{\boldsymbol{x}^n_{\mathrm{AR}j}, \text{“AR-markup”}\}_{n=1\ldots N_j} \right\} \tag{6.1}$$

contains the frontal image $\mathcal{I}_j$ of the object, the list of SIFT descriptors $\boldsymbol{\sigma}^i_j$ and their image locations $\boldsymbol{x}^i_j$, the locations of boundary points $\boldsymbol{x}^m_{\mathrm{B}j}$ to define the object extent, and lastly a number of positions $\boldsymbol{x}^n_{\mathrm{AR}j}$ tagged with graphical annotations. Fig. 6.2 shows a representation of the database.

6.3 Keyframe selection

As the recognition process runs independently from the mapping process any keyframe could be selected, and at any time, to detect objects in.

One obvious order of selection would be temporal, the order that they were added to the map. But when the user is exploring a new area of the scene, keyframes will be added at a more rapid rate than the object recognition thread can process, leading to a backlog. In this case another obvious approach might be to always process the most recently arrived keyframe first, and hope that there is time later to clear the backlog.

However, there are three considerations that make these poor stratagems. First is that this process must not only recognize objects, but also localize them, and second is that, at a minimum, two keyframes containing the same recognized object are required for localization. Last is that providing information on areas where the camera is currently looking is a priority, and here it is worth stressing that the order in which keyframes are added, does not necessarily correlate with their proximity. When the camera is exploring new areas there will be correlation, but if the user is revisiting an already mapped environment then keyframes will be added only occasionally, and temporal neighbours may well be from opposite ends of the mapped area.

The stratagem adopted therefore selects keyframes in pairs: the first processed is that keyframe whose position and orientation are closest to the current camera's; and the second is the keyframe that is most similar to the first, as explained shortly. First though, the correct map needs to be selected.

As the user may be using multiple maps, the object recognition process needs to check the status of the keyframes in the maps to decide which map's keyframes should be processed. There are two conditions that can occur: either keyframes in the current map are being processed, or keyframes in another map are. Fig. 6.3 details the map selection process used. The preference is to process new keyframes in the current map to ensure the best user experience.

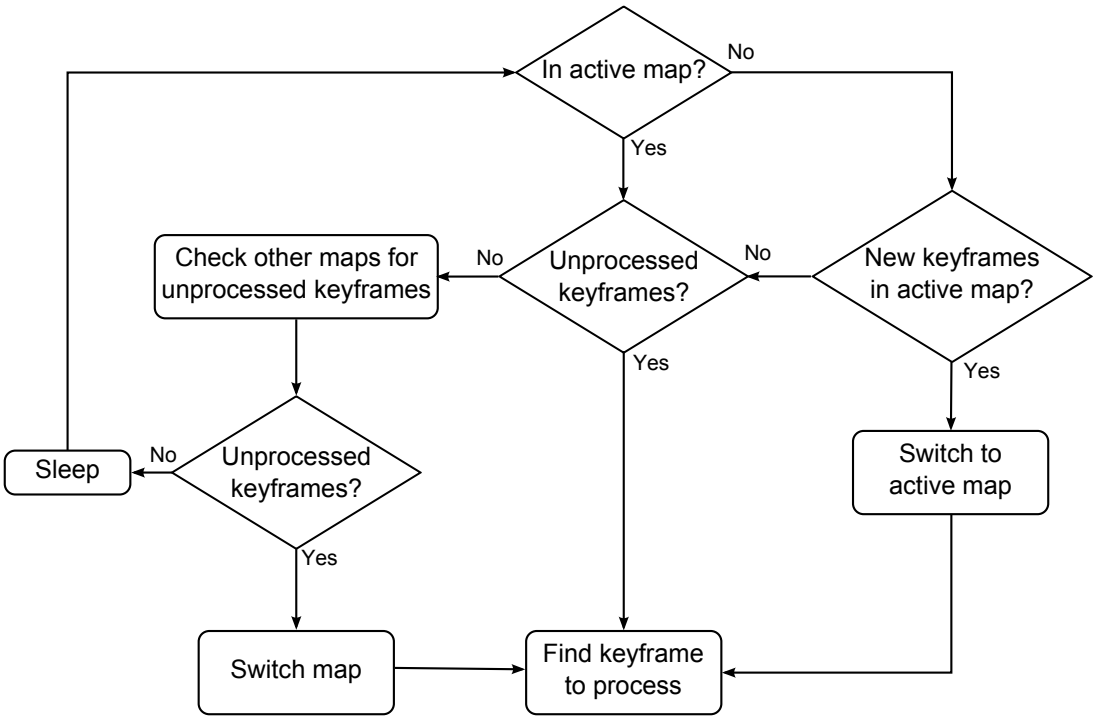


Figure 6.3: The process flow for selecting a map to process keyframes from.

However, if all the keyframes have been processed in the current map, the system seeks unprocessed keyframes in the remaining maps. In this way if the user makes multiple maps faster than the system can process the frames for each, the system will return to finish off when it has time. Recognized objects will now be there when the user returns to the earlier maps.

Once a map has been selected, a keyframe in the map is selected. The camera's location is known, and so are the locations of the keyframes, allowing the nearest keyframe to the user to be selected for processing. This provides the selection criteria for the first keyframe of a pair. The second keyframe needs to be one that is nearby and looking at the same area. To find this frame a modification to the map maker process is made. Whenever a keyframe is added to the map, the map maker records which keyframe already in the map is most similar to the new one. This becomes its parent, and the parent also records that this new keyframe is a child, forming a bidirectional tree. The similarity measure used is the number of map points the two keyframes have in common. This is an efficient search in the PTAMM framework as each map point has a list of keyframes it has been observed in. This bidirectional tree allows all of the

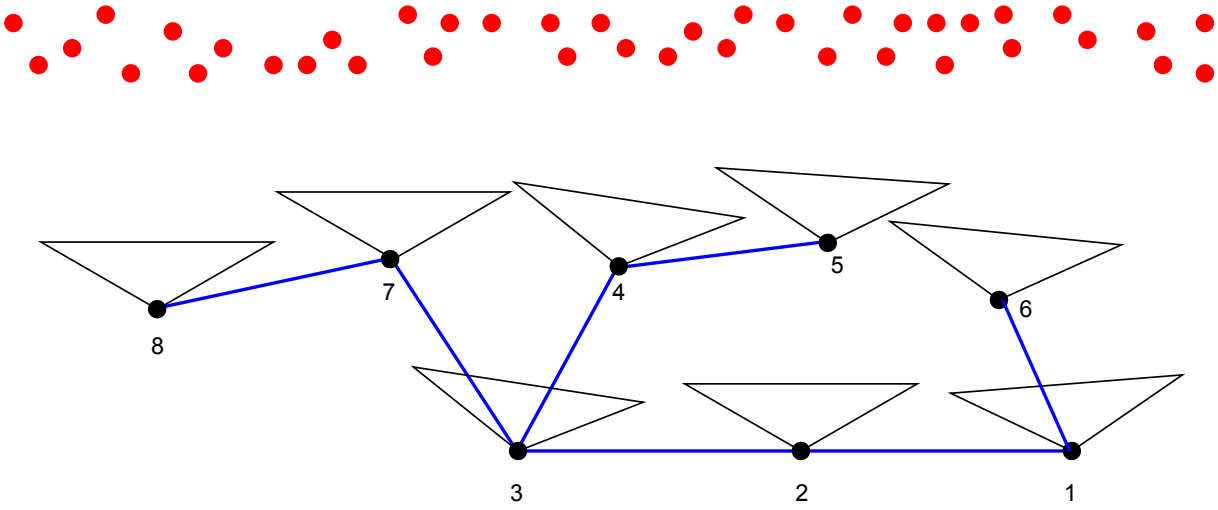


Figure 6.4: The bidirectional tree formed for the keyframes does not take on a linear structure. This diagram shows a top down view of keyframes (triangles), numbered in temporal order, and their observed points (red circles). The bidirectional tree (blue line) splits at keyframe 3, as this keyframe has more points in common with 4 and 7 than they do with each other.

most similar frames to be quickly located for any particular frame. A representation of this tree is shown in Fig. 6.4, where it can be seen how the tree structure selects frames that have the most points in common and not necessarily the last frame added. Fig. 6.5 details the keyframe selection process. The selection defaults to processing the most recent unprocessed frame, as this is the next most likely to contain pertinent information, if one of the following conditions occurs: the first keyframe of the pair contained no objects; any found objects had already been localized; all of the relatives had been processed.

6.4 Object recognition

Once a keyframe k has been selected for processing, SIFT descriptors and their locations ($\boldsymbol{\sigma}_k^l$, $\boldsymbol{x}_k^l$, $l = 1...L_k$) are extracted from its associated image. These keypoints are stored in the keyframe structure. The keypoint descriptors are compared with those in the database using Beis and Lowe's approximate best-bin-first modification of kd-trees [10], as reviewed in §3.4.2. If the number of keypoints matched between the keyframe image and any given object's database entry exceeds a threshold, that object is flagged as potentially visible. However, because of repeated structure or other scene similarity, some of the features may be incorrectly matched.

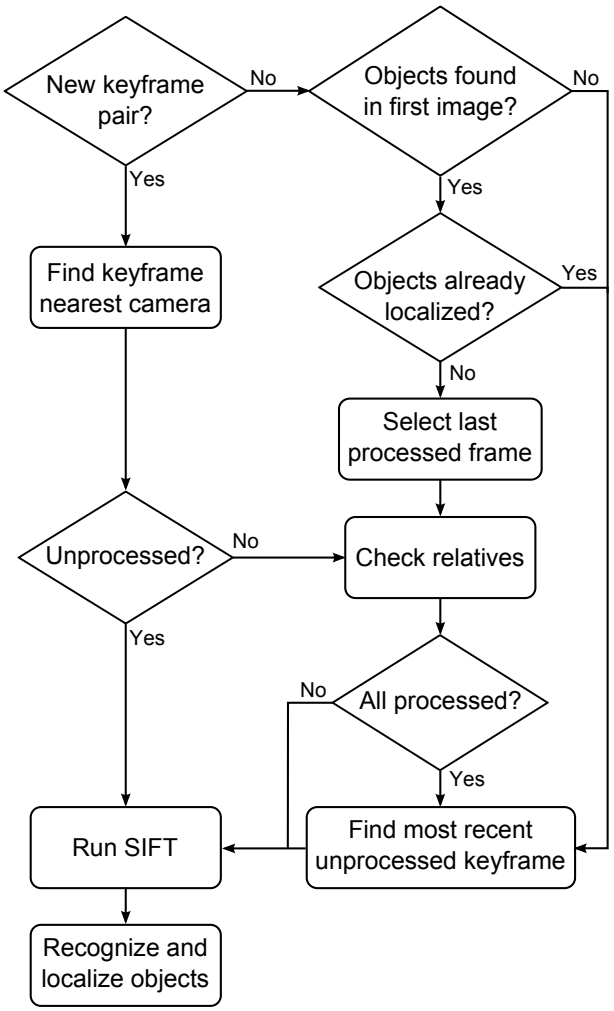


Figure 6.5: The process flow for selecting and processing pairs of keyframes.

Here the object's planarity is exploited to remove outliers, by using RANSAC to estimate the homography between the database feature positions and the keyframe feature positions, $\boldsymbol{x}_j = \mathtt{H}\boldsymbol{x}_k$, and inferring that the object is indeed visible if the consensus set of inliers is large enough. The inliers are added to a list of observations for their particular database keypoint, for object reconstruction and localization. It is stressed that this robust fitting is merely used as a method of segmentation, not localization. Indeed, the homography itself is discarded.

6.5 Object reconstruction and localization

At this stage there will be a subset of keyframes where SIFT keypoints have been extracted and matched to object keypoints in the database. Recognized objects now have a list of keypoints that occur in a set of keyframes, with some features occurring in multiple keyframes. All that is currently known is that particular objects occur in the scene, but not where in 3D.

To reconstruct and localize an object a method that is very similar to the map making process of the PTAMM system is used. Once an object has been found visible in two or more keyframes, there will be a subset of object keypoints that are observed in two or more of the keyframes. First their scene positions are reconstructed quite generally, and only then are they fitted to the underlying shape of the model to obtain the position and orientation of the object in the scene.

Once a database keypoint has been observed in at least two keyframes it can be triangulated in 3D as the locations and poses of the keyframes are known from the mapping process. Then, when multiple database keypoints have been triangulated, bundle adjustment is performed to robustly minimize the error in the locations of these points. A plane is then fitted to the points and the points are projected on to this plane. The location of the object in this plane is then found from the projected point locations.

6.5.1 Triangulation

To triangulate a keypoint, the poses of the keyframes are treated as fixed, as they have already been optimized in PTAMM's bundle adjustment process.

With just two views the usual algebraic residual is minimized [54]. Up to scale, the two observations of the homogeneous scene point $\boldsymbol{X}$ are $\boldsymbol{x}_1 = \mathtt{P}_1\boldsymbol{X}$, and $\boldsymbol{x}_2 = \mathtt{P}_2\boldsymbol{X}$, where the projection matrix for each view $\mathtt{P}_{1,2} = \mathtt{K}[\mathtt{R}_{1,2}|\boldsymbol{t}_{1,2}]$ is known. Combining these,

$$\mathtt{A}\boldsymbol{X} = \begin{bmatrix} x_1\boldsymbol{p}_{13} - \boldsymbol{p}_{11} \\ y_1\boldsymbol{p}_{13} - \boldsymbol{p}_{12} \\ x_2\boldsymbol{p}_{23} - \boldsymbol{p}_{21} \\ y_2\boldsymbol{p}_{23} - \boldsymbol{p}_{22} \end{bmatrix} \boldsymbol{X} = \boldsymbol{0} \tag{6.2}$$

where $\boldsymbol{p}_{ij}$ is the j-th row of $\mathtt{P}_i$, and the residual is minimized when $\boldsymbol{X}$ is, up to scale, the column of $\mathtt{V}$ corresponding to the smallest singular value in the SVD, $\mathtt{UDV}^\top \leftarrow \mathtt{A}$.

6.5.2 Bundle adjustment

As more observations are added, Levenberg-Marquardt [89] is used to minimize error in the image, and the inhomogeneous $\boldsymbol{X}$ is estimated so as to minimize the L_2 norm of errors in the image

$$\boldsymbol{X} = \arg\min_{\boldsymbol{X}^*} \left\{ \sum_k ||\boldsymbol{x}_k - \boldsymbol{x}_\mathrm{p}\left(\boldsymbol{X}^*, \mathtt{P}_k\right)||_2 \right\} \tag{6.3}$$

where $\boldsymbol{x}_k$ is the (inhomogenous) observation in keyframe k and $\boldsymbol{x}_\mathrm{p}()$ is the predicted image position.

The objective for minimization is of course similar to that used in the PTAMM system. However, note that here the camera poses are known, so triangulation is being performed, not bundle adjustment. The assumption that the keyframes are fixed is not completely accurate, as adding further keyframes causes the map maker to readjust the map, possibly causing keyframes to move. This is handled by checking for a change in keyframe poses and rerunning the bundle adjustment for the affected objects.

6.5.3 Plane fitting

The reconstructed points could be of any surface, however, here the points are known to lie on a planar surface. Hence, once at least three keypoints have been localized a plane is fitted to them using RANSAC to expose outlying data. For the inlying set, the mean and covariance

$$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{X}_i \quad \mathtt{C} = \sum_{i=1}^{n} (\boldsymbol{X}_i - \boldsymbol{\mu})(\boldsymbol{X}_i - \boldsymbol{\mu})^\top \tag{6.4}$$

are computed. The plane normal $\hat{\boldsymbol{n}}$ is the column of $\mathtt{U}$ corresponding to the smallest eigenvalue in $\mathtt{\Lambda}$ from the eigendecomposition $\mathtt{U\Lambda U}^\top \leftarrow \mathtt{C}$.

The inliers are now projected onto the plane,

$$\boldsymbol{X'}_i = \boldsymbol{X}_i - \hat{\boldsymbol{n}}(\hat{\boldsymbol{n}} \cdot (\boldsymbol{X}_i - \boldsymbol{\mu})), \tag{6.5}$$

and these in-plane point locations are now used to locate the object. The optimized locations of the keypoints found by the bundle adjustment process are left unchanged, and only the projected inlier set is used to fit the object.

6.5.4 Object fitting

The final stage in locating the object is to determine the optimum rotation and translation that takes the database keypoint locations $\boldsymbol{x}^i$ to those on the located plane $\boldsymbol{X'}_i$. Then the boundary points of the object can be found in 3D.

For n iterations, where here $n = 100$, two of the projected inlier points $\boldsymbol{X'}_i$ are selected at random to act as scaling and rotation reference points. The database keypoints of the inlier set are transformed from the image plane of the object to the estimated 3D plane, relative to the two projected inlier points. The pair that result in the minimum distance between points are accepted as the best match. The boundary points $\boldsymbol{x}_{\mathrm{B}}$ are then found relative to the best pair of in-plane points, and saved as additional data with the map. The same is also done for any AR annotations $\boldsymbol{x}_{\mathrm{AR}}$ located on the object. The object can now be used in the AR rendering process.

6.6 Rendering

Once the boundaries for the objects have been found their outlines can be drawn on the user's display. The rendering process is the same as the original underlying PTAM system. The rendering process takes account of lens distortion by first undistorting the current camera image and rendering it as a background, overlaying graphical elements, then distorting the entirety back so that the camera image has its original form.

```
<objectDB version="0.1">
  <Objects size="2">
    <Object Image="deer.jpg" Sift="deer.key" Name="Deer" Description="One hungry
        deer" />
    <Object Image="oscilloscope.jpg" Sift="oscilloscope.key" Name="Oscilloscope"
        Description="">
      <overlays>
        <overlay id="0" type="abs" x="320" y="240" image="welcome.png" />
        <overlay id="1" type="float" x="500" y="320" label1="Power Button"
        label2="Press to turn on oscilloscope" />
        <overlay id="2" type="float" x="586" y="100" label1="Vertical Position"
        label2="Dial until line on display is centred" />
      </overlays>
    </Object>
  </Objects>
</objectDB>
```

Listing 6.1: Example XML object database with two entries.

6.7 Implementation

The software developed for this system builds upon the PTAMM software of the last chapter. It also integrates a version of the object database, and SIFT libraries developed for the object-SLAM software in Chapter 4. These have been modified to use the CVD and TooN libraries instead of the VW library to be in line with PTAMM. They have also been modified to handle the simplified databases. The XML database listings now look like the example shown in listing 6.1. A basic entry now consists of a reference to an image file (used for display), a SIFT keypoints file, the name and a description of the object. An extended entry has an overlays section, and this is the same as the one described in §4.2.1.

6.8 Experimental evaluation

As in the previous experiments, the machine used is a 2.20GHz Intel Dual Core portable computer. As described earlier, this system builds upon the multi-camera PTAMM system, and there is no fundamental limitation to the number of cameras that can be used. However, in these experiments only one camera is used due to the processing limitations.

To compare the effectiveness of this new method with the monoSLAM based system the

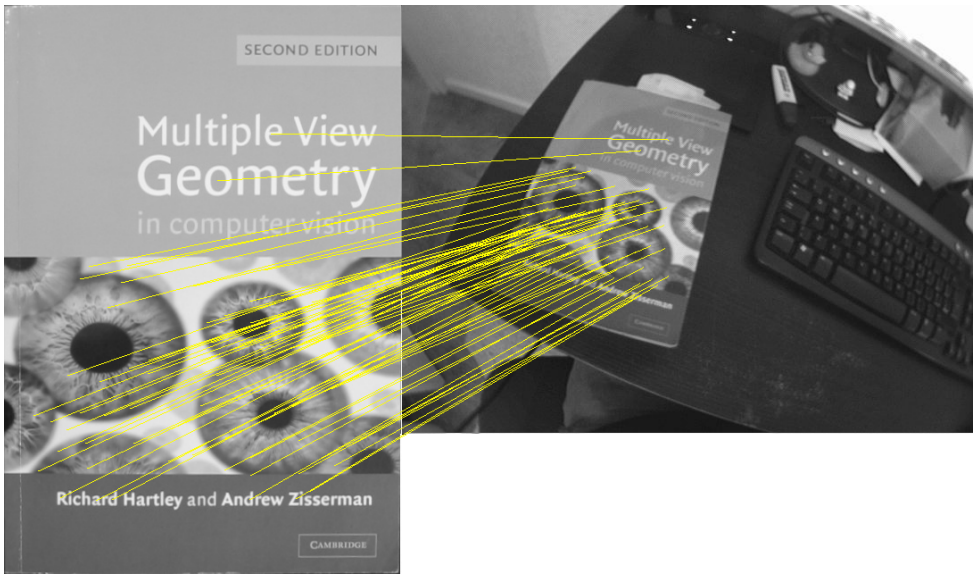
experiments from Chapter 4 are revisited. To begin with though, the first experiment shows how the developed method works in practice with a single object, demonstrating each stage of the processing pipeline. The following three experiments use the simple wall, poster wall, and cluttered desk sequences from Chapter 4. In these three experiments the accuracy of the recovery of objects is evaluated. The final three experiments evaluate the system in the real world scenarios of the gallery, the street, and the oscilloscope tutorial.

6.8.1 Evaluation

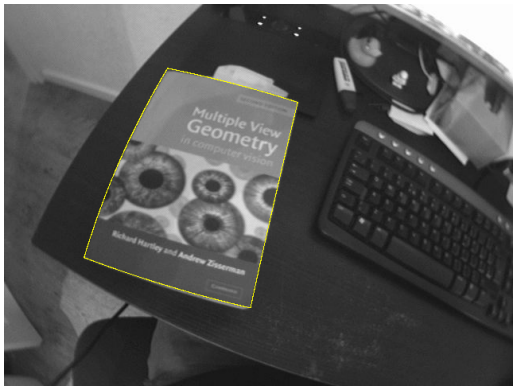
A single object is used here to illustrate the processing stages in the object recognition and localization thread.

Fig. 6.6(a) shows the frontal view of the database object. Of its 1 245 keypoints, some 67 keypoints have been matched to the particular keyframe image. Outlying matches were filtered using RANSAC to calculate the best fit homography, represented by the object's outline in (b). The object was found visible in 29 keyframes in this particular sequence, and (c) shows the plane fitted to the inliers of the localized points from these 29 keyframes. The object is then fitted to the inliers, represented by the outline drawn on the camera image in (d). Lastly, the AR related to the object is rendered to the display in (e). At the end of the sequence 224 database keypoints had been observed in the 29 keyframes, and 191 had been localized and all classified as inliers.

Fig. 6.7 shows a sample of the keyframes in which the object was detected, and the object is outlined using the calculated homography. The average times taken for each of the recognition and localization stages are shown in Table 6.1, and Fig.6.8 shows the timing graph for each iteration of the process. It can be seen that SIFT dominates, followed by the database matching. The "Find objects" stage includes the homography estimation, and the initial triangulation of points.

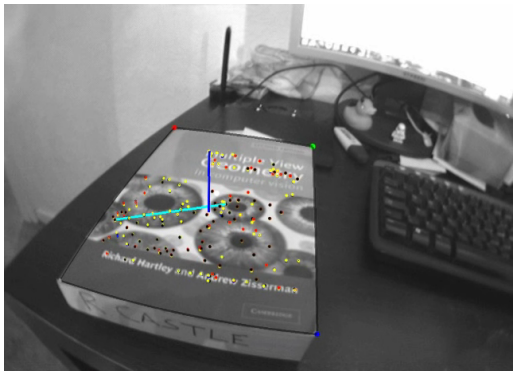


(a)

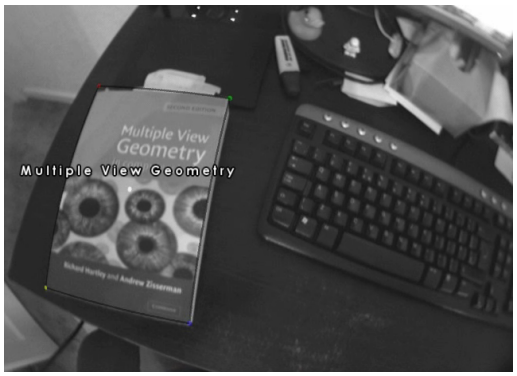


(b)

(c)



(d)



(e)

Figure 6.6: Object recognition and localization pipeline. (a) Detected keypoints are matched to the database keypoints. (b) Presence of object is confirmed. (c) Points are projected into 3D, optimized, and fitted to a plane. (d) Object is located in the map. (e) AR overlay is rendered (see video exp1_evaluation.avi).

Figure 6.7: Sample of 15 out of the 29 keyframes where the book was detected, with the calculated homography represented by the book's outline.

Section	Average time (ms)
Keyframe selection	0.6
SIFT feature extraction	1 529.9
Database matching	198.2
Find objects	33.9
Bundle adjustment	4.8
Plane fitting	0.4
Object fitting	2.2
Total	**1 770.0**

Table 6.1: Average times taken for each of the main processing blocks of the object recognition and localization process.

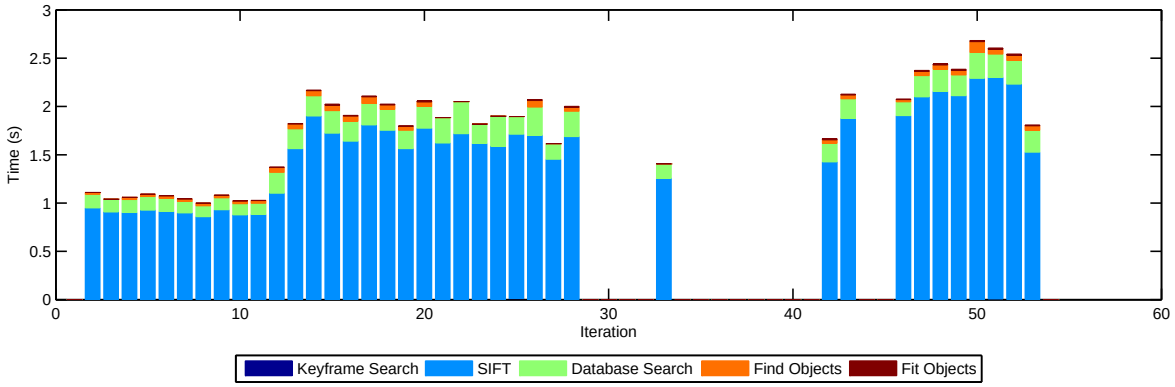


Figure 6.8: Timing graph of the book sequence. It can be clearly seen that SIFT and the database lookup dominate the processing time.

6.8.2 Simple wall

In this second experiment, the same simple wall sequence is as used as in experiment 1 in §4.3.1, along with the same small database of five planar objects, containing 9 433 keypoints. Fig. 6.9 shows various camera views and 3D map views of the recovered scene. In (a) the detected objects are shown labelled as the user would see them, and in (b) the points that have been found on each object can can be seen. In (c) and (d) the 3D map views show that the scene has been faithfully recovered, with all objects in their respective planes. Fig. 6.10 shows the timing graph for the object recognition process for this sequence.

As there is no calibration tile to act as a reference point, Table 6.2 instead compares the objects to each other. The upper triangle holds the expected angles, and the lower triangle the

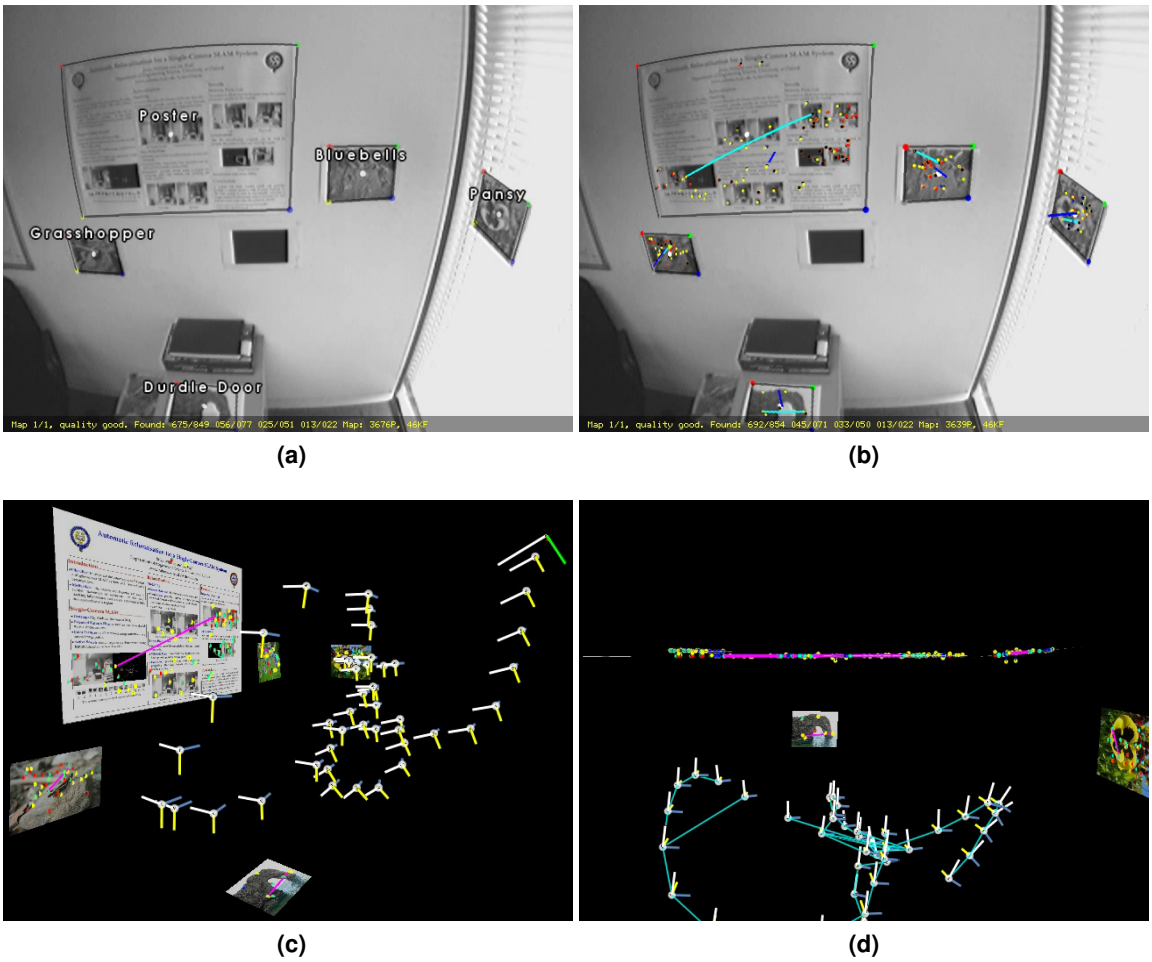


Figure 6.9: (a) The user's view with the detected posters labelled. (b) The same view, but with the localized keypoints shown, along with the object normals and a line between the two best fitting points. (c) A perspective view of the final map, and (d) an overhead view, showing the coplanarity of the three posters on the same wall (see video exp2_simple_wall.avi).

		Expected Angle (°)				
	Posters	Bluebells	Grasshopper	Durdle Door	Pansy	Poster
Angle (°)	Bluebells	—	0	90	90	0
	Grasshopper	8.8	—	90	90	0
	Durdle Door	85.5	91.7	—	90	90
	Pansy	95.5	88.3	98.5	—	90
	Poster	6.1	2.8	89.4	92.3	—

Table 6.2: The angles between the various posters.

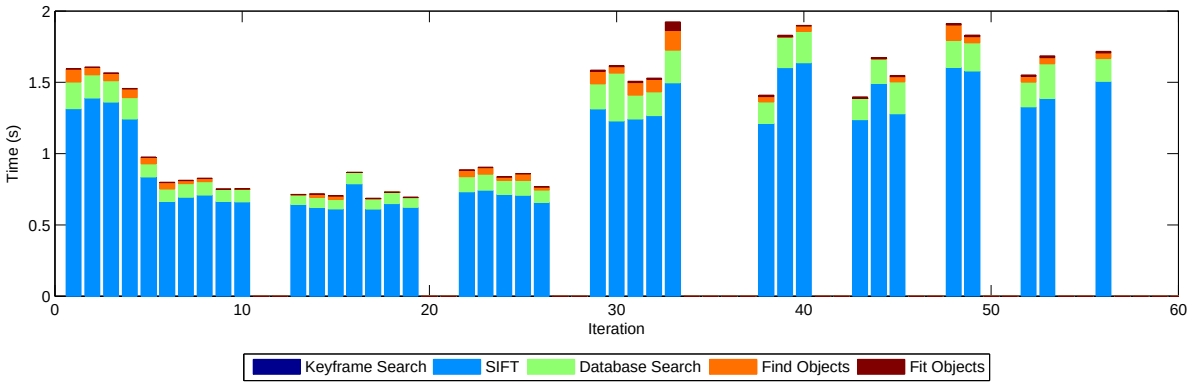


Figure 6.10: Timing graph of the simple wall sequence. It can be seen that the thread was able to finish processing all of the keyframes at numerous points during the sequence.

measured angles. The objects are located less than 3° out of plane with respect to reach other, with two notable exceptions. The Bluebells poster exhibits angles that are two to three times larger than the other posters, and the angle between the Pansy and Durdle Door posters is three times larger than the angle between these posters and the others. However, all of the posters are well located with respect to the local map points found by the underlying tracking system. There are two possible causes for this. The first is calibration, specifically the radial distortion, of the camera, and the second is too few measurements of the posters. If the camera calibration is not perfect then measured map points will be optimized incorrectly in the bundle adjustment, leading to the map points and the keyframes being located incorrectly. The recovered environment will end up with various curvatures. Table 6.3 addresses the second possibility of too few measurements to localize an object. It shows the number of keyframes each object was found in, the number of unique database points that were found in these keyframes, and the number of these that were localized. Points that have been localized were seen in two or more keyframes. It can be seen that the Durdle Door poster has the fewest measurements, making it the least certain to be placed correctly. The Bluebells poster however has a significant number of points, as many as the Pansy poster. This leads to the conclusion that the Durdle Door poster localization is poorer mainly due to the few measurements, and camera calibration is the main issue for the Bluebells poster, and possibly its neighbouring poster, the Pansy.

Poster	Keyframes found in	Database keypoints found	Database keypoints localized
Bluebells	7	51	25
Durdle Door	2	17	8
Grasshopper	7	61	44
Pansy	5	38	25
Poster	27	130	82

Table 6.3: The number of frames the objects were found in, along with the number of database keypoints that were found and localized.

6.8.3 Poster wall

In this third experiment the system handles a larger number of objects. The sequence used is the same as the one in §4.3.2. The same database of 16 objects with a total of 31 910 keypoints is used, and again all 13 observed objects are successfully recognized and localized. However, unlike the monoSLAM based system, where this was the maximum number of objects that the system could handle and maintain real time operation, here the system is unimpeded by the number of objects. Fig. 6.11(a) shows the final frame from the sequence, with the detected objects outlined and labelled. The keypoints that have been localized on each object are shown in (b), along with the each object's normal, and the pair of points that the object fitted to best.

Fig. 6.11(c) shows three camera frames demonstrating how one of these objects becomes better localized over time. As shown by the object's projected outline, the localization is poor when the object was first recognized when partially out of shot, but is improved as further measurements are made.

Fig. 6.12(a) shows a perspective graphic of the recovered map with the added objects, and (b) is a view from above showing that the individually located objects have a collective coplanarity. It should be noted that placing the objects together on the wall plane does not affect the individual localizations, but merely gives an opportunity to examine collective quality. Fitting a plane to the boundary points of the objects, and scaling the results to the known size of the scene shows that the standard deviation about the zero mean is around 2 cm.

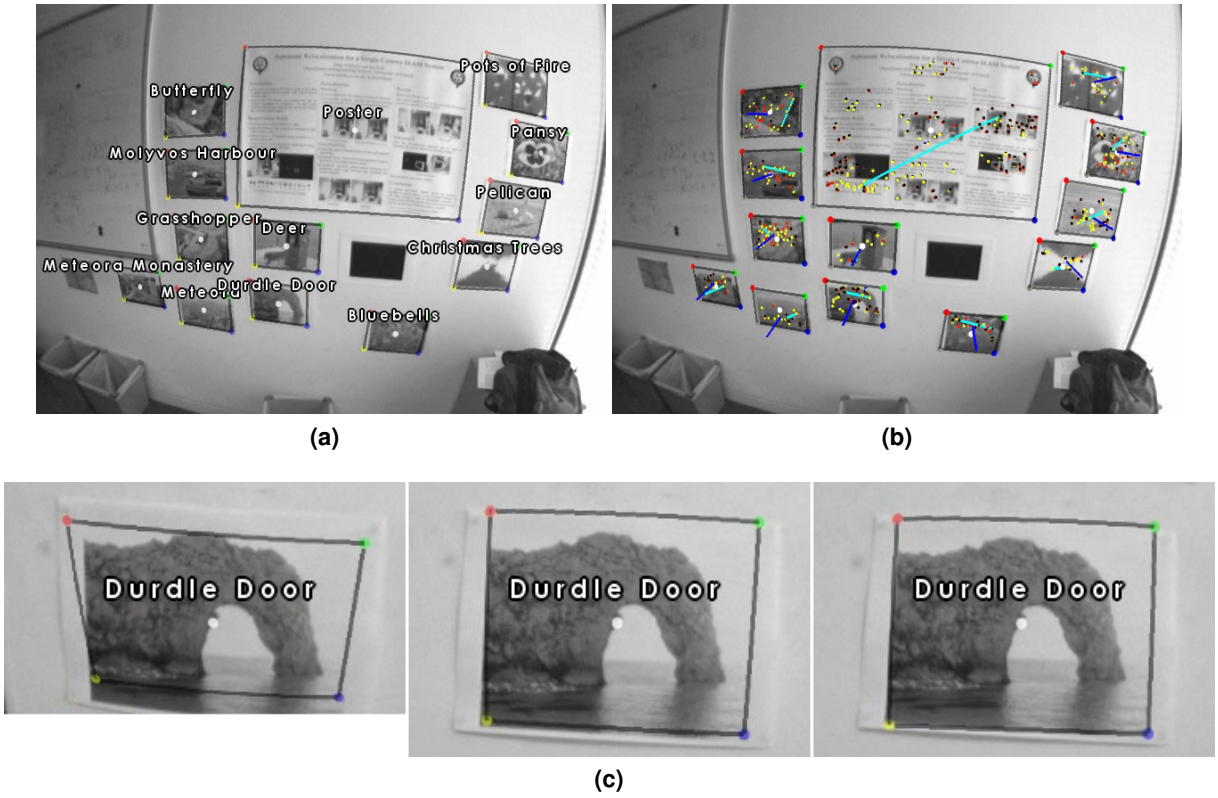
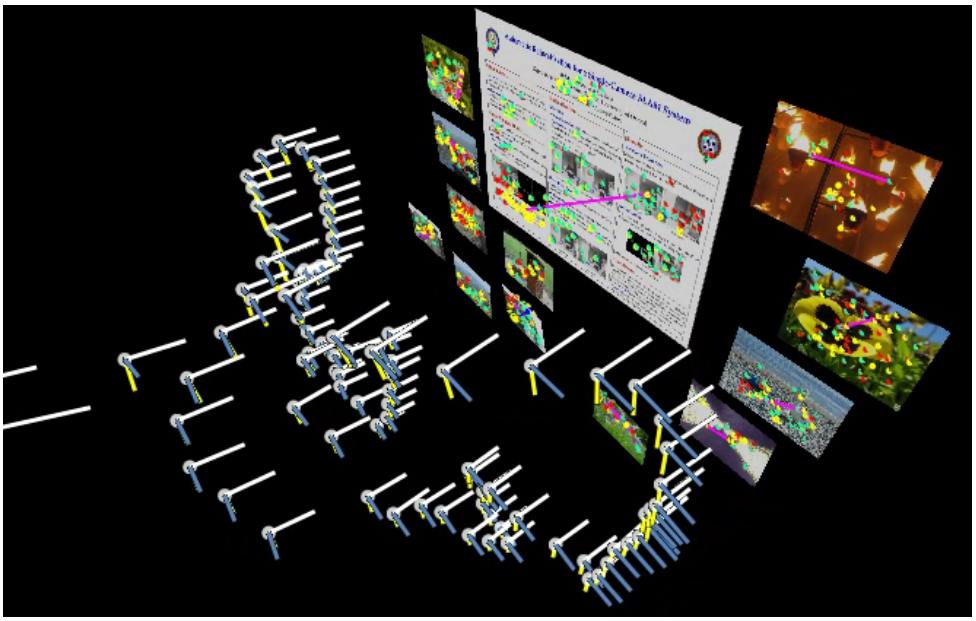
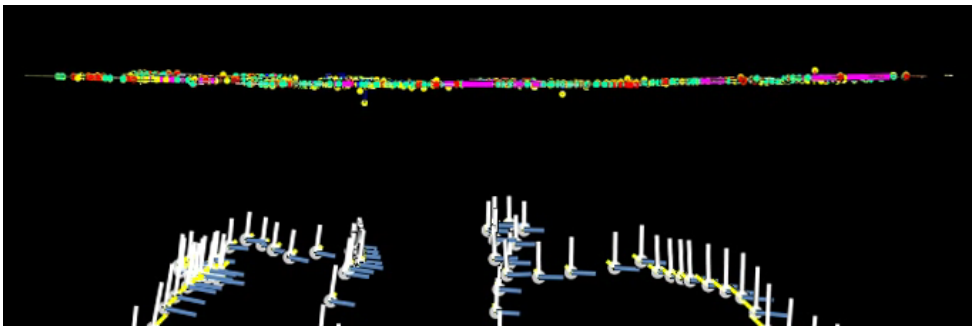


Figure 6.11: All 13 posters are successfully recognized and located within the plane of the wall. (a) A view showing the AR overlays of the object extents and names as the user sees the system. (b) A view showing the detected keypoints on the objects, along with the two best points (line between them) for fitting each object, and the object normals. (c) The improvement of localization as further keypoint matches are added to the triangulation of the object structure (see video exp3_posters.avi).

Fig. 6.10 shows the timing graph for this experiment. The database used is around 25 times larger than in the first experiment, however due to the BBF lookup the search time has only increased to 290 ms on average (45% longer). As more objects are located, the time taken to find and fit the objects increases as the number of points to triangulate and optimize increases. Per object, the time only increases as more keypoints are located.



(a)



(b)

Figure 6.12: 3D views of the recovered objects: (a) perspective and (b) overhead views. It can be clearly seen that the posters all share a collective coplanarity.

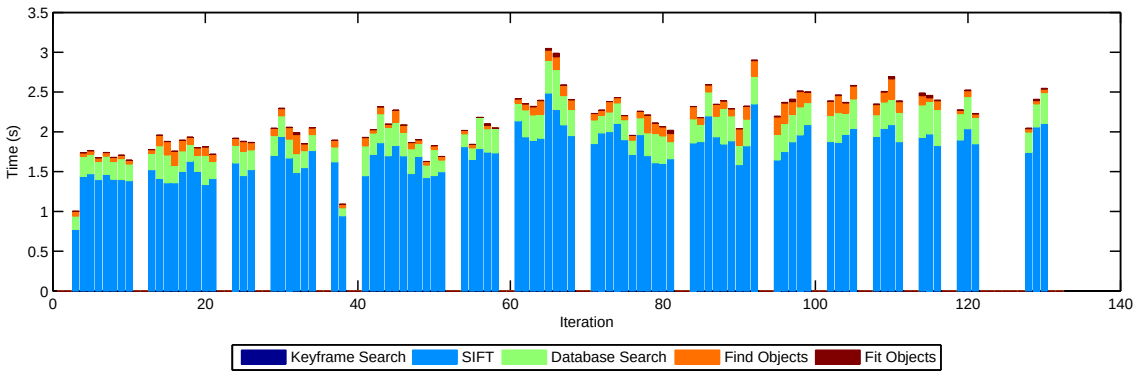


Figure 6.13: Timing graph of the posters sequence. It can be clearly seen that SIFT and the database lookup dominate the processing time.

Poster pair	Expected Angle (°)	Angle (°)
Pots of Fire and Grasshopper	90	84.9
Pots of Fire and Colosseum	90	86.9
Grasshopper and Colosseum	90	87.5

Table 6.4: The angles between the various posters.

6.8.4 Cluttered desk

In this fourth experiment the cluttered desk sequence from §4.3.3 is used, along with the same database of 16 objects. Fig. 6.14 shows various images from the sequence and the recovered map.

There are two important observations from this experiment. The first shows a disadvantage of using multiple views to reconstruct and locate the object, rather than the single view method of Chapter 3. Here at least two keyframes containing the object are required, along with at least three database points being observed in both frames to locate the object's plane. The book was detected in one frame with nine keypoints, but was not successfully observed in another, so it was not localized.

The second observation is that the well spaced keyframes from PTAMM, allow the Grasshopper poster to be localized well with its surrounding points. The depth issue present in the monoSLAM system is not present here. This can be clearly observed in Fig. 6.14(e) and (f) and compared with Fig. 4.12d. Table 6.4 shows the angles between each object.

6.9 Ashmolean gallery - real world scenario I

This experiment revisits the Ashmolean gallery. The gallery database has 37 paintings with some 75 000 features. PTAMM is able to track more successfully than monoSLAM, again a result of using more points. The removal of the need to measure the paintings also removes the uncertainty mentioned in §4.4.

PTAMM's multiple-map capability was used here, with separate maps made along each of

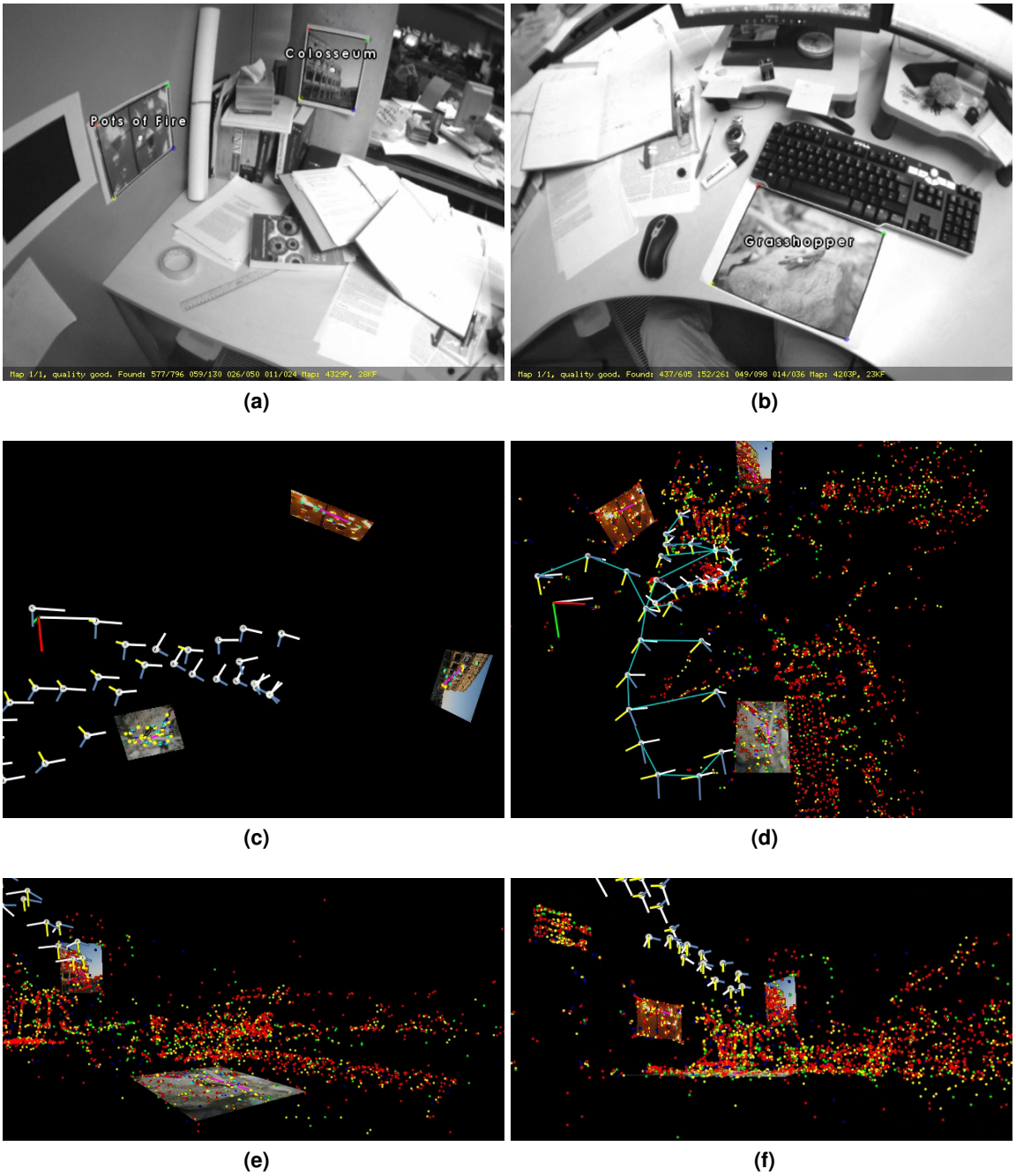
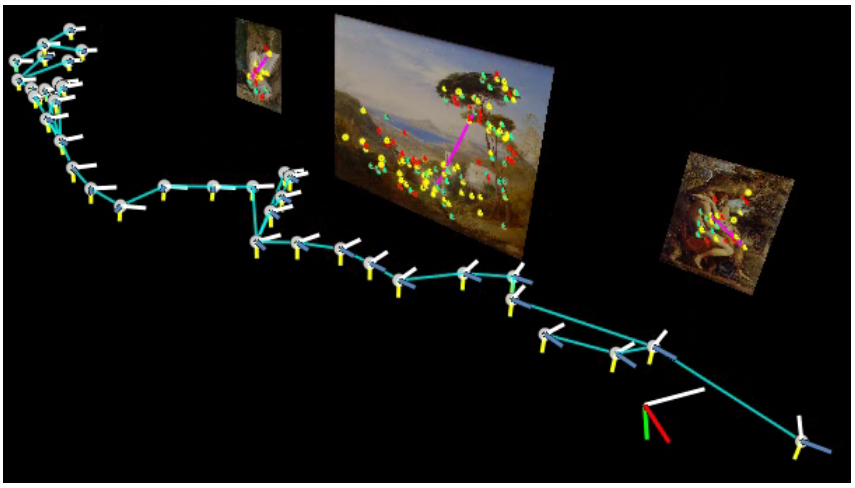


Figure 6.14: (a,b) Images from the cluttered desk sequence. (c) An overhead view of the map. (d) Another overhead view showing the bidirectional tree structure and the map points along with the detected objects. (e,f) Close-up views of the Grasshopper poster, showing that it is well localized with the surrounding map points (see video exp4_cluttered_desk.avi).

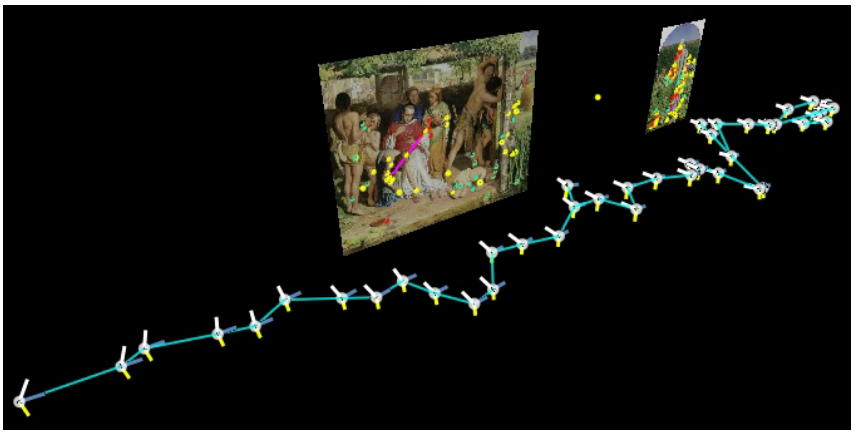
the three walls with paintings, and the relocalizer used to switch between them. 3D views of the three maps are shown in Fig. 6.15. In each of these, the detected paintings, keyframes, and the tree structure linking the keyframes is shown. Fig. 6.16(a) and (b) show an overhead view of the third wall's map in Fig. 6.15(c) shown without and with the map points respectively. It can be seen that the paintings are located within the centre of the point cloud. A slight curvature along the wall is also apparent. This is most likely due to the model of the radial distortion being slightly incorrect.

As the maps were created, the paintings were detected, localized and labelled for the user. Fig. 6.17 shows a selection of the recognized paintings with their AR labels detailing the painting's title and artist. However, of the 20 observed paintings, 9 were not fully detected and localized. There turned out to be several reasons: those not detected were either too small in the keyframes for SIFT to match; or had too few distinctive features to be recognized; or were observed in too few keyframes to be localized; or had too few matching keypoints across keyframes to triangulate the keypoints; or had too few points localized to estimate a plane. The majority of the failures to detect paintings were due to the failure in finding matches between the database keypoints and the keyframe keypoints.

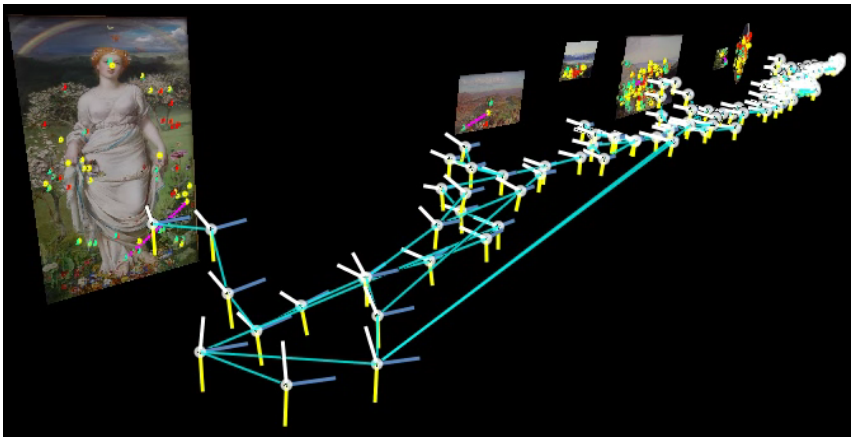
Once the maps had been constructed and the paintings detected, PTAMM was placed in a read only mode, allowing the user to explore the gallery, but not to add any further keyframes to the system. The user was free to explore the gallery without accidentally corrupting the maps. When the user left one mapped area the system became lost, and attempted to relocalize within one of the maps using the map switching mechanism of PTAMM. Although the system was able to recover into all three maps, the search was protracted. The reason for this is that the repeating pattern of the gallery wallpaper caused considerable visual aliasing.



(a) Map 1



(b) Map 2



(c) Map 3

Figure 6.15: Three maps are made around a gallery, and 11 paintings are located within these maps. The bidirectional tree for the keyframes can also be seen (see video exp5_gallery.avi).

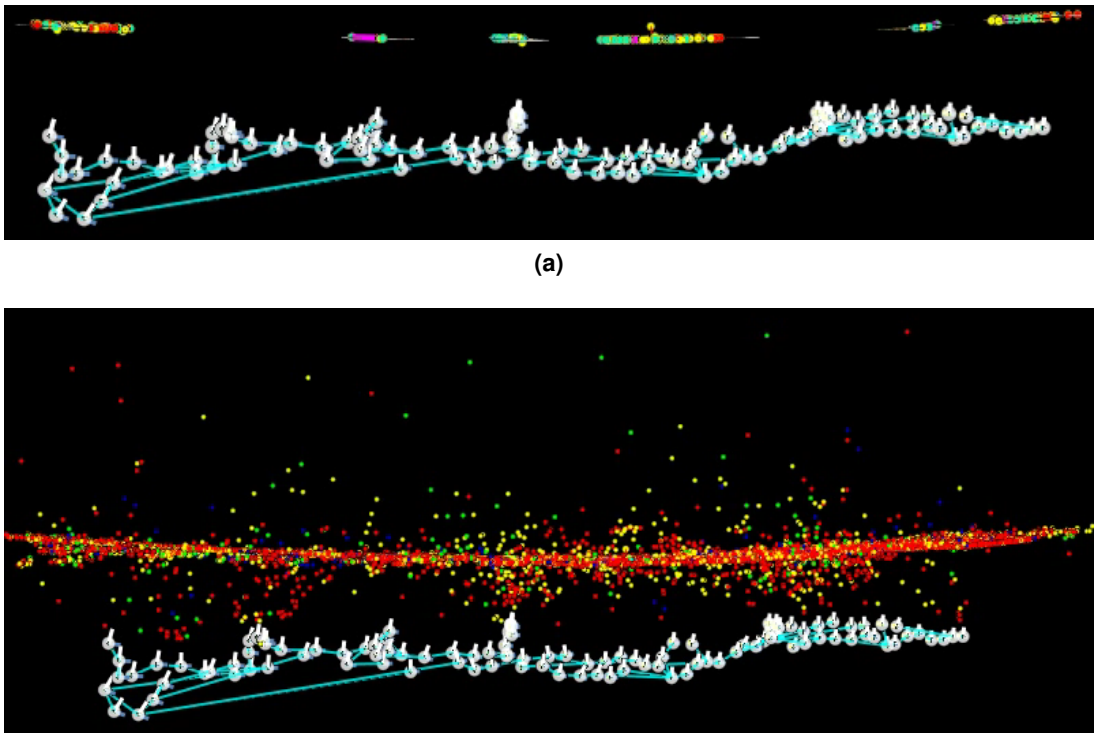


Figure 6.16: In (a) and (b) an overhead view of map 3 is shown without and with map points shown. The objects are clearly located within the the centre of this point cloud.

6.10 Street scene - real world scenario II

Recall that taking the system outdoors to recognize shop fronts was too much for monoSLAM §4.5. The PTAMM method developed here fairs better, but not perfectly. The same database of eight shop fronts with a total of 11 359 keypoints was used. The improved tracking and mapping allowed a map of the street to be built, unaffected by the moving vehicles and pedestrians. However, the object matching still presented a significant challenge. The same three shops are detected and localized, but the number of localized keypoints is very small, as shown in Table 6.5. Fig. 6.18(a-c) shows the detected shop fronts outlined and labelled. The full map in (d) can be seen to be densely featured and reflect the shape of the street, and (e) and (f) show the shop fronts within the point cloud, and from above, respectively.

Figure 6.17: The detected paintings are automatically, labelled with the title, and artist information (see video exp5_gallery.avi).

Poster	Keyframes found in	Database keypoints found	Database keypoints localized
All about hair	4	26	17
Oddbins	3	21	9
Duke of Cambridge	2	12	6

Table 6.5: The number of frames the objects were found in, along with the number of database keypoints that were found and localized.

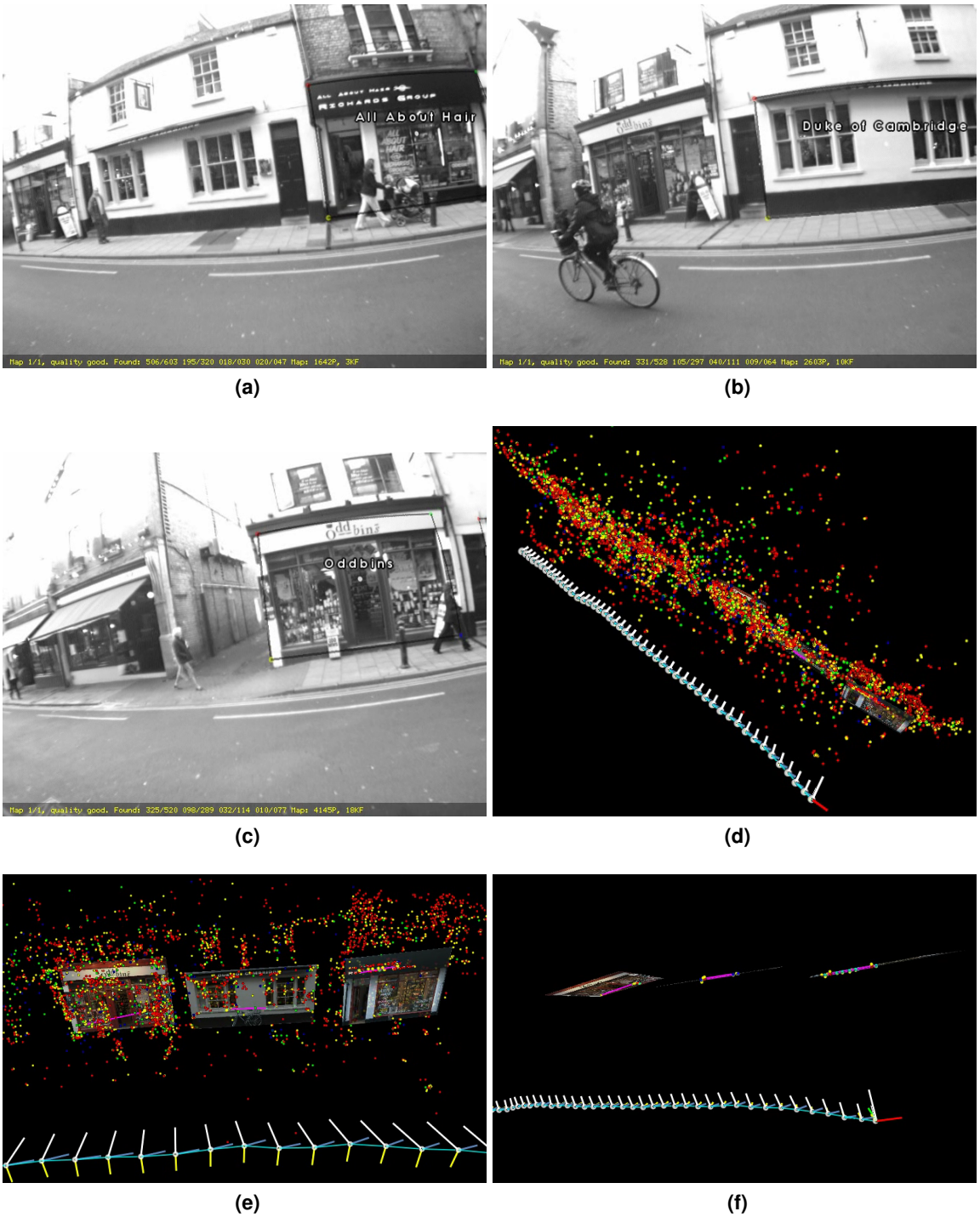


Figure 6.18: (a,b,c) Various images from the street sequence showing the three labelled shops. (d,e,f) Various map views showing the localization of the shops within the recovered map (see video exp6_street.avi).

Figure 6.19: In this AR oscilloscope tutorial, AR elements are placed with respect to locations within the object's extent. As the user advances through the tutorial different overlays are displayed (see video exp7_oscilloscope.avi).

6.11 Oscilloscope tutorial - real world scenario III

This final experiment recreates the oscilloscope AR tutorial. The same tutorial is carried out as before, and the results can be seen in Fig. 6.19. In the frames from the sequence, AR labels direct the user, and a circle is placed over the button or dial of interest. The new method presented in this chapter is able to perform this experiment almost equally well. However, the requirement for multiple keypoints to be seen in multiple keyframes causes the initial detection of the oscilloscope to take longer. The superior tracking of the PTAM based method becomes apparent as the camera moves to extreme views around the oscilloscope, as the system is able to track smoothly throughout. The monoSLAM system, by comparison, became lost at this point as shown in §4.6.

6.12 Limitations and improvements

The move from monoSLAM to PTAMM described in Chapter 5, and the different approach to managing objects proposed in this chapter have solved all of the major shortcomings that were evident in Chapters 3 and 4.

The removal of the need to measure the objects makes the implementation of a new database as simple as capturing an image of the desired object and naming it, allowing the system to work in a wider variety of environments. Though at the cost of increased processing time as two or more images are required for reconstruction. The increased flexibility of the method shown makes this trade off worthwhile. The use of keyframes allows for an improved search of the mapped environment than the previous approach.

SIFT. The main remaining problem is the significant time taken to run SIFT. A substantial reduction in this would allow image doubling to be used and faster keyframe processing for faster user feedback. A GPU based implementation of SIFT or a similar feature detector and descriptor would enable this. Currently a keyframe is processed once using SIFT without doubling. However, looking at the timing graphs it is apparent that the system does reach points where it has processed all keyframes and the user is no longer exploring. During these phases the keyframes could be reprocessed using image doubling. This benefit may be redundant depending on how well a GPU implementation works.

Measurements. In several of the experiments, objects observed in few keyframes were recognized but not localized. For objects that are only seen with a glancing side view there is not much that can be done. However, there are several changes that could be made for objects that pass in front of the camera. The failures occur for two reasons: (i) the keyframes are too separated causing the object to only be seen in one frame; (ii) keypoints on the object are detected, but these fail the initial RANSAC homography test as too few points are found.

The keyframe separation can be reduced by adjusting the new keyframe insertion parameters (§5.2.2), but this would increase the processing requirements. One way around this is to focus the PTAMM bundle adjustments on small local cells of keyframes. This technique, known as a sliding window, requires the keyframes not to be located with reference to a world origin but to each other in a relative representation. Work is in progress to implement this fully in the PTAMM framework [58], and has potential to allow much larger maps to be handled.

The rejection of good keypoint matches could be avoided by removing the need to find a homography. Instead, accepting *all* matches, and using epipolar geometry tests and bundle adjustment to weed out the outliers. This may result in more keypoint matches, and therefore object localizations. However, if many mismatches are found for the initial triangulation of keypoints, then the initial plane estimation may be wrong.

Mobile objects. The aim of the work presented in this chapter and in previous chapters has been to locate stationary objects in predominantly stationary environments. If an object moves, its outline will remain where it was located. In the monoSLAM based system this was also true, but once an object was not observed enough it was deleted. Here, the fixed nature of the keyframes stops this occurring. To detect objects that have moved one might replace the old keyframes with new ones when the camera revisits previously mapped areas. This would also aid the tracking system as points that have moved would be updated with the new keyframes. The second method might be to use a frame-rate tracker and not add objects as augmentations to the map.

3D objects. In this chapter planar objects have been considered, but the reconstruction method demonstrated could be extended to handle planar objects with distortions (*e.g.* a poster on a curved surface) and 3D objects. The current planar constraints (homography fitting and plane fitting) are only used because the objects are known to be planar. Instead a model could be inferred from the located keypoints, allowing deformations to be represented correctly. Extend-

ing this a step further would be to allow 3D models to be inferred from the detected keypoints, and for a more dense representation, from the map points. A 3D model could be learnt on-line as a user explores an environment.

6.13 Conclusion

In this chapter it has been shown how objects can be recognized, reconstructed and localized from keyframes captured as part of a parallel tracking and mapping process. The developed method is able to automatically recognize objects, and annotate a user's display with AR related to the objects, while building a map of the environment. As a minimum, to recognize an object only an image of it is required, but further information can be provided to enable a rich AR experience. By using the well spaced keyframes the whole of the mapped environment is searched for known objects. The method was demonstrated working successfully in laboratory environments and also in larger real world scenarios, as a gallery guide and as an interactive AR tutorial system.

7

Conclusions and future work

This concluding chapter summarizes the contents of the thesis and its contributions to the field of wearable visual computing. Suggestions for future work are made, both incremental and evolutionary.

7.1 Summary of contributions

The aim throughout this thesis has been to develop portable hardware and computational techniques that enable users of a wearable visual system to explore the environment freely and to be presented with information about their surroundings.

After an introduction which provided motivation for the work, Chapter 2 presented the design and build of a wearable system equipped with dual cameras, which functioned independently. The first was a shoulder mounted camera with two motorized axes for control either by a remote operator or by some autonomous tracking process. The second camera was attached to the rear of a hand held and touch-sensitive display tablet that provided output and input to and from the wearer. This configuration was used as a magic lens, allowing the user to see through to the environment behind the display, but with graphical augmentation overlaid. The aim was to use, where possible, readily available off-the-shelf hardware. The wearable system was used for the work presented in Chapters 3 to 6.

In Chapter 3, a method was presented where objects of interest to the user were automatically recognized and localized in the surroundings while a map of the surroundings was being constructed. The user was then presented with augmented reality constructs related to the recognized objects. This was achieved by computing SIFT features on planar objects, using the object recognition method of Lowe, and localizing the objects relative to the camera using a single frame decomposition of plane-to-plane homographies. At the same time, monocular SLAM recovered the structure of the environment and the camera's location in it, allowing in turn the objects to be located in this same coordinate frame, all at video rate. The objects' locations were used as further measurements in the SLAM process.

In Chapter 4, the approach was demonstrated first providing AR in laboratory conditions, before being used in the much more challenging environment of Oxford's Ashmolean Museum of Art and Archaeology, where objects were used to establish the scale in a previously uncalibrated map. An AR based tutorial showing how to use an oscilloscope was developed, demonstrating that effective AR could be created with rather little investment of time and, importantly, without the need to create CAD models.

It was concluded that while monoSLAM provided an effective geometrical foundation for desk top sized work such as the oscilloscope, it was less successful in larger scale environments: the quadratic complexity in the number of map points is too restrictive.

The purpose of the work described in Chapter 5 was then threefold. The first goal was to extend the range of each single map, and the second was to further liberate the wearer by introducing multiple maps. The third goal was to avoid the use of a "strong" motion model, which in Chapter 4 was found to inhibit the motion of the wearer. The first and third were achieved by adopting the recently reported method of parallel tracking and mapping which separates the urgent task of camera tracking from the more measured task of refining the map. The separation and parallel operation of these tasks and the use of keyframes was shown to be an ideally suited to (i) to building multiple maps which are automatically switched as the

camera moves in and out of them, and (ii) to allowing several cameras to build and work in each individual map. The method was again demonstrated in laboratory conditions and in an extended trial where multiple maps were built in the the Oxford University Museum of Natural History, and AR displayed to the wearer. The contribution here was to demonstrate the utility to the wearer of "weakly linked" multiple maps: ones between which there is no computed Euclidean transformation.

The last substantive chapter, Chapter 6, reintroduced recognized and located objects, but now into the framework of multiple maps and keyframes developed in Chapter 5. Significant changes were made to the approach in Chapter 3. Because of the greater feature numbers in PTAMM maps, objects were used to augment the map, but not to provide extra measurements into it. Another, highly significant modification, was that the objects were reconstructed using triangulation of observations made of them from PTAMM's keyframes. This allowed objects to be modelled without scale. The significance of the reconstruction is that although planar objects were used in Chapter 6 as in Chapter 3, the reconstruction is point-based and quite general, opening the way to the use of fully 3D objects.

7.2 Future work

There are many aspects where there is scope for improvement and significant addition of capability. These have been detailed in the individual chapters, but a few of the major ones are summarized below. They are split in terms of immediate incremental enhancements to the current system, and more radical changes.

7.2.1 Incremental enhancements

The immediate enhancements can be split into four categories: those for the base PTAM system; those for the PTAMM system; those for the object reconstruction implementation of PTAMM; and those for the wearable.

PTAM. Currently many poor features get initialized, but never deleted. This leads to many spurious features located throughout a map. Various measures need to be implemented to phase out these poor measurements. Along with improved feature culling is the need for keyframe culling. Over time features get deleted if they are unable to be remeasured by the camera. This can lead to keyframes that have very few or no features. In this case the bad keyframes needs to be removed and replaced with a new ones.

Features are deleted when the camera has been unable to measure them enough times, however this failure may be due to temporary occlusions. Occlusion reasoning needs to be improved beyond the simple check of "is the feature facing the camera?". This can be done by implementing surface estimation, which can be used to help estimate whether a feature may be behind a surface or not. Adding surfaces would also greatly improve the AR, as AR objects could then interact with the surfaces and be properly occluded.

A final change to the maps is to move to a completely relative framework, where each keyframe is relative to another and not a global coordinate frame. In this way the map can be locally adjusted with updates to a keyframe's position being propagated to its children. The advantage of this is that larger maps can be handled as the local adjustments lessen the need to perform a global adjustment. Work on this has already begun with Holmes *et al.* [58].

For the end-user, there are two significant barriers to usage. The first is the need to calibrate the camera prior to use. Though a one-off exercise, it is a reasonably complex procedure. Making the calibration automatic by adding the camera unknowns to the bundle adjustment would remove this barrier, but at the cost of increasing the complexity of the bundle adjustment. The second improvement is the map initialization that requires the user to start the initialization, move the camera horizontally enough, and then end the initialization. Instead, making the process a one click start, and allowing the system to automatically monitor the process. It would then determine when the camera has moved correctly, and enough to initialize. Visual guidance could be provided to the user to help them move the camera correctly.

PTAMM. For the PTAMM system, there are two significant improvements. The first being a method to detect overlapping maps and join them either automatically or with the approval of a user. This would benefit well from a relative framework. The second is to allow each tracker to have its own map maker, and work in their own maps when required. With a quad core machine this would be very feasible for a two camera system. Rather than an absolute one-to-one of map makers to trackers, instead a method where a map maker is created when a tracker switches to a map with no trackers, and therefore no map makers, associated would be better. When a tracker switches away, the map maker continues to optimize the map, and only once complete, dies. This way all maps are kept optimal, and the trackers become fully independent.

PTAMM with object reconstruction. For the object reconstruction addition to PTAMM the first and most pressing improvement is to switch to a GPU enabled feature extractor, whether it is SIFT or an equally adept derivative such as SURF. The next improvement would be to remove the planarity constraints and implement 3D model reconstruction and recognition in a manner similar to that of Gordon and Lowe [49]. Currently all objects are added off-line, allowing objects to be added when the system is running using segmentation and iterative learning of the object's appearance would be a useful addition.

Wearable. Finally, the wearable requires two main improvements. First, a roll motor to enable full stabilization, and secondly autonomous control from the PTAMM system or its derivative.

7.2.2 Evolutionary enhancements

Two evolutionary changes are suggested. The first is to adapt the system to use stereo cameras. By using a calibrated stereo pair map points can be localized instantly. A system that is broadly similar to PTAM that uses a stereo pair has recently been demonstrated by Sibley *et al.* [133]. This system has been able to traverse large areas, building a single map with very little drift. Well constructed stereo maps, could then be used by single camera systems.

The second is to build a massively multi-user system. Currently the system is a single user experience, with one user exploring a set of maps on their own. Other users could be given copies of the system, and explore the same set of maps, but there would be no interaction. One user changing a map is not propagated to the other users. Using the PTAMM system as a basis, a multi-user system could be constructed that not only allows more portable hardware to be used, but also leverages large computation power to provide a richer AR experience. The most processor intensive part of PTAMM is the map maker, so removing this from the portable computer removes a significant processing requirement from the device. Instead, place the map maker on a powerful server and the tracker on a low powered portable device. Now as many map makers as required can be used to build as many maps as needed. All maps are stored on a networked map server, allowing many users to connect to the map server and use the maps. The expandable architecture of PTAM as shown in this thesis with PTAMM and PTAMM with object reconstruction, allows further computationally intensive processes to be added on the server side. These include, object recognition, surface estimation, map joining, and more robust map relocalization. There are of course many issues that need resolving and careful execution, with the most important being the network bandwidth. Current wireless technology does not allow 30 Hz transmission of 640×480 images, and would struggle to send the well spaced keyframes of PTAMM without a backlog occurring. This may lead to the requirement of super users running a full PTAMM system who can initialize a map and upload it for further processing and sharing. However, this need would diminish with increasing network speeds. Other important issues include, map versioning, data propagation, and when and how many maps to send a user.

Bibliography

[1] 2d3 Ltd. Boujou. http://www.2d3.com — Last accessed 01/07/2009.

[2] Active Vision Group. VW. http://www.robots.ox.ac.uk/ActiveVision — Last accessed 01/07/2009.

[3] American Art Clay Co. Inc. Friendly plastic. http://www.amaco.com — Last accessed 01/07/2009.

[4] Apple Inc. http://www.apple.com — Last accessed 01/07/2009.

[5] ARToolKit. http://www.hitl.washington.edu/artoolkit — Last accessed 01/07/2009.

[6] D. Ashbrook and T. Starner. Using GPS to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, 7(5):275–286, 2003.

[7] Atmel Corporation. http://www.atmel.com — Last accessed 01/07/2009.

[8] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. *Computer Vision and Image Understanding (CVIU)*, 110(3):346–359, 2008.

[9] P. Beardsley. *Active Vision*. PhD thesis, University of Oxford, 1992.

[10] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proc IEEE Conf on Computer Vision and Pattern Recognition*, pages 1000–1006, 1997.

[11] S. Benhimane and E. Malis. Homography-based 2D visual tracking and servoing. *Special Joint Issue on Robotics and Vision. Journal of Robotics Research*, 26(7):661–676, July 2007.

[12] M. Brown and D. G. Lowe. Invariant features from interest point groups. In *Proc 14th British Machine Vision Conference*, pages 656–665, 2002.

[13] M. Brown and D. G. Lowe. Unsupervised 3d object recognition and reconstruction in unordered datasets. In *Proc International Conference on 3-D Digital Imaging and Modeling*, pages 56–63, 2005.

[14] P. Bunnun and W. Mayol-Cuevas. OutlinAR: an assisted interactive model building system with reduced computational effort. In *Proc 7th IEEE/ACM Int Symp on Mixed and Augmented Reality*, September 2008.

[15] Carnegie Mellon University. The CMU Sphinx Group Open Source Speech Recognition Engines. http://cmusphinx.sourceforge.net — Last accessed 01/07/2009.

[16] R. O. Castle, G. Klein, and D. W. Murray. Video-rate localization in multiple maps for wearable augmented reality. In *Proc 12th IEEE Int Symp on Wearable Computing*, pages 15–22, 2008.

[17] D. Chekhlov, A. Gee, A. Calway, and W. Mayol-Cuevas. Ninja on a Plane: Automatic Discovery of Physical Planes for Augmented Reality Using Visual SLAM. In *Proc 6th IEEE/ACM Int Symp on Mixed and Augmented Reality*, November 2007.

[18] D. Chekhlov, M. Pupilli, W. Mayol-Cuevas, and A. Calway. Robust Real-Time Visual SLAM Using Scale Prediction and Exemplar Based Feature Description. In *Proc 25th IEEE Conf on Computer Vision and Pattern Recognition*, June 2007.

[19] N. Cornelis and L. Van Gool. Fast scale invariant feature detection and matching on programmable graphics hardware. In *Proc Computer Vision and Pattern Recognition Workshops*, pages 1–8, 2008.

[20] Handykey Corporation. `http://www.handykey.com` — Last accessed 01/05/2009.

[21] Creative Display Systems. `http://www.creativedis.com` — Last accessed 01/07/2009.

[22] 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, 2008.

[23] Cybermind. `http://www.cybermindnl.com` — Last accessed 01/07/2009.

[24] O. Naroditsky D. Nistér and J. R. Bergen. Visual odometry. In *Proc 22nd IEEE Conf on Computer Vision and Pattern Recognition*, pages 652–659, 2004.

[25] A. J. Davison. SceneLib. `http://www.doc.ic.ac.uk/~ajd/Scene` — Last accessed 01/07/2009.

[26] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc 9th Int Conf on Computer Vision*, pages 1403–1410, 2003.

[27] A. J. Davison and D. W. Murray. Mobile robot localisation using active vision. In *Proc 5th European Conf on Computer Vision*, pages 809–825, 1998.

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

[29] T. E. de Campos. *3D Visual Tracking of Articulated Objects and Hands*. PhD thesis, University of Oxford, 2006.

[30] T. E. de Campos, W. W. Mayol-Cuevas, and D. W. Murray. Directing the attention of a wearable camera by pointing gestures. In *Proc Brazilian Symposium on Computer Graphics and Image Processing*, 8–11 October 2006.

[31] T. W. Drummond. CVD (Cambridge Video Dynamics). `http://mi.eng.cam.ac.uk/~twd20/libcvdhtml` — Last accessed 01/07/2009.

[32] T. W. Drummond. TooN (Tom's object oriented numerics library). `http://mi.eng.cam.ac.uk/~twd20/TooN/html` — Last accessed 01/07/2009.

[33] T. W. Drummond and R. Cipolla. Real-time tracking of complex structures with on-line camera calibration. In *Proc 10th British Machine Vision Conf*, pages 574–583, 1999.

[34] T. W. Drummond and R. Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):932–946, July 2002.

[35] A. T. Duchowski. *Eye Tracking Methodology: Theory and Practice*. Springer, 2007.

[36] E. Eade and T. W. Drummond. Scalable Monocular SLAM. *Proc 24th IEEE Conf on Computer Vision and Pattern Recognition*, 1:469–476, 2006.

[37] eMagin. `http://www.emagin.com` — Last accessed 01/07/2009.

[38] C. Engels, H. Stewénius, and D. Nistér. Bundle adjustment rules. In *Photogrammetric Computer Vision*, 2006.

[39] ETT. `http://www.ett.co.th` — Last accessed 01/07/2009.

[40] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proc 21st IEEE Conf on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, June 2003.

[41] M. Fiala. ARTag, a fiducial marker system using digital techniques. In *Proc 23rd IEEE Conf on Computer Vision and Pattern Recognition*, volume 2, pages 590–596, 2005.

[42] 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.

[43] A. W. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. In *Proc 5th European Conf on Computer Vision*, volume 1, pages 311–326, 1998.

[44] E. Foxlin. Generalized architecture for simultaneous localization, auto-calibration and map-building. In *Proc IEEE/RSJ Conf on Intelligent Robots and Systems*, pages 527–533, 2002.

[45] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[46] D. B. Gennery. Visual tracking of known three-dimensional objects. *International Journal of Computer Vision*, 7(3):243–270, April 1992.

[47] P. Georgel, P. Schroeder, S. Benhimane, S. Hinterstoisser, M. Appel, and N. Navab. An industrial augmented reality solution for discrepancy check. In *Proc 6th IEEE/ACM Int Symp on Mixed and Augmented Reality*, 2007.

[48] Giga-Byte Technology Company Ltd. `http://www.giga-byte.com` — Last accessed 01/07/2009.

[49] I. Gordon and D. G. Lowe. What and where: 3D object recognition with accurate pose. In *Toward Category-Level Object Recognition*, pages 67–82, 2006.

[50] R. Hariharan and K. Toyama. Project Lachesis: Parsing and modeling location histories. In *Proc 3rd Int Conf on Geographic Information Science*, pages 106–124, 2004.

[51] C. Harris and C. Stennett. RAPID: A video rate object tracker. In *Proc 1st British Machine Vision Conf*, pages 73–77, 1990.

[52] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc 4th Alvey Vision Conf*, pages 147–151, 1988.

[53] C. G. Harris. Camera calibration. Technical report, Roke Manor Research, Siemens, UK, 1992.

[54] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[55] A. van den Hengel, A. R. Dick, T. Thormählen, B. Ward, and P. H. S. Torr. Videotrace: rapid interactive scene modelling from video. *ACM Transactions on Graphics (SIGGRAPH special issue)*, 26(3):86, 2007.

[56] Hitec RCD. `http://www.hitecrcd.com` — Last accessed 01/07/2009.

[57] S. Hodges, L. Williams, E. Berry, S. Izadi, J. Srinivasan, A. Butler, G. Smyth, N. Kapur, and K. R. Wood. Sensecam: A retrospective memory aid. In *Ubicomp 2006*, pages 177–193, 2006.

[58] S. Holmes, G. Sibley, G. Klein, and D. W. Murray. A relative frame representation for fixed-time bundle adjustment in SFM. In *Proc Int Conf on Robotics and Automation*, pages 2264–2269, 2009.

[59] P. J. Huber. *Robust Statistics*. Wiley, 1981.

[60] i-o Display Systems. `http://www.i-glassesstore.com` — Last accessed 01/07/2009.

[61] IDS: Image Development Systems. uEye USB Cameras. `http://www.ids-imaging.com` — Last accessed 01/07/2009.

[62] H. Kato and M. Billinghurst. Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System. In *Proc 2nd IEEE and ACM International Workshop on Augmented Reality*, pages 85–94, 1999.

[63] G. Klein. GKTools. `http://www.robots.ox.ac.uk/~gk` — Last accessed 01/07/2009.

[64] G. Klein and D. W. Murray. Full-3D Edge Tracking with a Particle Filter. In *Proc 15th British Machine Vision Conference*, volume 3, pages 1119–1128, 2006.

[65] G. Klein and D. W. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc 6th IEEE/ACM Int Symp on Mixed and Augmented Reality*, 2007.

[66] G. Klein and D. W. Murray. Improving the agility of keyframe-based SLAM. In *Proc 10th European Conf on Computer Vision*, pages II: 802–815, 2008.

[67] J. G. H. Knight, A. J. Davison, and I. D. Reid. Towards constant time SLAM using postponement. In *Proc IEEE/RSJ Conf on Intelligent Robots and Systems*, pages I:406–412, 2001.

[68] M. Kölsch and M. Turk. Robust hand detection. In *Proc IEEE Intl. Conference on Automatic Face and Gesture Recognition*, pages 614–619, 2004.

[69] M. Kourogi, T. Kurata, and K. Sakaue. A panorama-based method of personal positioning and orientation and its real-time applications for wearable computers. In *Proc 5th IEEE Int Symp on Wearable Computing*, pages 107–114, 2001.

[70] T. Kurata, N. Sakata, M. Kourogi, H. Kuzuoku, and M. Billinghurst. Remote collaboration using a shoulder-worn active camera/laser. In *Proc 8th IEEE Int Symp on Wearable Computing*, pages 62–69, 2004.

[71] M. Lamming and M. Flynn. Forget-me-not: intimate computing support of human memory. In *Proc. of FRIEND21 Int. Symp. on Next Generation Human Interface*, 1993.

[72] T. Lee and T. Höllerer. Hybrid feature tracking and user interaction for markerless augmented reality. In *Proc 10th Int Conf on Virtual Reality*, pages 145–152, 2008.

[73] Lenovo. `http://www.lenovo.com` — Last accessed 01/07/2009.

[74] J. J. Leonard and H. F. Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer Academic, Boston MA, 1992.

[75] J. J. Leonard, H. F. Durrant-Whyte, and I. J. Cox. Dynamic map building for an autonomous mobile robot. *International Journal of Robotics Research*, 11(8):286–298, 1992.

[76] J. J. Leonard and P. M. Newman. Consistent, convergent and constant-time SLAM. In *Proc Int Joint Conference on Artificial Intelligence*, pages 1143–1150. Morgan Kaufmann, 2003.

[77] J. J. Leonard and R. Rikoski. Incorporation of delayed decision making into stochastic mapping. In *Experimental Robotics VII (Lecture Notes in Control and Information Sciences, Vol 271)*, pages 533–542, 2001.

[78] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, 2006.

[79] L. Liao, D. Fox, and H. Kautz. Extracting Places and Activities from GPS Traces Using Hierarchical Conditional Random Fields. *International Journal of Robotics Research*, 26(1):119–134, 2007.

[80] Lilliput Technology. `http://www.lilliput.cn` — Last accessed 01/07/2009.

[81] T. Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):77–116, 1998.

[82] Liteye. `http://www.liteye.com` — Last accessed 01/07/2009.

[83] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[84] D. G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):441–450, May 1991.

[85] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[86] Lynxmotion Inc. `http://www.lynxmotion.com` — Last accessed 01/07/2009.

[87] J. MacCormick and M. Isard. Partitioned sampling, articulated objects, and interface-quality hand tracking. In *Proc 6th European Conf on Computer Vision*, volume 2, pages 3–19, 2000.

[88] S. Mann. Smart clothing: The wearable computer and wearcam. *Personal Technologies*, 1(1):21–27, March 1997.

[89] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[90] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 207(1167):187–217, 1980.

[91] E. Matias, I. S. MacKenzie, and W. Buxton. A wearable computer for use in microgravity space and other non-desktop enviroments. In *Companion of the ACM CHI '96 Conference on Human Factors in Computing Systems*, pages 69–70, 1996.

[92] W. W. Mayol, A. J. Davison, B. J. Tordoff, N. D. Molton, and D. W. Murray. Interaction between hand and wearable camera in 2D and 3D environments. In *Proc. British Machine Vision Conference*, 2004.

[93] W. W. Mayol, A. J. Davison, B. J. Tordoff, and D. W. Murray. Applying Active Vision and SLAM to Wearables. In *Robotics Research: The Eleventh International Symposium*, volume 15 of *Springer Tracts in Advanced Robotics, volume 15*, pages 325–334, 2005.

[94] W. W. Mayol, B. J. Tordoff, and D. W. Murray. On the choice and placement of wearable vision sensors. *IEEE Trans on Systems, Man and Cybernetics*, 39(2):414–425, March 2009.

[95] W.W. Mayol and E. Rodriguez. Wearclam wearable keyboard. `http://www.robots.ox.ac.uk/~wmayol/WearClam` — Last accessed 01/07/2009.

[96] W. W. Mayol-Cuevas. *Wearable Visual Robots*. PhD thesis, Department of Engineering Science, University of Oxford, 2004.

[97] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.

[98] P. Mistry, P. Maes, and L. Chang. WUW - wear Ur world: a wearable gestural interface. In *Proc 27th International Conference on Human Factors in Computing Systems, Extended Abstracts*, pages 4111–4116, 2009.

[99] 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 *Proc Int Joint Conference on Artificial Intelligence*, 2003.

[100] J. M. M. Montiel, J. Civera, and A. J. Davison. Unified inverse depth parametrization for monocular SLAM. In *Proc Robotics: Science and Systems II, Philadelphia PA*, 2006.

[101] Motion Computing. `http://www.motioncomputing.com` — Last accessed 01/07/2009.

[102] E. Mouragnon, F. Dekeyser, P. Sayd, M. Lhuillier, and M. Dhôme. Real time localization and 3d reconstruction. In *Proc 24th IEEE Conf on Computer Vision and Pattern Recognition*, pages 363–370, 2006.

[103] Myvu. `http://www.myvu.com` — Last accessed 01/07/2009.

[104] D. Nistér. *Automatic dense reconstruction from uncalibrated video sequences*. PhD thesis, Royal Institute of Technology KTH, Stockholm, Sweden, 2001.

[105] 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.

[106] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23(1), 2006.

[107] Nuance. `http://www.nuance.com` — Last accessed 01/07/2009.

[108] NVIS. `http://www.nvisinc.com` — Last accessed 01/07/2009.

[109] A. Opelt, A. Pinz, and A. Zisserman. Learning an alphabet of shape and appearance for multi-class object detection. *International Journal of Computer Vision*, 80(1):16–44, 2008.

[110] M. Özuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99(1), 2009.

[111] M. Özuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Proc 25th IEEE Conf on Computer Vision and Pattern Recognition*, 2007.

[112] Palm Inc. `http://www.palm.com` — Last accessed 01/07/2009.

[113] A. Pentland. Looking at people: Sensing for ubiquitous and wearable computing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):107–119, 2000.

[114] W. Piekarski. *Interactive 3D Modelling in Outdoor Augmented Reality Worlds*. PhD thesis, University of South Australia, 2004.

[115] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. Visual modeling with a hand-held camera. *International Journal of Computer Vision*, 59(3):207–232, 2004.

[116] M. Pollefeys, R. Koch, and L. Van Gool. Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. *International Journal of Computer Vision*, 32(1):7–25, 1999.

[117] M. Pupilli and A. Calway. Real-time Camera Tracking Using Known 3D Models and a Particle Filter. In *Proc International Conference on Pattern Recognition*, August 2006.

[118] Qt Software. `http://www.qtsoftware.com` — Last accessed 01/07/2009.

[119] J. Quarles, S. Lampotang, I. Fischler, P. Fishwick, and B. Lok. Collocated AAR: Augmenting After Action Review with Mixed Reality. In *Proc 7th IEEE/ACM Int Symp on Mixed and Augmented Reality*, 2008.

[120] G. Reitmayr and T. W. Drummond. Going out: Robust tracking for outdoor augmented reality. In *Proc 5th IEEE/ACM Int Symp on Mixed and Augmented Reality*, pages 109–118, 2006.

[121] G. Reitmayr, E. Eade, and T. W. Drummond. Semi-automatic annotations in unknown environments. In *Proc 6th IEEE/ACM Int Symp on Mixed and Augmented Reality*, 2007.

[122] B. Rhodes and T. Starner. Rememberance agent: A continuously running automated information retreival system. In *Proc. of Pract. App. of Intelligent Agents and Multi-Agent Tech*, London, 1996.

[123] Rockwell Collins. `http://www.rockwellcollins.com` — Last accessed 01/07/2009.

[124] E. Rosten and T. W. Drummond. Fusing points and lines for high performance tracking. In *Proc 10th Int Conf on Computer Vision*, volume 2, pages 1508–1511, October 2005.

[125] E. Rosten and T. W. Drummond. Machine learning for high-speed corner detection. In *Proc 9th European Conf on Computer Vision*, volume 1, pages 430–443, 2006.

[126] SAAB. `http://www.saabgroup.com` — Last accessed 01/07/2009.

[127] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):824–840, 2009.

[128] G. Schall, E. Mendez, and D. Schmalstieg. Virtual redlining for civil engineering in real environments. *Mixed and Augmented Reality, IEEE / ACM International Symposium on*, 0:95–98, 2008.

[129] D. Schmalstieg and D. Wagner. Experiences with handheld augmented reality. In *Proc 6th IEEE/ACM Int Symp on Mixed and Augmented Reality*, pages 1–13, 2007.

[130] Sensics. `http://www.sensics.com` — Last accessed 01/07/2009.

[131] SGI. OpenGL. `http://www.opengl.org` — Last accessed 01/07/2009.

[132] J. Shi and C. Tomasi. Good features to track. In *Proc IEEE Conf on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[133] G. Sibley, C. Mei, I. Reid, and P. Newman. Adaptive Relative Bundle Adjustment. In *Proc Robotics: Science and Systems V*, 2009.

[134] S. N. Sinha, J. Frahm, M. Pollefeys, and Y. Genc. GPU-Based Video Feature Tracking and Matching. In *Workshop on Edge Computing Using New Commodity Architectures*, 2006.

[135] A. Smailagic, D. Siewiorek, R. Martin, and D. Reilly. CMU wearable computers for real-time speech translation. In *Proc 3rd IEEE Int Symp on Wearable Computing*, 1999.

[136] P. Smith, I. Reid, and A. J. Davison. Real-time monocular SLAM with straight lines. In *Proc 15th British Machine Vision Conference*, 2006.

[137] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous Robot vehicles*, pages 167–193. 1990.

[138] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5(4):56–68, 1986.

[139] S. M. Smith and J. M. Brady. SUSAN—a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, 1997.

[140] Sony Computer Entertainment. Eye of Judgment. `http://www.us.playstation.com/EyeofJudgment` — Last accessed 01/07/2009.

[141] Sony Corporation. `http://www.sony.com` — Last accessed 01/07/2009.

[142] H. Stewénius, C. Engels, and D. Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284–294, 2006.

[143] R. Swaminathan and S.K. Nayar. Non-metric calibration of wide-angle lenses and polycameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1172–1178, 2000.

[144] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, September 2005.

[145] 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(7-8):693–716, 2004.

[146] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. FastSLAM: An efficient solution to the simultaneous localization and mapping problem with unknown data association. *Journal of Machine Learning Research*, 2004.

[147] B. J. Tordoff. *Active Control of Zoom for Computer Vision*. PhD thesis, Department of Engineering Science, University of Oxford, 2002.

[148] P. H. S. Torr and D. W. Murray. Stochastic motion clustering. In *Proc 3rd European Conf on Computer Vision*, pages B:328–337, 1994.

[149] Toshiba. `http://www.toshiba.com` — Last accessed 01/07/2009.

[150] Trivisio. `http://www.trivisio.com` — Last accessed 01/07/2009.

[151] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, first edition, 1977.

[152] Vuzix. `http://www.vuzix.com` — Last accessed 01/07/2009.

[153] M. Walter, R. Eustice, and J. Leonard. A provably consistent method for imposing exact sparsity in feature-based SLAM information filters. In *Proc 12th International Symposium of Robotics Research*, 2005.

[154] S. Wangsiripitak and D. W. Murray. Avoiding moving outliers in visual SLAM by tracking moving objects. In *Proc Int Conf on Robotics and Automation*, pages 375–380, 2009.

[155] B. Williams, G. Klein, and I. D. Reid. Real-time SLAM relocalisation. In *Proc 11th Int Conf on Computer Vision*, 2007.

[156] B. Williams, P. Smith, and I. D. Reid. Automatic relocalisation for a single-camera simultaneous localisation and mapping system. In *Proc Int Conf on Robotics and Automation*, pages 2784–2790, 2007.

[157] R. G. Willson. *Modeling and Calibration of Automated Zoom Lenses*. PhD thesis, Carnegie Mellon University, 1994.

[158] J. Wither, S. DiVerdi, and T. Höllerer. Evaluating display types for ar selection and annotation. In *Proc 6th IEEE/ACM Int Symp on Mixed and Augmented Reality*, 2007.

[159] Xsens Technologies B.V. `http://www.xsens.com` — Last accessed 01/07/2009.

[160] Z. Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and Vision Computing*, 15(1):59–76, January 1997.