An Infrastructure for Video-Augmented Environments

Daniel Parnham

PhD
Department of Electronics
University of York

December 2006

Abstract

The development of a Video-Augmented Environment requires an integrated set of programming and image processing elements. Each of these elements is presented here as both an independent subject and part of a coherent infrastructure. The primary aim of this research was to realise an infrastructure that would provide a basis for the rapid development of Video-Augmented Environment applications. The *OpenIllusionist* framework is introduced as an embodiment of this infrastructure and provides a platform for further research and stable application development.

This thesis presents agent-based design as an efficient technique for development. It then explains how a framework can support agents along with other necessary infrastructure elements. Fiducials are suggested as a means for both calibration and interaction. A new fiducial design is presented and shown to be more reliable than that used in a large proportion of augmented reality research. Automated and efficient calibration algorithms are introduced that are straightforward to incorporate into a framework. Shadow removal is discussed and then one particular technique is adapted for use in a finger detection algorithm providing an improvement over background subtraction methods. Finally a practical realisation of a Video-Augmented Environment is presented that was developed using *OpenIllusionist*.

Overall, this thesis not only shows the feasibility of rapid VAE development, but also provides a practical framework, comprising a range of experimentally-verified technologies, to support efficient and robust applications.

Contents

1 Introduction **15**
1.1 Augmented Reality 15
1.2 Video-Augmented Environments 16
1.3 History of VAEs 16
1.4 Towards a Software Framework for VAEs 21
1.4.1 Platform Decisions for OpenIllusionist 22
1.5 Infrastructure 24
1.6 Contributions 26

2 Agent-Based Design **27**
2.1 Introduction 27
2.2 Background 28
2.3 Requirements 30
2.3.1 Object-Oriented Design 30
2.3.2 Threading 31
2.3.3 Sensors 32
2.3.4 Imaging 32
2.3.5 Messaging 33
2.3.6 Memory 33
2.4 The OpenIllusionist Agent 33
2.4.1 Agent Details 34
2.4.2 The Agent Cycle 36
2.4.3 Sensors 37
2.5 Rapid Development 38

2.6 Conclusions 38
2.6.1 Further Work 39

3 Framework 40
3.1 Introduction 40
3.2 Requirements for a Framework 40
3.2.1 Support for the Agent 41
3.3 Design 42
3.4 The OpenIllusionist Framework 43
3.4.1 Major Elements of the Framework 45
3.4.2 Major Methods of the Framework 48
3.5 Rapid Development 50
3.6 Conclusions 50
3.6.1 Further Work 50

4 Fiducials 52
4.1 Introduction 52
4.1.1 What is a fiducial? 52
4.1.2 Examples 52
4.2 Background 53
4.3 Requirements 57
4.3.1 General Requirements for a Fiducial System 57
4.3.2 Specific Requirements for a Video Augmented Environment 58
4.4 Proposed Design 60
4.4.1 Introduction 60
4.4.2 Ellipses 61
4.4.3 Segments 61
4.4.4 Centre Spot 62
4.4.5 Ternary 62
4.5 Implementation 62
4.5.1 Edge Detection 63
4.5.2 Object Finding 64

4.5.3 Ellipse Fitting 65
4.5.4 Fiducial Transform Calculation 71
4.5.5 Fiducial Verification 72
4.5.6 Fiducial Orientation and Identification 75
4.6 Optimisation 80
4.6.1 Test Platform 81
4.6.2 Buffer Size 83
4.6.3 Segment Start and End 83
4.6.4 Centre Spot Radius 90
4.6.5 Local Optimisation 91
4.7 Testing 93
4.7.1 False Positives 93
4.7.2 Size of Fiducial 93
4.7.3 Comparison 94
4.7.4 Binary or Ternary? 98
4.8 Results 98
4.8.1 False Positives 98
4.8.2 Size of Fiducial 100
4.8.3 Comparison 101
4.8.4 Binary or Ternary? 105
4.9 Conclusions 106
4.9.1 Further Work 107

5 Calibration **110**
5.1 Introduction 110
5.2 Scenarios 111
5.3 Background 111
5.4 Specification 117
5.5 Photometric Calibration 118
5.5.1 Algorithm Overview 119
5.5.2 Algorithm Description 122
5.5.3 Background Compensation 123
5.5.4 Plane Calibration 123

5.6 Geometric Calibration 126
5.6.1 Projective Transform 127
5.6.2 Optimising the Transform 129
5.7 Results 129
5.7.1 Plane Calibration 130
5.8 Conclusions 131
5.8.1 Further Work 133

6 Shadow Removal 134
6.1 Introduction 134
6.2 Background 135
6.3 Invariance Method 137
6.3.1 Colour Constancy 137
6.3.2 Entropy Minimisation 138
6.3.3 Retinex 139
6.4 Implementation 140
6.4.1 Preliminary Results 140
6.5 Conclusions 142
6.5.1 Further Work 142

7 Finger Detection 144
7.1 Introduction 144
7.1.1 Requirements 144
7.2 Background 144
7.3 Letessier and Bérard Method 146
7.3.1 Problems 148
7.4 Proposed Method 150
7.4.1 Overview 150
7.4.2 Analysis of the Invariance Image 150
7.4.3 Thresholding 151
7.5 Implementation 154
7.5.1 Overview 154
7.5.2 Frame Capture 154

7.5.3 Finger Verification 155
7.5.4 Testing Framework 156
7.5.5 Test Setup 156
7.6 Results 157
7.6.1 Optimisation 157
7.6.2 Comparison 161
7.7 Conclusions 163
7.7.1 Problems 163
7.7.2 Further Work 164

**8 Robot Ships:
An OpenIllusionist Application 165**
8.1 Introduction 166
8.2 Overview 167
8.3 Design 168
8.4 Development 169
8.4.1 Engine 169
8.4.2 Queen 169
8.4.3 Scout 170
8.4.4 Beacon 171
8.4.5 Worker 171
8.4.6 Tanker 174
8.4.7 Spill 174
8.4.8 Code 174
8.5 Final Evaluation 175
8.6 Conclusions 175

9 Conclusions 177
9.1 Summary 177
9.2 Overall 178
9.3 Further Work 179

A Agent Example 181
A.1 Header 181

A.2 Source 182

B Framework Example **184**

B.1 Header 184

B.2 Source 185

List of Figures

1.1 *OpenIllusionist* layer diagram. 23

2.1 *OpenIllusionist* agent class diagram. 34
2.2 OpenIllusionist agent flow diagram. 36

3.1 OpenIllusionist full class diagram. 44

4.1 A selection of example fiducials. 54
4.2 A selection of example fiducials. 55
4.3 Proposed fiducial in some example configurations. 60
4.4 Flow diagram of the edge detection algorithm. 63
4.5 Flow diagram of the object finding algorithm. 65
4.6 Parameters of an ellipse. 66
4.7 Focus points of an ellipse. 70
4.8 Images of fiducials indicating how they are processed at each stage. 73
4.9 Images of fiducials indicating how they are scanned. 77
4.10 Fiducial parameter dimensions. 80
4.11 Internal rendering of fiducials for optimisation tests. 81
4.12 Graph of buffer size results. 84
4.13 Reasons for failure. 85
4.14 Graph of segment end results. 86
4.15 Reasons for failure. 87
4.16 Graph of segment start results. 88
4.17 Reasons for failure. 89
4.18 Graph of centre spot results. 90

4.19 Reasons for failure. 91
4.20 Local optimisation of parameters. 92
4.21 The optimised fiducial design. 93
4.22 Fiducial test sheet. When printed the fiducials each have a width of 47mm. 95
4.23 An example from each group of test images. 97
4.24 Binary against ternary fiducial test sheet. 99
4.25 False positives detected in test footage. 100
4.26 Fiducial size performance results. 101
4.27 An example of output images from the testing software. 102
4.28 Fiducial comparison results — not found. 102
4.29 Fiducial comparison results — incorrect identification. 103
4.30 Fiducial comparison results — true positives. 104
4.31 Binary against ternary — simulation test results. 106
4.32 Binary against ternary — captured image test results. 106

5.1 A typical camera-surface-projector configuration. 116
5.2 A flowchart of the photometric calibration algorithm. 120
5.3 Flowcharts of processes involved in the photometric calibration algorithm. 121
5.4 Pixel values for an image captured from a VAE. 130
5.5 Pixel values for an image captured from a VAE after plane calibration. 131
5.6 An image captured from a VAE before and after plane subtraction. 132
5.7 The corresponding histograms of an image captured from a VAE before and after plane subtraction. 132

6.1 Inaccuracies in edge detection caused by shadows. 134
6.2 Entropy minimisation example 138
6.3 Results of invariance method with a single light source. 141
6.4 Results of invariance method with multiple light sources. . . . 141

7.1 Captured frames (left) and the corresponding invariance images (right). 150
7.2 Results of invariance method without plane correction. 152
7.3 An invariance image thresholded at −3 SD. 153
7.4 An invariance image after thresholding and filtering 154
7.5 An image being processed by the finger verification application 155
7.6 Optimisation of LB method — True positives. 158
7.7 Optimisation of invariance method — True positives. 158
7.8 Optimisation of LB method — False positives. 159
7.9 Optimisation of invariance method — False positives. 160
7.10 Optimisation of methods — True positives against false positives for all lighting conditions. 160
7.11 Comparison of methods after optimisation — Overhead lights only. 161
7.12 Comparison of methods after optimisation — Overheads and overcast daylight. 162
7.13 Comparison of methods after optimisation — Overheads and bright daylight. 162
7.14 Example output from the proposed finger detection algorithm. 163

8.1 Photo of the installation at the Royal Museum in Edinburgh. . 165
8.2 Screenshot of *Robot Ships*. 166
8.3 Arrangement of sensors for the scout agent. 170
8.4 Arrangement of sensors for the worker agent. 173

List of Tables

4.1 Number of codes available in each of the possible fiducial configurations. 61
4.2 Parameter ranges for testing. 82
4.3 Optimised parameter values. 92
4.4 False positive results. 99
4.5 Algorithm performance results. 105

5.1 Setup scenarios. 111
5.2 Background scenarios. 112
5.3 Foreground scenarios. 113
5.4 Dealing with different types of clutter during calibration. . . . 113
5.5 Indirect lighting scenarios. 114
5.6 Direct lighting scenarios. 115

7.1 Table showing the size and type of test image sets. 156
7.2 Table showing the ranges over which parameters were varied. . 157

8.1 Number of lines of code per class in *Robot Ships*. 174

List of Acronyms

AR	Augmented Reality
BDI	Belief-Desire-Intent
CED	Chrominance Euclidean Distance
ELF	Elliptical Luminance Fiducial
FRF	Fast Rejection Filter
GPL	General Public License
GUI	Graphical User Interface
HMD	Head-Mounted Display
LC	Log-Chromaticity
L-GPL	Library General Public License
LMS	Least Mean Squares
MR	Mixed Reality
OCR	Optical Character Recognition
RAG	Region Adjacency Graph
RANSAC	Random Sample Consensus
TRIP	Target Recognition using Image Processing
TUI	Tangible User Interface
VAE	Video-Augmented Environment
V4L	Video For Linux
VR	Virtual Reality

Acknowledgements

I would like to thank my supervisor, John Robinson, for his support and guidance throughout my research. I would also like to express my gratitude to the National Museums of Scotland for giving us the opportunity to develop Robot Ships.

Many thanks to Justen Hyde for his part in creating and developing the OpenIllusionist framework, for his support and friendship, and for the many late nights spent solving a variety of implementation issues.

Finally, I would like to thank Caroline and my family for all of their love and encouragement.

Chapter 1

Introduction

The aim of this research is to examine, define and then meet the requirements of the Video-Augmented Environment (VAE). A software framework for the rapid development of stable VAE applications will be implemented in order to validate the solutions provided by the research.

1.1 Augmented Reality

Augmented Reality (AR), sometimes referred to as Mixed Reality (MR), brings together the real, physical world surrounding us and the synthetic world of Virtual Reality (VR). In the real world we interact with everything around us using all of our senses and motor abilities. In VR we interact with virtual objects and although the technology attempts to simulate the real world as accurately as possible, it is not yet at a point where a user finds the differences imperceptible, especially with regards to senses other than vision and hearing.

AR is concerned with enhancing the real world with virtual information. This augmentation could be anything from overlaying instructions on how to use a particular piece of machinery, to generating a virtual pet that appears to exist in the physical world and responds to the environment accordingly.

Although AR often makes use of VR technology, the interactions of the user remain anchored in the real world and therefore a different set of applications present themselves. These applications extend to areas such as education, entertainment, surgery and urban planning to name but a few. The implementation of AR also presents different design problems to those in VR, including the accurate detection of a user's actions and an appropriate response in real-time. These problems are investigated and solutions proposed in this thesis.

1.2 Video-Augmented Environments

Although a large proportion of research is carried out in the field of Head-Mounted Display (HMD) and hand-held AR, a significant amount of research is concerned with VAEs which are based upon projected AR.

The typical setup for a VAE is the camera-surface-projector configuration where a surface such as a wall or a tabletop is both projected onto and observed by the system. The projector is used to augment physical objects on or in front of the surface and can also render virtual objects directly onto the surface. Although a VAE can be used by a single person, a major advantage is that it can provide group-oriented interaction and collaboration. This type of AR is the focus of the thesis.

1.3 History of VAEs

The seminal work of Wellner [73, 74] describes the *DigitalDesk* which attempted to bridge the gap between a desktop computer where files are manipulated and the physical desktop where paper is manipulated. He introduced a vision-based system that gave rise to the VAE and inspired researchers in AR.

The *DigitalDesk* incorporated two important techniques in order to provide an example of what was possible then and what could be possible in the future. The first technique was Optical Character Recognition (OCR) so that the system could read what was written on a printed page and the second was finger detection so that it knew where the user was pointing. Using these image processing methods the *DigitalDesk* would allow a user to point at a number on a printed page and copy that number to a virtual calculator which was projected onto the desk. The user could then tap buttons on the virtual calculator to perform a calculation (finger taps were detected by a microphone attached to the desk).

An interesting point to note from [74] is that users of the *DigitalDesk* were not bothered by the occlusion and shadows produced when using projection from above. This was attributed to the fact that they were already accustomed to shadowing on desks from overhead lights.

In 1991, Stafford-Fraser and Robinson introduced the *BrightBoard* which was an augmented whiteboard [64]. Instead of developing an active whiteboard, which can be expensive, the system uses an ordinary one and monitors what is happening through a vision system. It was designed so that the software could be run on a workstation continuously and therefore uses a triggering system to detect when it should capture an image to process. This same triggering system attempted to capture images in which a user is not obstructing the whiteboard. Commands could be drawn on the whiteboard by the user in the form of letters surrounded by a box. The commands would then activate scripts for specific tasks such as printing or saving what was written on the board. One of the principal limitations of the system is that it required the camera to be looking straight at the whiteboard resulting in a greater probability of occlusion by the user.

An update to the *DigitalDesk* was presented in [42] named *Marcel*. Instead of relying on a high-resolution camera focussed on a small part of the desk to perform OCR, *Marcel* used a low resolution camera. It simply matched sheets of paper with pre-processed scans via a simple set of checks involving the length of lines, paragraph heights and word breaks on a page. This meant that it could monitor the entire desk for activity, although at the time it still took around 8 seconds to process a full image. An additional application to the virtual calculator was also presented in which the user could select a word on a page and a translation would be projected elsewhere on the desk.

The *Origami Project* was yet another modification to the *DigitalDesk* described in [60]. This relied on pages that were marked with an identification number in an OCR font for easy recognition. It could then provide augmented content that was correctly aligned with the page. The project also included a method of interaction using a pen fitted with an LED for easy recognition by a vision system.

The *LivePaper* system [59] provides a VAE in which individual sheets of paper are treated as objects with additional "magical" properties. A sheet of paper is detected through boundary extraction in a captured image and then the contents of that page are recognised by the system and the associated augmentations are displayed. The augmentations will remain attached to the page, as if written on it, when moved and a finger-triggered menu system is displayed beside it to provide additional functionality.

One of the most active groups in AR is that of the Tangible Media Group at MIT led by Hiroshi Ishii. In [32] they present the Tangible User Interface (TUI) which provides a link between the elements of the Graphical User Interface (GUI) that computer users are familiar with, such as windows and menus, and the physical desktop used in the real world. Their research focusses on interactive surfaces that use physical, tangible objects to provide the interaction. Using a combination of image processing and sensor technology they are able to detect the position and orientation of objects and provide feedback by way of projection onto the surface.

An interesting point that is made by Ishii and Ullmer in [32] is the importance of ambience and how we are aware of many details within our environment even if we are concentrating on just one. If a change occurs in our peripheral awareness we are able to instantly shift focus to that event. It is possible to exploit this ability by incorporating subtle elements within our surroundings that indicate the status of information which will attract our attention if that status changes. An example given by Ishii and Ullmer is a web page hit counter which produces the sound of rain drops; when the hit rate changes the rain becomes heavier or lighter accordingly providing the user with ambient feedback about digital information.

A TUI for musical performance is presented in [48] which uses RF tagged pucks to control various parameters of sampled sounds. It describes how the use of both hands provides a more natural interface to a computer than the typical mouse. Although the paper concentrates on the iterative design of the TUI elements, the particular application highlights the need to "*make interactions legible for observers*". In other words, an application should provide an interesting and understandable experience for both users and spectators.

In [56] they present two implementations of a VAE application for the process of landscape design. The most accurate, and expensive, method is called *Illuminating Clay* which incorporates a projector and laser scanner. With this method simple clay can be used as a 3D landscaping construct and the system is capable of dealing with objects or even hands placed on the surface. The alternative method, named *SandScape*, employs infra-red light which is transmitted through a bed of glass beads (the sand) and detected by a camera above. The intensity of the transmitted light can be used to calculate the height of the surface at any given point. This method, however, will only work with transparent beads and will not cope with objects placed on the surface. In both cases the applications can then provide information for the user graphically, such as how water will flow in different areas of the surface model.

A further application developed by the Tangible Media Group is an urban planning workbench called *Luminous Table* [31]. The workbench allows urban designers to construct a combination of 3D physical models and 2D drawings and then overlay them with digital simulation information, as in the landscape design application. This allows designers to see the effects of shadows, reflections and wind flow caused by buildings along with traffic flow around roads. The application makes use of two projectors and two cameras to provide a large working space that can support many users. This provides a good scenario in which a VAE promotes social interaction.

The *Luminous Room* was presented in [69] where Underkoffler et al. discuss various applications of the VAE. Their applications rely on an appliance called the *I/O Bulb* which is intended as a replacement for the ordinary light bulb and incorporates both projection and camera imaging technologies. Currently in a prototype stage, the *I/O Bulb* is supposed to unobtrusively integrate VAE technology into a room. By marking simple real-world objects with retro-reflective coloured dots in specific patterns the system can identify and locate those objects to provide interactive elements for the application.

The *Luminous Room* is perhaps one way of realising the "*office of the future*" as described by Raskar et al. [54]. In this paper they describe an office in which surfaces are augmented to overlay information and walls are augmented to provide the user with views of other offices in such a way as to effectively create one large room. This uses a dynamic image-based model of a room to generate a 3D view that can be then be transmitted to users in other rooms. Those views are projected so that the perspective is correct for a user by tagging them and thus knowing their location. Their proof-of-concept implementation introduces an impressive technology called *imperceptible structured light* which allows patterns to be hidden within normal projected images. The patterns are invisible to humans but can be detected by a correctly synchronised camera. This structured light is used to calculate the model of a room by providing depth information. The 3D model can be further improved by merging information from multiple devices.

At present many of the research groups continue their work on VAEs and some of the individual contributions will be reviewed in more detail at appropriate points throughout this thesis.

1.4 Towards a Software Framework for VAEs

Researchers in AR often make use of software libraries that provide support for particular technological features. The most widely used is probably the *ARToolkit* [35] which provides marker tracking capabilities. It does not provide a framework on which AR applications can be built, but it does provide useful functionality to use within a framework.

Studierstube [65] is a project that provides a freely available framework with which researchers and developers can build AR applications. The primary aim of the framework is to incorporate 3D virtual information into a real world scene using HMD technology and hand-held devices.

Another development framework is the *MR Platform* [68] for *Linux*. As with *Studierstube*, it provides a framework on which to develop applications aimed at HMD devices which require very accurate and complex calibration routines. The project that resulted in the development of the *MR Platform* was also responsible for producing an innovative HMD named *COASTAR* which allows the optical axes of the see-through displays and the cameras to coincide; this greatly improves the accuracy with which 3D augmentation can be applied.

At the beginning of the current thesis work there were no publicly available software frameworks directed towards VAEs. In response to this, the author and Justen Hyde initiated the *OpenIllusionist Project* [45] to develop an open-source framework with which to produce VAE applications. It is not concerned with specialist equipment such as the HMD but with readily available hardware (projectors, modern PCs and webcams) meaning that anybody should be able to setup their own VAE. This ease of use is a vital part of helping to take AR out of the research lab and into people's homes and offices.

The framework is not only designed to produce simple to setup applications, but to also simplify the application development process itself. By providing the underlying image processing, calibration, threading and general structure, *OpenIllusionist* allows a developer to concentrate on the important aspects of the application without having to worry about producing a stable infrastructure.

The inspiration behind *OpenIllusionist* as an agent-based system came from *PenPets* [44] developed by Sean O'Mahony. *PenPets* was a VAE application that allowed users to interact with virtual pets by drawing lines or placing objects on a whiteboard.

The *OpenIllusionist* framework was created as a means to reproduce the *PenPets* demo, to help create future applications and to assist in further research. The potential of rapid development for both developers and researchers alike led to the release of the framework under the GNU General Public License (GPL).

1.4.1 Platform Decisions for OpenIllusionist

Even though much of the research presented in this thesis can apply to any VAE, *OpenIllusionist* provides a practical constraint and a means with which to check the feasibility of possible solutions. Before explaining the major areas of research in section 1.5, the remainder of this section provides a little more detail about the nature of the constraint.

OpenIllusionist was originally developed as a *Windows* framework that accessed the *Windows* API directly. However, it was later decided to move to a cross-platform library which would also provide some useful utility classes. The main two libraries to choose between were *wxWidgets* [76] and *Qt* [52].

Qt has restrictive licensing and prohibitive costs for commercial development whereas *wxWidgets* is released under a modified Library General Public License (L-GPL) license meaning that developers can release both commercial and open-source software with any license they wish. The flexibility of the *wxWidgets* licensing and the ease with which it can be used made it the obvious choice.

The rendering aspect of the framework required a likewise cross-platform and flexible library. *Direct3D* is powerful but difficult to develop with and restricted to the Windows platform only. *OpenGL* has been implemented on all major platforms, and most minor ones too, and provides a simple method with which to render graphics; it is therefore used by *OpenIllusionist* extensively.

Capturing frames from an external device such as a camera usually relies on very platform specific methods. Therefore *OpenIllusionist* uses a frame-grabber library designed by the author to provide a level of abstraction such that different frame-grabbers can be developed for different platforms. Since the primary platform on which the framework is currently used is *Windows*, the method of capturing frames uses *DirectShow* since it provides access to almost any available capture device on the system (including all webcams with appropriate drivers). *DirectShow* also provides a standard mechanism for adjusting camera parameters which thereby allow the framework to change the camera setup during calibration. Future developments are intended to include a frame-grabber library for *Linux* using Video For Linux (V4L).

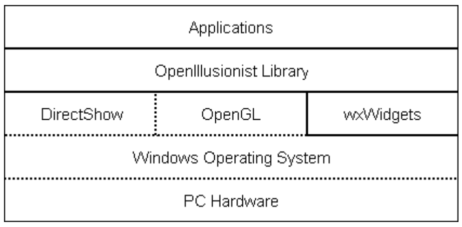


Fig. 1.1: *OpenIllusionist* layer diagram.

The diagram in figure 1.1 depicts the relationship between *OpenIllusionist*, the *Windows* platform and VAE applications.

OpenIllusionist contains all of the elements discussed in this thesis except those concerning shadow removal and finger detection, although they will be included in the future. It currently consists of approximately 5500 lines of code and is compiled as a binary library file for simple inclusion in a VAE application.

1.5 Infrastructure

This thesis is concerned with all of the necessary elements of an infrastructure for the rapid development of stable VAE applications. It also discusses how some of those elements have been implemented within the *OpenIllusionist* framework. The primary input to a VAE is through a vision system and therefore many of the elements presented deal with aspects of image processing. Relevant image processing topics are selected that are required to support an intuitive user interface and that can also provide the means with which to calibrate the system and cope with different environmental conditions. The remaining elements discussed in this thesis approach the software development concerns.

Although some research has been published that combines agents with AR there is currently no other VAE development platform that inherently supports agents. Therefore chapter 2 introduces agent-based design and explains how it can provide an appropriate model with which to develop VAE applications. It goes on to describe how *OpenIllusionist* provides an agent-based development platform and then chapter 3 illustrates how a framework can support an agent-based design. It also shows how all of the infrastructure elements can fit together in a cohesive and efficient manner.

The detection of fiducials (markers) is an important requirement for an infrastructure of this kind since the system must have a method with which it can recognise interactive objects and perform automated calibration. Therefore chapter 4 provides an extensive look at this subject and proposes a new design of fiducial that incorporates multi-level data.

A VAE must be able to calibrate itself, preferably in an automated and efficient way. Many of the earlier VAE systems, as discussed in the background section, were based in controlled laboratory conditions and required manual calibration. Chapter 5 presents an automated method for calibration that can be incorporated into a framework.

Surprisingly the problems caused by shadows in vision systems are often ignored or disregarded, but they can cause serious problems for image processing algorithms. Chapter 6 tackles the problem of shadows that a VAE is likely to face, especially when placed in uncontrolled lighting conditions.

In the *DigitalDesk*, single fingertip detection was employed to allow interaction with a virtual calculator. Finger detection is a desirable, although not a necessary, element for any augmented environment. Recent advances in finger detection methods and faster computers mean that multi-finger detection is efficient enough to incorporate into a VAE system. Chapter 7 reviews finger detection methods and then presents an algorithm that uses an alternative to background differencing.

A practical example of an application developed with *OpenIllusionist* is described in chapter 8 and finally the overall conclusions are discussed in chapter 9.

Since this thesis covers a wide range of topics most of the chapters contain their own background sections and conclusions.

The terms “infrastructure” and “framework” can usually be used interchangeably. However, within this thesis “infrastructure” refers to the combination of elements required for a practical VAE and "framework" refers to the actual implementation of such an infrastructure.

1.6 Contributions

The research discussed here has led to the development of the *OpenIllusionist* framework both as a means to validate the research and to develop full applications. Although the framework itself does not provide an advancement of "state-of-the-art", since many of the techniques used are already well known in the field, it does bring together these techniques in a coherent and practical way. This provides a solid foundation on which future research can be built.

A method for calibration is introduced that attempts to cancel out the effects caused by uneven lighting across a planar surface. Initial testing suggests that it has potential for use in a VAE.

This thesis proposes a new fiducial design that can incorporate either binary or ternary identification codes. It is proved to be a reliable alternative to other widely used fiducials.

An adaptation to an existing finger detection algorithm is also proposed that provides a significant increase in performance.

Chapter 2

Agent-Based Design

2.1 Introduction

This chapter describes the motivation for using an agent-based design and the specific features required by an agent within a VAE. A framework built upon the model of an autonomous mobile agent gives the flexibility of producing artificial-life style applications with ease whilst still being able to produce a user interface with static controls. A static control can simply be an agent with rudimentary behaviour and very little, if any, animation.

Consider an object in the physical world such as a pen; it is an autonomous entity and as such will move completely independently of other objects on a desk when pushed. Virtual objects in an augmented environment must be able to respond in a similarly independent and parallel way to reinforce the illusion of the augmentation, to further blur the boundaries between the physical and virtual world. When a physical object is capable of moving itself it has an even higher degree of autonomy, in fact it appears to have agency. Similarly, virtual objects that move and interact with each other and the physical world can naturally be thought of, and therefore designed, as agents.

A developer working on a custom application should be able to create individual elements in a modular way. Productivity would be greater if the methods with which these elements are executed, how they interact and how they communicate are already defined and implemented. An agent-based system would allow this modularity and support the requirement of autonomy along with other desirable characteristics such as complexity and adaptability. An agent could be any number of things that contribute to the VAE experience, some examples are:

- A graphical button that responds to a user's finger.
- A simulation of an animal that can move around and react to its environment.
- A simple animated object that does not respond to anything but is simply there to provide information or context.
- An invisible agent that responds to user interaction by triggering events elsewhere.

The diverse properties and capabilities of agents allow developers to create complex systems with emergent behaviours and has the advantage that a particular unforeseen circumstance is far less likely to bring down the whole application. Additionally an agent-based framework means that the agents inherently have limited information available to them and therefore provide a good foundation for developing an application such as an artificial-life simulation.

2.2 Background

The Belief-Desire-Intent (BDI) agent model [9, 28], which is the most well known practical reasoning method (i.e., based around the same practical reasoning used by humans), defines an agent as having:

- *Belief*
  An agent has beliefs about the world based on information it has about its environment.

- *Desire*
  An agent has desires such as goals it must achieve.
- *Intent*
  An agent has intentions meaning that it has committed to plans based on its beliefs and desires.

Although this chapter is not considering the development of support for systems such as those proposed in [9, 28], the structure should be flexible enough that it could be added if required. The BDI model is a logical way of describing how even the simplest of agents function.

Incorporating a knowledge-based system such as that discussed in [18] into an agent-based system is also possible and could act as an alternative to the BDI agent model.

One of the earliest uses of an agent in an AR application was the ALIVE system [40]. Maes et al. introduce a mirror paradigm in which the user can see themselves in a virtual reflection of the room on a projection screen. Through gesture a user can interact with autonomous virtual characters within the scene.

Although there is a large amount of literature on agent research, a good overview being provided in [33], there is very limited information about agents currently being used in augmented reality. The only active development that the author is aware of in this area is within the *Studierstube* project headed by Dieter Schmalstieg.

The *Studierstube* project is concerned with producing a framework for developing augmented reality applications. However, unlike *OpenIllusionist*, they tend to concentrate on HMD technology and hand-held systems. Barakonyi et al. [3] discuss an agent framework referred to as "AR Puppet" which is based upon a hierarchy as used in theatre where there is a director, choreographer, puppeteer and puppet. In this scenario the director and choreographer have global views of the environment and agents whereas the puppeteer and puppet only have local views and deal with low-level details.

Barakonyi et al. continue in [4] to discuss applying the agent model to computer entertainment and education and then in [5] introduce an AR agent-based game called "MonkeyBridge". The game relies on a set of physical building blocks that shape both the physical surface and the virtual world, HMD technology and fiducials (refer to chapter 4) to provide a novel interactive environment in which autonomous agents are integrated.

2.3 Requirements

The following section describes some of the requirements necessary or desirable to produce an extensible and flexible agent structure. As with all elements of a VAE, the agent and the code that supports the agent must be efficient since many elements will be running concurrently.

2.3.1 Object-Oriented Design

Although an object in object-oriented programming is not an agent it can offer a good basis for agent design:

- Objects can be designed to run concurrently through multi-threading (modern computing power can allow many agents to run concurrently).
- Autonomy can be achieved by concealing and controlling their internal state.
- They can be made aware of their local environment through feedback mechanisms such as sensors.
- Communication abilities can be included that convey information to other agents or users.

A primary concern is efficiency and so when dealing with a large, complex infrastructure that is required to respond quickly it may not be sensible to adhere to the strict rules of object-oriented design. For example, making all member variables private and only providing access through function calls could seriously inhibit the speed of the system. However, the structure can still take advantage of some of the desirable features of object-oriented design such as classes, inheritance and polymorphism.

2.3.2 Threading

Within the framework there must be a means by which agents can run concurrently and the simplest way to achieve this is to create each agent as a thread in the current process. Additionally the VAE requires other elements to be running simultaneously such as frame-grabbing, image processing and rendering, all of which can be allocated threads.

Current computer architectures can actually take advantage of threaded applications and improve their performance with the use of multi-processor and hyper-threading technology.

Unfortunately there are many potential dangers concerning the use of threads, primarily to do with the synchronisation between them. Multiple threads cannot be permitted to write to the same location at the same time and so a critical section (or a mutex) must be used. Overuse of critical sections, however, can have a detrimental effect on performance so keeping the critical parts of the code as brief and rarely used as possible is desirable. Performance is a key issue with regards to this kind of system since so many elements are required to run concurrently.

Ideally the framework should hide as much of the threading as possible from the application developer to reduce the chances that they produce unsafe code whilst also improving their productivity by obscuring the complexity.

2.3.3 Sensors

An agent requires a method through which it perceives the physical and virtual world; this will govern its beliefs. It must be simple and efficient since complex sensors, such as giving an agent binocular vision, would require huge amounts of calculation, rendering and data exchange. This would result in major performance issues with even a small number of agents running. Another point is that the sensor system should not ordinarily give agents a view of the entire world since this could result in less believable behaviour.

A sensor could simply be a line, corresponding to a single ray in a vision paradigm, along which the agent can "see" other objects. A vector sensor like this would be quick to scan and simple to implement. Strategic placing of the sensors, or even dynamic adjustment of them, can allow the agent to have a reasonable view of the world.

2.3.4 Imaging

An addition, or alternative, to the sensor system can be provided through imaging. By allowing an agent to request a localised rectangular image it can perform its own image processing internally. Since the image provided would be taken from that of the original captured frame it would not provide clear information about other virtual agents but could allow the agent to extract information about the physical world.

As with the sensor system, the image should provide an agent with a limited view of the world. This should be based on what is reasonable for that agent to detect or "see" from its current location. However, both the sensor and imaging systems should be flexible enough that a developer could provide an agent with a global view of the world but only if it is absolutely necessary for a particular application.

2.3.5 Messaging

Another important attribute of an agent is the ability to communicate with others. A messaging system needs to be a part of the infrastructure so that the problem of communication between threads can be dealt with internally. This means that developers building a custom application do not have to concern themselves with the complexity of messaging, but can just create and post a message in a straightforward way.

Communication can allow agents to share beliefs, desires and intents so that they can collaborate to tackle much bigger goals, alongside attempts to complete personal goals.

2.3.6 Memory

An agent must be able to store information, especially if a developer intends to design an agent capable of learning. The base class should contain storage for all of the information that is relevant to all agents, however storage for information that a particular agent requires can simply be included in the derived class. This ensures that the base class remains as clear and efficient as possible.

The memory is not only useful for storing the current state and beliefs of an agent, but can also be used to store previous beliefs, desires and intents that could govern future decisions.

2.4 The OpenIllusionist Agent

The base agent class in *OpenIllusionist* (oiAgent) has been developed such that it meets the requirements described in section 2.3. A UML class diagram is shown in figure 2.1 indicating all of the most important attributes of the class and its associated structures.

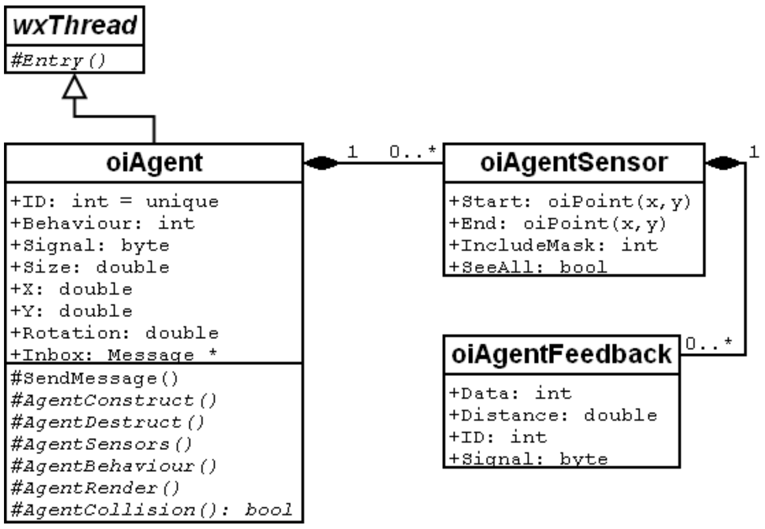


Fig. 2.1: *OpenIllusionist* agent class diagram.

This class-based approach provides a logical, hierarchical process to developing agents. The complicated functionality vital to the running of the agent is completely contained in the base class. Therefore any derived class need only implement the functionality that is specific to the behaviour and graphic design of that particular agent.

2.4.1 Agent Details

The agent class is derived from a generic thread class (from the *wxWidgets* library) to provide it with inherent threading capabilities. An agent retains information relevant to the infrastructure in public variables, which are public out of a necessity for efficiency since other parts of the framework can access them directly. These variables provide data such as the unique identity of the agent, which behavioural model it has and its size, location and orientation in the virtual world.

The “Inbox” is a message pointer that is either NULL or contains a message for the agent to process during that cycle (a cycle, in this case, referring to a single pass through the main execution loop of the thread).

The "SendMessage" function is provided as a straightforward means of posting a message. The messaging process within *OpenIllusionist* is described in more detail in chapter 3.

The remaining functions are virtual and must be overridden when creating a custom (derived) agent:

- *AgentConstruct*
  This is executed during the start-up of the thread and is where any extra memory that a custom agent uses must be declared. Agent variables should be initialised here plus any sensors that are required must be created.

- *AgentDestruct*
  Executed during the shut-down of the thread and is where extra memory must be cleaned up.

- *AgentSensors*
  This is where the sensors are setup. If an agent has moving sensors then this function can be called each cycle or whenever is necessary. Alternatively, it can be called just once if the agent has fixed sensors.

- *AgentBehaviour*
  The most important function of all and where the sensor information is processed, messages are read, decisions are made, movements are performed, messages are posted and much more.

- *AgentRender*
  Called from outside the agent thread during the *OpenGL* render cycle, it is a request for the agent to draw itself.

- *AgentCollision*
  Also called from outside the agent thread, this function asks the agent whether or not a specific coordinate collides with it and the function returns true or false. This means that it is the responsibility of the agent to decide what collision boundaries it has, therefore, it can have a

complex shape without burdening the rest of the system with knowledge about that shape. If a particular agent simply returns false here then it is invisible to all other agents.

2.4.2 The Agent Cycle

The main execution loop of the agent (see figure 2.2) consists of a simple set of function calls which are then implemented in the derived agents as described above. The “AgentFeedback” function is a call-back to the *oiIllusionistEngine* which owns all of the agents. The call-back represents the point at which the agent synchronises with the main process and acquires updated information about the local environment.

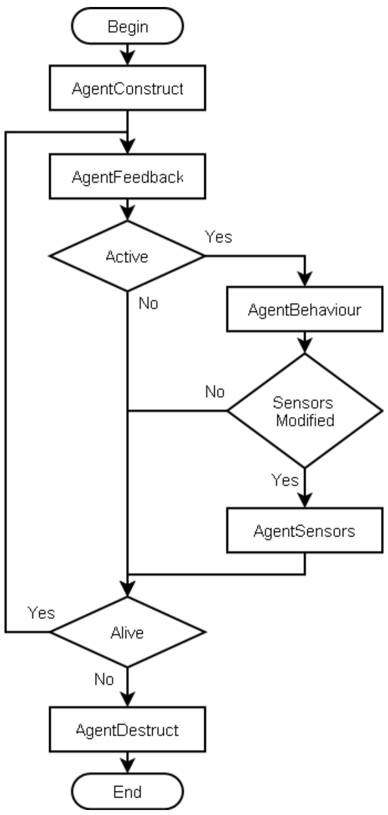


Fig. 2.2: OpenIllusionist agent flow diagram.

The feedback function provides the agent with beliefs about the environment. The "AgentBehaviour" is where an agent forms intents based on what it is trying to achieve ("desires") and then acts upon them accordingly.

2.4.3 Sensors

A sensor in *OpenIllusionist* is a vector defined by start and end coordinates which are relative to the agent. The "IncludeMask" variable is a bit mask that tells the engine what this sensor is permitted to see. The "SeeAll" flag indicates whether or not the agent is permitted to see more than one item along a sensor. If set to false a sensor will act like vision in that everything behind the first item will be occluded whereas if set to true an item could be sensed even when occluded, similar to hearing.

The agent initialises its sensors and modifies them if and when necessary. Each sensor also contains a fixed number of feedback structures that are updated by the engine. When the "AgentFeedback" function is called the engine populates the feedback structures with information relating to the local environment. Each item of feedback contains the following information (as seen in figure 2.1):

- *Data*
  Indicates what is visible on the sensor. It could be a physical object, another agent or nothing at all.
- *Distance*
  Informs at what distance along the sensor an item was seen.
- *ID*
  If another agent is visible on the sensor then this provides the ID of that agent. This allows one agent to post a message to another if in close proximity.
- *Signal*
  Each agent contains a signal that can be used to indicate their current status. Another agent can read this signal via the sensors.

2.5 Rapid Development

The C++ source code for an example agent has been provided in appendix A. When compiled into an *OpenIllusionist* application this agent will be rendered as a simple red rectangle capable of moving forwards and then turning on the spot when it encounters an obstruction (virtual or physical). Comments in the source code explain in detail how the agent works.

The emphasis on agent design within the framework is that a developer should be able to concentrate on the behaviour of an agent and not need concern themselves with issues such as threading and messaging. Although messaging is not used in this example, sending a message asynchronously is simply a case of calling the "SendMessage" function (more information is provided about the messaging system in chapter 3).

The rendering example uses a few simple OpenGL commands but a developer could use any of the available OpenGL features for rendering. Therefore an agent could be anything from a simple coloured polygon, as in the example, to a complex, textured and animated mesh.

The agent was produced in approximately 80 lines of code and yet is capable of moving and interacting with other virtual and physical objects. This shows that an agent-based approach is appropriate to the development of VAEs. Due to the class-based implementation of the agent, many instances of it can be created and run concurrently.

2.6 Conclusions

This chapter provides the motivation and specification for an agent-based design and then proposes an implementation that allows a developer to produce reliable agents in an efficient manner. The example in appendix A shows that very little code is required to get a basic agent running since the implementation of threading, messaging and synchronisation is handled within the library itself and is therefore hidden from the application developer.

A real example that makes extensive use of the agent model and messaging system is *Robot Ships* which is described in more detail in chapter 8.

The primary drawback to using an agent-based model is apparent where a large number of agents are required. Since the agents are threaded to allow them to run concurrently this means that there could be a large number of threads which would seriously affect the responsiveness of the application. This problem is only likely to occur with many hundreds of complex agents when using a modern computer and the advancement of multi-processor systems should soon provide the power to deal with many thousands of agents.

2.6.1 Further Work

Multiple threaded agents can reduce the responsiveness of the system and so the majority of further work should be concerned with optimisations to the code that supports the agents. In particular, the point at which an agent synchronises with the main thread provides the largest bottleneck. Removing as much calculation as possible from the feedback process and keeping it as short and efficient as possible will help with the overall speed of the process.

In order to promote the development of intelligent agents the base agent class could be modified to provide additional functionality for developing BDI agents.

By producing subsets of behavioural functions it would be possible to allow agent design to consist of simply plugging in the behaviours that are desired. This can then be further extended by providing a graphical design interface for the agents that allows anybody (not just developers) to construct agents using “plug-and-play” behaviours, appearances and sensors.

Chapter 3

Framework

3.1 Introduction

To develop an agent-based VAE requires a framework that can support the agents and perform the other necessary tasks such as image processing. This chapter discusses some of the requirements for such a framework, it describes the implementation within OpenIllusionist and then how that implementation allows for rapid development of VAE applications.

3.2 Requirements for a Framework

One of the primary features of a VAE is the ability to provide interaction through an interface that does not encumber the user. This interface is vision and so the framework should support the following capabilities:

- Capturing images from a camera.
- Concurrent execution with other parts of the framework.
- Self-calibration.
- Extensible image processing.

The framework must provide multi-threading support so that various sections are able to run concurrently including individual agents. Ideally the threading support must be transparent to the developer using the framework library and so all issues with synchronisation should be dealt with internally.

Agents, and indeed other parts of the framework, must be able to send messages to each other. The process of sending and receiving messages must be implemented in a thread-safe way and yet remain efficient enough to prevent slowing down the system.

The input to the framework is via the image capture and processing system, but it also needs an output that can provide the user with feedback. This output could provide information to any of the senses, however, this framework is primarily concerned with video-augmentation and so should concentrate on the visual aspect, therefore providing graphical rendering capabilities.

3.2.1 Support for the Agent

The framework must be designed to deal with many agents. As mentioned before they require threading and messaging capabilities that should be implemented to run as efficiently as possible. They must also have a fast method of retrieving sensor information from the main thread so as not to produce a bottleneck in the system.

As suggested in chapter 2 it would also be useful if an agent could request a small section of image (captured from the camera) so that it would be possible to perform localised image processing within that agent. To this end, the framework should provide the capabilities to request and provide image segments during the same feedback method in which sensor information is updated.

3.3 Design

Since object-oriented design is well suited to developing an agent model then it makes sense to use a class-based structure for the entire framework. Ordinarily an application has a main execution thread from which it launches child threads, therefore the main thread requires an overall controlling class that instantiates, monitors and communicates with all of the other major classes. It should have the following responsibilities:

- Launching agents and providing them with feedback about their environment.
- Administering the messaging system.
- Controlling the synchronisation of threads.
- Controlling the classes related to image capture.
- Handling user interaction such as key-presses.
- Rendering anything an application requires that is not handled by the agents themselves.

An object-oriented approach will also provide a framework that is simpler to extend since classes can be overridden and functionality added. The design of the system should be adaptable to this type of modification by providing logical and flexible base class structures.

As mentioned in the agent-based design chapter, the sensor system for an agent (i.e., the method with which it perceives the virtual and physical world) should be as efficient as possible. Sensor feedback requires synchronisation with the main thread and if an application contains many agents then this could result in a major processing bottleneck. Ideally, the system would render the world view from the perspective of the agent, but this would be very inefficient. Instead a simple vector sensor design, as proposed in chapter 2, is more appropriate since it is very adaptable and will consume far less processing time.

wxWidgets contains a well established, cross-platform and simple to use thread class which can provide a good basis for all objects within *OpenIllusionist* that require threading. With additional support in the core infrastructure, threading can be completely obscured from the developer of an application. This can be achieved by completely handling the thread loop inside the framework and then calling various functions at the appropriate points in the thread cycle. These functions can be overridden in a derived class without the need to consider how and when they will be executed.

The rendering process must maintain a reasonable frame-rate and so cannot rely on waiting for an agent to render itself during the thread cycle. In fact, a rendering context will not normally support rendering from multiple threads. One possibility is to allow an agent, on initialisation, to provide the framework with a description of how it is rendered which the renderer can simply refer to each frame. The major disadvantage to this solution is that an agent cannot dynamically change how it looks very easily. Therefore it would be better if an agent was responsible for rendering itself and so functionality must be provided that allows an agent to safely render itself even while its thread cycle is running in parallel with the rendering thread.

3.4 The OpenIllusionist Framework

Refer to figure 3.1 for an overview of the framework and how the elements relate to each other. Some of the less important variables and functions have been excluded to avoid further complication of the diagram.

The high-level design of the *OpenIllusionist* framework [45] was the result of an equal contribution from both the author and Justen Hyde, although much of the implementation was by the author. All other topics covered in this thesis are comprised entirely of the author's own research.

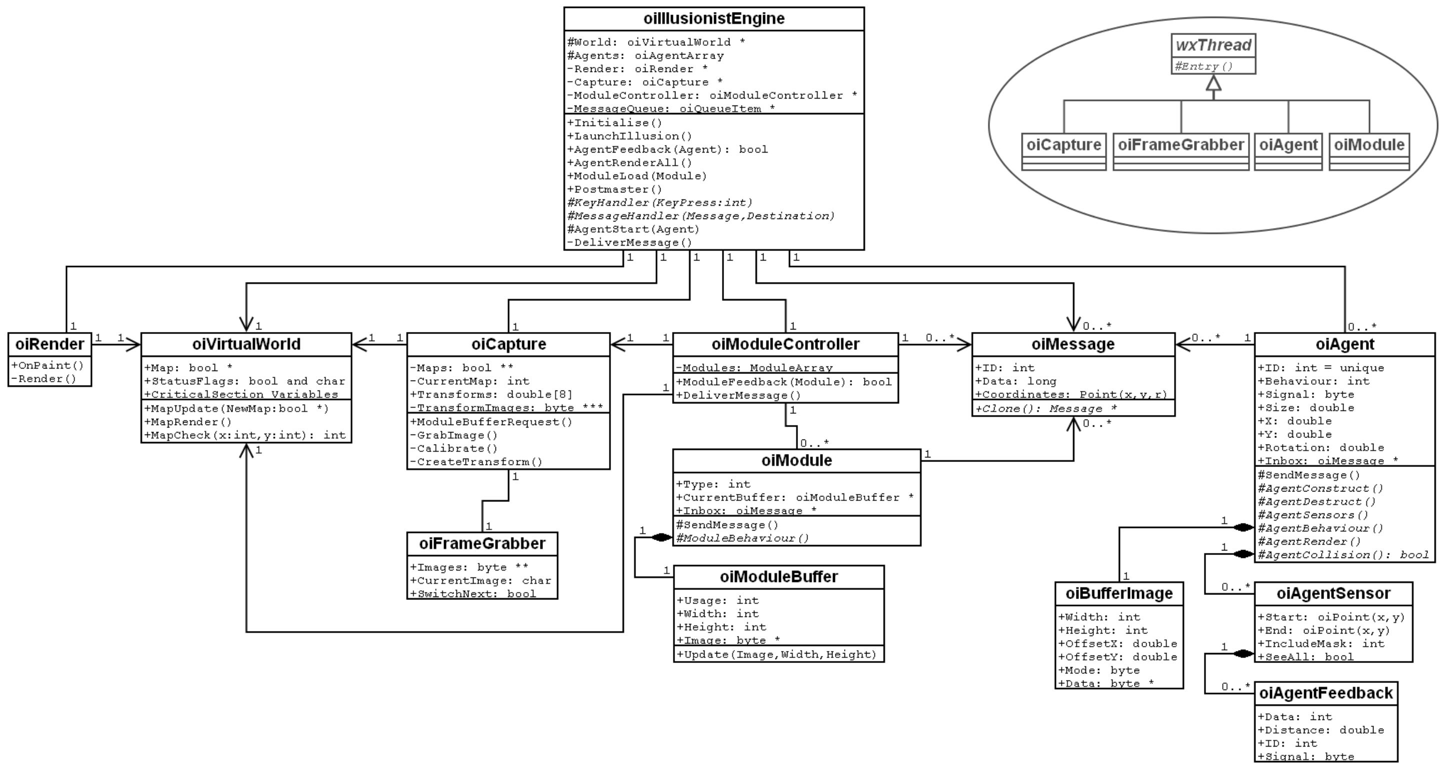


Fig. 3.1: OpenIllusionist full class diagram.

3.4.1 Major Elements of the Framework

This section provides descriptions of each major element within the framework, as seen in the class diagram.

oiVirtualWorld

The *VirtualWorld* contains most of the status flags and critical section variables and is also where the map is referenced. The map is what provides the link to the physical world; it is updated by the capture system and represents a boolean image of the world where any pixels that are “true” indicate the presence of a physical object. Since the capture system uses a fast edge detection method to generate the map it results in an image of lines that are referred to as walls.

The critical section variables are used by all of the threaded elements to ensure that safe synchronisation can be achieved. This includes critical sections for updating the agents, modifying flags and posting messages.

The status flags are used to inform the renderer what should be visible and to store the current state of the calibration process. Utility functions are also provided to allow safe access to the current map.

oiCapture

The capture thread is responsible for acquiring live video images from the frame-grabber, performing the primary image processing and updating the map. It also handles the photometric and geometric calibration of the camera, which is explained in more detail in chapter 5.

Since the copying of an image is a time consuming process the framework employs a double-buffering system. This ensures that updating the map does not unduly affect other framework elements that rely on the data. It is very fast because the map is copied into *Buffer[1]* while the rest of the system is looking at *Buffer[0]*, then within the critical section the system is told that the current map is now *Buffer[1]*. This means that the only map updating code contained within a critical section is a simple flag switch.

oiRender

This class performs all of the relevant *OpenGL* initialisation and is not threaded but instead runs as part of the main process. At regular intervals a timer triggers the “OnPaint” event, which in turn calls the internal “Render” function. The “Render” function informs the engine to render all of the agents as well as ensuring that any other elements that need to be displayed are rendered in the appropriate order.

Agents are sorted into order based on their given type, this then provides the order of precedence for rendering. Performing the rendering in this way allows particular agent types to always be rendered before others which is important for alpha-blending. For example, if a developer was producing a semi-transparent agent that is capable of moving over other agents then those other agents must be rendered first so that the transparent agent can be alpha-blended correctly.

oiMessage

The messaging system is based around a simple class that can be extended, if necessary, in a custom application. The class contains some basic useful variables such as ID and coordinate information, but additional member variables could easily be added by creating a new class that is derived from it. A “Clone” function is provided which must be overridden in any child class since it allows the system to make a copy of a message when needed.

oiModule

A module is similar to an agent in that it can send or receive messages and is threaded, however its purpose is to perform any secondary image processing that is not as time critical as the updating of the map. The framework contains core modules that can be activated by a custom application, but a developer can also create custom modules descended from this class. Currently a module informs the rest of the application of its findings via the messaging system but there are plans to extend it (see section 3.6.1).

Each module owns a buffer (*oiModuleBuffer*) in which a captured image can be stored for processing. However, if this buffer was to be updated every time the module finished processing an image and requested another then there would be a significant amount of copying going on, especially if there were a large number of modules. Therefore the framework uses a shared buffer system in which each module contributes its buffer to a pool.

If a module requests a new image to process then the system first checks whether or not this module has already processed the latest image in the buffer pool. If it has then a new image is requested from the capture device and stored in the next free buffer to which the module is then provided with a pointer; if the module has not then it is simply given a pointer to the buffer containing the latest image. Using a thread-safe counting system ensures that each buffer is correctly handled by the system and that a module will not delete itself whilst other modules are accessing its buffer.

The benefit of this method becomes apparent if you consider four modules, one of which is capable of processing images at intervals of one second and the other three which are capable of processing images at intervals of three seconds. If each module was provided with its own copy of the latest image to process then in the space of 9 seconds image copying would have occurred a total of 18 times. Using the proposed method the fastest module would dictate how much copying would be required and so only 9 images would have been copied in total in the same scenario.

oiModuleController

The *ModuleController* is the owner of all of the modules and handles the assignment of images to process (as described above) along with the delivery of messages. It liaises with the capture thread to acquire new images to process but in such a way as to not hinder the performance of the primary image processing.

oiIllusionistEngine

The engine is the heart of the framework. It owns all of the agents, provides them with feedback and is the common structure through which they are synchronised. It is responsible for initialising most of the other classes within the system and is where the global message queue is handled. A custom application must override this class and provide implementations for the "KeyHandler" and "MessageHandler" functions but the developer does not need to know about the internal workings of the base class and the synchronisation of threads.

3.4.2 Major Methods of the Framework

Agent Feedback

The "AgentFeedback" function is a call-back that each agent uses to synchronise with the world during its cycle. It consists of three main stages:

- *Sensors*
  Each sensor is scanned and the corresponding coordinates are checked depending on the configuration of the sensor. The map may be checked for physical objects and other agents may be checked for collision. The current information is updated via the *oiAgentFeedback* class which can then be processed by the agent behaviour method later in the cycle.

- *Inbox*
  If there is a message currently in the inbox then it is assumed that the agent has already processed it during the previous cycle and is therefore deleted. This prevents the message queues getting clogged if an agent does not bother to read its messages. The following section describes how the inbox is filled.

- *Buffers*
  An agent is not just restricted to viewing the world using sensors, if necessary it can request small buffer images that are relative to that agent (i.e., located and oriented with respect to the agent). This stage updates the buffers within the agent so that it can then perform localised image processing.

Messaging System

When an agent or module sends a message it is handled by the "PostMaster" function inside the engine. If the message is destined for the engine itself then the function simply calls the "MessageHandler" function (implemented in the custom application) and then deletes the message. On the other hand, if the message is destined for agents or modules, the main message queue is searched to find an empty slot (the length of the message queue is determined during initialisation of the engine). The message is inserted into the queue and a *recipients* counter is set to zero. The postmaster then checks to see if the message is intended for a module or number of modules and if so then it is passed to the module controller. If the message is intended for an agent or number of agents then the postmaster adds a reference for the queue item to each of the intended agent's message queues (FIFO implemented as a circular buffer with a head and tail index). The *recipients* counter is incremented for each agent that receives the message. The module controller delivers messages to the modules in exactly the same way.

During the feedback function an agent or module will check their queue for a message and if one exists then it clones the message and gives the reference to the agent or module via the inbox variable. It then decrements the *recipients* counter and if it was the last object to copy the message then it deletes the original and makes that slot in the main queue available again.

The messaging system uses critical sections appropriately to avoid potential conflicts and the structure of it allows for an efficient delivery system that will not have too great an adverse effect on the overall performance of the system.

The primary drawback to this method of message handling is that an agent or module is only able to process a single message each cycle.

3.5 Rapid Development

The C++ source code for a derived engine can be found in appendix B. This complements the sample agent discussed in chapter 2 and together they represent all that is needed to create a full VAE application. The comments in the source code provide more detail about how the engine works. As with the agent, very little code is actually required to produce a working engine.

3.6 Conclusions

In this chapter the requirements of a framework for a VAE are specified. The elements of the *OpenIllusionist* framework are then described showing that it meets those requirements. The examples provided in the appendices show that a fully-functional VAE can be developed with minimal code. The technical issues such as threading, image processing, calibration and message handling are dealt with in the library. This leaves a developer with only the visual and behavioural design of the application to worry about and allows for very rapid development.

3.6.1 Further Work

As mentioned in chapter 2, the point at which most thread synchronisation occurs is the "AgentFeedback" function and is therefore the point at which speed of execution is crucial. Currently the algorithm for updating the sensor information calculates each location to be checked based on the start and end coordinates. This could be improved by incorporating a calculation step in the base agent class that is performed before the feedback synchronisation occurs and stores the relevant coordinates in an array. Then, inside the critical section, the array can simply be stepped through thus removing the need for calculation. This means that the sensor coordinate calculations are executed during the agent's own time and not that of the main process. Further optimisation can then be achieved by only updating the sensor coordinate arrays when the agent has actually changed the sensor configuration or has moved.

The module system for extra image processing was a recent addition to the OpenIllusionist framework and so there are still some major modifications planned in future development of the library. The calibration system is currently integrated into the *oiCapture* class, however modules may require calibration steps of their own. Therefore, by pulling out the calibration system into a separate entity and providing it with access to the modules, it should be feasible to allow those modules to "plugin" their own calibration steps alongside the primary ones.

Another plan for the modules is to implement a map overlay system as an addition or alternative to using messaging. This would allow modules to present information to the system that can be "seen" via the agent sensors. It would work in much the same way as the wall map except that each module could also contain a map which may or may not be seen by an agent depending on the configuration of its sensors. To implement this the sensor configuration system must be changed to keep the task of setting up the sensors as clear as possible. Efficiency is a major concern since it would provide even more information that the feedback function must search through.

The experience of an augmented environment is not just limited to vision. To expand the immersive nature of a VAE it would be desirable to include audio feedback along with the visual aspect. There are plans to extend the capabilities of the framework to include an audio thread to which the engine, agents and modules alike can post messages requesting sounds to be played.

Chapter 4

Fiducials

4.1 Introduction

4.1.1 What is a fiducial?

A fiducial (also known as a marker or target) is a symbol that can be easily recognised and interpreted by a computer vision system in a real scene. Fiducials have a variety of uses such as:

- *Calibration*: Fiducials placed in known locations allow a vision application to calibrate the orientation and position of a camera relative to the scene.
- *Tracking*: An object can be marked so that it may be detected and then tracked by a vision application.
- *Interaction*: A fiducial can be used to interact with a computer system.
- *Information*: They can also be used to associate some information with a location or object

4.1.2 Examples

One of the earliest uses of the fiducial is in robot placement tasks both in automated manufacturing and research systems. For example, fiducials can be printed onto circuit boards so that robots equipped with machine vision can precisely attach components. The accuracy and reliability of this method is dependent on tightly controlled conditions such as lighting and camera resolution.

The BBC developed a tracking system for their virtual studios [67] to allow them to accurately calculate the location of a camera. It allows the computer rendered elements to always match the actors and objects in the live scene regardless of where the camera is or how much it moves during a shot. This system uses a set of concentric-ring fiducials mounted on the ceiling of the studio, the normal cameras are then fitted with auxiliary cameras pointing upwards that can track the fiducials. This allows each camera to work out exactly where it is in the studio and which way it is pointing.

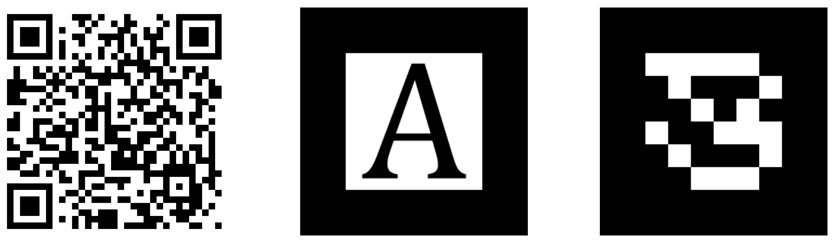
A potential use, in terms of studio production, could be motion capture where major sections of the body are labelled with fiducials and can be tracked to produce motion data for a virtual character. Another use could be the tracking of an individual throughout a building by giving them a fiducial badge and having cameras in each room. As an example, this would be useful for analysing the path that people take through a museum and which sections they spend the most time in.

A few companies are now using fiducials combined with mobile phone software to provide links to websites. One example is ShotCode [63] who provide markers which other companies can then place in their advertising thus allowing customers to connect to a website just by taking a photo of it with their phone.

4.2 Background

Bar codes are probably the most commonly used fiducials. They are reliable and although they can potentially be used to provide coarse location data they do not provide accurate information about the orientation and location of the object that they are attached to. They also normally rely on a set of laser scanners to read them and can be difficult to detect and read via a vision system with an ordinary camera. Therefore they are not really suitable for AR applications.

QR Code developed by Denso Wave [51] is a 2-dimensional bar code that can contain a reasonably large amount of data (such as a URL), with error correction, in a very small space but like normal bar codes they require a scanner or camera to be very close for processing. However they do contain enough information so that orientation could be determined if necessary. See figure 4.1a for an example *QR Code*.



(a) QR Code (b) ARToolkit (c) ARTag

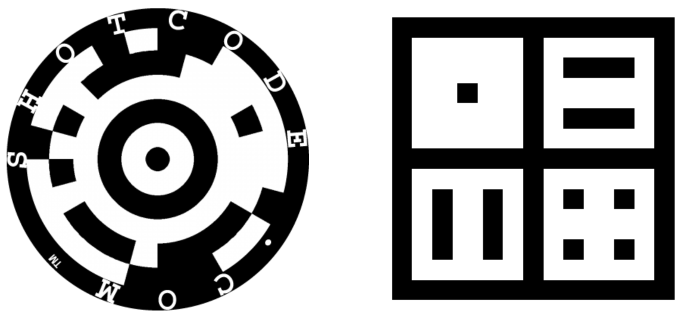
Fig. 4.1: A selection of example fiducials.

The most well known fiducial system for AR applications is probably the *ARTookit* [35] which uses square fiducials with a thick black border and containing a greyscale pattern or image (figure 4.1b). The detection algorithm works as follows:

1. Threshold the image.
2. Determine outlines (edge detection).
3. Find any outlines that can be fitted with four connected line segments.
4. The internal image is then compared with known patterns to identify the fiducial (or eliminate it).

The *ARToolkit* has been used to develop applications where virtual 3D objects are overlaid on the fiducial [36], to calibrate for shared spaces [35] and positioning systems [2] and even to provide novel input devices [75]. It is also one of the few fiducial libraries available open-source and is released under the GNU GPL.

To address the false detection and misidentification issues that can occur with *ARToolkit* an alternative has been developed named *ARTag* [19]. *ARTag* still relies on the outer black border and square shape, but internally it uses a robust binary coding system arranged in a grid (see figure 4.1c).



(a) ShotCode (b) Region Adjacency Tree

Fig. 4.2: A selection of example fiducials.

The *ShotCode* fiducials (figure 4.2a) mentioned in the examples (section 4.1.2) appear to be based on a similar system to that developed in Cambridge called Target Recognition using Image Processing (TRIP) [16]. *TRIPcodes* are circular fiducials that incorporate a central black spot and solid ring; the code, which is stored in two concentric rings around the central ring, can be broken up into sectors where each sector represents a value. There are two data rings present in every sector thereby providing two binary digits, but the combined value 11_2 is reserved to indicate the orientation of the fiducial and is referred to as the synchronisation sector. The method contains the following stages:

1. Threshold the image.

2. Edge detection.
3. Edge following and filtering.
4. Apply ellipse fitting algorithm to any potential ellipses.
5. Find concentric ellipses.
6. Decipher the code (with parity check).

Topological fiducials are different to the aforementioned types since they do not rely on geometry. Instead they rely on the hierarchical structure of regions or elements within an image, in the case of region adjacency trees [13] this corresponds to regions of black and white (refer to figure 4.2b). The method of detecting a topological fiducial is as follows:

1. Adaptive thresholding.
2. Build a region adjacency tree of the entire image.
3. Search within the tree for sub-trees corresponding to known fiducials.

This system has the advantage that a fiducial can take any form such as a square (as in figure 4.2b), a circle or even a binary picture. It is also far more resilient to fiducial warping than geometric fiducials which generally need to be printed onto a planar surface, therefore a topological fiducial could easily be printed onto clothes for example.

Johnston and Clark [34] developed a topological fiducial system that incorporates a tri-level thresholding algorithm. Areas that are definitely white or black are assigned appropriately and areas that are ambiguous are marked as such. The system will build a Region Adjacency Graph (RAG) of the entire image that includes both the known and unknown regions and then, using a simple set of rules, will merge the unknown regions with known regions. Since this is performed at the graph level and not the image level it is much faster. Once the graph has been thinned, a search is performed for a known key RAG which corresponds to a pattern that all of the fiducials contain. The identity of the fiducial can then be determined since the identification RAG is always connected to the key RAG via a single black node.

A good evaluation of coding schemes for the data within fiducials can be found in [57] and although they are not covered in this thesis they can certainly be applied to many of the fiducial designs presented here, including the proposed one.

4.3 Requirements

4.3.1 General Requirements for a Fiducial System

The design of a fiducial is greatly dependent on the requirements for the system and can affect both the way it looks and how it is processed. Some of the criteria and constraints that a system may include are as follows:

- Reliability
  - *Detection*: A true fiducial should be detected.
  - *Identification*: The identity (code) of a fiducial should be determined without error.
  - *Mis-detection*: Natural elements in a scene should not be interpreted as fiducials.
  - *Efficiency*: The detection and identification algorithms should be as fast as possible.
- Robustness
  - *Lighting conditions*: Detection should not be affected by illumination effects such as shadows.
  - *Deformation*: Detection should be possible even when the fiducial is displayed upon a deformed surface.

  - *Occlusion*: If part of the fiducial is occluded it should still be found and identified correctly.
  - *Pose*: The fiducial should be detected when seen from any angle.

- Size
  - *Compactness*: The fiducial should be as compact as possible.
  - *Information content*: The fiducial should contain as much information as possible.

- Accuracy
  - *Location in image*: The location of the fiducial should be accurately determined in the image.
  - *Location in world*: The location and orientation of the fiducial should be accurately determined in the world.

- Simplicity
  - *Ease of rendering*: The fiducial must be easy to reproduce, for example, monochrome may be preferable to colour.

4.3.2 Specific Requirements for a Video Augmented Environment

Many AR systems deal with HMDs and the use of fiducials for registration of a scene so that computer graphics can be accurately overlaid. The interest of this thesis lies with the VAE, concentrating on the requirements for such a system and therefore with specific fiducial properties. The two primary uses of a fiducial within an augmented environment are calibration and interaction.

Calibration

Geometric calibration (see chapter 5) of a projector-camera system is necessary so that interaction between the physical and virtual world is seamless.

The system must therefore be able to accurately translate between pixel locations in a captured frame and the corresponding locations in the projected image. Since calibration is usually performed during the start up phase of the application, efficiency of fiducial detection is not too important although excessive times would prolong the calibration stage. The accuracy with which the system can locate fiducials in the image is of the utmost importance since this will ultimately affect how well the transform between projector and camera performs.

The system must perform well regardless of whether fiducials are projected or printed since a calibration system could use either. Although the ability to detect fiducials that have been warped (such as those projected onto uneven surface) is desirable it is not essential since the majority of VAE systems are projected onto planar surfaces. Calibration fiducials do not need to contain much information since even if only one fiducial is used it can simply be displayed in different locations at different times.

Interaction

A fiducial system for calibration may only be run at the beginning, but a fiducial system for interaction must be running continually. Hence it must be as efficient as possible to avoid interfering with the operation of the rest of the VAE and yet also provide believable interaction through fast response times. There is more of a requirement than with calibration to be able to uniquely identify fiducials so that different ones can provide different interactions, but there is no need to have a huge number of combinations since an application with too many types of interaction would prove too complex for an end-user.

Accuracy is fairly important but not as critical as for calibration; if a fiducial is being used as a controller for one or more parameters a lack of accuracy could produce jitter or aliasing. The system must be able to detect a fiducial regardless of pose (assuming a sensible proportion of the fiducial is still visible) and if being used as a controller it would be advantageous to be able to exploit the pose if it can be measured. For example, if a fiducial was to be used as a controller much like a joystick, then accurate calculation of the pose would allow the system to extract the roll, pitch and yaw along with the location.

Overall

Both the calibration and interaction elements need a fiducial system that can function in a variety of lighting conditions since that is a requirement for the VAE as a whole, combined with the complication that a marker could be printed or projected. A system that is both fast and accurate would be ideal since only one implementation would need to be developed to cover the two elements.

4.4 Proposed Design

4.4.1 Introduction

In this section a fiducial design is proposed that is similar to that in [16] except that it has an outer boundary and the option of using ternary codes instead of binary. The system, named Elliptical Luminance Fiducial (ELF), contains a novel method of decoding the fiducials that means it is resilient to both shadowing effects and perspective distortion. The algorithms described here also include a new adaptive edge detection method which provides a potential improvement in efficiency over the Canny edge operator [11] used in the original design described in [47].

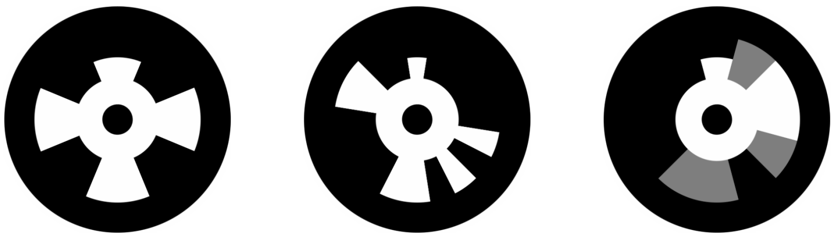

(a) Binary - 7 (b) Binary - 19 (c) Ternary - 11

Fig. 4.3: Proposed fiducial in some example configurations.

4.4.2 Ellipses

Circles were chosen for this design because of certain properties that can aid in their detection:

- A circle under perspective projection is an ellipse.
- There is a unique affine transformation between a plane circle and the same circle under perspective projection if the orientations of both are known.
- Efficient and robust ellipse detection methods already exist.

4.4.3 Segments

Examples of the design can be seen in figure 4.3 where 4.3a represents a binary configuration with 7 code segments, 4.3b represents a binary configuration with 19 code segments and 4.3c represents a ternary configuration with 11 segments. Due to optimisations the system only deals with integer angles (0–359) when scanning a fiducial and so segment sizes are chosen accordingly. Table 4.1 describes the possible configurations and combinations for the ELF.

Segments	**Binary**	**Ternary**
3	8	27
7	128	2187
11	2048	177147
17	131072	129140163
19	524288	1162261467

Table 4.1: Number of codes available in each of the possible fiducial configurations.

The number of segments available for coding is odd in each case because one segment is reserved for indicating the orientation of the fiducial. This is clearest to see in the ternary fiducial (4.3c) and consists of a segment with outer-half black and inner-half white. The orientation segment is referred to as the marker.

The use of a split segment for the marker suggests that all of the segments could be divided to hold twice as many data bits, with one particular combination reserved for the marker. However, in practice, locating a change from white to black required to detect the marker is simpler and requires less accuracy than scanning two separate rings around the entire fiducial. Scanning multiple rings would require a much greater accuracy in the ellipse parameters since they must also include compensation for perspective distortion.

4.4.4 Centre Spot

The centre spot in this design provides two functions - verification and perspective hint. After detecting the outer ellipse the system will then search for the spot to confirm whether or not it is currently dealing with a valid fiducial and thus reducing the number of false positives. Additionally, the centre of a circle under perspective is actually slightly shifted in the minor axis compared with the exact centre of the ellipse. This shift can provide a useful clue as to which direction the fiducial is actually tilted.

4.4.5 Ternary

The ternary option (black, grey and white) for this design has remained in place following the developments since [79]. Ternary has been shown to be almost as robust as binary [47] and yet provides a considerably larger number of code combinations (see table 4.1).

4.5 Implementation

The steps necessary for detecting and decoding fiducials of the proposed design are as follows:

1. Edge detection.

2. Object finding.

3. Ellipse fitting.
4. Fiducial transform calculation.
5. Fiducial verification.
6. Fiducial orientation and identification

4.5.1 Edge Detection

Earlier implementations of the system incorporated an optimised Gaussian blur and Canny edge operator (as mentioned in section 4.4). However, these were found to be the most time consuming parts of the system and so a new adaptive threshold based edge detection algorithm was adopted (developed by John Robinson [58]). This algorithm also addresses the issue of ringing edges that often occur in images captured from low-end cameras such as webcams. The ringing can cause edge detectors to find multiple edges where there is only a single edge and it also reduces the accuracy of the real edge which can affect the results in later stages of image processing.

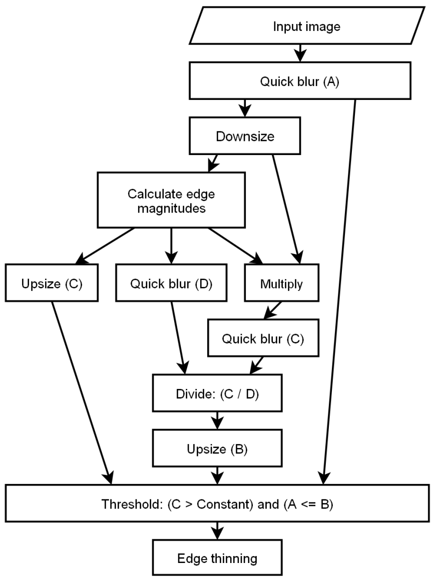


Fig. 4.4: Flow diagram of the edge detection algorithm.

Blur

During optimisation of the algorithm it was found that the edge detection performed almost as well using a simple 3×3 box filter as with using a Gaussian blur filter. Since even a heavily optimised Gaussian filter is slower than a box filter it was decided that the fiducial library should use the latter.

A 3×3 filter would usually require an extra border of pixels around the image but this means that the border must be created and extra pixels analysed. Instead, as a compromise for optimisation, the outside pixels of the image are simply copied to the new blurred image and the filter is applied to every pixel just inside the image boundaries.

Algorithm Overview

The diagram (figure 4.4) provides an overview of the steps involved in the adaptive edge detection algorithm. There are a greater number of steps in this process than there would be in an alternative method such as Canny, but since most of the steps are being performed using quarter-size images (half the width and height of the original captured image) this method still provides a significant performance increase.

4.5.2 Object Finding

Object detection is simply a case of following edge chains within the edge detected image. It is implemented using a recursive function to follow an edge in any direction. It would normally be necessary to store the nodes (x,y) of a chain in the order that they were found, especially if later processing stages rely on this order (e.g. if there was a need to calculate the smoothed gradient of the edge so that peaks and troughs could be located). However the ellipse fitting function described next is not affected by the node order and therefore the algorithm can be optimised. Ordinarily an object finder would use a reference image where it can indicate which pixels have already been processed (otherwise the recursive function would get stuck repeating pixels and cause a stack overflow) and it would add nodes to a chain along

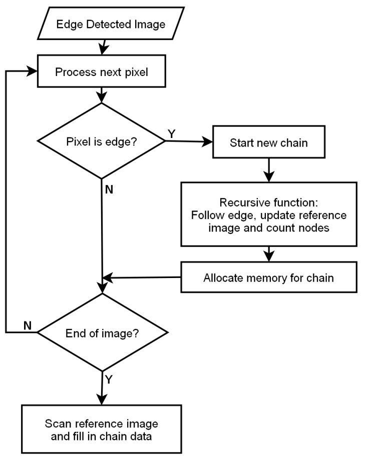

Edge Detected Image
Process next pixel
Pixel is edge?
Y
Start new chain
N
Recursive function:
Follow edge, update reference image and count nodes
Allocate memory for chain
N
End of image?
Y
Scan reference image and fill in chain data

Fig. 4.5: Flow diagram of the object finding algorithm.

the way (a linked list being the most efficient way of doing this). This means that memory is being declared each time a node is added and since memory allocation can be slow it has a negative effect on the efficiency. To take advantage of the fact that the nodes are not required to be in order, an alternative has been implemented (figure 4.5).

4.5.3 Ellipse Fitting

An ellipse can be represented in a number of ways, usually as either a polynomial or $[\ x_o \quad y_o \quad \theta \quad a \quad b\ ]$ where (x_o, y_o) are the coordinates of the ellipse origin, θ is the angle, a is the length of the major axis and b is the length of the minor axis (see figure 4.6). Ellipse fitting is concerned with finding ellipse parameters that fit the majority of nodes in an object.

When first implementing ellipse fitting a simple method was developed based around the known properties of an ellipse. The method worked as follows:

1. Find the two nodes that are furthest apart and assume that these lie on the major axis.
2. Find the centre of the major axis to determine the ellipse centre.

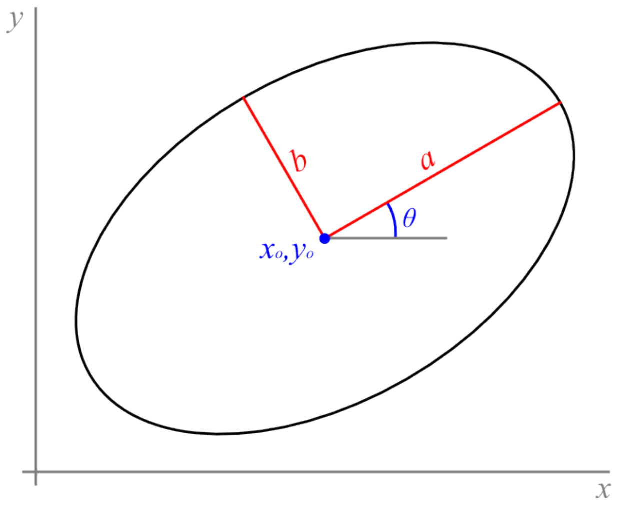


Fig. 4.6: Parameters of an ellipse.

3. Calculate a line from the centre that is perpendicular to the major axis.
4. Find the closest nodes to that line to determine the length of the minor axis.

This algorithm proved to be very unstable and a slight improvement was made by adjusting the rotation of the major axis based on distances from half-way along the axis to the outer edge. Equation (4.1) is the calculation for adjustment where θ' is the new angle for the major axis, θ is the old angle and a is length of the estimated major axis. d_1 and d_2 are the distances between the major axis and the ellipse edges on a vector that is perpendicular to the major axis and intersects it at $0.5a$.

$$\theta' = \theta + \tan^{-1}\left(\frac{d_1 - d_2}{0.5a}\right) \tag{4.1}$$

This adjustment did not provide enough of an improvement and the ellipse fitting still suffered severe errors primarily caused by a lack of sub-pixel accuracy in the edge detection combined with the fact that only a few of the nodes are actually used. A slightly more advanced method that is still based around the geometric properties of the ellipse is proposed in [77]. It processes all of the edge pixels in an image and attempts to classify them as belonging to different ellipses at the same time as producing the ellipse parameters. Although it is fairly efficient it also suffers from problems with accuracy and requires that every combination of pixel pairs are considered.

At this stage it was decided to attempt an approach based around statistical techniques. The Hough transform (originating from Hough's work in [30]) is probably the most common way to extract lines and circles. It calculates straight lines through each node at different angles and then calculates at what angle and distance a perpendicular line would go through the origin. The data for all nodes is overlaid and the point at which the majority meet provides an estimate of a straight line through the nodes. This can be expanded by including circle radius as a third parameter to allow the estimation of circle or arc segment parameters. The main problem with the Hough transform is that it is rather slow and even with an optimisation where the edge direction provides a starting point for the angle it is too inefficient for real-time fiducial detection.

A probabilistic method [70] that incorporates elements of Random Sample Consensus (RANSAC) [24] is based around tracking of ellipses. At the time it was developed real-time robust ellipse detection was not easy to achieve on commodity hardware and so this method presented a unique technique that used tracking lines and was aimed specifically at ellipses. These lines were arranged around an ellipse (already detected using conventional methods) at perpendicular angles to the gradient of the point at which they were placed. In each frame that followed, these lines were used to find potential edges and then a RANSAC like voting method was employed to choose which edge points should be used to update the ellipse parameters.

Other regression techniques such as Least Mean Squares (LMS) can provide a much faster solution to the problem of calculating ellipse parameters. The method described in [37] uses a fast line extraction algorithm to find arc segments as short straight lines and then a least squares ellipse fitting algorithm is applied to extract the ellipse parameters.

Direct Least Squares Ellipse Fitting

The chosen technique for ellipse fitting was developed at the University of Edinburgh by Fitzgibbon et al. [25, 49] and is a direct least squares method. Their research shows that a direct solution can be achieved without the need for an iterative method. This solution has been found to be stable, efficient and provides very accurate results. In their workings they show that ellipse fitting can be solved by minimisation of (4.2) with an imposed constraint (4.3).

$$E = \|Da\|^2 \tag{4.2}$$

$$a^T C a = 1 \tag{4.3}$$

where D is the design matrix (see (4.4)), a are the parameters of a polynomial that describes the ellipse ($a = [\; a \;\; b \;\; c \;\; d \;\; e \;\; f \;]^T$), C is the constraint matrix defined as (4.5) and n is the number of data points to be processed.

$$D = \begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^2 & x_ny_n & y_n^2 & x_n & y_n & 1 \end{bmatrix} \tag{4.4}$$

$$C = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.5}$$

This leads to the simultaneous equations (4.6) of which the solution to the eigensystem produces a single positive eigenvalue. The corresponding eigenvector then represents the parameters of an elliptical polynomial. Whereas other methods of ellipse fitting can be destabilised by noisy data, the unique solution provided by this method has a low eccentricity bias which will always produce an ellipse and therefore remain stable.

$$\begin{aligned} Sa &= \lambda Ca \\ a^T Ca &= 1 \end{aligned} \tag{4.6}$$

where S is the scatter matrix ($S = D^T D$).

C++ Realisation

Fitzgibbon et al. provide an implementation of the ellipse fit algorithm using 6 lines of Matlab code [25]. However, the OpenIllusionist framework is written in C++ and so an implementation in this language was required. The following list outlines the algorithm used in the fiducial library:

1. Construct the design matrix D.
2. Calculate the scatter matrix $S = D^T D$.
3. Apply the constraint matrix C.
   (a) Find the Cholesky decomposition of the scatter matrix (since Cholesky is twice as fast as standard LU decomposition for solving linear equations and is far simpler than finding the inverse of the scatter matrix) L where $S = LL^T$.
   (b) Calculate the inverse Cholesky L^{-1} which provides the inverse of the scatter matrix as $S^{-1} = (L^{-1})^T L^{-1}$.
   (c) Multiply the inverse Cholesky by the constraint matrix by the transpose of the inverse Cholesky $M = L^{-1}C(L^{-1})^T$ which is equivalent to $M = S^{-1}C$.
4. Initialise the eigenvector and eigenvalue matrices E_{vec} and E_{val}.
5. Solve the generalised eigensystem using a Jacobi transformation.

6. Multiply the transpose of the inverse Cholesky with the eigenvector matrix $E'_{vec} = (L^{-1})^T E_{vec}$.

7. The method described here results in negated eigenvalues so determine the only negative eigenvalue and extract the corresponding eigenvector. This vector (a) corresponds to the solution of the polynomial and the ellipse parameters can be calculated directly from it.

Descriptions of the Jacobi transformation and the various decomposition methods can be found in [50].

Final Checks

Since the direct least squares method always returns an ellipse solution the system must then confirm whether or not the object really is an ellipse and whether its parameters are sensible. First of all the origin of the ellipse is checked to ensure it is actually within the boundaries of the image. Then the major axis is checked to make sure it is not larger than the image.

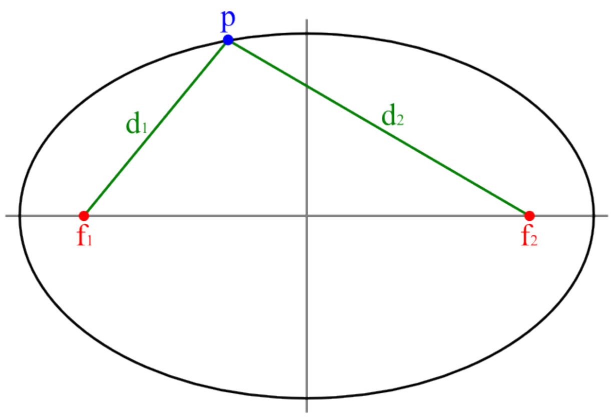


Fig. 4.7: Focus points of an ellipse.

Finally the fitness of the ellipse is evaluated by exploiting a property of the elliptical focus points. These foci are found along the major axis at a distance (f) from the origin given by (4.7). If the distance from any point of the perimeter to one focus point (d_1) is summed with the distance to the other focus point (d_2) then it is equal to twice the length of the major axis as in (4.8). The diagram in figure 4.7 is an example where d_1 is the distance between a node on the ellipse p and focus point f_1 and d_2 is the distance between p and focus point f_2.

$$f = \sqrt{a^2 - b^2} \tag{4.7}$$

$$d_1 + d_2 = 2a \tag{4.8}$$

Scanning through all of the nodes in an object, tallying those that satisfy the equation (4.8) within a threshold and then dividing the tally by the total number of nodes results in a fitness value between 0 and 1. Any object in which more than 95% of the nodes fit the ellipse parameters (i.e., a fitness value > 0.95) is accepted as an ellipse and passed through to the next stage of fiducial detection.

4.5.4 Fiducial Transform Calculation

To allow all of the following stages to treat each fiducial in the same way, regardless of their orientation in the image, a transform is required. This will provide a conversion between fiducial space (circular) and image space (elliptical). Since an elliptical space can be described by an affine transformation (as mentioned in section 4.4.2) the calculation of a transform matrix is trivial.

$$A = \begin{bmatrix} a\cos\theta & -b\sin\theta \\ a\sin\theta & b\cos\theta \end{bmatrix} \tag{4.9}$$

$$A^{-1} = \begin{bmatrix} \frac{\cos\theta}{a} & \frac{\sin\theta}{a} \\ \frac{-\sin\theta}{b} & \frac{\cos\theta}{b} \end{bmatrix} \tag{4.10}$$

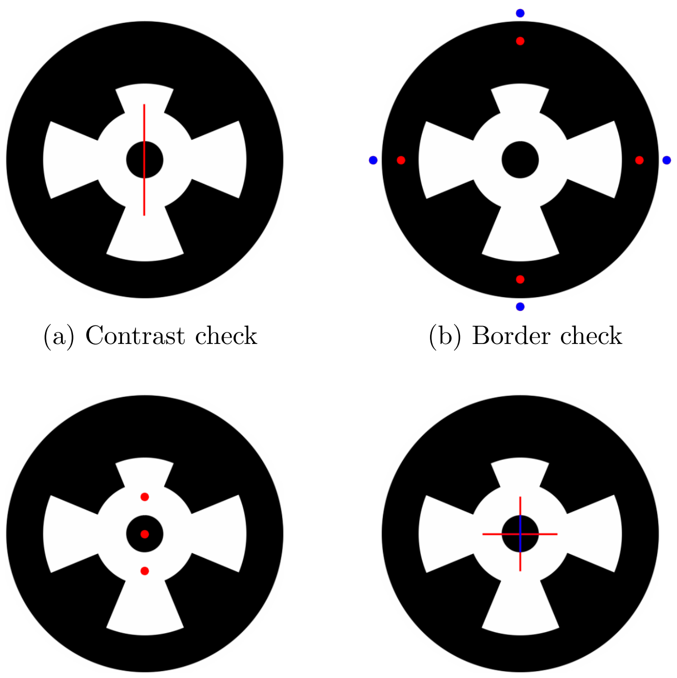
The equation (4.9) describes the affine transformation A where a is the length of the major axis, b is the length of the minor axis and θ is the angle of the ellipse relative to the image axes. The second matrix (A^{-1}) provides the inverse of the transformation and although not needed for fiducial detection it can be used by an application to apply 2D or even 3D overlays.

4.5.5 Fiducial Verification

Fiducial verification is broken up into a small set of tests and corrections. They not only verify that the object currently being analysed is a fiducial but they also help correct for slight errors due to perspective. The centre of the ellipse can be found very accurately, but as described in section 4.4.4 the centre of a fiducial at an acute angle does not necessarily lie precisely in the elliptical centre. This can lead to errors in the fiducial identification stages that follow. The centre spot allows the system to correct for the perspective shift and therefore improve the accuracy of later stages. The verification stage consists of the following elements:

1. Contrast check.
2. Confirmation of the fiducial border (inside and out).
3. Centre spot check.
   (a) Check for an offset.
   (b) Correct the temporary centre coordinates.
4. Centre spot tally.
   (a) Scan centre spot.
   (b) Correct fiducial centre and calculate perspective direction.

The example images in figure 4.8 are orthogonal to the camera and are therefore perfectly circular. For the purpose of illustration it is assumed that the elliptical minor axis is vertical and the major axis is horizontal.



(a) Contrast check (b) Border check

(c) Centre spot check (d) Centre spot tally

Fig. 4.8: Images of fiducials indicating how they are processed at each stage.

Contrast

With the proposed fiducial design there will be both black and white regions found towards the centre of the ellipse. This first step simply scans a line through the centre as indicated in figure 4.8a by the red line. Maximum and minimum grey levels are recorded and if the difference between them is lower than a threshold then the fiducial is rejected since it is deemed to have too low a contrast level for accurate identification.

Border

The next step confirms that the fiducial has a black border surrounded by white. Figure 4.8b shows the points at which the fiducial is tested along the major and minor axes. The points represented by the red dots are measured exactly in the centre of the fiducial border and are expected to be black. The points represented by the blue dots are measured at a fixed distance outside of the fiducial and are expected to be white.

Centre Spot Check

A check, as illustrated in figure 4.8c, indicates whether or not the centre spot is actually located close to the origin of the ellipse. This simply tests whether the very centre is black and either side is white. If this fails then it could mean one of two things: either the ellipse being processed is not a fiducial or the shift due to perspective is significantly large. The check is performed along the minor axis since that will always be the direction of tilt and therefore the axis in which perspective shift will occur.

In case a failure is due to shift the algorithm then attempts to compensate for it. It scans up and down the minor axes, alternating direction with each step to avoid bias. At each step it performs the check of the same three points and if it finds the correct combination of white and black then it stores the current origin coordinates temporarily. If the correct combination is not found then the ellipse is rejected.

Centre Spot Tally

The temporary origin coordinates are now used to provide a more accurate centre location using a tally. A line is scanned (as in figure 4.8d) along the minor axis and a tally of the consecutive white-black-white regions is stored. The same scan is also performed in the major axis except a tally is not taken. If there are more than two transitions in either scan then the ellipse is rejected. The two white tallies from the minor axis then provide the means with which to calculate a more accurate fiducial centre using the following equations:

$$d = \frac{(t_1 - t_2)}{2} \tag{4.11}$$

$$\begin{bmatrix} x_o \\ y_o \end{bmatrix} = A[\ 0 \quad d\ ] + \begin{bmatrix} x \\ y \end{bmatrix} \tag{4.12}$$

In (4.11), d is the shift distance, t_1 is the lower white tally and t_2 is the upper white tally. In (4.12), (x_o, y_o) are the corrected origin coordinates, A is the affine transform matrix (calculated in section 4.5.4) and (x, y) are the temporary origin coordinates.

The reliable extraction of ellipse parameters should ensure that scanned lines will always be appropriately placed. Although analysing areas may provide slightly more accurate information, the scanning of a single line will always be faster.

4.5.6 Fiducial Orientation and Identification

The final stage of detection is concerned with scanning the segments within the fiducial and determining both its orientation and identity. To improve upon the efficiency of the steps within this section the algorithms described use an integer step size of 1° combined with sine and cosine lookup tables. The following list describes the steps involved in reading a fiducial:

1. Grey Ranges - Measure the grey range at each angle.

2. Gather Data - Construct an array containing the information at each angle.
3. Collate Data - Classify and tally data.
4. Merging
    (a) Remove boundaries.
    (b) Merge consecutive segments.
    (c) Merge ends if necessary.
5. Marker - Locate the marker segment.
6. Fit Data - Attempt to fit the tallied data (multiple pass).
7. Identify - Calculate the identity of the fiducial.

Each of the steps, described in the following sections, must be able to read pixel values from the original image. Therefore the transform described earlier combined with simple linear interpolation provides sub-pixel sampling. Every point within the fiducial is represented as polar coordinates (angle θ and radius r) which is converted to cartesian coordinates (x_f, y_f). The ellipse transform is then applied to give (x', y') where (x_o, y_o) is the centre of the fiducial.

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = \begin{bmatrix} r\sin\theta \\ r\cos\theta \end{bmatrix} \tag{4.13}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A \begin{bmatrix} x_f \\ yf \end{bmatrix} + \begin{bmatrix} x_o \\ y_o \end{bmatrix} \tag{4.14}$$

Grey Ranges

Calculating the minimum and maximum grey levels for every angle that is to be scanned around the fiducial provides the means with which to adapt to local light and shadow effects within the image. To this end, the first step is

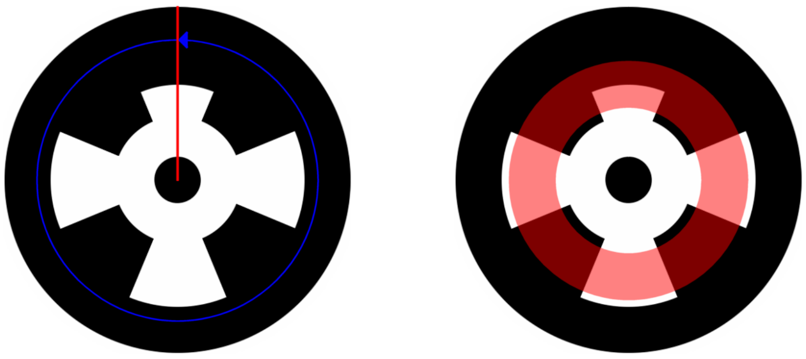

(a) Grey Scan (b) Data Scan

Fig. 4.9: Images of fiducials indicating how they are scanned.

to gather the levels as illustrated in figure 4.9a where the red line represents a single angle to scan and the blue line indicates the direction of circular scanning.

Gather Data

Using the local minimum and maximum grey levels to scale values, data is scanned from around the fiducial and stored in an array. Each scan line starts and ends inside the data segment with a buffer region each side as shown by the red region in figure 4.9b. The buffer helps to prevent slight inaccuracies due to perspective from causing erroneous readings.

During a single scan line the algorithm calculates the average value (after scaling) and also monitors any transitions. If a single transition is found then the array item is tagged with the value -1 else the average value for the scan line is used.

Collate Data

Each value (v) in the array is then classified (c) as described in equation (4.15) for binary fiducials and equation (4.16) for ternary fiducials. If $v = -1$ the item is classified as a marker. LOW, MID and $HIGH$ are the classification thresholds (systematic preliminary tests have shown that suitable values are 64, 128 and 196 respectively).

$$c_b = \begin{cases} 0 & \text{, if } v < MID \\ 1 & \text{, if } v \geq MID \end{cases} \tag{4.15}$$

$$c_t = \begin{cases} 0 & \text{, if } v < LOW \\ 1 & \text{, if } LOW \leq v \leq HIGH \\ 2 & \text{, if } v > HIGH \end{cases} \tag{4.16}$$

The data is tallied into groups and the system creates a new group each time there is a change in the data. The boundary flag is determined at each change and is dependent on the size of the group. For each group it stores:

- A tally of the number of items.
- The angle at which the section started.
- The value of the section (0, 1 or 2).
- Whether or not the section is a marker.
- Whether or not the section is a boundary.

Merging

Data merging has three sections. First of all boundary removal simply removes any groups that are small enough to have been classed as boundaries. The removal of boundaries could mean that two or more groups of the same value then become adjacent and so the second section involves combining these into larger groups. Finally the algorithm performs a check of the two items at each end of the array. Since the previous steps involved scanning a full circle there is a good chance that the scan will have started in the middle of a segment and so the items at each end of the array will be the same; if this is the case then they are merged.

Marker

Determining the direction of the marker is simply a case of finding the appropriate group. All of the groups are processed and any that are flagged as markers are checked that they are a reasonable size and therefore not an anomaly. If only one marker is found then this is a valid fiducial else it is rejected. Since the starting angle (s) and the tally (t) for the group are known, the marker angle (ϕ radians) is simply calculated as in equation (4.17).

$$\phi = \frac{\pi(s + \frac{t}{2})}{180} \tag{4.17}$$

Fit Data

This is the most important step in identifying the fiducial and involves a multi-pass attempt (if necessary) to fit the data to the expected fiducial type. In the first attempt a segment size threshold (S_T) is set which is adjusted on subsequent attempts. Each group is then processed in sequence starting with the first one following the marker segment, looping around and ending with the one before the marker. The fiducial type dictates how many segments there are and the size of a segment (S_S). Each group size (g) could potentially represent a single or multiple data segments and so it is divided by the segment size to provide an integer estimate of the number of segments (n). However, inaccuracies in the angle stepping due to perspective distortion can result in disproportionately sized groups and so the remainder (r) of the aforementioned division is then compared against the threshold and if it is greater then the number of segments is incremented by one, refer to equation (4.18).

$$n = \lfloor\frac{g}{S_S}\rfloor + \begin{cases} 0 & \text{, if } r \leq S_T \\ 1 & \text{, if } r > S_T \end{cases} \tag{4.18}$$

When the estimated number of segments has been found the corresponding number of digits are added to the code string with the value for that group. If, after processing all of the groups, the number of digits is found to correspond to the expected number of segments for that fiducial type then the algorithm proceeds to the final stage. If not then the segment size threshold (S_T) is adjusted and another iteration is attempted. The threshold is incremented if the string length was too high and decremented if too low. If the number of attempts reaches a specified limit then the fiducial is rejected.

Identify

The final step simply involves converting the binary or ternary string of numbers into a single identification number.

4.6 Optimisation

A simulation over a range of typical fiducials was performed to optimise the design and implementation of the proposed method. A test platform was developed which can vary some of the parameters involved in drawing and detecting the fiducial. All parameter values are normalised such that a radius of 1.0 corresponds to the outside edge of the fiducial.

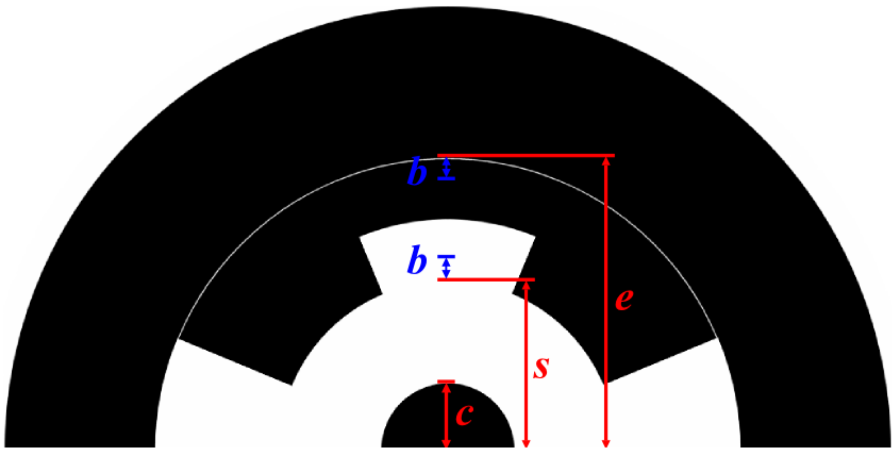


Fig. 4.10: Fiducial parameter dimensions.

The diagram in figure 4.10 depicts the main parameters of interest which are to be optimised. The buffer size (b) has an effect on most elements of the fiducial and is therefore the first parameter to be tested. It represents the gap between the radius at which an element is drawn (such as a data segment) and the radius at which it is scanned. The other parameters consist of the segment start (s), segment end (e) and the centre spot radius (c).

4.6.1 Test Platform

The test platform consists of a renderer capable of drawing fiducials at any orientation and scale, the fiducial library and two threads to test combinations of parameters. This use of two threads allows the platform to take advantage of multiple processors if available.

Renderer

The renderer utilises the fiducial library to draw a particular fiducial into a bitmap to be used as a texture in OpenGL. This texture is mapped onto a single polygon which is then rendered to an off-screen buffer at the requested angle and scale. It generates images such as those in figure 4.11.

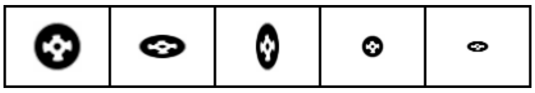


Fig. 4.11: Internal rendering of fiducials for optimisation tests.

Test Threads

Each thread is presented with a different range of fiducial types to deal with. It loops through parameter combinations, fiducial types, codes, angles and scales requesting images from the renderer. It processes each image with the fiducial library and then checks whether it succeeded in detecting and identifying the current fiducial. The thread tallies the successes and failures, writing them to file as it goes (one line for each parameter combination).

Parameter Ranges and Combinations

The following table (4.2) provides the ranges over which the various parameters are to be tested. The ranges were selected based on the requirements of the proposed method and to also yield as much variation as is possible and sensible.

Parameter		**Range**	**Step Size**
Buffer Size	(b)	0.01 – 0.10	0.01
Segment Start	(s)	0.20 – 0.80	0.01
Segment End	(e)	0.60 – 0.95	0.01
Centre Spot Radius	(c)	0.05 – 0.40	0.01

Table 4.2: Parameter ranges for testing.

Some combinations of parameters over the ranges described can be ignored, for example the segment start cannot be greater than the segment end ($s < e$ must hold true). The test platform automatically detects invalid combinations and skips over them.

The remaining variables are concerned with the inner iterative loops and provide multiple tests for each combination of parameters. The fiducial types and number of possible codes were listed in table 4.1. The angles used are $-60°$ to $60°$ at increments of $30°$ in both the x-axis and the y-axis (as seen in figure 4.11). Most of the parameter tests involve little variation in the scale except for those dealing specifically with the centre spot radius (see section 4.6.4 for more details). This is due to the centre spot being the smallest feature of the fiducial and is therefore likely to be the first element affected by distance.

Method

Although an optimisation via a method such as gradient descent could provide a much faster result, the following sections perform an exhaustive search instead. The reasoning behind this is apparent from the following graphs in which some parameter changes can cause sudden shifts or discontinuities thus producing an optimisation surface that is not suitable for a descent method.

4.6.2 Buffer Size

The sheer magnitude of parameter combinations and other variables would result in a prohibitively long test and so for the buffer size estimation a subset of the combinations was selected. The full variation of parameters as described in table 4.2 were used, but only a single fiducial type and code were included.

Each line in the results, shown in figure 4.12, represent the performance against buffer size for a fixed set of the remaining parameters (centre spot, segment start and end). Although rather variable in shape there is a point at which every line reaches 100%. This point corresponds to a buffer size of 0.05 and so this value is used in all further optimisation tests.

The next graph (figure 4.13) presents the reasons for detection failure at each buffer size, averaged over the remaining parameters. It can clearly be seen that the dominant effect of buffer size is on the ability to successfully check outside of the border. This could be due to the point of measurement being too close to the edge of the fiducial and therefore detecting low values instead of high when the edge is out of focus.

4.6.3 Segment Start and End

With the buffer size fixed, the segment start and end were considered next. The tests were performed over the full ranges described in section 4.6.1 and the results analysis first concentrated on the segment end. The segment end is important for the detection of the fiducial perimeter since a narrow border could result in the outer edge not being visible at greater distances or smaller scales. It also has an effect on the data segment size and so the results are

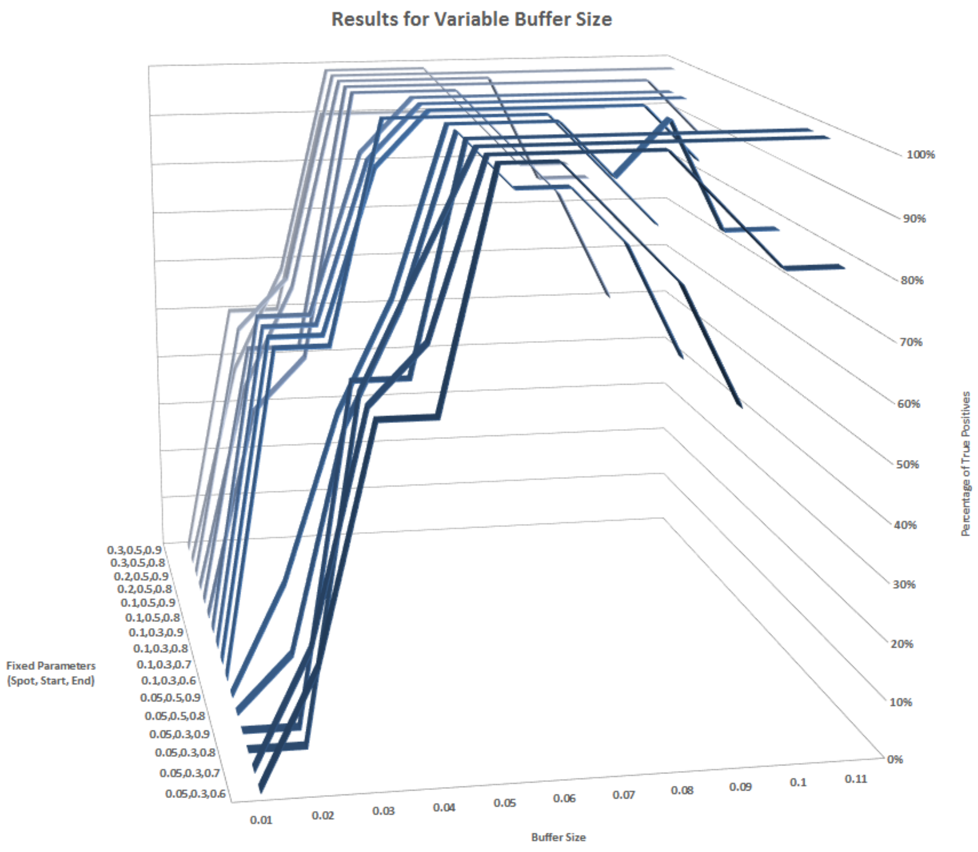


Fig. 4.12: Graph of buffer size results.

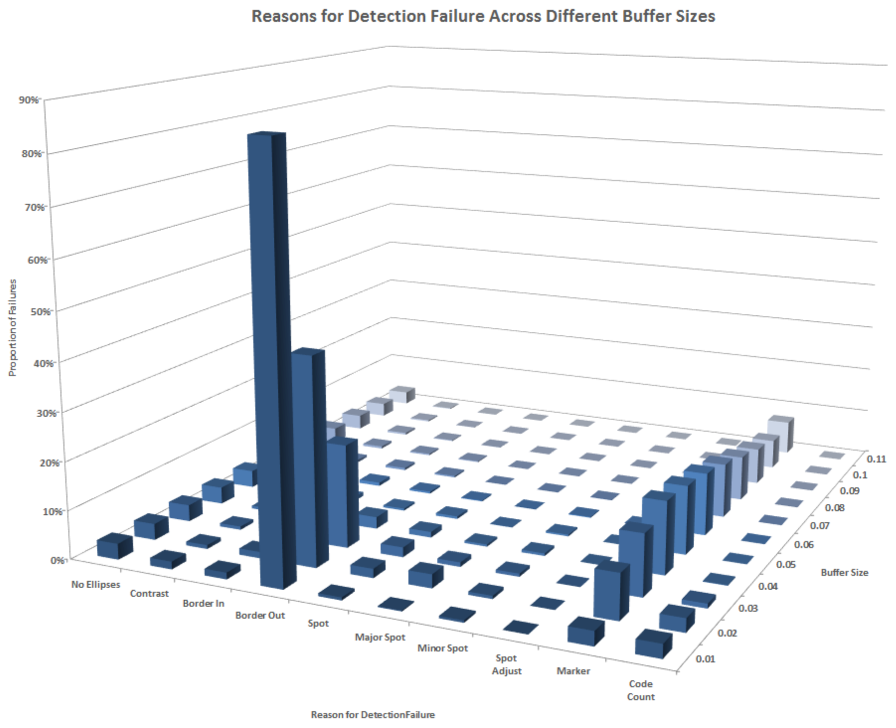


Fig. 4.13: Reasons for failure.

presented as in figure 4.14. Each line in the graph represents the performance against segment end for a fixed centre spot radius and segment start.

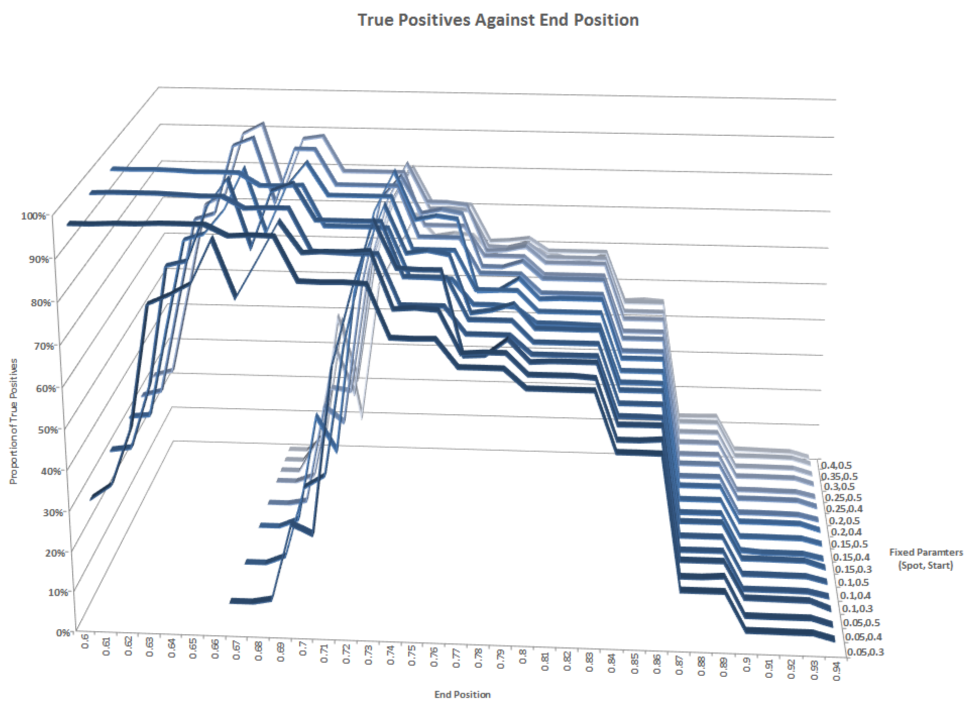


Fig. 4.14: Graph of segment end results.

A common peak amongst those lines which actually start at a value of 0.6 is that at a segment end radius of 0.66. When compared with the graph of figure 4.15 this point corresponds to a combined dip in the failures at marker detection and the low end of the failures due to ellipse and contrast detection.

The segment start radius is considered next. As with the previous graphs, each line in figure 4.16 represents the performance of this parameter against other fixed parameters (centre spot and segment end).

The results show a distinct stepping although this is actually directly related to the segment end value. With a lower segment end the results for the

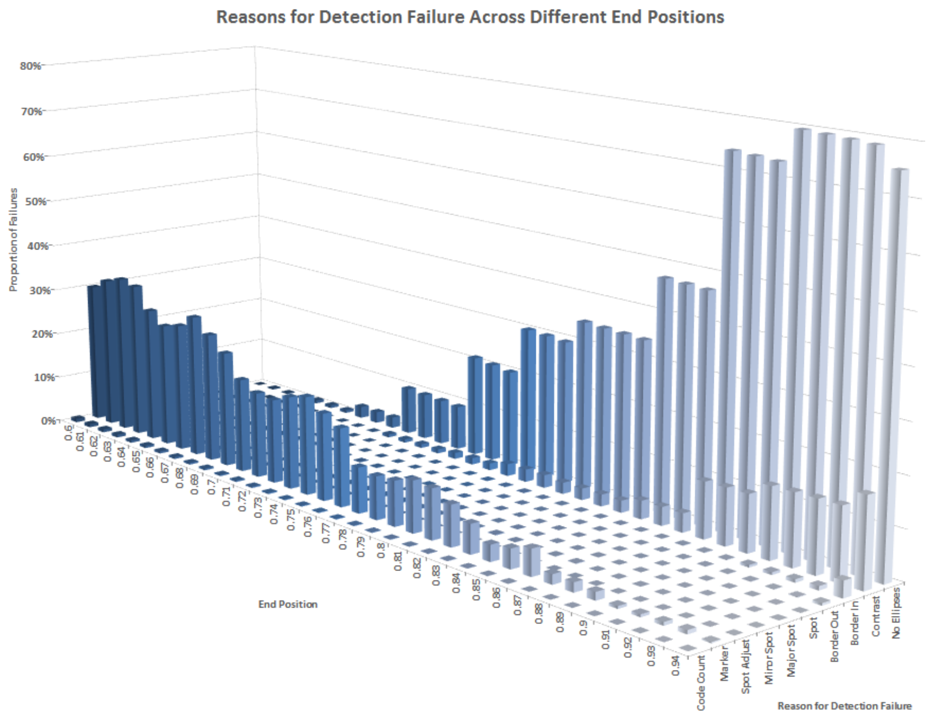


Fig. 4.15: Reasons for failure.

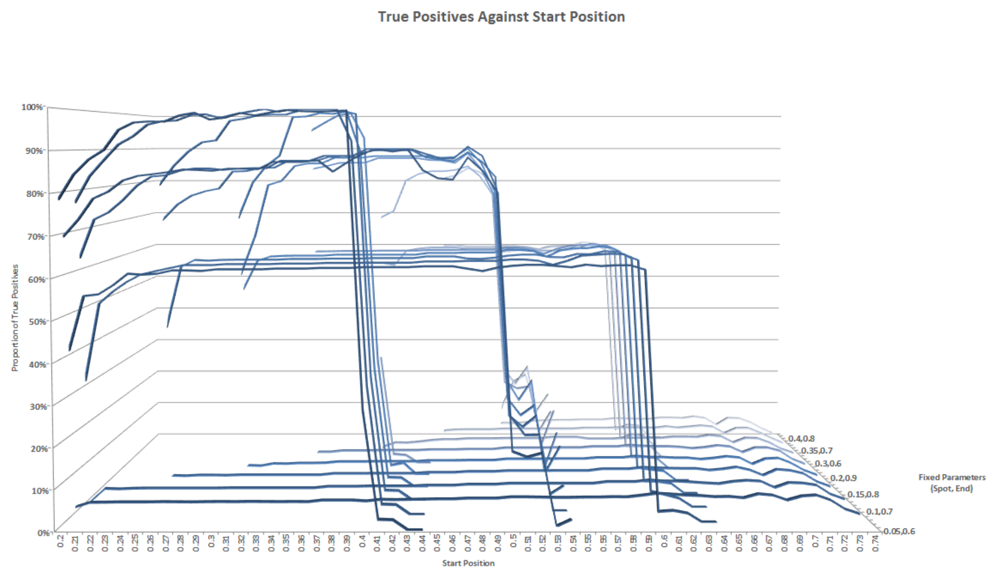


Fig. 4.16: Graph of segment start results.

segment start are confined, due to the restrictions described in section 4.6.1, to even lower values. This causes a drop-off in performance as the segment start comes into close proximity with the segment end. Since the highest step corresponds to the lower values for segment end this corroborates the results seen in figure 4.14. The value of 0.38 provides the highest result across all of the lines in the topmost step.

The reasons for failure (figure 4.17) across all of the tested parameters and variables are as expected. Ellipse detection remains fairly consistent up until the point where the segment start position reduces the possibilities for the segment end position. The marker detection failures increase, again due to the segment start and end points getting closer together, although the drop towards the end is purely related to the number of parameter combinations.

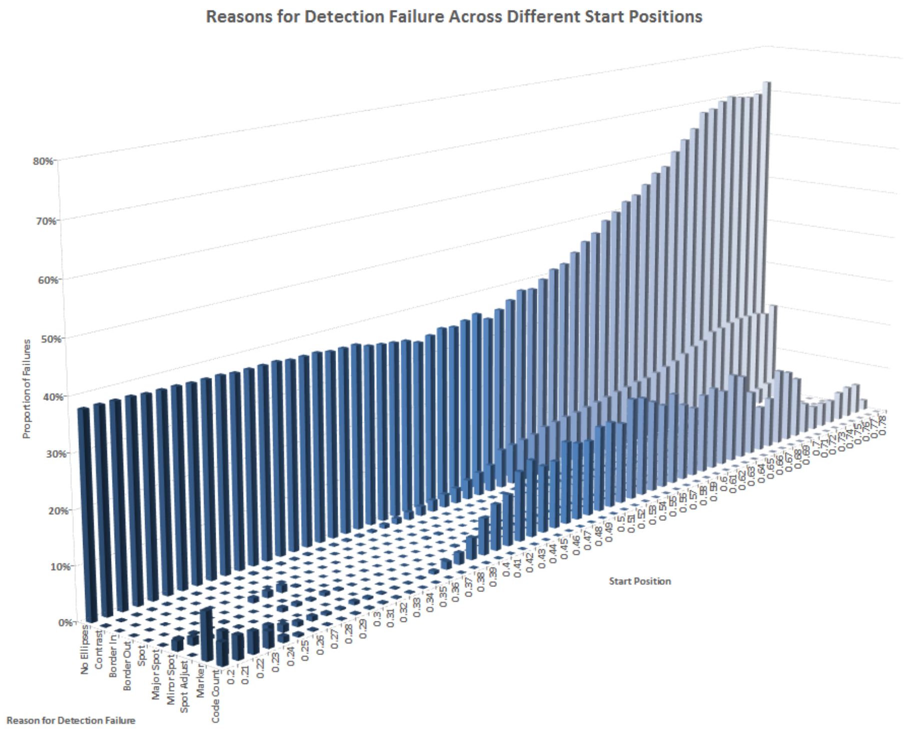


Fig. 4.17: Reasons for failure.

4.6.4 Centre Spot Radius

Next, with all other parameters fixed, the centre spot radius is optimised. Since the only parameter now being adjusted is the centre spot the effects of the scale variable can be fully tested. A difference in scale of up to 0.3 in increments of 0.02 are applied via the rendering system alongside the same fiducial types, codes and angles used in previous tests.

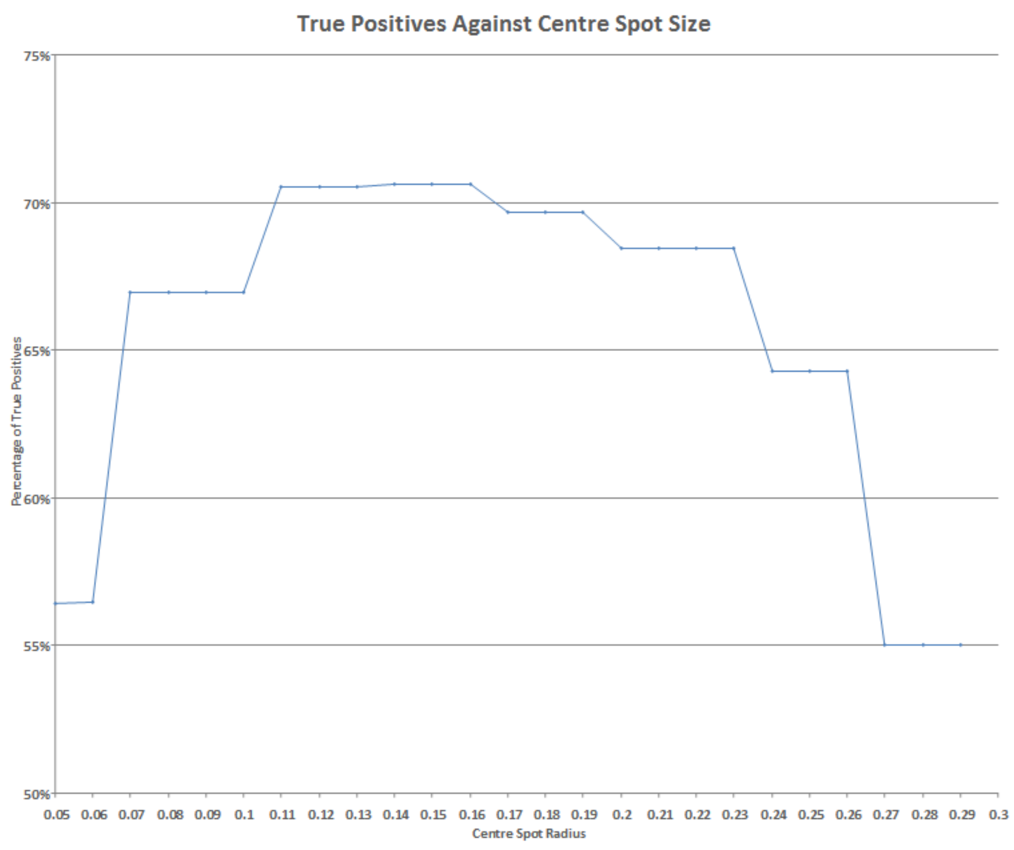


Fig. 4.18: Graph of centre spot results.

The graph for the centre spot results (figure 4.18) is much simpler than the others because it represents the last parameter to be varied. It shows a clear peak around the value of 0.15 and in the failure graph (figure 4.19) most elements remain fairly constant.

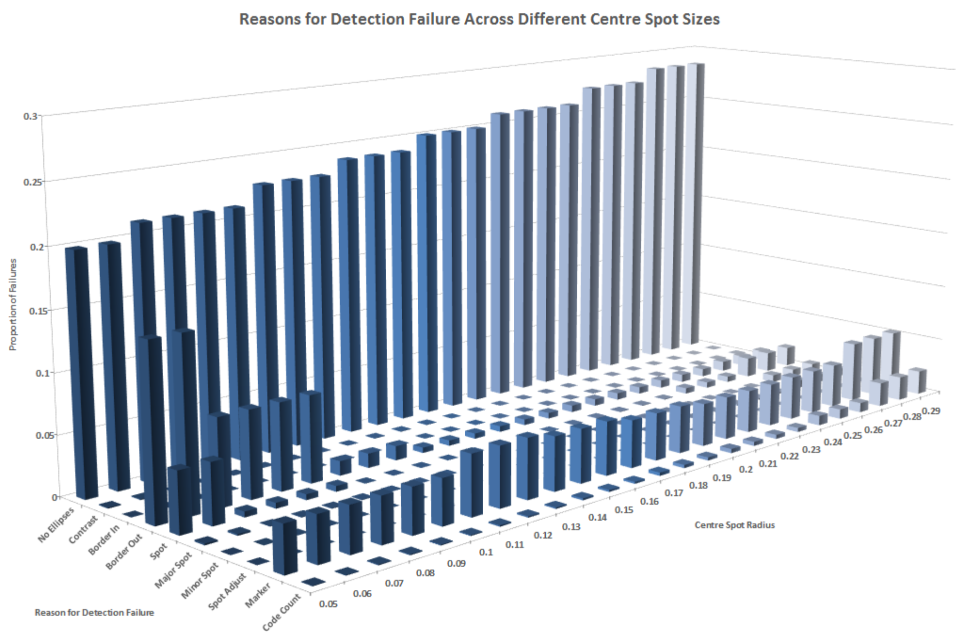


Fig. 4.19: Reasons for failure.

4.6.5 Local Optimisation

Finally the segment start, end and centre spot are all varied locally around their optimised values (±0.01 at increments of 0.002). It was found that the centre spot radius had little effect on the overall results and so was left at a fixed value of 0.15. The segment start and end are then plotted as a 3D surface (figure 4.20).

The 3D surface contains distinct peaks and troughs running in channels diagonally through the segment start and end axes. The point at which the largest average peak area occurs is at a start value of 0.378 and an end value of 0.664. To recap, the optimised fiducial parameters to be used in all further tests can be found in table 4.3. The resulting fiducial is shown in figure 4.21.

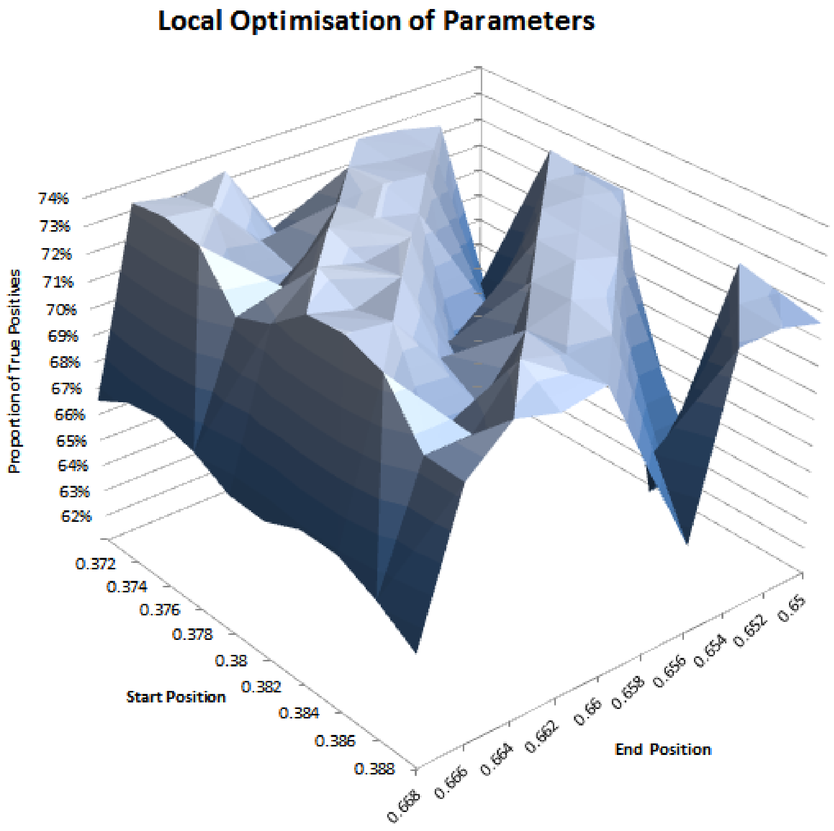


Fig. 4.20: Local optimisation of parameters.

Parameter		**Value**
Buffer Size	(b)	0.05
Centre Spot Radius	(c)	0.15
Segment Start	(s)	0.378
Segment End	(e)	0.664

Table 4.3: Optimised parameter values.

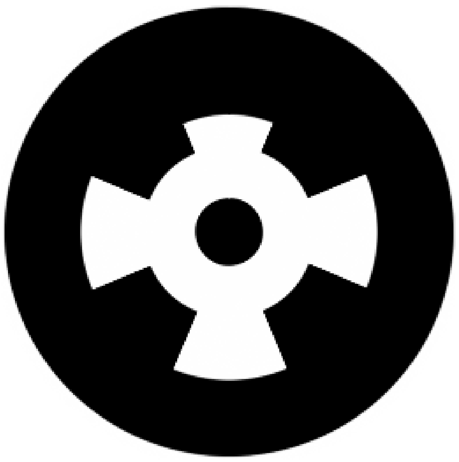


Fig. 4.21: The optimised fiducial design.

4.7 Testing

The following sections describe a number of tests performed using the optimised fiducial design. The results are then presented in section 4.8.

4.7.1 False Positives

The ability of the ELF algorithms to reject non-fiducials is important since false positives could cause calibration mechanisms to fail or user interaction could become confusing. To this end, the algorithms can be tested by applying them to process a large number of images from a variety of sources and counting the number of fiducials found. Since the proposed fiducial design is circular, including source footage that contains ellipses will allow the rejection capabilities at later stages of detection to be tested.

4.7.2 Size of Fiducial

A camera within a VAE could be placed a large distance from the surface that is to be augmented. If the system requires fiducials of a large scale in the image this could lead to prohibitively large fiducials, especially for user interaction. Therefore the fiducial detection algorithms must be able to cope with fiducials of a reasonably small size. This can be tested using the framework developed for optimisation through the rendering of increasingly distant (or decreasingly scaled) fiducials over the ranges of codes and angles described in section 4.6.1.

Since testing in this way is artificial, actual captured images of small fiducials are also tested in the comparison section (4.7.3).

4.7.3 Comparison

The *ARToolkit* was selected for comparison since it is probably the most widely used fiducial library to date and the C++ source code is freely available to download.

A simple test framework was developed consisting of two applications. The first simply captures frames from a USB camera and saves them as bitmaps. The second application utilises both the *ARToolkit* library and the ELF library to process a sequence of bitmaps. The output consists of text files of results and copies of the processed bitmaps in which any detected fiducials are indicated.

A test sheet was produced containing both ELF and *ARToolkit* fiducials (see figure 4.22). Capturing the test data using a single sheet means that both fiducial types are analysed under the same lighting and camera conditions. The two *ARToolkit* fiducials used are labelled “Hiro” and “Kanji” and are standard patterns that are included with the library. The ELF fiducials are 7-segment ternary with codes 0, 42, 588 and 2186.

The *ARToolkit* has a potential advantage here in that it is only concerned with matching the two aforementioned patterns. The ELF system, however, works by reading the code and so there are 2187 combinations that it could identify in this particular case.

Groups of images were then captured under the following conditions:

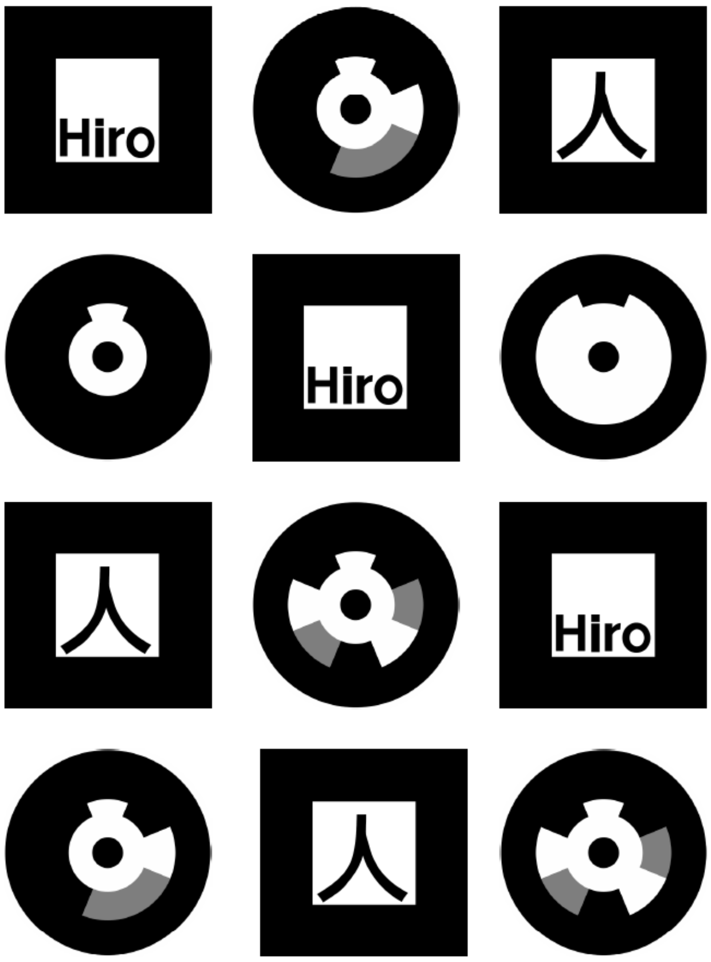


Fig. 4.22: Fiducial test sheet. When printed the fiducials each have a width of 47mm.

- *Distance* - see figure 4.23a
  The camera was slowly moved further away with the test sheet fixed.

- *Fast Moving* - see figure 4.23b
  The camera was fixed and the test sheet was moved quickly so as to create motion blur.

- *Forced Over-Exposure* - see figure 4.23c
  The camera was fixed and manually set to over-expose the image while the test sheet was moved and tilted slowly.

- *Forced Under-Exposure* - see figure 4.23d
  The camera was fixed and manually set to under-expose the image while the test sheet was moved and tilted slowly.

- *Local Shadows* - see figure 4.23e
  The camera and test sheet were fixed and a single light source was used to produce changing local shadows.

- *Low Light* - see figure 4.23f
  The camera was fixed and the lighting level set low while the test sheet was moved and tilted slowly.

- *Normal Light* - see figure 4.23g
  The camera was fixed while the test sheet was moved and tilted slowly.

- *Very Low Light* - see figure 4.23h
  The camera was fixed and the lighting level set very low while the test sheet was moved and tilted slowly.

The images were processed using the second application and the results collated manually. The software was then modified to measure the efficiency of different aspects of the ELF detection algorithm and run again using the same test images.

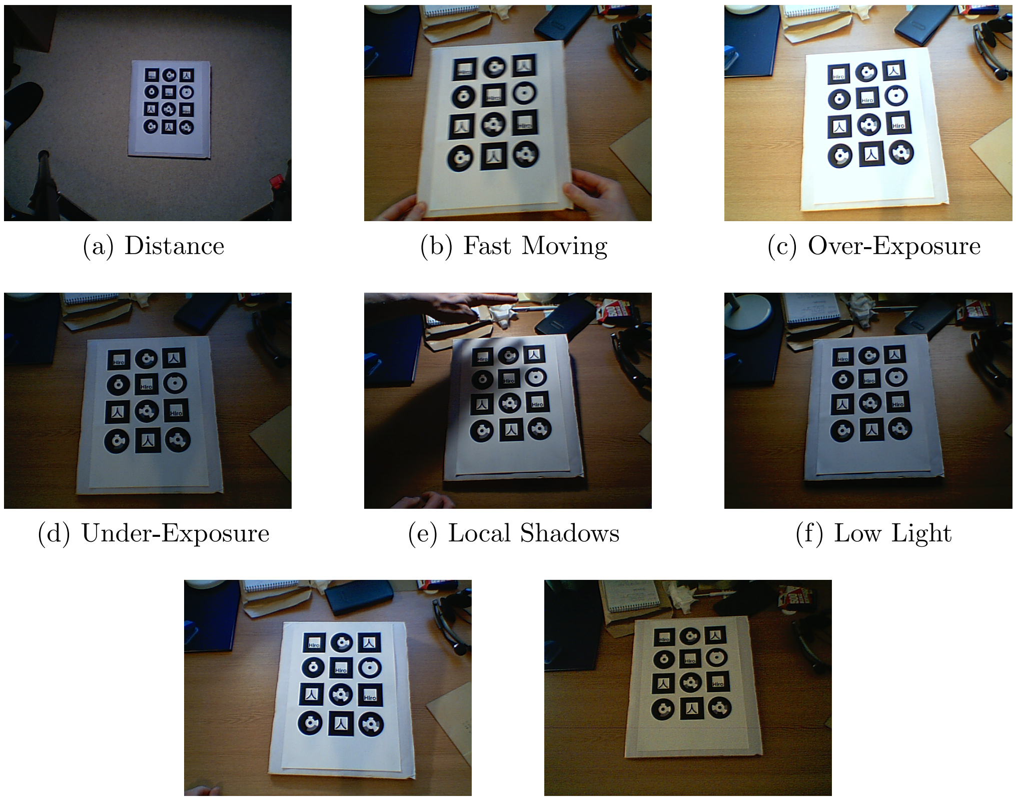

(a) Distance
(b) Fast Moving
(c) Over-Exposure
(d) Under-Exposure
(e) Local Shadows
(f) Low Light
(g) Normal Light
(h) Very Low Light

Fig. 4.23: An example from each group of test images.

4.7.4 Binary or Ternary?

The final test is concerned with the reliability and benefits of using ternary fiducials. Building on the same tools used for the comparison with *ARToolkit* in section 4.7.3 another test sheet was produced (see figure 4.24). The test sheet contains a single fiducial of each type with codes chosen that provide a reasonable amount of variation within the structure of the fiducial.

As in the previous section, images were captured under different conditions; in this case distance, low light and normal light. The software would then perform multiple passes on the images looking for each type of fiducial and outputting the results as images with overlaid fiducial identities.

The system developed for the optimisation of the fiducial design was also adapted so that tests could be performed on simulated data. Since the captured images are limited to testing only one of each type of fiducial, the simulation allows for testing of a large selection of code combinations. Each fiducial was tested at the same scales and angles as in section 4.6.1 and the results were then collated and output to file.

4.8 Results

4.8.1 False Positives

The false positives test was run on a total of 198,631 frames from a variety of sources and the results are displayed in table 4.4. A section of the film “Le Mans” was included due to the large number of circles present in the footage (racing car tyres), although there was only a single false positive found in 32,229 frames which was not a tyre and can be seen in figure 4.25c. The Honda television commercial “The Cog” was also included because of a high quantity of ellipses in the footage and fiducials were mistakenly identified in four of the frames (see figure 4.25a).

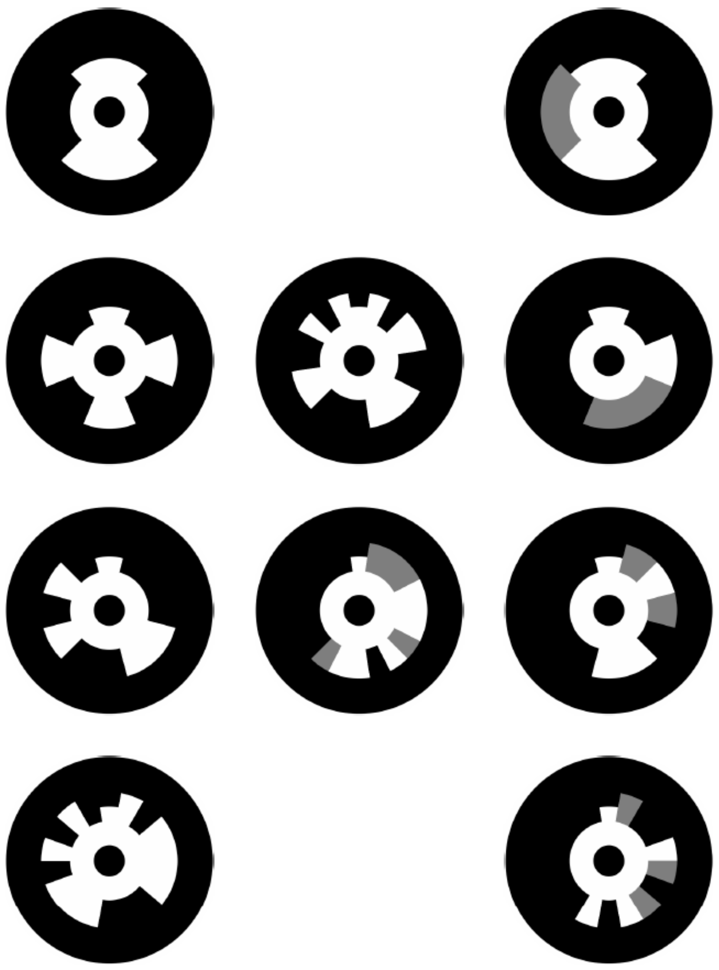


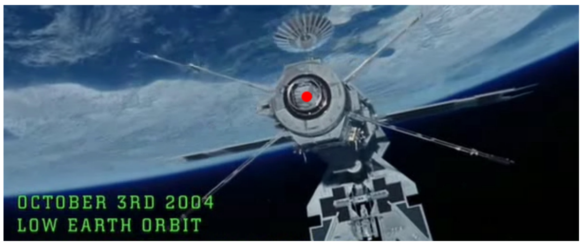
Fig. 4.24: Binary against ternary fiducial test sheet.

Source Footage	Dimensions	Frames	Fiducials
Alien vs. Predator (Film)	704 × 288	78914	8
Hex (TV)	640 × 352	67506	0
Honda (Commercial)	720 × 480	3131	4
Kill Bill (Film)	400 × 160	3875	0
Le Mans (Film)	347 × 234	32229	1
Reservoir Dogs (Film)	416 × 176	3278	0
Scrubs (TV)	512 × 384	6326	0
Shaun of the Dead (Film)	432 × 184	3372	0

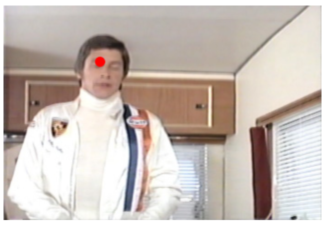
Table 4.4: False positive results.

The "Alien vs. Predator" clip contained 78,914 frames in which eight false positives were found. These were detected in a circular section of a satellite with dark and light patches that could easily be mistaken for fiducial elements (refer to figure 4.25b). No fiducials were found in any of the remaining footage which consisted of clips from both film and television. The false positive detection rate is therefore 0.0065% across all of the test data.

(a) Honda



(b) Alien vs. Predator



(c) Le Mans

Fig. 4.25: False positives detected in test footage.

4.8.2 Size of Fiducial

The graph in figure 4.26 indicates that a fiducial with a radius of 11 pixels or more should be detectable. This equates to a fiducial area of 380 pixels which is a large improvement over the original design proposed in [47] which started detecting fiducials reliably at a size of approximately 800 pixels.

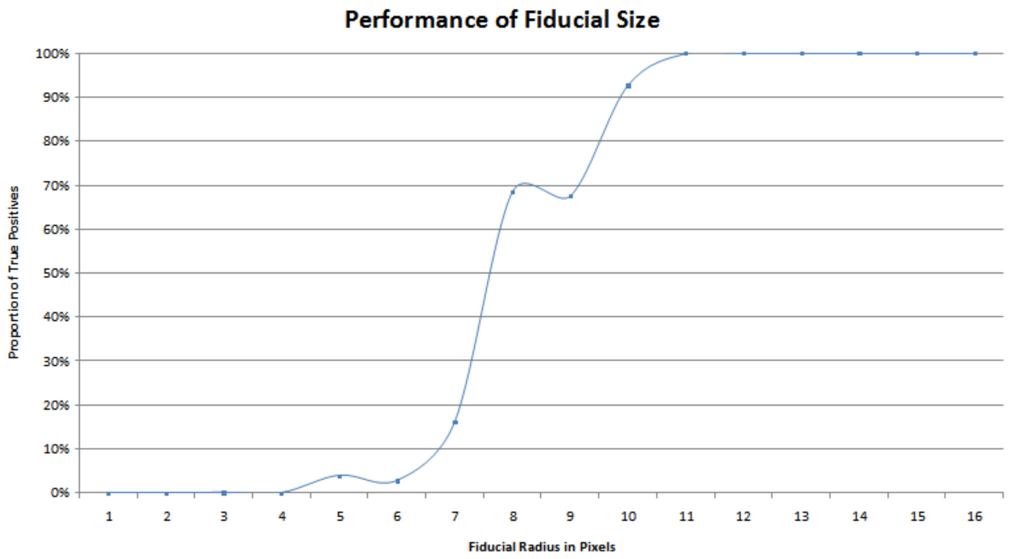


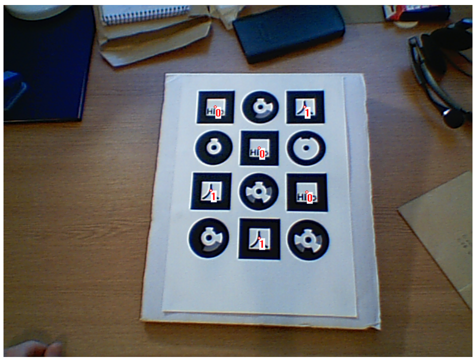
Fig. 4.26: Fiducial size performance results.

The range between 4 and 11 is not a smooth, steep slope as expected. This is primarily due to the sub-pixel changes in the rendered images which cause sudden failures in the ellipse detection when the inside sections merge with the outer ring. Another major reason is the failure of the marker detection and since both of these reasons are binary (either pass or fail) they can produce rather erratic results when approaching the thresholds.

4.8.3 Comparison

Figure 4.27 provides examples of the output images from the testing software. Image 4.27a is the output from the *ARToolkit* thread where the red numbers 0 and 1 represent successful detection of the “Hiro” and “Kanji” fiducials respectively. Image 4.27b is the output from the ELF thread and each red number represents the code read from that fiducial.

Figure 4.28 shows the proportion of fiducials not detected at all. Under the conditions of shadowing, low light and under-exposure the ability of the *ARToolkit* to detect the outer square of the fiducial is seriously affected whereas the proposed system shows far greater resilience.


(a) ARToolkit

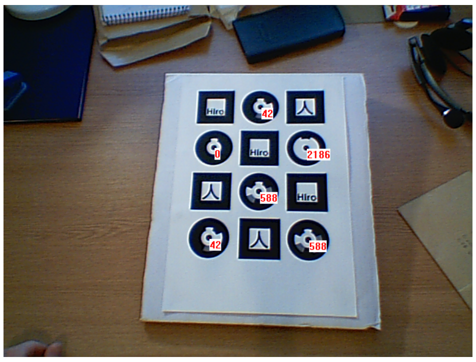

(b) ELF

Fig. 4.27: An example of output images from the testing software.

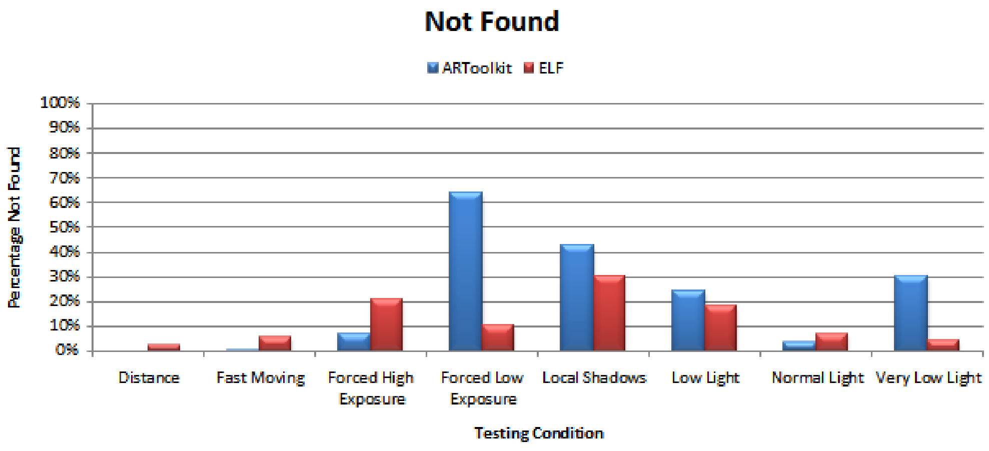


Fig. 4.28: Fiducial comparison results — not found.

Figure 4.29 presents the proportion of fiducials that were detected but incorrectly identified. The *ARToolkit* suffered the most problems with fiducial identification during the distance tests with nearly 30% incorrect. The proposed system was most affected when the camera exposure was forced high or low which causes the fiducial centre estimate to be perturbed. This in turn affects the accuracy of the fiducial scanning stages resulting in a higher number of misidentifications.

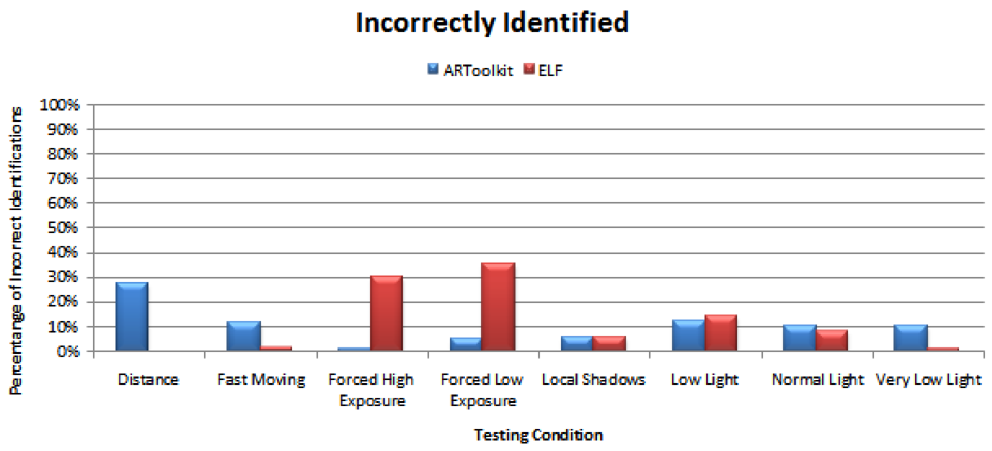


Fig. 4.29: Fiducial comparison results — incorrect identification.

The graph in figure 4.30 compares the true positives (i.e., detected and correctly identified) found by each fiducial system through all of the testing conditions. It is clear that the proposed fiducial design performs better or as well as the *ARToolkit* in all but one of the conditions. In the case of forced over-exposure the ability of the ELF detection falls well below that of the *ARToolkit*. This appears to be due to inaccuracies in the calculation of the fiducial centre and is most likely caused by the severe ghosting generated by the camera inside the edges which affects both the edge detection and the pixel values scanned inside the fiducial.

The section of the graph relating to normal lighting condition actually reports slightly lower performance than some of the other sections. This is due to the variation in angle of the test sheet as it was moved during frame capture. This resulted in that particular set of images containing fiducials at greater angles than in other sets and thereby reducing the detection rate.

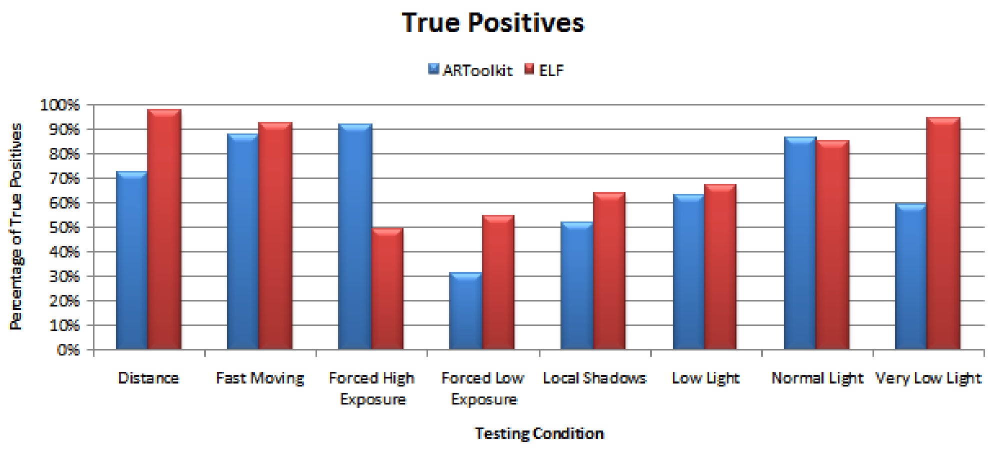


Fig. 4.30: Fiducial comparison results — true positives.

As mentioned earlier, the *ARToolkit* has the advantage here since it only needs to discriminate between two fiducial types whereas the ELF contains 2187 combinations.

Efficiency

The following table (4.5) provides the time taken for each stage of the fiducial detection system to run. The times are an average across all of the test images from the *ARToolkit* comparison. The images were captured from a USB camera at a resolution of 640 $\times$ 480 and processed on a 2GHz AMD machine with DDR memory.

The total time of processing was 28.73 ms which corresponds to an average frame rate of 34.8 fps. The largest bottleneck in the system is the edge detection which accounted for half of the overall processing time. The scanning of the fiducial for verification and identification is also a major concern when it comes to optimising the algorithms further.

Stage of Fiducial Detection	**Time (ms)**
Edge Detection	14.33
Object Finding	3.74
Ellipse Fitting	0.83
Verification and Identification	9.82
Total	**28.73**

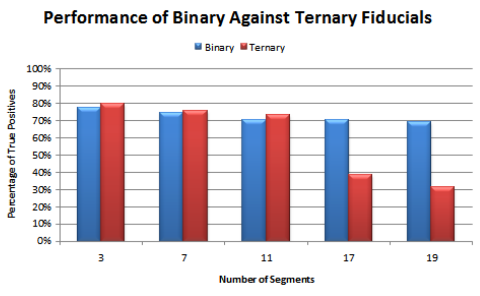
Table 4.5: Algorithm performance results.

In comparison, the *ARToolkit* averaged 5 ms for the entire detection process. Although it has the advantage of many more years of development over ELF with plenty of people working on optimisations, it does have the problem of scalability. In the test scenarios presented here it was only concerned with matching two patterns, but what if it was given the same number of combinations that are possible with the 7-segment ternary ELF? The *ARToolkit* supports up to 50 patterns by default and measurements over this range suggest that it could take over 50 ms if 2000 patterns are used.

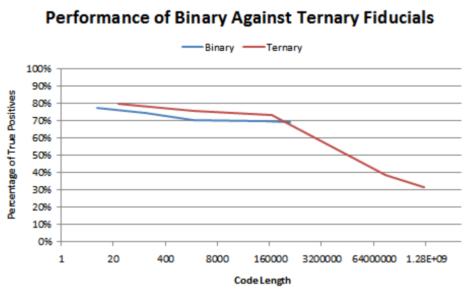
4.8.4 Binary or Ternary?

The final results compare the binary and ternary options of the proposed fiducial system. The graphs in figure 4.31 provide the results for the simulated tests which covered a large number of code combinations. In these simulations the binary fiducials showed a slight drop in performance with an increase in the number of segments as expected. However, the ternary fiducials showed a significant drop in performance at 17 and 19 segments.

The graphs of results for the captured image test are shown in figure 4.32. In this case the binary and ternary fiducials perform at a similar level throughout, both dropping as the number of segments increases. There is an anomaly in which the 19 segment fiducials actually perform better than the 17 level segments but this could simply be due to the position of the fiducials on the test sheet or even that the particular codes chosen for 19 were more reliable to read. Since only a single code was used for each type it is unreasonable to draw conclusions from this data alone and hence the



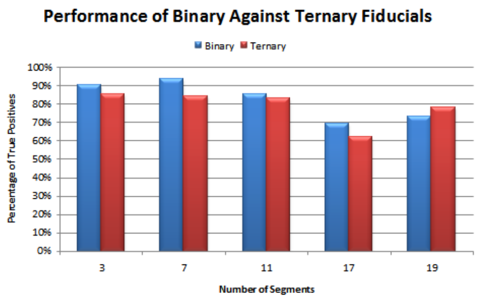
(a) Against Number of Segments



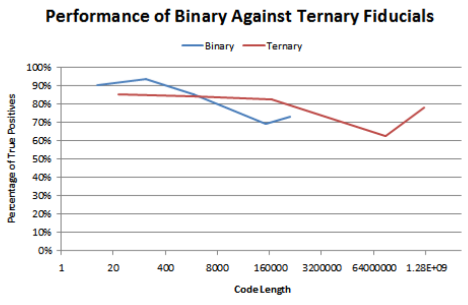
(b) Against Code Length (Log Scale)

Fig. 4.31: Binary against ternary — simulation test results.

inclusion of simulated data.



(a) Against Number of Segments



(b) Against Code Length (Log Scale)

Fig. 4.32: Binary against ternary — captured image test results.

4.9 Conclusions

This chapter has proposed a fiducial design which has proven to be more reliable than the popular *ARToolkit*. In section 4.3 a set of requirements were specified and then two of these in particular were described as being important to a VAE. Accuracy is vital when it comes to calibrating an augmented environment and through the use of circles in the design, very precise ellipse centres can be calculated. Efficiency is important since a fiducial system could be running alongside many other elements within a VAE. Although the proposed design may not be as fast as some other designs available it is still in the early stages of development and further optimisation is possible (see section 4.9.1).

The single false positive detected in the "Le Mans" footage is inconsequential since the addition of a tracking layer to the fiducial system would ignore it. The remaining false positives are of more concern since fiducial-like objects were detected over a number of consecutive frames. However, considering the similarity of those objects to actual fiducials the small number of false positives found are acceptable.

The ELF design performed as well as, and in some cases better than, the *ARToolkit* over a range of tests. The only problem area was that of forced over-exposure which had a detrimental effect on the accuracy of the fiducial centre calculations.

The binary and ternary comparison shows that ternary can potentially perform as well as binary especially considering the number of combinations available in each. For example, if smaller fiducials were required (or fiducials at a greater distance) then it is logical to select a fewer number of segments to improve the reliability and therefore if 2000 codes were needed it may be better to choose 7-segment ternary over 11-segment binary.

The ELF design proposed in this chapter has been successfully incorporated into the *OpenIllusionist* framework and provides support for the calibration system (see chapter 5) along with an additional approach for interacting with a VAE.

4.9.1 Further Work

Optimisation

The results indicated that the major bottlenecks of the ELF algorithm are the edge detection and the fiducial verification and identification. Improvements can certainly be made to the verification stage since it is currently too complex; the set of criteria that accept or reject a fiducial must be reviewed, simplified and can then be further optimised. The identification stage relies on a more holistic approach to read the fiducial code and the processing time could be reduced by adapting the method to reach the solution in fewer iterations.

The edge detection method used in the proposed system is also a recent development. Improvements in the efficiency could certainly be achieved by optimising certain internal techniques such as image rescaling and blurring. Additionally, other stages of the method could be merged to reduce the number of independent loops within the code.

Object finding can be modified not only to improve efficiency but also remove the possibility of a stack overflow by replacing the recursive algorithm with an iterative equivalent.

Reliability

The majority of problems with the proposed system were either related to incorrect rejection of the fiducial or incorrect reading of the fiducial code. Both of these can be linked to an accuracy problem with the fiducial centre calculation which is first estimated by starting at the centre of the ellipse and then adjusted using the centre spot as reference. Since a simplification of the verification stage was suggested to improve efficiency it would also be a good opportunity to revise the method of adjustment, perhaps using a technique adapted from [38].

Although the edge detection method is adaptive it still relies on an initial threshold value and so in low contrast situations it can struggle to distinguish edges. Therefore a fast pre-processing stage could be included that would calculate a sensible threshold value to pass on to the edge detector.

The algorithm currently rejects a fiducial if more than one marker is found. It may be worth investigating the performance when the system is modified to instead select the most likely candidate for the marker. This modification could, however, increase the number of false positives detected.

Binary vs. Ternary

The results certainly suggest that ternary is a viable option when choosing a fiducial specification. The methods described in this chapter would benefit from further development which could improve the reliability of ternary fiducial identification. The testing stage relied on both simulated data and real captured data but the range of tested fiducials and situations in the real data was limited. Therefore further testing is necessary before it can be stated that ternary is as reliable as binary and the results here imply that additional research is justified.

Chapter 5

Calibration

5.1 Introduction

Accurate and reliable calibration is vital for any augmented reality system so that it can maintain the illusion that virtual elements are responding to physical user interaction. For a VAE infrastructure, the calibration process must provide the means with which to sustain a coherent spatial relationship between physical objects and virtual elements in a continually changing visual environment. The system must perform this task reliably and as close to real-time as possible, even when objects are being moved.

This chapter begins by presenting an extensive, although not exhaustive, range of possible scenarios, each with a possible technical solution. Based on a suitable subset of the scenarios, a specification is then defined that provides a practical basis for developing a calibration system. A review of existing calibration methods is presented and then techniques for dealing with the two aspects of calibration, photometric and geometric, are considered.

The proposed photometric algorithms are concerned with practical, automated adjustment of camera parameters and performing pre-processing on captured frames. The geometric aspect deals with calculating a projective transform and then providing an optimised method with which to apply that transform to live video images.

5.2 Scenarios

The following tables describe a large number of possible situations which could require differing calibration techniques. A possible solution to each problem is also suggested. The first table (5.1) considers setup scenarios such as the hardware being used and how it is aligned.

Scenario	**Possible Solution**
Camera does not see the entire projection area	Project many fiducials to gauge the area that can be seen and then provide projected feedback as to which way to move the camera or projector.
Non-adjustable camera	Throughout the calibration routine provide feedback to the user about what to adjust (focus, aperture etc).
Fully-adjustable camera	Calibration system can automatically make changes to the camera setup.
Grey-scale camera	Calibrate for brightness, contrast and exposure.
Colour camera	Same as grey-scale camera but include colour and white balancing.

Table 5.1: Setup scenarios.

Table 5.2 presents likely scenarios for the background, i.e., what will be visible to the camera all of the time. Table 5.3 considers possibilities for the foreground and table 5.4 expands on some particular examples of foreground clutter.

Tables 5.5 and 5.6 cover potential indirect and direct lighting conditions.

5.3 Background

A large proportion of AR developments are concerned with HMDs. However, the infrastructure requirements of this thesis are based upon augmented environments where the user is not encumbered with equipment but can, for

Scenario	**Possible Solution**
Plain background	Set background level to mid-luminance level by adjusting the camera. Check the luminance range by projecting full white.
Patterned background	Attempt using histogram analysis on the background to provide a reasonable dynamic range and adjust the camera accordingly. Capture the background with and without full projection. Subtract and scale to allow geometric calibration when the fiducials are projected. Subtract background when running to prevent the pattern features being detected by the system.
Using monitor instead of projector	Account for flicker due to the refresh rate by ensuring that the camera exposure time is great enough. Display black, set low-level luminance of the camera. Request that a user points a finger in the centre of the screen, set the mid-level luminance of the camera. Display white, set high-level luminance of the camera. Perform geometric calibration and then repeat the previous steps but only considering the area of interest (it could possibly skip the stage requiring user interaction on the first pass).
Non-flat surface	Perform photometric calibration as above. Project many fiducials and allow the area to be broken up into blocks and then perform geometric calibration on each block independently. Merge this information to form an approximate calibration grid.

Table 5.2: Background scenarios.

Scenario	**Possible Solution**
Empty	Request that a hand or object is placed in the centre of the area to determine the low and mid-levels of the camera. Then remove the hand and project full white to set the high-level ready to perform geometric calibration.
Clutter	Calculate the level peaks based on the histogram and then adjust the camera accordingly. Project full white and use background subtraction and scaling to allow fiducials to be detected even when they are overlapped by shallow objects. Project many coded fiducials so that at least some should be visible. Pick the outermost and least distorted four to perform the geometric calibration.

Table 5.3: Foreground scenarios.

Clutter	**Possible Solution**
Pen Lines	Request that lines be drawn so that levels for edge detection can be determined.
Objects	Leave in the scene to assist level finding during calibration.
Hands	Keep still but leave in the scene.
Heads	Request that all but hands are removed and that hands are kept still. The process could start by checking for motion.

Table 5.4: Dealing with different types of clutter during calibration.

Scenario	**Possible Solution**
Low Ambient	Possibly provide advice on how to adjust the camera (iris) if either the camera cannot be automatically adjusted or it is hitting the noise level when the system tries changing the brightness, contrast or exposure. Another option is to use the projector as a back-light and gradually add illumination until the system can set reasonable levels for the camera.
Extreme Ambient	Reduce the camera settings automatically. If this is not enough (or not possible) then provide advice on how to adjust the camera manually (i.e., closing the iris).
Soft Shadows	Smoothing of the input images can reduce the chances of detecting shadow edges. Consider using colour-space based shadow removal (see chapter 6).
Hard Shadows	A shadow removal technique is necessary to prevent false detection of edges.

Table 5.5: Indirect lighting scenarios.

Scenario	**Possible Solution**
Bright Spots	A similar technique to shadow removal could be employed or simply clip the values at a certain level so that the spots are classed as projector light and thus ignored.
Light Source in Frame	Should not be allowed since this would greatly affect the camera exposure and could cause severe blooming.
Weak Projector	If the projection cannot be detected then request that the ambient light be reduced.
Strong Projector	Allow it to clip and either suggest turning up the ambient light (which would improve the realism of the display) or use the projector as a partial direct light source and therefore reduce the effect of shadows produced by other light sources.
Projector Refresh	Ensure that the camera exposure is long enough to capture a full projection frame.
Dynamic Lighting	This would normally be caused by the projector. Set the levels so that the dynamic lighting is above the ambient level threshold and therefore ignored.

Table 5.6: Direct lighting scenarios.

example, simply walk up to a table and begin interacting. It therefore requires calibration for camera-surface-projector systems, as in figure 5.1, or occasionally camera-monitor systems.

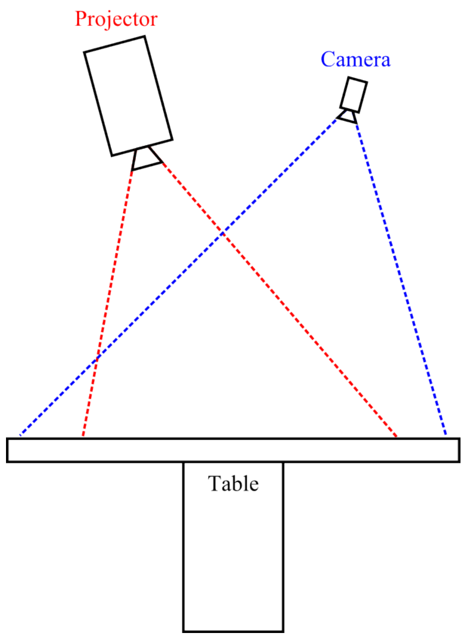


Fig. 5.1: A typical camera-surface-projector configuration.

Calibration techniques for both HMDs and projected systems are discussed in [27]. However, in both cases they rely on the use of a magnetically tracked stylus which is not relevant for the framework that this thesis is proposing.

Raskar et al. present a calibration technique in [55] that is capable of dealing with non-planar surfaces and multiple, overlapping projectors. In this way it can provide a wide field-of-view display that is correct from one particular location (i.e., where the camera used for calibrating is placed). It projects one pixel at a time and registers the corresponding location in the captured image thus building a transform between projected space and the view-port of the camera. The projection is then warped on rendering such that it appears correctly to a spectator in the perfect position. Overlap of projectors is dealt with by alpha-blending edges using a linear gradient (or non-linear if the function of projector intensity is known).

A technique that extends non-planar surface projection by also adapting for radiometrically complex surfaces is described in [8, 7]. This method of calibration allows an image to be projected onto a patterned background and yet appear as if displayed on a plain surface. In [8], Bimber et al. also deal with the problem of view-dependent stereoscopic vision where the projection must adjust the displayed images based on the physical position of the user. They propose a solution that does not require an accurate 3D model of the projection surface but instead uses a set of known parameters measured from multiple camera viewpoints. The parameters are then weighted and interpolated appropriately to estimate the correct parameters for a user in an arbitrary position. They also propose a method with which to cope with projector focus on a non-planar surface since a projector can only provide planar focus. The proposed method involves overlapping multiple projectors, focussed on different planes, and measuring how well focussed every pixel is from each projector. The contributions of each projector are then combined appropriately so as to produce a display of consistent focus.

A review of calibration techniques is presented in [46] covering aspects of geometric and appearance problems for both single and multiple projector configurations. Of particular interest is the discussion of curved surface projection in which the problem can be simply broken up and dealt with as a piecewise planar system.

5.4 Specification

The scenarios described above cover a wide range of possibilities but to begin developing a calibration stage for the framework this chapter only considers a subset of them. To begin with a foundation that provides a general, yet practical solution, the following assumptions are made:

- A camera-surface-projector setup is used.

- The camera can see the entire projection area and is grey-scale or colour.
- The camera is at least partially adjustable (exposure, brightness and contrast).
- The background is a planar surface and is of a plain colour.
- The ambient light level is variable, multiple soft shadows are present.
- A powerful projector is used which can also provide direct lighting if the ambient is too low.
- Foreground objects of reasonable contrast can be exploited to help adjust camera levels.

In a situation where the entire projection area is not visible the second assumption can still be applied by only using the section of projection area that is visible.

The problem can be split into two parts, one which deals with photometric calibration (section 5.5) and the other which deals with geometric calibration (section 5.6).

5.5 Photometric Calibration

Photometric calibration is concerned with adjusting the camera and lighting situation so that the VAE has a good dynamic range to work with. It can be achieved through direct control of the camera or as a set of instructions to the user. Since a camera that is partially adjustable has been specified, coupled with the fact that many of the available USB and *FireWire* cameras provide some degree of control over their parameters, this method will aim to adjust the camera directly.

As mentioned in section 1.4.1, *DirectShow* was chosen as the method for capturing images from a camera on the *Windows* platform. The *DirectShow* libraries provide the ability to control all aspects of a camera and this functionality was therefore incorporated and simplified in the experiments here, and subsequently in the *oiFrameGrabber* class.

If the exact response of a camera is known then it may be possible to perform a measurement and then jump directly to the appropriate settings for exposure, brightness and contrast. However, as mentioned above, any type of camera could be used with the proposed system and therefore an adaptable method is desirable. By adjusting the parameters a little at a time and monitoring the effect it is possible to calibrate for any connected camera.

In some situations it may even be feasible to allow the camera to automatically adjust itself. This is usually performed by the driver software, which has been designed to deal with images containing plenty of detailed objects or people. A VAE, as proposed in this thesis, will often have a plain white background with maybe a few sparse objects in the foreground and rapidly changing images being projected, all of which can confuse the normal automatic control algorithms. It is therefore desirable to take full control of the camera within the framework and calibrate specifically for the aforementioned scenarios.

5.5.1 Algorithm Overview

The main aspect of the photometric calibration is adjusting the camera using a set of logical decisions as described by the flowchart in figure 5.2. This part of the photometric calibration is performed with the projector set to display nothing (i.e., the render output is completely black).

The process labelled “Adjust high peak” is shown in more detail in figure 5.5.1 and the one labelled “Adjust low peak” is shown in figure 5.5.1.

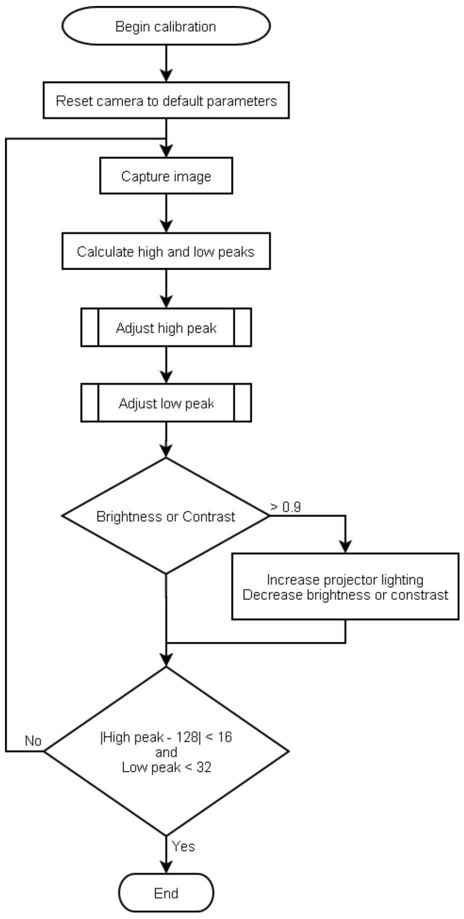
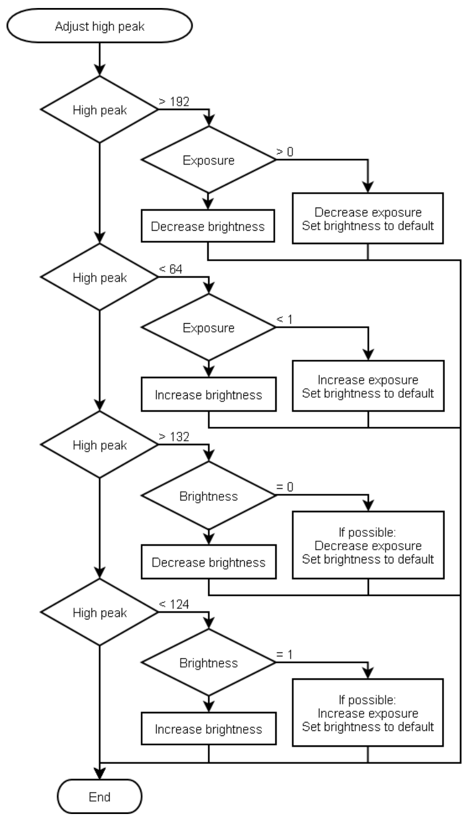
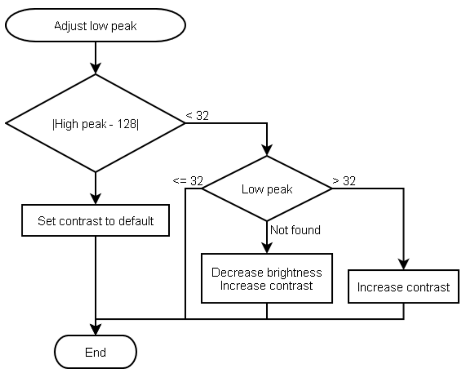


Fig. 5.2: A flowchart of the photometric calibration algorithm.



(a) Adjust high peak



(b) Adjust low peak

Fig. 5.3: Flowcharts of processes involved in the photometric calibration algorithm.

The values involved are the high and low peak values which are calculated from the histogram of the greyscale input image and are therefore in the range 0–255. The adjustable camera parameters are exposure, brightness and contrast which are all scaled to the range 0–1.

5.5.2 Algorithm Description

The algorithm attempts to simultaneously adjust the exposure, contrast and brightness to achieve a greyscale image in which the predominant range is 0–128 and anything projected would then appear above the 128 level. In the early iterations it performs a coarse adjustment of the exposure to adjust the high peak into the range 64–192. If the limits of the exposure parameter are reached then the brightness is adjusted instead.

Once the coarse adjustment is complete a finer level of adjustment is performed by varying the brightness. If the limits of the brightness are reached then the exposure is increased or decreased accordingly, the brightness is reset to default and the fine adjustment continues. The aim is to achieve a high peak level of approximately 128.

Throughout the coarse and fine adjustments for exposure and brightness the contrast level is also modified so that the low peak drops below 32. If a low peak cannot be ascertained then both the brightness and contrast are adjusted until it can.

If the system approaches the upper limit for either the brightness or contrast (> 0.9) then it decides that the light level is too low for the camera and therefore begins to use the projector to provide illumination. Each time the level of projected light is raised the parameter that triggered this response is decreased back below the threshold (0.9).

5.5.3 Background Compensation

The previous steps require a plain surface with objects placed on it in order to gauge the high and low levels detected by the camera. Ideally this should be a white background with black objects or vice versa. Since these objects could be placed anywhere there is a good possibility that they will affect the geometric calibration stage because black objects will reflect much less light from the projector than a white background. The system therefore attempts to compensate for this as much as possible.

A reference frame is captured with the projector rendering full black (all pixels switched off) and then a second frame is captured with the projector rendering full white (all pixels switched on). During the next stage of calibration each image that is processed undergoes an initial modification step. This modification rescales each pixel based on the corresponding pixels in the two reference frames. Equation (5.1) describes how each pixel is scaled where i is the pixel index, C is the current frame, R_b is the black reference frame, R_w is the white reference frame and C' is the rescaled image.

$$C[i]' = 255 \times \frac{C[i] - R_b[i]}{R_w[i] - R_b[i]} \tag{5.1}$$

A pixel in the frame that corresponds to a dark object would have a lower dynamic range when light from the projector hits it and so the rescaling stretches this range. A pixel that corresponds to the white background will have a much higher dynamic range and so the rescaling will have a minimal effect. Overall this results in an image in which the dark objects have been almost completely removed and is therefore more suitable for further processing.

5.5.4 Plane Calibration

Setting a single threshold level when performing image processing tasks within a VAE, such as edge detection, can be problematic since the lighting level across the projection surface could vary considerably. This is particularly apparent in uncontrolled lighting situations where there is likely to be multiple light sources coming from different directions.

This section proposes a calibration method to reduce the effects of uneven lighting. The theory is based on the assumption that the variation in lighting across a planar surface with constant reflectance is also a plane.

The steps involved in the method to cancel the effect of lighting variation are as follows:

1. Capture a reference image with no illumination from the projector.
2. Capture a reference image with full illumination from the projector.
3. Subtract the reference images from each other and threshold to create a mask of only the background surface.
4. Fit a plane to the pixel values that fall within the mask (for each colour channel independently).

Fitting a Plane

Fitting a plane equation (5.2) to the pixel values requires a minimisation method such as least squares where the system attempts to minimise the sum of the squares of the residuals.

$$f(x, y) = ax + by + c \tag{5.2}$$

In this case the summed, squared residuals are given by the error equation (5.3) where a, b and c are the parameters of the plane (to be determined), x and y are the image coordinates, v is the pixel value, i is the pixel index and N is the number of pixels.

$$E = \sum_{i=1}^{N} (ax_i + by_i + c - v_i)^2 \tag{5.3}$$

Finding the derivative of the error equation with respect to each of the three plane parameters results in three equations which can then be used as a set of linear simultaneous equations to solve for the parameters, see (5.4), (5.5) and (5.6).

$$\frac{\partial E}{\partial a} = 2\sum_{i=1}^{N}(ax_i + by_i + c - v_i)x_i = 0 \tag{5.4}$$

$$\frac{\partial E}{\partial b} = 2\sum_{i=1}^{N}(ax_i + by_i + c - v_i)y_i = 0 \tag{5.5}$$

$$\frac{\partial E}{\partial c} = 2\sum_{i=1}^{N}(ax_i + by_i + c - v_i) = 0 \tag{5.6}$$

Expanding the equations gives (5.7), (5.8) and (5.9).

$$a\sum_{i=1}^{N} x_i^2 + b\sum_{i=1}^{N} x_i y_i + c\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} x_i v_i \tag{5.7}$$

$$a\sum_{i=1}^{N} x_i y_i + b\sum_{i=1}^{N} y_i^2 + c\sum_{i=1}^{N} y_i = \sum_{i=1}^{N} y_i v_i \tag{5.8}$$

$$a\sum_{i=1}^{N} x_i + b\sum_{i=1}^{N} y_i + cN = \sum_{i=1}^{N} v_i \tag{5.9}$$

These can be arranged in matrix form (5.10) and then rearranged as an augmented matrix (5.11).

$$\begin{bmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i & N \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum x_i v_i \\ \sum y_i v_i \\ \sum v_i \end{bmatrix} \tag{5.10}$$

$$\left[\begin{array}{ccc|c} \sum x_i^2 & \sum x_i y_i & \sum x_i & \sum x_i v_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i & \sum y_i v_i \\ \sum x_i & \sum y_i & N & \sum v_i \end{array}\right] \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{5.11}$$

For simplicity the augmented matrix will be represented as follows:

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{array}\right] \tag{5.12}$$

Performing Gaussian elimination on the augmented matrix provides a solution in upper-triangular form as in equation (5.13).

$$\left[\begin{array}{ccc|c} s_{11} & s_{12} & s_{13} & s_{14} \\ 0 & s_{22} & s_{23} & s_{24} \\ 0 & 0 & s_{33} & s_{34} \end{array}\right] \tag{5.13}$$

Rearranging from an augmented matrix back into normal matrices allows the system of simultaneous equations to be solved giving:

$$c = \frac{s_{34}}{s_{33}} \tag{5.14}$$

$$b = \frac{s_{24} - s_{23}c}{s_{22}} \tag{5.15}$$

$$a = \frac{s_{14} - s_{12}b - s_{13}c}{s_{11}} \tag{5.16}$$

Subtracting the Plane

Now that the plane parameters have been determined each frame can be adjusted by simply subtracting the $ax + by$ term of the plane which results in an image with greatly reduced illumination shift across the surface and that can be thresholded around the value of c.

5.6 Geometric Calibration

As described earlier, accurate geometric calibration is necessary to maintain the perception that the user is interacting directly with virtual objects. The aim of geometric calibration is to produce a transform that maps pixels in image space (captured from the camera) to pixels in rendered space (displayed by the projector) with as little error as possible.

The algorithm must begin by locating points in image space for which the coordinates in rendered space are known. Methods to locate known points can involve projecting straight lines or coloured points which simplify the image processing task. Another common method is to use fiducials and since the proposed fiducials in chapter 4 are circular the centres can be located very accurately [25].

The fiducial detection algorithm is easily modified to search for concentric circles projected onto the surface. Concentric circles are simpler than coded fiducials and are therefore less likely to be affected by objects on the surface. Objects not only cause a reflectance problem (as discussed in section 5.5.3) but also a displacement problem if the objects are too tall. Displacement occurs because the projector and camera are not "looking" along the same vector (see figure 5.1) and therefore the camera can potentially see breaks in a projected pattern as it overlaps an object. Even if they were both pointing along the same vector, the overlap would still cause distortion of the projected marker.

Since the renderer can be controlled to project a single pair of concentric circles at a time, the identification of fiducials is no longer an issue (although rendering a single screen with multiple coded fiducials would be faster). However, the lag between the rendering system and the camera could result in the same fiducial being detected twice and so simple logic can be included that ignores a marker that is located, within tolerance, at the same coordinates as the previous marker.

5.6.1 Projective Transform

An affine transformation, equation (5.17), is the result of a linear combination of translation, rotation, scaling and sometimes shearing. This transform preserves straight lines and the parallelism of lines which can work well if the camera is aligned perpendicular to the projection surface and the projection has had the correct keystone adjustment. In general the camera will not be perpendicular and so the affine transform will produce inaccurate results.

$$\begin{aligned} x' &= ax + by + c \\ y' &= dx + ey + f \end{aligned} \tag{5.17}$$

The projective transform (5.18) is similar to the affine except that it contains a denominator term. This results in a transform that preserves straight lines but not the angles between them and is therefore capable of accurately mapping geometry that has undergone projective distortion.

$$\begin{aligned} x' &= \frac{ax + by + c}{gx + hy + 1} \\ y' &= \frac{dx + ey + f}{gx + hy + 1} \end{aligned} \tag{5.18}$$

Since there are eight unknowns in the projective transform, four coordinate pairs are required to solve it and thus four fiducials must be projected and then located in image space. Following that, calculation of the unknown parameters is simply a case of solving the eight simultaneous equations using either a direct algebraic method or an iterative method.

Alternatively, many more points could be detected and then a least squares method used to fit the projective transform to the data. This method could provide better accuracy but it would also be slower and less able to deal with objects laying on the surface. However, the proposed method has been found to provide acceptable accuracy and could easily be adapted to avoid areas of the surface where tall objects are present.

5.6.2 Optimising the Transform

Once the parameters for the projective transform have been found the system has an equation it can use to transform coordinates from image space to rendered space. The problem with this is that a number of multiplications and divisions need to be performed for every pixel in each frame which will cause a serious performance hit especially with so many other processes running within the framework simultaneously.

This problem can be solved since the projective transform never changes (assuming the camera, projector and surface are static) and, using pointer indirection, a form of lookup table can be generated. The lookup table is handled by the system as an image that is the same dimension as the rendered space except that at every pixel location lies a pointer to the transformed pixel location in image space. This means that the VAE can simply read the pixel values using world coordinates (render space) and not be concerned with calculating the appropriate coordinates in the captured frames. Transformed coordinates are not likely to be integer values and so the main limitation of this method is the inability to interpolate pixel values, although the gain in performance far outweighs the loss in pixel accuracy. Some specific applications may require better pixel accuracy and so it could be beneficial to implement a slower alternative within the framework that provides the developer with a choice.

5.7 Results

Although there has been no formal testing of the base photometric and geometric calibration algorithms, they have been successfully used within the *OpenIllusionist* framework for more than two years (at the time of writing). Furthermore, the *Robot Ships* exhibit (chapter 8) has been running daily for almost a year, encountering a variety of lighting conditions and foreground object configurations.

5.7.1 Plane Calibration

The graph in figure 5.4 shows the pixel values for a single colour channel (green in this case) as a surface. This data supports the assumption that the change in lighting level across the surface is approximately planar, the spikes in the corners are where the edge of the table ends and the spikes in the middle are caused by dark objects on the surface. The graph was generated using a sample of data every 20 pixels in both the x and y dimensions.

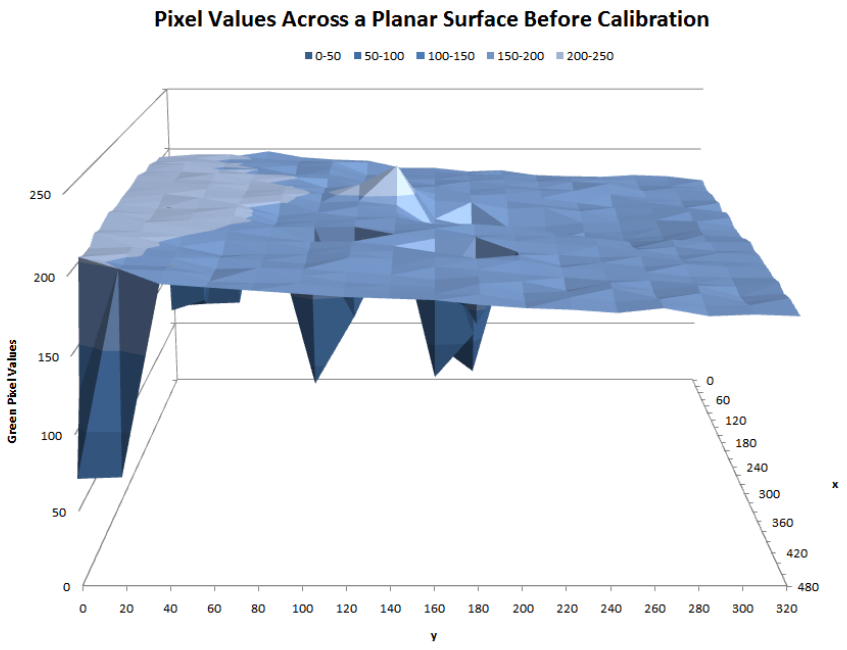


Fig. 5.4: Pixel values for an image captured from a VAE.

This indicates that fitting a tilted plane to the data will provide a very good estimation of the ambient lighting variation across the surface.

The graph in figure 5.5 is the result of applying plane calibration to the data and provides a clear view of the positive effect of plane calibration.

The images in figure 5.6 show a captured frame before and after plane subtraction, with the graphs in figure 5.7 being the corresponding histograms

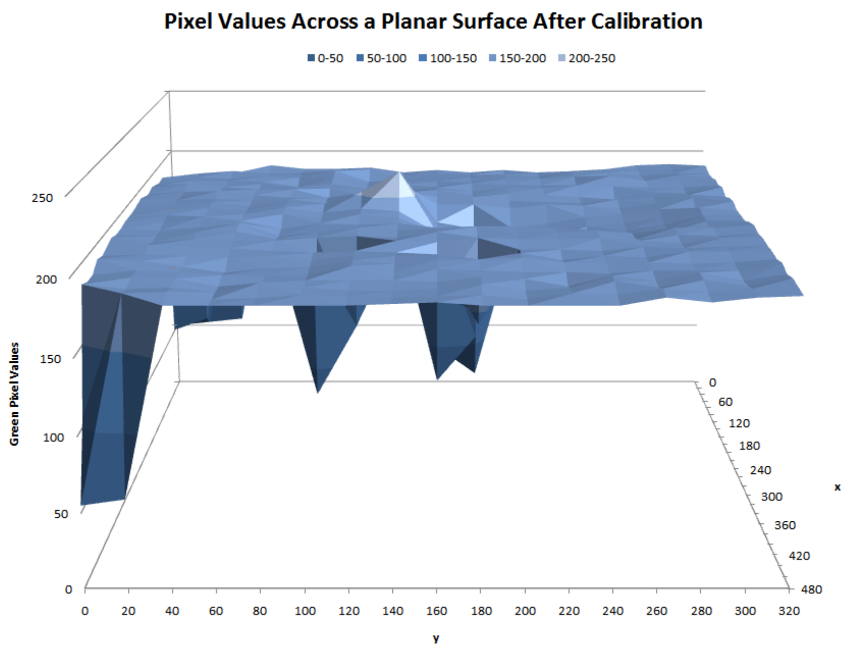


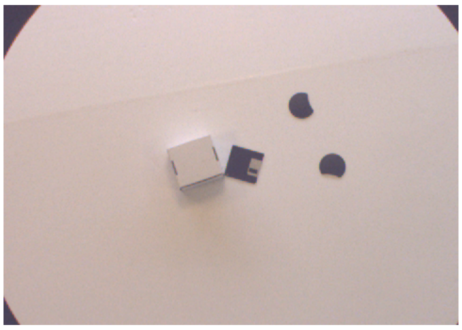
Fig. 5.5: Pixel values for an image captured from a VAE after plane calibration.

(for the green channel).

5.8 Conclusions

This chapter presents methods to calibrate for both photometric and geometric conditions in a typical camera-surface-projector system. Although not a comprehensive test, these methods have been used reliably in *OpenIllusionist* for more than two years.

Plane calibration was also proposed as a means to reduce the variation in illumination across the background surface. The results were positive and warrant further research into the use of this method as a preliminary stage in image processing. Chapter 7 takes it a stage further by combining plane calibration with shadow removal to provide a better image for finger detection. It also shows the effectiveness of the method under a number of different lighting conditions.


(a) Before plane subtraction

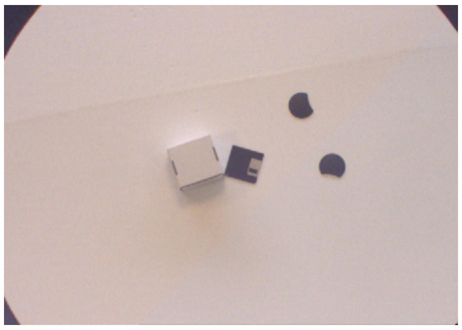
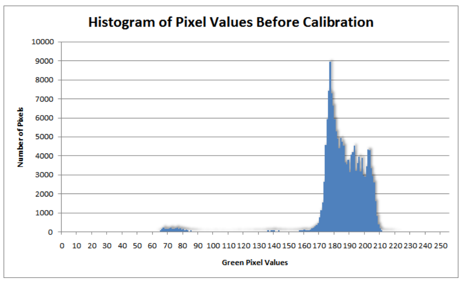
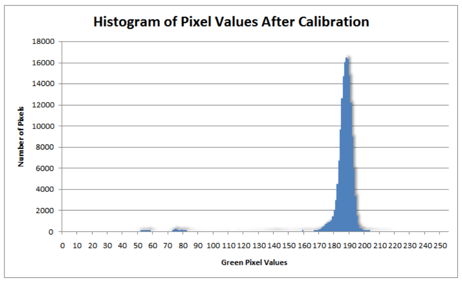

(b) After plane subtraction

Fig. 5.6: An image captured from a VAE before and after plane subtraction.



(a) Before plane subtraction



(b) After plane subtraction

Fig. 5.7: The corresponding histograms of an image captured from a VAE before and after plane subtraction.

5.8.1 Further Work

The photometric calibration algorithm currently suffers from problems with efficiency, especially if the camera is slow at adjusting its parameters. Adapting a binary search or gradient descent method could improve the speed at which the system calibrates.

The purpose of geometric calibration is to provide an accurate registration method between the image space of the camera and the rendered space of the projector. As it stands, the accuracy of the system will be affected by lens distortion which is not currently considered. However, methods for determining the intrinsic camera parameters can be widely found in the literature. Therefore, correction for lens distortion and camera-projector registration could easily be combined in future development.

As discussed in section 4.9.1 the accuracy of the fiducial centre calculation is reduced at acute angles and this is accentuated when there is no centre spot for reference. The geometric calibration method proposed here makes us of concentric circles instead of coded fiducials, nevertheless there are unique properties of concentric circles that allow very accurate centre coordinates to be calculated [38].

The next step with the plane calibration method is to incorporate it into the framework so that further image processing tasks, such as simple thresholding, can benefit from a uniform background.

Further adaptations to the framework calibration system could include methods to calibrate multiple overlapping cameras and projectors which would allow much larger surfaces to be augmented. Breaking up the geometric calibration into a piecewise planar problem would provide the ability to automatically correct projection onto non-planar surfaces.

Chapter 6

Shadow Removal

6.1 Introduction

Shadows present a significant problem with regards to image processing. A VAE is often subject to numerous shadows due to multiple light sources such as windows and overhead lights. Shadows, for example, can result in edge detection inaccuracies (as in figure 6.1) or they can be seen by the system as additional objects in the scene.

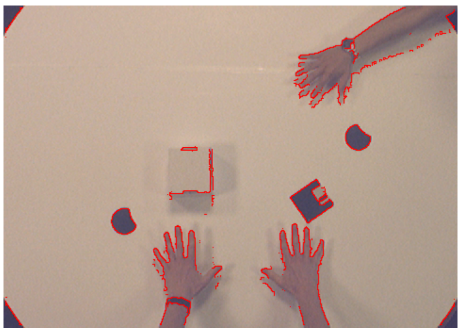


Fig. 6.1: Inaccuracies in edge detection caused by shadows.

This chapter aims to provide an overview of possible solutions to the problem of shadow removal. It also presents a review of one particular method that shows potential and could be used as a pre-processing stage for other systems. This is then extended and applied in chapter 7.

6.2 Background

The majority of shadow removal techniques tend to be rather complex and are therefore not aimed at real-time application but instead deal with post-processing of single images or in some cases sequences of images. However, with the most recent advances in processing power it is feasible that some of these techniques may be suitably adapted for use in a VAE.

A technique of real-time shadow removal that is concerned with tracking and pose estimation of people is given in [29]. This method is based around image differencing and provides a way to remove segmented pixels that are a result of shadowing effects. It describes an algorithm called colour normalised cross correlation which produces a brightness and contrast invariant comparison of the texture of a region in the reference image and the same region in the current frame. If the textures are similar then the region is likely to be a shadow on the background and the corresponding pixel is therefore removed from the segmented image.

The term "intrinsic images" refers to a set of images in which each represents an intrinsic characteristic and was first used by Barrow and Tenenbaum [6]. For example, it would be possible to represent a scene in a photo by the reflectance of objects within the scene and the quantity of illumination across the scene; taking the product of these two images effectively results in the original image. If the intrinsic images can be determined from the original image then they could prove very useful in further image processing stages: A reflectance image is ideally suited to tasks such as edge detection since shadows are not present.

The primary problem remains that determining these intrinsic images is not a simple task. Barrow and Tenenbaum suggest a method that actually uses edge classification by analysing pixels on either side of an edge. These edges are used to initialise the corresponding intrinsic images and certain local constraints are then applied (such as continuity) to adjust all other values within those images iteratively.

Weiss [72] describes a method that requires a sequence of images in which the reflectance is constant (camera and subject remain the same) and the illumination varies (such as change in sun position). It is based on the statistical assumption that derivative filters applied to a natural illumination image tend to result in a sparse histogram. By applying a temporal median filter to the derivatives of the original image both the reflectance and illumination images can be estimated. This method is not suitable for real-time processing since it requires multiple images to produce a single reflectance image.

A method capable of estimating intrinsic images from a single image is presented in [66]. This is also based around some of the assumptions described by Weiss except that it uses both colour and greyscale information to determine whether pixel value changes are due to reflectance or shading. In colour space, changes due to shading tend to be proportional across all three colour channels (RGB) whereas changes due to reflectance are not. A greyscale classifier is trained using sets of images containing no shading and others containing no reflectance. This relies on the assumption that local patterns due to shading have a unique appearance which is clearly different to most reflectance patterns. After using both classifiers there are still some parts of the images which are ambiguous and so contours and probabilities are used to propagate known information.

An alternative improvement over the Weiss method is proposed in [12] which works with a single image. Three metrics are applied to derivatives of the original image — intensity, blur and chromaticity. These metrics exploit different properties that result from shading and reflectance and can then be combined to more accurately classify pixels.

Another technique [61] that is only concerned with detecting the shadow itself also begins by classifying edge pixels. By calculating an edge map for each colour channel independently the gradients in each map can be compared. An edge pixel is classified as shadow where the gradients are in the same direction since it is assumed that shadowing affects each colour channel proportionally. The edge pixels are further refined by eliminating unconnected elements.

The ShadowFlash method [78] can be used to analyse pairs of images that either show the independent effects of two light sources or a single light source in two different positions. However, the limitation of only handling up to two light sources means that it would be unsuitable for use in a VAE system. ShadowFlash could be extended to cope with a greater number of light sources but it would then require the same number of images as sources where each image shows the result of a single source. Since this would be needed for each frame it is impractical.

6.3 Invariance Method

6.3.1 Colour Constancy

In 2001 Finlayson and Hordley introduced a novel method of calculating invariance images [21]. Instead of generating a 2D colour constancy image they proposed a 1D invariance image based on two straightforward assumptions. Firstly, that illuminant chromaticities lie on a Planckian locus and secondly, that a camera sensor response is a Dirac delta function (i.e., each sensor for red, green and blue has an impulse response at a specific wavelength).

By applying these assumptions Finlayson and Hordley show that a variation in illumination corresponds to a linear shift in Log-Chromaticity (LC) space. This shift is parallel for all reflectance chromaticities and therefore rotating the colour space so that the shifts are vertical will result in a graph in which the x-axis corresponds to surface reflectance and the y-axis corresponds to illumination. Converting only the values along the x-axis into a greyscale range produces an image that is invariant to illumination.

They then prove that although most illuminants do not lie perfectly on a Planckian locus they do tend to lie close enough for the assumption to remain valid. Camera responses, on the other hand, vary significantly and each sensor could actually respond to a reasonably wide bandwidth of light. However, the findings in [21] show that if the correct calibration angle for a camera can be found then calculating an invariance image with reasonable accuracy is possible.

6.3.2 Entropy Minimisation

In later work, Finlayson et al. describe a method of automatically selecting an illumination invariant angle using entropy minimisation [20] which removes the requirement for camera calibration. After calculating an intrinsic image they then build on previous work [22, 17, 26] to reconstruct a full colour image without shadows. However it is the construction of a 1D invariance image which is of interest since many common image processing tasks, such as edge detection, only require greyscale images.

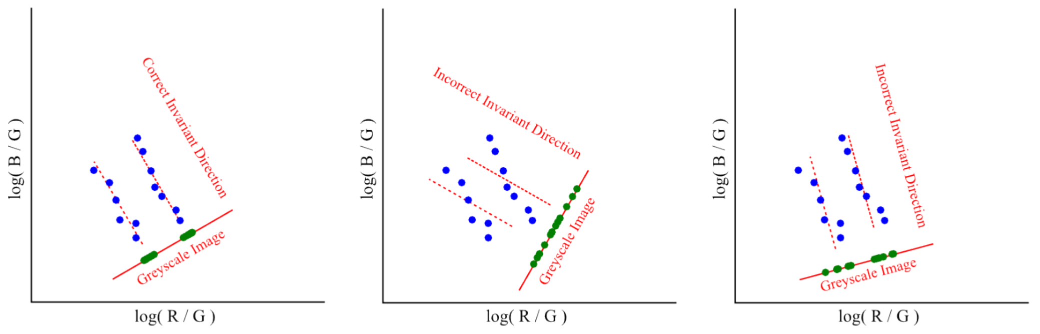


Fig. 6.2: Entropy minimisation example (based on a similar diagram in [20]).

The graphs in figure 6.2 provide an example of distributions in LC space. The blue dots represent the colour of individual pixels from the source image, the red dashed lines represent the invariant direction, the solid red lines are perpendicular to the invariant direction and the green dots represent the projection of the blue dots onto the perpendicular line.

The first graph shows the effect of projecting onto the perpendicular line when the invariant direction has been correctly identified — the pixels are tightly clustered with those of similar reflectance. The second and third graphs show the result when the invariant direction has been incorrectly identified — the pixels are spread out along the perpendicular line.

It is clear from the graphs that at the appropriate invariant angle the resulting projection to 1D results in an image with the least amount of entropy. This suggests that an estimate of the invariant angle can be found by calculating the entropy of each 1D image over a range of invariance angles and finding the minimum. Finlayson et al. go on to show that this is a reliable method of calculating the invariant angle for their test camera and further results show that it should apply to any camera.

6.3.3 Retinex

In a later paper [23] Finlayson et al. present an adaption to Land's Retinex algorithm that allows removal of shadows from full colour images.

Retinex theory uses human vision as a model for understanding how we interpret colour and shading information. The algorithm looks at relative pixel values in each colour channel independently by averaging the ratios of one pixel to many other pixels. By thresholding the ratios such that values close to 1 are set to 1, the method effectively removes gradual illumination changes across an image.

Finlayson et al. propose a modification to the thresholding that incorporates shadow edge information retrieved by comparing an invariance image with the original image. By setting the ratios of pixels that lie on a shadow edge to 1 alongside the aforementioned threshold, both gradual changes of illumination and abrupt changes caused by shadows are removed from the image.

6.4 Implementation

A C++ implementation of Finlayson's invariance method was developed. It begins by calibrating for the first captured frame using the reviewed entropy minimisation technique. Varying an angle in 1° steps and projecting values from LC space onto a line perpendicular to that angle produces a 1D image. By calculating the entropy of each 1D image over the range of angles the system can then pick the point of least entropy as the invariant angle to be used in all further processing.

Every frame is then transformed into LC space where each pixel is calculated using equations (6.1) and (6.2)

$$C_1 = \ln R - \ln G \tag{6.1}$$

$$C_2 = \ln B - \ln G \tag{6.2}$$

where C is the LC value of the pixel and R, G, B are the red, green and blue values of the pixel respectively. This part of the implementation is easy to optimise since the range of values in each colour channel is only 0–255 and so the corresponding logarithms can be stored in a small lookup table.

The LC values are then projected along the calibrated angle to produce an invariance image using equation (6.3).

$$I = \exp(C_1 \cos\theta + C_2 \sin\theta + 4.844) \tag{6.3}$$

where I is a pixel in the invariance image, θ is the invariant angle and the value 4.844 ($\ln 127$) adds a constant illumination back into the image.

6.4.1 Preliminary Results

Figure 6.3 presents the result of applying the invariance algorithm to a simple image with a strong shadow lit from a single light source (incandescent). Although the invariance image is noisy it is very clear that the shadow has been successfully removed.

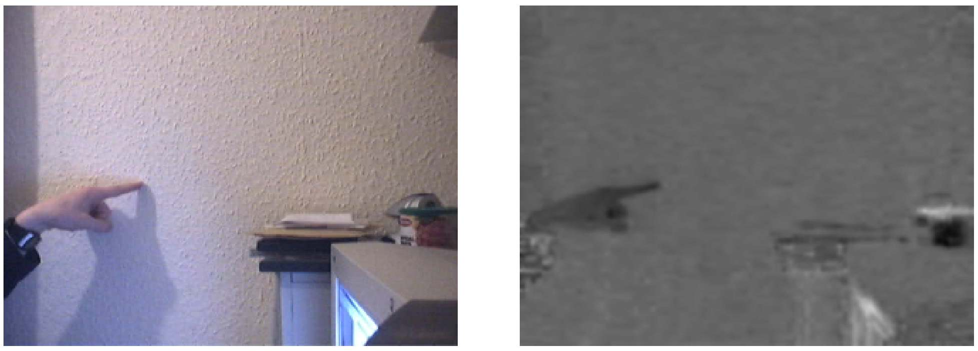

(a) Captured frame (b) Invariance image

Fig. 6.3: Results of invariance method with a single light source.

The next figure (6.4) shows a frame captured from a VAE and the resulting invariance image. In this case multiple overhead light sources were present (flourescent) causing numerous shadows. Only some of the shadows have been removed in the invariance image, but even with those that are left the difference in value between background and shadows has been reduced whilst the difference between hands and shadows has been increased. This will allow for much clearer segmentation of hands in later stages of image processing.

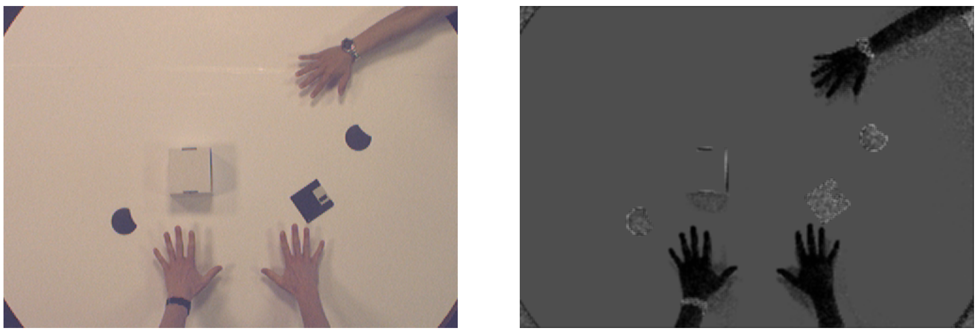

(a) Captured frame (b) Invariance image

Fig. 6.4: Results of invariance method with multiple light sources.

The objects placed on the table actually become less clear in the invariance image. This is due to the fact that all greyscale values (i.e., those of equal red, green and blue components) are 0 in LC space and the objects themselves are black and white. With the background surface also being white the differences between objects and surface become far less discernible in the invariance image.

6.5 Conclusions

The invariance method was originally developed by Finlayson et al. as a post-processing stage for photos. However, it has been found that with current processing capabilities it is possible to perform invariance image calculation on live video.

Initial results look promising with the method working very well on images where the scene was illuminated by a single light source. Where multiple light sources are concerned the algorithm seems unable to completely remove all of the shadows but it does appear to have reduced the effect of those that are left.

The production of full colour images without shadows requires that the adapted Retinex algorithm be applied to each colour channel separately. This is currently too processor intensive for use on live video within a VAE infrastructure.

6.5.1 Further Work

In [20] they suggest an improvement over $\left[\log\frac{R}{G}, \log\frac{B}{G}\right]$ by using the geometric mean ($\sqrt[3]{R \times G \times B}$). Dividing by the geometric mean instead of a single colour channel removes the dependency on one colour, which could cause problems if that colour is not very prevalent in a particular image. Since this chapter is considering methods suitable for real-time applications this improvement would seriously affect processing time; the log values of individual colour channels can be optimised with lookup tables but the calculation of the geometric mean cannot. However, informal results during the development of the finger detection system show that using the green channel as a reference provides reasonable results.

The main concern with the method described here is the time taken to determine the invariant angle. The use of a search method such as gradient descent could provide a performance increase.

The production of a greyscale image from the invariance projection could benefit from further investigation. Currently the implementation used in the finger detection method simply adds a constant value before conversion to greyscale so that artificial light is added to what is otherwise a purely reflectance image. Finlayson et al. suggest a way of calculating how much light to add back into the equation based on the brightest pixels in the original image since they are assumed to not be in shadow.

Chapter 7

Finger Detection

7.1 Introduction

The principal feature of a VAE is intuitive user interaction. Since this research has focussed on an infrastructure for tabletop projection this user interaction is normally concerned with hands. Therefore finger detection capabilities should be a core feature of any VAE framework and hence this chapter considers some of the currently available techniques, then it proposes a new method based around invariance images.

7.1.1 Requirements

A finger detection system for use in an augmented environment application must be efficient so as to be responsive and not adversely affect the performance of the rest of the system. It must also require little manual calibration, preferably none at all, so that it can be inserted into a framework with ease. The system must be resilient to the effect of various lighting conditions and shadows.

7.2 Background

The majority of current finger detection techniques rely on either background differencing [41] or skin colour detection [1] for the initial stage of image processing. Other methods require the use of gloves or markers [15, 10] but this section will not be considering those since a user should be able to walk up to a VAE system and begin interacting without the need for extra equipment.

A colour detection method described in [1] distinguishes hands using a Bayesian classifier plus a small set of training data. It then finds the contours of the hands and employs a curvature analysis algorithm, at a variety of scales, to determine peaks which could correspond to fingertips.

Using a mask to perform template matching [14] is another way to detect fingertips. This requires that the system already has a template with which to match and orientation also becomes an issue. Since a finger could be pointing in any direction the template must attempt to track not only the position of the fingertip but also the orientation by updating the template. If the tracking of a fingertip is lost it can be very difficult to re-acquire and also the algorithm could be rather computationally expensive.

FingerMouse [53] was a system developed to allow a pointing finger to control the cursor of a desktop machine thus removing the need for a mouse. The purpose was to eliminate the time wasted by moving a hand back and forth between keyboard and mouse. Their method involved segmentation via a probabilistic colour look-up table which required training (user-specific) and then a principal axis-based fingertip detection algorithm.

A method is presented in [43] that works with plain background, greyscale images. It involves spatial filtering and then a trained neural network to detect the fingertip locations. The paper provides results to show that it works even when there is very low contrast, such as when a finger is viewed over the palm of a hand.

Using an infra-red camera can provide a very clean binary image for further processing. In [62] they make use of this type of camera in an augmented desktop scenario and then perform fingertip detection using template matching within the estimated hand region.

7.3 Letessier and Bérard Method

This chapter focusses on a technique developed by Letessier and Bérard [39] that combines a method for image differencing and a simple, yet effective algorithm for the detection of potential fingertips that they have named the Fast Rejection Filter (FRF). The FRF does not rely on template matching, special gloves or hand-colour detection but a simple set of rules to classify pixels. It is efficient, straightforward to implement and thus a good choice for use in an augmented environment.

The FRF is concerned with detecting fingertips but not hand shape and is therefore unable to detect fingers that are pressed together.

Image Differencing

Letessier and Bérard propose a method for image differencing which is based upon a colour-space metric referred to as Chrominance Euclidean Distance (CED). The following equation (7.1) provides a summary of the algorithm used for CED where p refers to a pixel in the current image and R, G, B the corresponding channels, p' refers to a pixel in the background image and R', G', B' the corresponding channels.

$$d(p,p') = \left\| \left[\frac{R}{R+G+B}, \frac{G}{R+G+B}\right] - \left[\frac{R'}{R'+G'+B'}, \frac{G'}{R'+G'+B'}\right] \right\| \tag{7.1}$$

One of the reasons for using chrominance space is that it should reduce the effect of shadows during processing. This may work in a laboratory situation where the lighting can be controlled, but in a more common situation it may not be as reliable. An example is a room with overhead lights combined with daylight admitted through windows.

The final stage of image differencing is thresholding which will provide the finger detection algorithm with a binary map of changed pixels to analyse. The CED method described in this section results in a difference image with a histogram typically containing two modes. Therefore the threshold must be chosen to separate those modes since the lower one represents background noise. Letessier and Bérard describe a threshold based on the median and median absolute deviation.

Background Maintenance

To be able to perform image differencing a background image must be stored for comparison. The simplest way to maintain a background image is to use a recursive temporal filter as follows:

$$Bg^{t+1}_{(x,y)} = \alpha Im^{t}_{(x,y)} + (1-\alpha)Bg^{t}_{(x,y)} \tag{7.2}$$

The basic equation (7.2) uses a learning rate (α) to impart a proportion of power from each pixel in the current frame (Im) to the background image (Bg). This simple method, however, produces some unwanted problems such as dark areas of the image not updating as quickly as lighter areas. Results can be improved somewhat by biasing the learning rate using results from the image differencing. This leads to a slightly modified background formula (7.3), as suggested in [39].

$$Bg^{t+1}_{(x,y)} = \alpha^{t}_{(x,y)} Im^{t}_{(x,y)} + (1-\alpha^{t}_{(x,y)})Bg^{t}_{(x,y)} \tag{7.3}$$

In this case the learning rate is adapted on a pixel-by-pixel basis depending on the image differencing results.

Finger Detection

The FRF is a simple set of criteria applied to each pixel in the thresholded difference image and is an improvement over the earlier filter proposed in [71]. It provides the basis for finger detection in both the method currently being described and the proposed method for this chapter. This method of pixel classification means that the algorithm is simple, robust and there is no requirement to implement a more complex object recognition system or employ template matching. The filter, as proposed in [39], relies on the following criteria:

1. Pixel is not background.
2. Pixel is connected to a region of foreground pixels large enough to be a hand.
3. Pixel is completely surrounded by foreground pixels at a radius small enough to fit inside a fingertip.
4. While scanning at a radius large enough to fit around a fingertip exactly one arc of foreground pixels and one arc of background pixels are found.
5. The distance between the two endpoints of the foreground arc found in 4 is approximately the size of a finger.

This method is based on a simple model of the finger (a semi-circle connected to a rectangle) and because it relies on radial scans it is not affected by finger orientation which can cause serious problems for other methods such as template matching. The output of the FRF is an image that indicates at which stage each pixel was rejected or whether it is potentially part of a fingertip. A simple fill algorithm can then be used to find areas of pixels that pass all of the criteria. Any region that is large enough is classified as a fingertip and the centroid for that region provides the coordinates.

7.3.1 Problems

As described earlier, existing methods tend to rely on a system of background differencing and then perform the actual finger location on the resultant image. There are two major problems with this method, first of all the background image needs to be maintained and it usually requires a constant as in equation (7.2). This constant affects how quickly something becomes part of the background and so if set incorrectly it can cause a hand to disappear or an object that, long after it has been moved, remains visible in the original position. The second problem is that caused by shadows; a shadow in a live image appears as a change along with the hand itself and so can obscure features such as the actual edges of a hand.

The technique described in section 7.3 approaches these problems by using an adaptive method of background maintenance, as in equation (7.3). It also uses CED instead of RGB euclidean distance in an attempt to reduce the effect of shadows.

Another problem that will affect any system that relies on the FRF is to do with scale. A typical VAE setup could result in the camera being positioned high above the area of interest which would mean that hands appear small and fingertips could end up as being only a few pixels in area. This would make the task of eliminating false positives based on the region size far more difficult, although this can be partly overcome through the use of temporal filtering and tracking.

Their results seem to show that this method works reliably, but tests performed as part of this research found problems. A VAE must be able to work well under any lighting condition and the most common situation is that with multiple light sources. The test setup used consisted of overhead fluorescent lights plus daylight entering through windows on one side of the room. It was found that the CED method did not deal with shadows as well as described. It also had severe problems with very dark areas of a frame causing a large amount of noise in the corresponding difference image. This is a common problem with chromaticity methods due to variations in small values being amplified in log space.

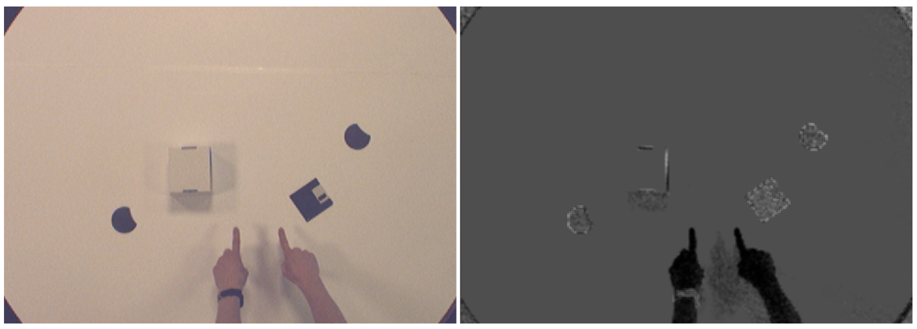
7.4 Proposed Method

7.4.1 Overview

Ideally finger detection should be resilient to shadows and not rely on background images. Therefore, if a user does leave their hand in one location for an extended period of time, it would not disappear. This section proposes a technique involving a combination of plane calibration (section 5.5.4), shadow removal (chapter 6) and the FRF (section 7.3).

7.4.2 Analysis of the Invariance Image

Figure 7.1 shows the result of applying plane calibration and the invariance method presented in the shadow removal chapter to captured frames in a VAE.


(a) Overhead lighting only

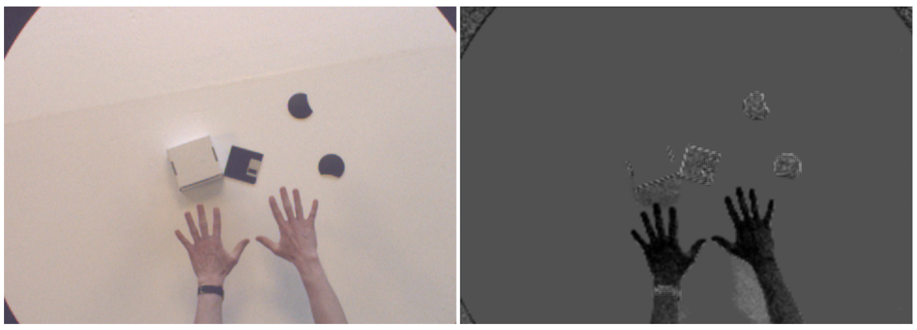

(b) Overhead lights and bright daylight

Fig. 7.1: Captured frames (left) and the corresponding invariance images (right).

Both images show that the plane calibration results in a very uniform background which remains that way even after conversion to an invariance image. Shadows still remain visible however since the invariance method can only minimise the effect of shadows caused by a single light source and even the overhead lighting will vary as each fluorescent bulb could be a different colour temperature. In the majority of cases the calibration of the invariance method simply attempts to reduce the impact of all of the shadows as much as possible.

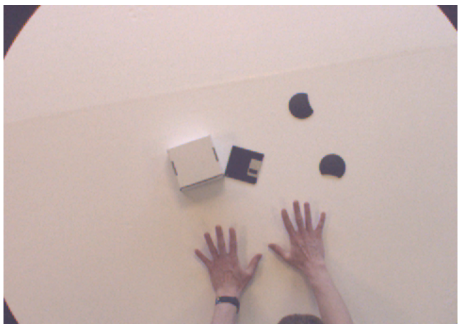
Although the shadows in the upper image of figure 7.1 remain it still represents a significant improvement. The difference in greyscale values between the hand and the shadow in the original frame is approximately 20 levels whereas in the invariance image this is more like 60 levels. This difference means that the invariance image can be thresholded with less chance of overlapping values causing distortion of the hand shape.

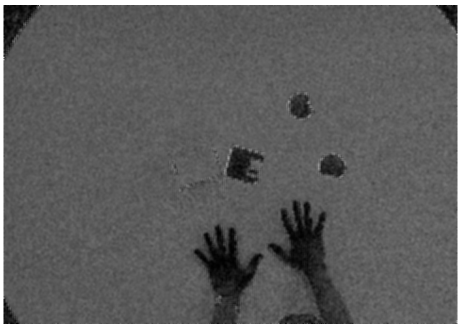
In the lower image of figure 7.1 it can be seen that the dominant shadow is actually lighter than the background and this is caused by the daylight coming through the windows. Daylight has a colour temperature that is further towards the blue end of the spectrum than the internal lights. Therefore the colour space conversion results in the pixel values of daylight shadows moving in a different direction to values of other shadows. In this particular case, where the shadow is lighter than the background in the invariance image, thresholding becomes a simpler task since a threshold to distinguish hand from background will also completely eliminate the shadow.

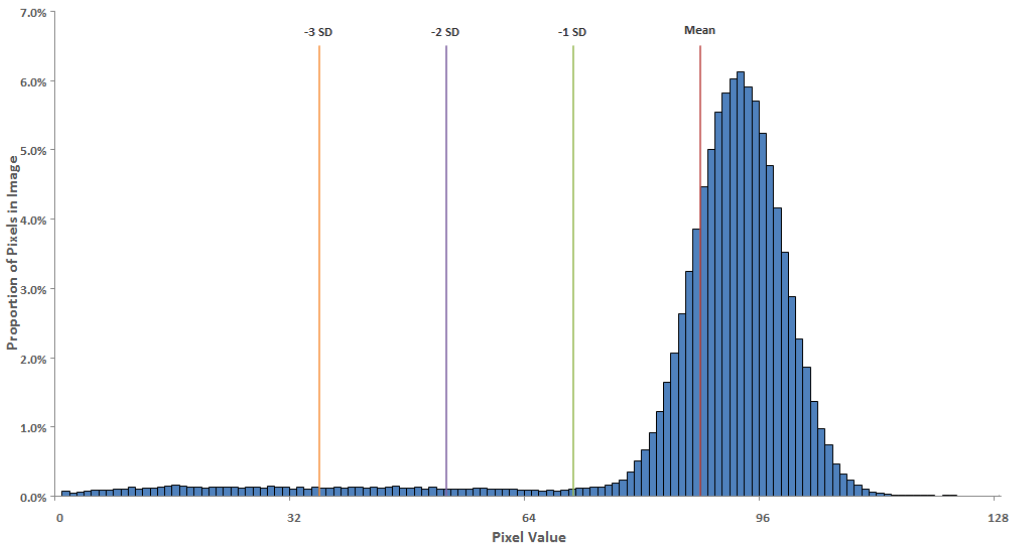
7.4.3 Thresholding

It is difficult to visually analyse the histogram of a thresholded invariance image since the plane calibration results in a large spike at the value for the background. This spike causes the rest of the histogram to be scaled down so much that the detail is lost; instead, an invariance image that has not been subjected to plane correction is analysed (figure 7.2).

The red line on the histogram represents the mean pixel value and the


(a) Captured frame


(b) Invariance image



(c) Histogram

Fig. 7.2: Results of invariance method without plane correction.

other lines show -1, -2 and -3 standard deviations (SD) from the mean. With the background being clearly shown in an almost Gaussian distribution it has been found that a stable and effective threshold can be automatically selected based on a multiple of standard deviations from the mean. The following (figure 7.3) is an example of the invariance image from figure 7.2 after thresholding at a level of 3 SD below the mean.

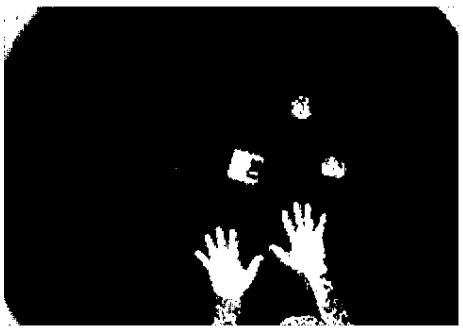


Fig. 7.3: An invariance image thresholded at -3 SD.

This thresholded image is then at a stage where it can be passed through the FRF to determine possible fingertips within the image. Figure 7.4 is the result of thresholding the invariance image from chapter 6 and applying the FRF. The colours represent the stage at which a pixel was rejected:

1. *White* - Pixel is background or is not connected to a region of foreground pixels large enough to be a hand.
2. *Black* - Pixel is not completely surrounded by foreground pixels at a radius small enough to fit inside a fingertip.
3. *Green* - While scanning at a radius large enough to fit around a fingertip exactly one arc of foreground pixels and one arc of background pixels are not found.
4. *Blue* - The distance between the two endpoints of the foreground arc is not the approximate size of a finger.
5. *Red* - This pixel could be part of a fingertip.

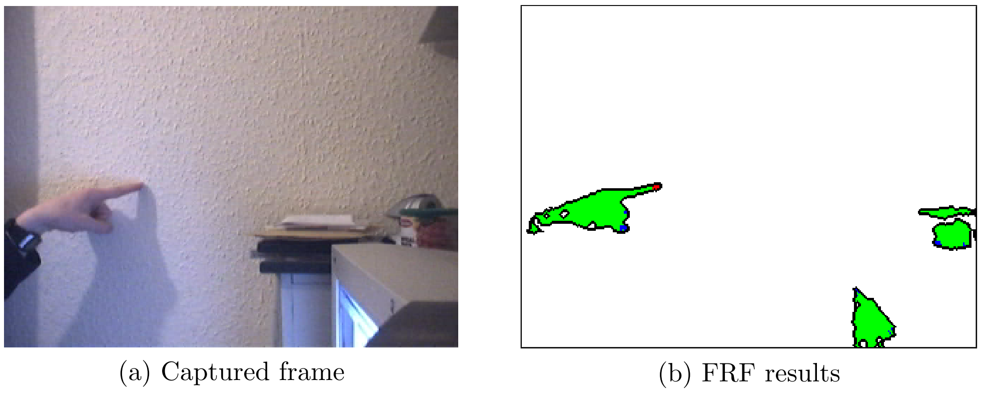

(a) Captured frame (b) FRF results

Fig. 7.4: An invariance image after thresholding and filtering where the different colours represent the stages of rejection for each pixel.

7.5 Implementation

7.5.1 Overview

Three test programs were developed to aid in the development and testing of the invariance-based finger detection. The first (section 7.5.2) provides live frame capture and initial image processing, the second (section 7.5.3) provides a utility to manually select fingertip locations and the third (section 7.5.4) provides a framework within which to fully test and compare the finger detection methods.

7.5.2 Frame Capture

An application was developed to test the Letessier-Bérard (LB) method of finger detection on live video. It was then later adapted to allow experimentation with the invariance method and to capture frames to disk. It was developed to provide full-screen rendering capabilities on the primary display plus a feedback and control window on the secondary, so that when run on a VAE with a projector connected to the primary display it provides a good testing platform. The frame-grabber library from *OpenIllusionist* was used to provide the program with live video images and elements of the *OpenGL* renderer were also utilised.

The renderer can be used to generate calibration feedback for whatever algorithm is being used and in this case it assists in selecting the area of interest when performing plane calibration. When switched to capture mode, the application begins saving consecutive frames to disk. For each frame a number of images are produced:

- Original image
- Original image with mask to show area of interest
- Plane calibrated image with mask

7.5.3 Finger Verification

This is a simple application that allows a sequence of images (generated by the frame capture program) to be loaded one at a time. The user can then manually click the image in the centre of every fingertip present (see figure 7.5). This data is saved out to a text file and provides the ground truth for later tests.

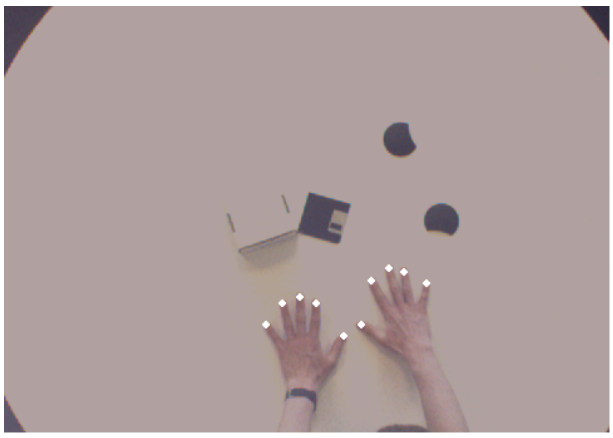


Fig. 7.5: An image being processed by the finger verification application, the white spots indicate where the user has clicked to specify the location of a fingertip.

7.5.4 Testing Framework

The testing framework is an application which allows both the LB and invariance methods of finger detection to be run and the results to be automatically compared against the ground truth data. It provides the ability to vary parameters within each method to produce a large set of results for a group of images. Since this can be rather time consuming, the program was adapted so that each method runs in a thread thus improving efficiency when run on a multi-processor system.

Both methods share a common class which contains functions to perform the final stage of the finger detection (the FRF) as well as functions to load and check against the ground truth data. This means that the tests are fair and will prove which method generates the most effective binary images for the filter.

7.5.5 Test Setup

Three sets of images were captured in different lighting conditions (see table 7.1) and the ground truth data manually generated. The daylight entered the room from windows along one side and so interacted with the scene at an acute angle; this produced elongated shadows and caused shifts in the ambient light level on the table when a person moved to that side.

Lighting Condition	**Number of Images**
Overheads only (fluorescent)	138
Overheads and overcast daylight	126
Overheads and bright daylight	99

Table 7.1: Table showing the size and type of test image sets.

The testing threads were setup as in table 7.2 where "first image" represents what the method does with the first image in each set. The parameter type is a description of the variable parameter for that method, the range is the range of values used for testing and the step is the step-size used to iterate through that range. The finger scale range and step are common to both methods since the scale has not been calibrated, but the variation of that parameter provides a good indication of whether a particular method works well regardless of the finger scale accuracy.

	LB Method	**Invariance Method**
First Image	Set as the background image	Used to calibrate the invariance angle
Parameter Type	Median Absolute Deviation multiplier	Standard Deviation multiplier
Parameter Range	0.0 to 25.0	0.0 to 5.9
Parameter Step	1.0	0.2
Finger Scale Range	1.0 to 4.0	1.0 to 4.0
Finger Scale Step	0.02	0.02

Table 7.2: Table showing the ranges over which parameters were varied.

7.6 Results

The results are in two sections of which the first is concerned with optimisation and the second with comparison of the methods. The optimisation stage was performed using all of the even numbered images in the test set and then the comparison with the odd numbered images.

7.6.1 Optimisation

Figures 7.6 and 7.7 show the number of true positives found by each method over the parameter ranges given in table 7.2 under each lighting condition. Both methods have a definite curve at which the peak represents the optimum parameter value for finding true positives.

The next figures (7.8 and 7.9), however, are not so clear since the production of false positives is a little more erratic. In the LB method an increase in the threshold parameter corresponds to a continuous drop in the number of false positives found and yet in the invariance method the number of false positives drops and then increases again. This could be due to breaking up

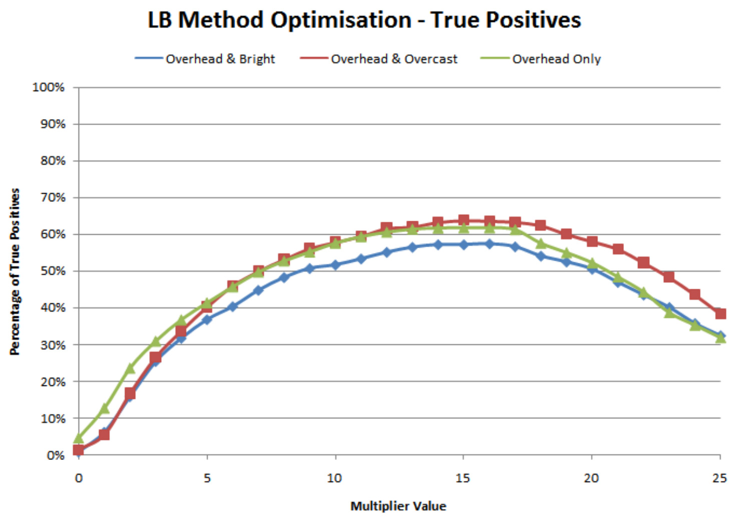


Fig. 7.6: Optimisation of LB method — True positives.

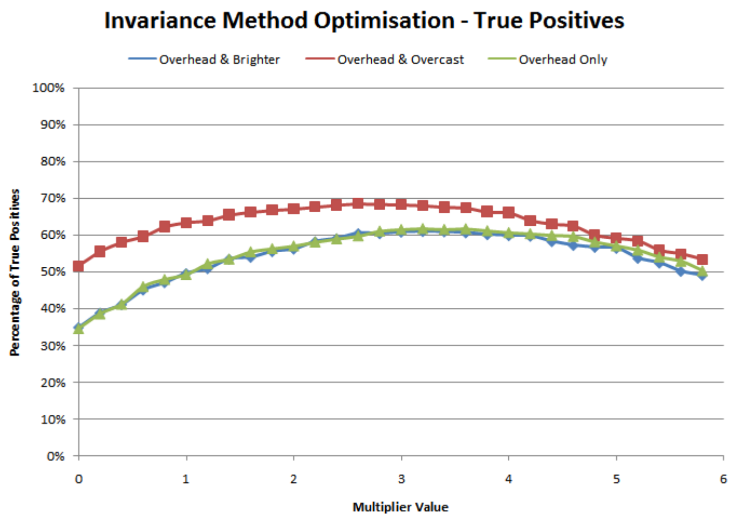


Fig. 7.7: Optimisation of invariance method — True positives.

of foreground areas as the threshold increases thereby creating smaller areas that are roughly the size and shape of fingers.

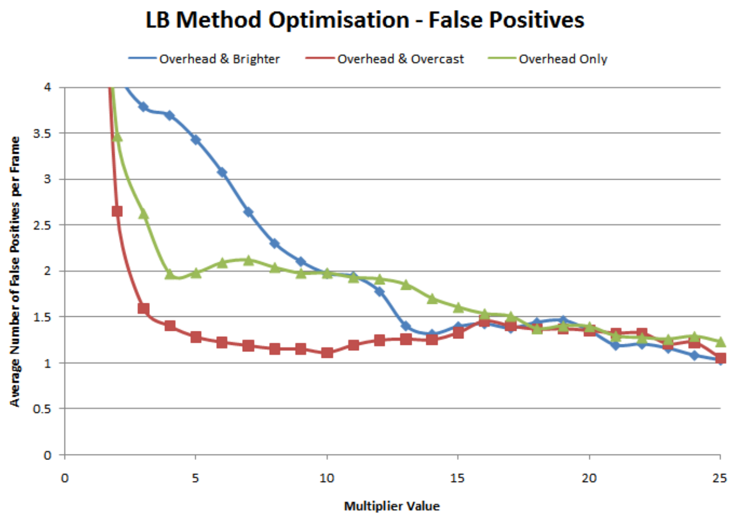


Fig. 7.8: Optimisation of LB method — False positives.

The final graph (figure 7.10) in this section is a combination of the data used in the previous ones. It shows the percentage of true positives against the average number of false positives per frame for each threshold parameter value across both methods. The labelled point is that of highest true positive rate which also corresponds to a level of false positives that is low enough to be ignored assuming an algorithm is employed to track the fingertips. The indicated threshold parameter values are used in the following section when comparing the two methods.

This graph and the following ones are similar to Receiver Operating Characteristics (ROC) graphs commonly used in signal processing, however they differ in that the true positive scale is along the horizontal axis and the false negative scale has not been normalised.

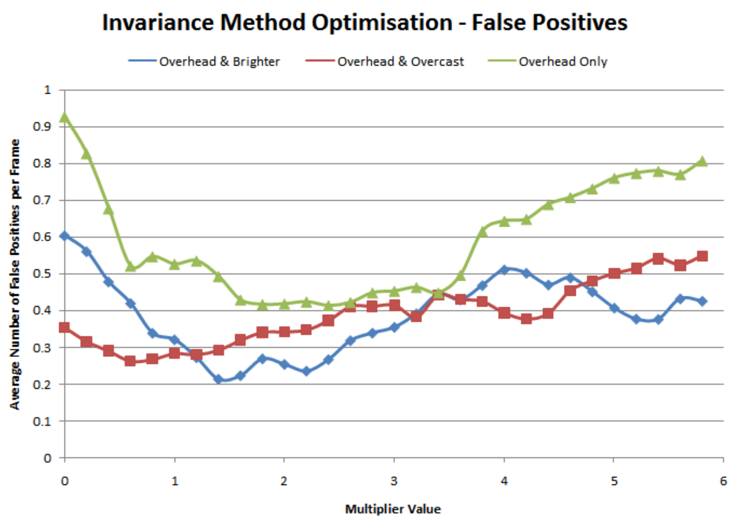


Fig. 7.9: Optimisation of invariance method — False positives.

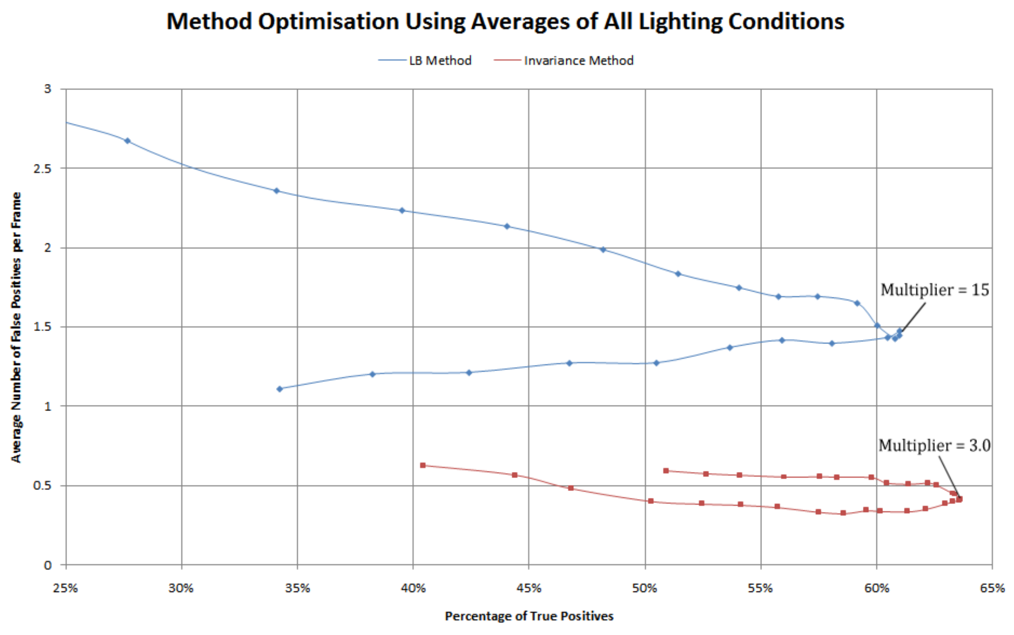


Fig. 7.10: Optimisation of methods — True positives against false positives for all lighting conditions.

7.6.2 Comparison

The following graphs (figures 7.11, 7.12 and 7.13) are scatter plots where each point represents the percentage of true positives against the average number of false positives per frame for a particular finger scale value. The blue points correspond to the results of the LB method and the red points to those of the invariance method.

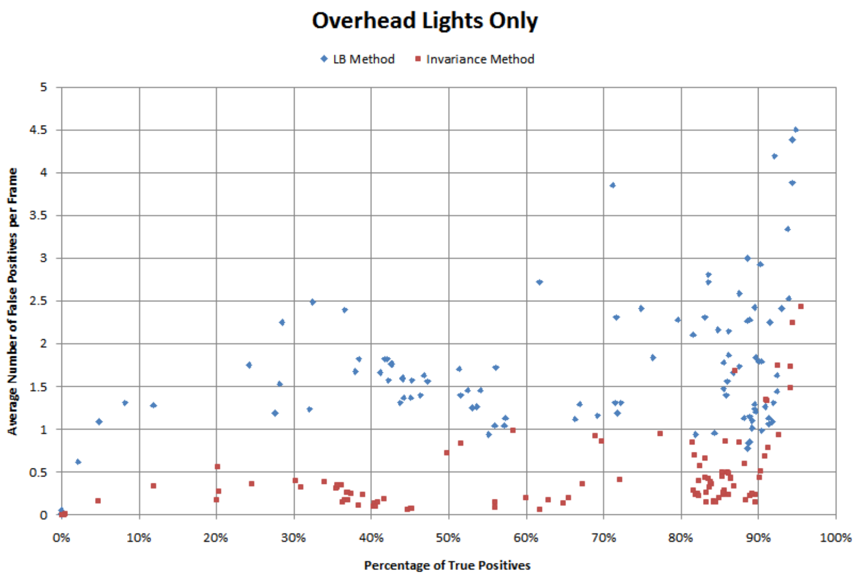


Fig. 7.11: Comparison of methods after optimisation — Overhead lights only.

It is clear from these that the invariance method performed better than the LB method overall since it managed to achieve higher true positive values and lower false positive values over a wide range of finger scales and under different lighting conditions.

The images in figure 7.14 are an example of the output from the proposed finger detection algorithm; the white dots represent detected fingertips.

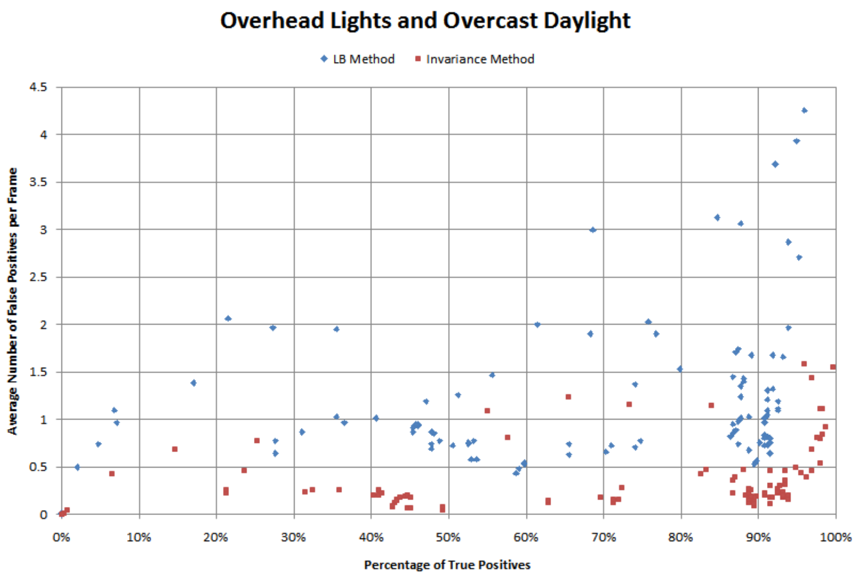


Fig. 7.12: Comparison of methods after optimisation — Overheads and overcast daylight.

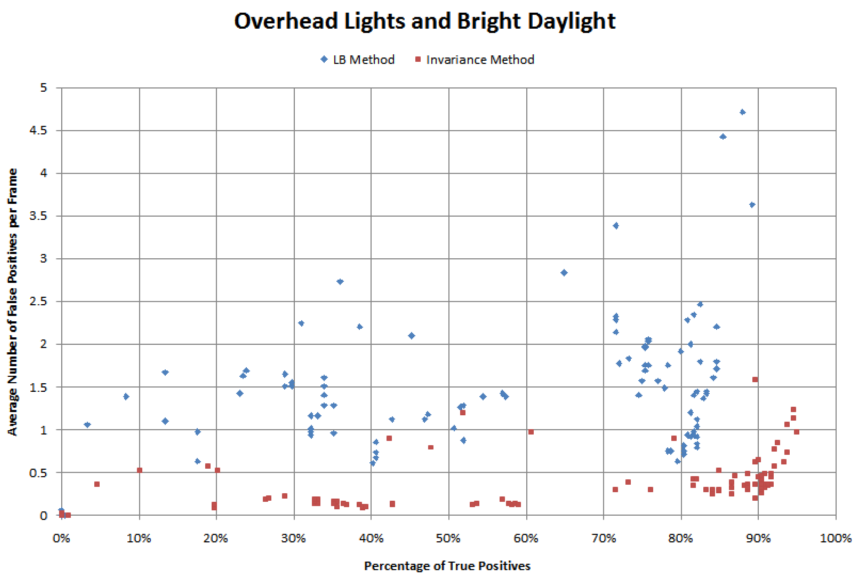


Fig. 7.13: Comparison of methods after optimisation — Overheads and bright daylight.

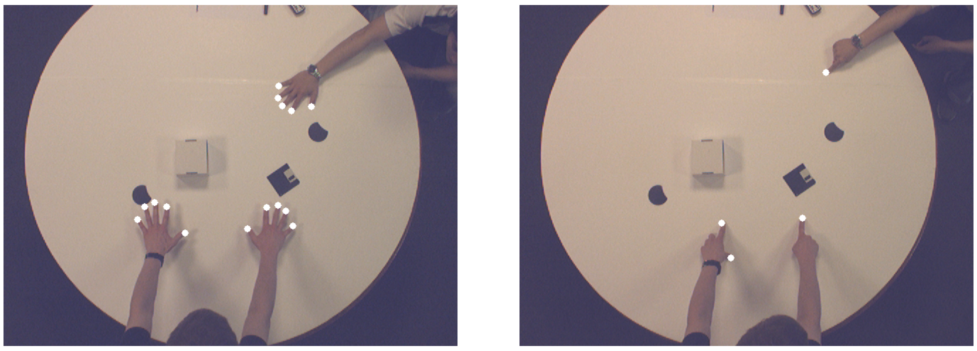


Fig. 7.14: Example output from the proposed finger detection algorithm.

7.7 Conclusions

This chapter shows that the invariance method of finger detection is at least as effective as the more common background differencing methods and in fact provides better results in complex lighting situations. There is a greater separation in the effectiveness of both methods in the second two graphs (7.12 and 7.13) where the lighting was more varied than in the first (7.11) which had only overhead lights. The situation with only overhead lighting is approaching a controlled laboratory condition whereas the other two are far more likely scenarios when trying to set up a VAE as an exhibit. Therefore the invariance method is definitely worth investigating further, not just for finger detection but for other image processing tasks as well.

7.7.1 Problems

One of the main advantages of the LB method is that it does not require any calibration (except perhaps finger scale, but that is common to both methods). Unfortunately it does have the overhead of having to maintain a background image which can hamper the performance of the algorithm. The invariance method, however, requires an initial calibration to find the optimum angle at which to transform from 2D LC space to 1D invariance space (refer to chapter 6). Fortunately this calibration stage can be automated and would therefore not require user interaction to initialise it.

Another minor problem is that of selecting which side of the mean to threshold; as shown in figure 7.1 some shadows can actually appear on the opposite side of the mean to the hands. Therefore it can be beneficial to ignore that half of the histogram and thus ignore the corresponding shadows altogether. But how can the system tell which side of the histogram contains hands and which side shadows? They could be either way around depending on the angle that is chosen during the calibration stage. This may have to be solved through user interaction although an automated method would be preferable and so further investigation is desirable.

7.7.2 Further Work

Testing under a greater variety of lighting conditions would be beneficial as would comparisons with other finger detection systems. Since finger detection is required for a VAE infrastructure it would also be worth testing how well the system performs when projecting onto the surface. Coloured light from a projector could cause problems in LC space and thus corrupt the invariance images.

The proposed technique for finger detection could be incorporated into *OpenIllusionist* as a core feature. This would provide a developer with an additional input to an application and would also present a familiar method of interaction for the user.

Chapter 8

Robot Ships: An OpenIllusionist Application

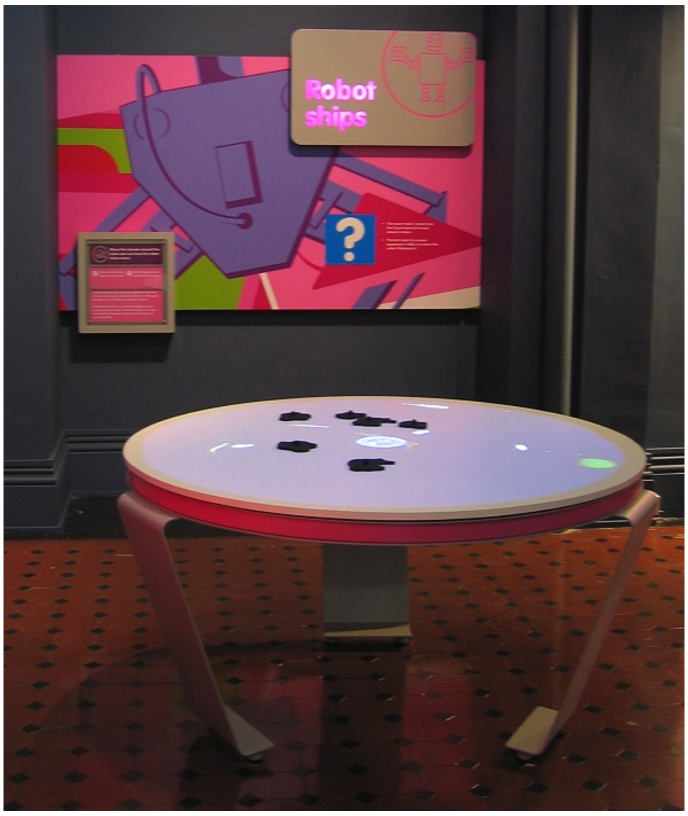


Fig. 8.1: Photo of the installation at the Royal Museum in Edinburgh.

8.1 Introduction

Robot Ships is an interactive exhibit on permanent display in the "Connect" gallery of the Royal Museum in Edinburgh. It is an example of using groups of simple autonomous agents to perform a more complex task, in this case the clearing up of a simulated toxic oil spill. The application is loosely modelled on the process with which ants collect food by laying down a trail of pheromones that other ants can follow.

Figure 8.1 shows the exhibit running in the museum and figure 8.2 illustrates the rendered output of *Robot Ships*. This chapter provides an overview of the *Robot Ships* application, an outline of the development and some evaluation results.

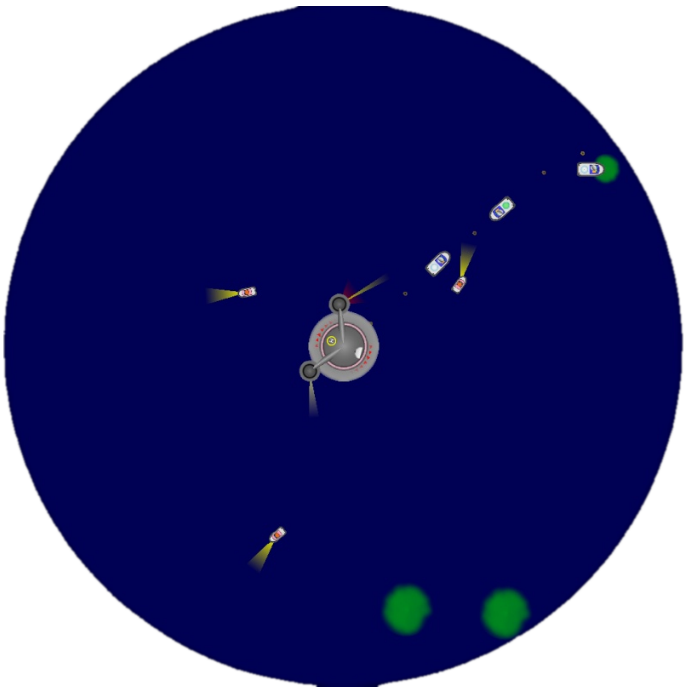


Fig. 8.2: Screenshot of *Robot Ships*.

8.2 Overview

Robot Ships has been developed with six agents, three of which are stationary and three that are mobile.

- *Queen*
  A stationary agent that is displayed as the base in the centre of the world. It is responsible for keeping track of all of the other primary agents and spawning new ones when required.

- *Scout*
  The scout searches the world by picking random points to head towards. If it finds a spill then it heads back to base dropping beacons along the way. It will attempt to go around any objects, physical or virtual, in its path.

- *Beacon*
  This agent does not move and contains a decay value. When the value reaches zero the agent will kill itself.

- *Worker*
  The worker is the most complicated agent and follows beacons laid down by the scouts until it reaches the spill. It proceeds to clear some of it up and return to base with a full load. They will attempt to follow the strongest beacons and if they are returning with a full load then they reinforce the beacons passed on the way back.

- *Tanker*
  A very simple mobile agent that appears on one side of the world and moves slowly to another. If it hits anything along the way then it sinks and spawns a spill.

- *Spill*
  A spill is another simple agent that decreases in size as the worker agents clear it up.

Robot Ships is configured to run the calibration routines automatically on start-up and to launch a Queen agent that in turn spawns most of the other agents. Spawning of agents via other agents is achieved through messages posted to the engine. The agents rely on their sensors to view the world and utilise the messaging system extensively.

8.3 Design

A simple prototype of *Robot Ships* was developed with no textures but simple coloured rectangles and circles. An initial evaluation by the museum provided the following suggestions to improve the application:

- Scouts need to look like they are searching.
- Scouts should move slower so that it is easier for users to interact.
- Workers must make it clear that they are removing part of the spill.
- Workers must be better controlled in that they only head out from the base when there is a trail to follow.
- Spill must be cleaned up by no more than 3 visits from workers.
- Beacons must be rendered so as their purpose is obvious.
- Scouts and workers that are not currently being useful should disappear inside the base.

Possible scenarios for *Robot Ships* were considered before the toxic spill clean-up application was decided upon. These included a search and rescue scenario in which an ambulance had to get to a casualty or a coastguard to a stranded boat. Another suggestion was rubbish collection where the worker would be a dustbin lorry.

These decisions and suggestion led to the final development of the application which is described in detail below.

8.4 Development

This section discusses how each of the classes function within *Robot Ships* and how they work together to shape the full exhibit.

8.4.1 Engine

The *Robot Ships* engine is derived from the *oiIllusionistEngine* class within the framework. When initialised it switches to calibration mode and ensures that if the calibration takes too long a previously saved setup will be loaded as a fail-safe and continue running the exhibit as normal.

During execution the derived engine is responsible for the following:

- Rendering of the blue background (water) and a black mask so that the exhibit fits to a circular table.
- Handling key presses (although this is primarily for debugging since the installed exhibit does not use a keyboard).
- Processing messages from agents and spawning new agents accordingly.

This is on top of the duties that the ancestor class is already performing internally in the framework library.

8.4.2 Queen

The queen is an agent that is rendered as the base of operations in the centre of the display. It is the location from which scout and worker ships are launched and is not only responsible for rendering itself but also manages most of the other agents.

There are two control structures owned by the queen each of which is responsible for a fixed number of scouts and workers. The queen performs the following tasks:

- Process messages from other agents and react appropriately.
- If there are no spills to clean up then launch a new tanker.
- If there are no discovered spills then launch scouts.
- If there are discovered spills then launch workers to clean them up.

8.4.3 Scout

The purpose of the scout agent is to search for the spills. Although the queen keeps count of the number of spills it has no knowledge of where the spills are located and the same is true of the scouts and workers. The scouts must rely on their sensors to detect a spill as they move randomly from waypoint to waypoint.

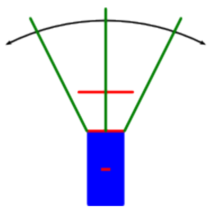


Fig. 8.3: Arrangement of sensors for the scout agent.

The diagram in figure 8.3 illustrates the sensor array used by the agent. The agent itself is represented as a blue rectangle, the green lines represent the “radar” which is swept back and forth searching for spills as indicated by the black arrows (for aesthetic purposes the agent rendering routines employ a searchlight effect which mimics the sweeping motion of the sensors). The red line in the centre of the agent detects when the scout has actually moved over a spill, the line at the front of the agent (on the bow) is the collision detection sensor pair and the horizontal line out in front of the scout is the collision avoidance sensor pair.

The scout has two modes of operation: searching and homing. When searching it chooses a random location (waypoint) and heads towards it as directly as possible. If a spill is detected on any of the radar sensors the agent steers in the corresponding direction until it travels over the spill and the centre sensor is triggered; this switches the scout into homing mode.

In homing mode the agent attempts to head back to the base at the centre in as straight a line as possible. As the scout moves it drops beacons (it actually spawns beacon agents) at regular intervals until it reaches the base at which point it terminates itself.

During both modes of operation the agent relies on a simple collision avoidance algorithm to move around the table. The collision detection and avoidance sensors are capable of seeing other agents within the virtual world alongside objects in the real world. The collision avoidance sensors are placed at a suitable distance that is determined by the speed of the agent. When something is detected by these sensors the agent attempts to swerve around it as it continues to travel forwards. If the scout does not manage to avoid an object and ends up detecting it with the collision sensors then the agent stops, reverses whilst turning until the sensors are clear and then continues moving forwards. While the scout is reversing the collision sensors are switched to the rear so that it does not accidentally reverse over another object or agent.

8.4.4 Beacon

A beacon is simply rendered as a small flashing marker. It contains a power value that is attenuated over time until it reaches zero at which point the beacon destroys itself. The agent accepts messages from workers which can either give it a power boost or a drain.

8.4.5 Worker

The purpose of the worker is to follow the trail of beacons created by the scouts, clear up some of the spill and then return to base. They are bigger and much slower than the scouts and rely on their sensors to detect both the beacons and spills.

The worker has three primary modes of operation: searching, filling and homing. It also has a secondary mode (stopped) which changes the behaviour depending on whether the primary mode is searching or homing.

In searching mode a worker scans the immediate surroundings for beacons and attempts to follow the trail. If multiple beacons are detected then the strongest one is followed and since the beacons get charged and discharged accordingly this produces an effect similar to that of ants using pheromones where the strongest trail is that leading to the food.

When the worker reaches a spill it stops and begins cleaning it up, sending messages to the spill agent throughout indicating that a unit of toxic waste has been taken. When either the worker is full or the spill is gone the worker switches to homing mode.

A worker will retrace its steps in homing mode so that it should pass by all of the beacons that it originally followed until it reaches the base. If the agent is full it sends messages to every beacon it detects so that the beacon is recharged. If the agent is not full (i.e., the spill has now been cleared) then it sends messages to discharge each beacon it passes. This method ensures that a redundant trail is removed while a valid trail is reinforced.

If the worker reaches the end of the beacon trail and there is no spill to be found, or it becomes lost, then it heads back home discharging beacons along the way.

The diagram in figure 8.4 shows the configuration of the sensors in the two main operational modes. The red sensors are identical to those found in the scout and are used for collision avoidance. The centre green sensor is scanned back and forth searching for either beacons or spills. The two

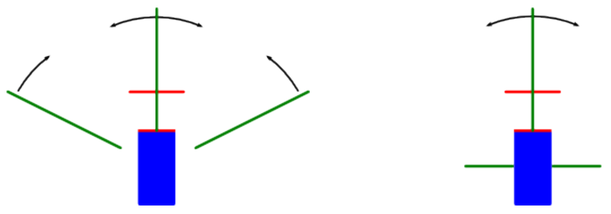

(a) Searching (b) Homing

Fig. 8.4: Arrangement of sensors for the worker agent.

remaining green sensors are scanned synchronously in a sweeping motion (see black arrows in the diagram) also looking for beacons and spills.

In homing mode the centre green sensor remains the same and continues scanning ahead for beacons. The other two sensors switch to stationary positions at right angles to the agent, they are also much shorter than before. This reduction in sensor size means that the agent is less likely to modify beacon energy levels if those beacons are further away from the actual trail.

The secondary stopped mode allows the agents to adhere to a right-of-way rule:

- Worker A collides with Worker B.
- A is searching. B is homing.
- A stops for a random time.
- B attempts to continue using the collision avoidance algorithm.
- A continues after a time or if the way ahead clears.

The random time-out prevents agents becoming locked in the stopped mode. If an agent is searching and it collides with a scout then it also stops and allows the scout to continue. The main rule emphasised here is that a searching agent must stop since collision avoidance could result in it losing the trail of beacons it had been following.

8.4.6 Tanker

The tanker is a large agent which moves very slowly in a straight line across the virtual world. It has a sensor at the front and another that runs through the middle so that the agent can respond to collisions with virtual and real objects and can also be hit by a user. If the tanker detects a hit on either sensor then it sinks and spawns a spill at that location.

8.4.7 Spill

The spill agent is spawned at a random size and is rendered as a green blob. It accepts messages from the worker which inform the spill that it is being cleared up. The agent responds by shrinking in size until it disappears and then destroys itself.

8.4.8 Code

The list in table 8.1 describes the approximate number of lines of code for each class of *Robot Ships*. It can be seen that the majority of coding was concerned with the agents but even these were not actually very long.

Robot Ships Class	**Lines**
Engine	250
Queen	525
Scout	550
Beacon	145
Worker	850
Tanker	245
Spill	150
Total	**2715**

Table 8.1: Number of lines of code per class in *Robot Ships*.

8.5 Final Evaluation

Before the "Connect" gallery was opened to the public *Robot Ships* was evaluated by the museum and the results (courtesy of the National Museums of Scotland via personal communication, 28th April, 2006) are summarised below.

- *Sample*
  Eight family groups consisting of 21 individuals.
- *Observations*
  Visitors spent quite a long time interacting with the exhibit (on average $9\frac{1}{2}$ minutes). Only one third interacted fully without any hints.
- *Comments*
  - A lot of social interaction.
  - Suitable for any age group.
  - Visitors have a tendency to find novel ways to interact with the exhibit.
  - Visitors liked figuring it out and the fact that it was "hands-on".
  - It was memorable and addictive.

The fact that two thirds of visitors required hints is due to there being no instructions or information provided during testing. Moreover, VAE technology is not yet commonplace in everyday life and so people are not aware that they can interact with it.

8.6 Conclusions

Once people understand that they can interact with a VAE it proves to be a popular form of exhibit and makes a welcome change from the standard touch-screen displays that only one person can use at a time.

OpenIllusionist was not yet fully developed when the *Robot Ships* project began and so the project itself prompted changes to the calibration system as well as the addition of a messaging system. The final application was supported by approximately 5500 lines of code in the *OpenIllusionist* library.

Robot Ships is an example of a practical augmented environment as a commercial development. It was designed and implemented using *OpenIllusionist* and the agent model presented in this thesis. The application itself consisted of 2715 lines of code which is small considering the overall complexity of the exhibit and it was implemented by a single programmer (the author). This demonstrates the potential of the framework for use in the rapid development of VAE applications and therefore implies that the claims regarding "infrastructure" in this thesis are valid.

Chapter 9

Conclusions

9.1 Summary

All of the chapters presented in this thesis contain their own conclusions which are summarised as follows:

- *Agent-Based Design*
  A design method for rapid development is presented and justified here. The only real limitation is the number of agents that a computer is capable of dealing with, but this will increase as computers continue to improve.

- *Framework*
  The *OpenIllusionist* framework is shown to provide a basis for rapid application development. Thus, a VAE application developer need only concern themselves with the visual and behavioural design and not the lower-level technical issues of creating an infrastructure.

- *Fiducials*
  A new design of fiducial and corresponding detection algorithm is proposed in this chapter. The design is shown to be as reliable as the leading brand and that ternary can provide a greater number of fiducial combinations without compromising that reliability.

- *Calibration*
  Methods for the complete calibration of a VAE framework are presented. An additional calibration method to reduce the illumination variation across a surface was also proposed and the effectiveness was substantiated.

- *Shadow Removal*
  A technique for shadow removal was analysed in this chapter. Results suggest that the method, although originally designed for post-processing photos, could be applied to live video using modern hardware.

- *Finger Detection*
  A finger detection algorithm is reviewed and then improved performance is achieved through a combination of plane calibration and shadow removal.

- *Robot Ships*
  A realisation of a VAE is presented in this chapter and proves that rapid development of a stable application is possible using *OpenIllusionist*.

9.2 Overall

The results of this research have been embodied in a practical realisation. *OpenIllusionist* provides a useful platform on which to develop stable VAE applications. It already contains all of the requirements for an infrastructure and can be easily expanded to include extra features. It allows for rapid application development and will hopefully encourage a shift of the technology from the research lab to homes and offices.

Some of the earlier VAE implementations have employed distributed computing to handle the various aspects of a software infrastructure, but with the advances in modern computing, frameworks such as *OpenIllusionist* are becoming more practical to run on a single, off-the-shelf PC. Recent availability of multi-core processors at reasonable prices will allow the heavily threaded framework to run even more efficiently. This will mean that it can be further expanded without degrading the responsiveness of the system.

This thesis shows that all of the infrastructure elements necessary to produce a VAE cannot only be merged into a cohesive framework but can also run simultaneously on commodity hardware at interactive speeds.

9.3 Further Work

As mentioned before, many of the infrastructure elements would benefit from further optimisation of the code and thus improve the responsiveness of the system as a whole. Optimisation would also provide more processing time such that dealing with greater numbers of agents or including additional complex modules becomes feasible.

Allowing users to build agents by selecting from a library of predefined behavioural functions would then allow non-programmers to experiment with topics such as interactive artificial-life simulations without having to worry about producing code.

There are many planned improvements for the framework in general as described in section 3.6.1. One addition that should be implemented is audio, especially when considering the usefulness of ambience in an augmented environment as discussed in the introduction. As with vision, audio can provide both an input and an output. By setting up a number of calibrated microphones the system could determine where sounds originated relative to the projection surface. Multiple speakers would allow the system to generate spatialised audio which could, for example, be used to provide the illusion that a particular agent is making a sound.

The proposed fiducial algorithm is worth researching further, in particular the comparison between binary and ternary fiducials. The calibration system could be expanded to deal with more of the possible scenarios. The framework should be modified to allow calibration to be overridden in a custom application and also allow modules to request calibration steps.

Communications would be another big step for *OpenIllusionist* since it could allow collaborative interaction in multiple locations simultaneously. It could even allow overlapping systems within the same room to self-organise and produce larger interactive areas that are calibrated to merge correctly.

Although *OpenIllusionist* has been used by the author (and colleagues in the same research group) to develop applications, the full usability of the framework can only be validated through third party development of VAE applications.

Appendix A

Agent Example

A.1 Header

```cpp
#include <Agent.h>

// Define a class that is derived from oiAgent
class oiSimpleAgent : public oiAgent
{
public:
    // Standard constructor and destructor
    oiSimpleAgent(void *Illusionist, int Type, double X, double Y, double Rotation,  int ID) :
        oiAgent(Illusionist, Type, X, Y, Rotation, ID, true) {}
    virtual ~oiSimpleAgent() {}
protected:
    // Overridden primary functions
    void AgentConstruct();
    void AgentDestruct();
    void AgentBehaviour();
    void AgentSensors();
    void AgentRender();
    bool AgentCollision(double SensorX, double SensorY);
private:
    // Custom private variables and functions for this particular agent
    bool m_bForward;
    double m_dTurnDirection, m_dTurnAngle;

    void ProcessSensors(int &LeftCollision, int &RightCollision);
};
```

Listing A.1: simple_agent.h

A.2 Source

```cpp
#include "simple_agent.h"


void oiSimpleAgent::AgentConstruct()
{
    // Initialise with a random size
    m_dSize = (int)(30 * (float)rand() / (float)RAND_MAX);
    if (m_dSize < 15) m_dSize += 15;

    // Initialise with a random speed
    m_dStepSize  = 0.5+(float)(0.02 * (float)m_dSize * (float)rand()/(float)RAND_MAX);

    m_dTurnAngle        = 2;    // Set the speed at which it can turn
    m_dTurnDirection    = 0;    // Initialise the current direction
    m_bForward          = true; // Initialise the movement flag

    CreateAgentSensors(2);      // Utility function for allocating sensors
}


// No memory was dynamically allocated for this agent, so nothing to cleanup
void oiSimpleAgent::AgentDestruct() {}


void oiSimpleAgent::AgentBehaviour()
{
    int  LeftCollision, RightCollision;

    if (m_Inbox)
    {
        // If there is a message in the inbox then it would be handled here
    }

    // Process sensor information
    ProcessSensors(LeftCollision, RightCollision);

    // If either of the sensors was triggered then respond accordingly else allow agent to move
         forwards
    if (LeftCollision || RightCollision)
    {
        m_bForward = false;     // Stop the agent moving forwards

        // If a direction to turn has not been set already then do so
        if (!m_dTurnDirection)
        {
            // If both sensors were triggered then pick a random direction to turn
            if (LeftCollision && RightCollision)
            {
                if (rand() > (RAND_MAX/2)) m_dTurnDirection = 1;
                else m_dTurnDirection = -1;
            }
            else if (LeftCollision) m_dTurnDirection = 1;
            else m_dTurnDirection = -1;
        }
    }
    else m_bForward = true;

    // If agent is permitted to move forwards then do so, else turn on the spot
    if (m_bForward)
    {
        m_dTurnDirection = 0;

        // Update the location of the agent according to speed and current rotation
        m_dX = m_dX + (m_dStepSize * sin((m_dRotation/180.00) * M_PI));
        m_dY = m_dY + (m_dStepSize * cos((m_dRotation/180.00) * M_PI));
    }
    else m_dRotation += m_dTurnDirection * m_dTurnAngle;
}


void oiSimpleAgent::ProcessSensors(int &LeftCollision, int &RightCollision)
{
    // If either sensor was triggered them return the corresponding data
    LeftCollision   = (m_pSensors[0].FeedbackNumber > 0) ? m_pSensors[0].Feedback[0].Data : 0;
    RightCollision  = (m_pSensors[1].FeedbackNumber > 0) ? m_pSensors[1].Feedback[0].Data : 0;
}


void oiSimpleAgent::AgentSensors()
{
    // Set the left collision sensor positions
    m_pSensors[0].Start.x =  0;
    m_pSensors[0].End.x   = -(int)(0.2 * m_dSize) - 1;

    // Set the right collision sensor positions
    m_pSensors[1].Start.x =  0;
    m_pSensors[1].End.x   =  (int)(0.2*m_dSize) + 1;

    for (int i=0; i<2; i++)
    {
        m_pSensors[i].Start.y =  (int)(0.6 * m_dSize);  // The y coords are common to both sensors
        m_pSensors[i].End.y   =  (int)(0.6 * m_dSize);
        m_pSensors[i].IncludeMask = 0xFFFFFFFF;         // See any agent type and walls
        m_pSensors[i].SeeAll      = false;              // Only see first item along sensor
    }

    m_bCalculateSensors = false;                        // Do not calculate the sensors again
}


void oiSimpleAgent::AgentRender()
{
    // Render a simple red rectangle
    glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
    glBegin(GL_QUADS);
        glVertex3f(-0.2f, 0.6f, 0.0f);
        glVertex3f(-0.2f, -0.6f, 0.0f);
        glVertex3f( 0.2f, -0.6f, 0.0f);
        glVertex3f( 0.2f, 0.6f, 0.0f);
        glNormal3f( 0.0f,  0.0f, 1.0f);
    glEnd();
}


bool oiSimpleAgent::AgentCollision(double SensorX, double SensorY)
{
    double RotX, RotY;

    // Utility function to calculate the relative coordinates to this agent
    RotateSensor(SensorX, SensorY, &RotX, &RotY);

    // If the coordinates lie within the rendered rectangle then report a collision
    if ((RotX >= -(0.2*m_dSize)) && (RotX <= (0.2*m_dSize)))
    {
        if ((RotY >= -m_dSize) && (RotY <= m_dSize)) return true;
    }

    return false;
}
```

Listing A.2: simple_agent.cpp

Appendix B

Framework Example

B.1 Header

```
#include <Illusionist.h>

// Definition of an agent type
#define SIMPLE 1

// Define a class that is derived from oiIllusionistEngine
class oiSimpleEngine: public oiIllusionistEngine
{
public:
    // Standard constructor passing a reference to the parent window to the base class
    // and setting some initialisation parameters
    oiSimpleEngine(wxFrame *Parent): oiIllusionistEngine(Parent, false, 255, CALIBRATION_RADIUS,
         CALIBRATION_COORDS) {}

protected:
    // Overridden primary functions
    virtual void KeyHandler(int Keypress);
    virtual void AgentLaunch(int AgentType, double X, double Y, double Rotation);
    virtual void MessageHandler(oiMessage *Message);

    virtual void PreAgentRender();
    virtual void PostAgentRender();
};
```

Listing B.1: simple_engine.h

B.2 Source

```
#include "simple_engine.h"
#include "simple_agent.h"

void oiSimpleEngine::KeyHandler(int Keypress)
{
    switch(Keypress)
    {
                    // If F1 is pressed then create and initialise the example simple agent
        case WXK_F1:    AgentLaunch(SIMPLE, m_World->m_SpawnPoint.dX, m_World->m_SpawnPoint.dY,
             m_World->m_SpawnPoint.dR);
                    break;
                    // If F8 is pressed then kill the last agent to be created
        case WXK_F8:    AgentKill(-1);
                    break;
                    // If F9 is pressed then toggle the calibration mode
        case WXK_F9:    if (!m_World->m_cCalibrationMode) CalibrationSetMode(CM_FULL);
                    else CalibrationSetMode(CM_OFF);
                    break;
                    // If F11 is pressed then toggle the render sensors flag. This will inform
                    // the renderer to draw any sensors that the agents may have.
        case WXK_F11:   m_World->m_bRenderSensors = !(m_World->m_bRenderSensors);
                    break;
                    // If F12 is pressed then toggle the render walls flag. This will inform the
                    // renderer to provide visualisation of the walls that have been detected by
                    // the primary image processing stage.
        case WXK_F12:   m_World->m_bRenderWalls = !(m_World->m_bRenderWalls);
                    break;
    }
}


void oiSimpleEngine::AgentLaunch(int AgentType, double X, double Y, double Rotation)
{
    // Do not create more agents than defined by the constant MAX_AGENTS
    if (AgentGetCount() > MAX_AGENTS) return;

    switch(AgentType)
    {
                    // Create a new oiSimpleAgent and pass the reference to the AgentStart
                    // utility function (implemented in the library)
        case SIMPLE:    AgentStart(new oiSimpleAgent(this, AgentType, X, Y, Rotation,
             m_iAgentIDCounter++));
                    break;
    }

}


void oiSimpleEngine::MessageHandler(oiMessage *Message)
{
    switch (Message->m_iID)
    {
        // Any messages sent to the engine are handled here
    }
}

// Any rendering that is required before the agents have been rendered is placed here
void oiSimpleEngine::PreAgentRender()
{
    // Render the spawn point as a small green triangle
    glPushMatrix();
        glTranslatef((float)m_World->m_SpawnPoint.dX, (float)m_World->m_SpawnPoint.dY, 0.0);
        glRotatef((float)-m_World->m_SpawnPoint.dR, 0.0f, 0.0f, 1.0f);
        glColor4f(0.1f, 0.4f, 0.1f, 0.5f);
        glBegin(GL_TRIANGLES);
            glNormal3f(0.0f, 0.0f, 1.0f);
            glVertex2f(-5.0f, -10.0f);
            glVertex2f( 5.0f, -10.0f);
            glVertex2f( 0.0f,  10.0f);
        glEnd();
    glPopMatrix();
}

// Any rendering that is required after the agents have been rendered is placed here
void oiSimpleEngine::PostAgentRender() {}
```

Listing B.2: simple_engine.cpp

Bibliography

[1] Argyros, A. A., and Lourakis, M. I. A. Vision-based interpretation of hand gestures for remote control of a computer mouse. In *In proceedings of the HCI '06 workshop* (Graz, Austria, May 2006), pp. 40–51.

[2] Au-Yeung, K., Johnston, D. J., and Clark, A. A comparison of fiducial-based visual positioning systems. In *18th International Conference on Pattern Recognition* (Hong Kong, August 06), pp. 758–761.

[3] Barakonyi, I., Psik, T., and Schmalstieg, D. Agents that talk and hit back: Animated agents in augmented reality. In *ISMAR '04: Proceedings of the Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)* (Arlington, VA, USA, November 2004), IEEE Computer Society, pp. 141–150.

[4] Barakonyi, I., and Schmalstieg, D. Augmented reality agents in the development pipeline of computer entertainment. In *Proceedings of the 4th International Conference on Entertainment Computing* (Sanda, Japan, September 2005), pp. 345–356.

[5] Barakonyi, I., Weilguny, M., Psik, T., and Schmalstieg, D. Monkeybridge: autonomous agents in augmented reality games. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology* (Valencia, Spain, June 2005), ACM Press, pp. 172–175.

[6] Barrow, H. G., and Tenenbaum, J. M. Recovering intrinsic scene characteristics from images. In *Computer Vision Systems.* New York: Academic Press, 1978, pp. 3–26.

[7] Bimber, O., Emmerling, A., and Klemmer, T. Embedded entertainment with smart projectors. *Computer 38*, 1 (2005), 48–55.

[8] Bimber, O., Wetzstein, G., Emmerling, A., and Nitschke, C. Enabling view-dependent stereoscopic projection in real environments. In *ISMAR '05: Proceedings of the Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality* (Santa Barbara, California, USA, October 2005), IEEE Computer Society, pp. 14–23.

[9] Bratman, M. E., Israel, D., and Pollack, M. E. Plans and resource-bounded practical reasoning. *Computational Intelligence 4* (1988), 349–355.

[10] Buchmann, V., Violich, S., Billinghurst, M., and Cockburn, A. FingARtips: gesture based direct manipulation in Augmented Reality. In *2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and SouthEast Asia (Graphite 2004)* (Singapore, June 2004), pp. 212–221.

[11] Canny, J. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 6 (1986), 679–698.

[12] Chung, Y.-C., Chang, S.-L., Wang, J. M., and Chen, S.-W. An improved method for extraction of intrinsic images from a single image with integrated measures. In *Artificial Intelligence and Applications* (Innsbruck, Austria, February 2005), pp. 356–361.

[13] Costanza, E., and Robinson, J. A. A Region Adjacency Tree approach to the detection and design of fiducials. In *Video, Vision and Graphics* (Bath, UK, July 2003), pp. 63–70.

[14] Crowley, J., Bérard, F., and Coutaz, J. Finger tracking as an input device for Augmented Reality. In *International Workshop on*

Gesture and Face Recognition (Zürich, Switzerland, June 1995), pp. 195–200.

[15] Davis, J., and Shah, M. Visual gesture recognition. In *IEEE Proceedings - Vision, Image and Signal Processing* (1994), vol. 141, pp. 101–10.

[16] de Ipiña, D. L., Mendonça, P. R. S., and Hopper, A. TRIP: a Low-Cost Vision-Based Location System for Ubiquitous Computing. *Personal and Ubiquitous Computing 6*, 3 (2002), 206–219.

[17] Drew, M., Finlayson, G., and Hordley, S. Recovery of chromaticity image free from shadows via illumination invariance. In *IEEE Workshop on Color and Photometric Methods in Computer Vision* (Nice, France, October 2003), pp. 32–39.

[18] Feiner, S., Macintyre, B., and Seligmann, D. Knowledge-based augmented reality. *Communications of the ACM 36*, 7 (1993), 53–62.

[19] Fiala, M. ARTag, an improved marker system based on ARToolkit. National Research Council Canada (NRC/ERB-1111), July 2004. NRC Publication Number: NRC 47166.

[20] Finlayson, G. D., Drew, M. S., and Lu, C. Intrinsic images by entropy minimization. In *Proceedings of the 8th European Conference on Computer Vision* (Prague, Czech Republic, May 2004), pp. 582–595.

[21] Finlayson, G. D., and Hordley, S. Colour constancy at a pixel. *Journal of the Optical Society of America 18*, 2 (2001), 253–264.

[22] Finlayson, G. D., Hordley, S. D., and Drew, M. S. Removing shadows from images. In *Proceedings of the 7th European Conference on Computer Vision* (Copenhagen, Denmark, May 2002), pp. 823–836.

[23] Finlayson, G. D., Hordley, S. D., and Drew, M. S. Removing shadows from images using retinex. In *Tenth Color Imaging Conference: Color Science and Engineering Systems, Technologies, Applications* (Scottsdale, Arizona, USA, November 2002), pp. 73–79.

[24] Fischler, M. A., and Bolles, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6 (1981), 381–395.

[25] Fitzgibbon, A., Pilu, M., and Fisher, R. B. Direct Least Square Fitting of Ellipses. *IEEE Transactions on Pattern Analysis and Machine Intelligence 21*, 5 (May 1999), 476–480.

[26] Fredembach, C., and Finlayson, G. Hamiltonian path based shadow removal. In *British Machine Vision Conference* (Oxford, UK, September 2005), p. 49.

[27] Fuhrmann, A. L., Splechtna, R., and Prikryl, J. Comprehensive calibration and registration procedures for augmented reality. In *Proceedings of Eurographics Workshop on Virtual Environments* (Stuttgart, Germany, May 2001), pp. 219–228.

[28] Georgeff, M. P., and Lansky, A. L. Reactive reasoning and planning. In *The Sixth National Conference on Artificial Intelligence* (Seattle, Washington, July 1987), pp. 677–682.

[29] Grest, D., Frahm, J.-M., and Koch, R. A color similarity measure for robust shadow removal in real time. In *Vision, Modeling and Visualization* (Munich, Germany, November 2003), pp. 253–260.

[30] Hough, P. V. C. Machine analysis of bubble chamber pictures. In *International Conference on High-Energy Accelerators and Instrumentation* (1959), pp. 554–556.

[31] Ishii, H., Ben-Joseph, E., Underkoffler, J., Yeung, L., Chak, D., Kanji, Z., and Piper, B. Augmented urban planning workbench: Overlaying drawings, physical models and digital simulation. In *ISMAR '02: Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR'02)* (Darmstadt, Germany, September 2002), IEEE Computer Society, p. 203.

[32] Ishii, H., and Ullmer, B. Tangible bits: towards seamless interfaces between people, bits and atoms. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems* (Atlanta, Georgia, March 1997), ACM Press, pp. 234–241.

[33] Jennings, N. R., Sycara, K., and Wooldridge, M. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems 1*, 1 (1998), 7–38.

[34] Johnston, D. J., and Clark, A. F. A vision-based location system using fiducials. In *Vision, Video and Graphics* (Bath, UK, July 2003), pp. 159–166.

[35] Kato, H., and Billinghurst, M. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *IWAR '99: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality* (San Francisco, California, 1999), IEEE Computer Society, p. 85.

[36] Kato, H., Billinghurst, M., Poupyrev, I., Imamoto, K., and Tachibana, K. Virtual object manipulation on a table-top AR environment. In *IEEE and ACM International Symposium on Augmented Reality* (Munich, Germany, October 2000), pp. 111–119.

[37] Kim, E., Haseyama, M., and Kitajima, H. Fast and Robust Ellipse Extraction from Complicated Images. In *International Conference on Information Technology and Applications* (Bathurst, Australia, November 2002), pp. 138–143.

[38] Kim, J.-S., Gurdjos, P., and Kweon, I.-S. Geometric and algebraic constraints of projected concentric circles and their applications to camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 4 (2005), 637–642.

[39] Letessier, J., and Bérard, F. Visual tracking of bare fingers for interactive surfaces. In *UIST '04: Proceedings of the 17th annual ACM*

symposium on User Interface Software and Technology (Santa Fe, New Mexico, October 2004), ACM Press, pp. 119–122.

[40] Maes, P., Darrell, T., Blumberg, B., and Pentland, A. The alive system: Wireless, full-body interaction with autonomous agents. *Multimedia Systems 5*, 2 (1997), 105–112.

[41] Malik, S., and Laszlo, J. Visual touchpad: a two-handed gestural input device. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces* (Pennsylvania, USA, October 2004), ACM Press, pp. 289–296.

[42] Newman, W., and Wellner, P. A desk supporting computer-based interaction with paper documents. In *CHI '92: Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (Monterey, California, USA, June 1992), ACM Press, pp. 587–592.

[43] Nölker, C., and Ritter, H. Detection of fingertips in human hand movement sequences. In *Proceedings of the International Gesture Workshop on Gesture and Sign Language in Human-Computer Interaction* (Bielefeld, Germany, September 1997), pp. 209–218.

[44] O'Mahony, S., and Robinson, J. A. Penpets: a physical environment for virtual animals. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems* (Fort Lauderdale, Florida, USA, April 2003), ACM Press, pp. 622–623.

[45] The OpenIllusionist Project. `http://www.openillusionist.org.uk`. Retrieved December 2006.

[46] Park, H., and Park, J.-I. Modern approaches to direct-projected augmented reality: A review. In *International Symposium on Ubiquitous VR* (Yanji City, China, July 2006), pp. 5–8.

[47] Parnham, D., Robinson, J. A., and Zhao, Y. A Compact Fiducial for Affine Augmented Reality. In *Visual Information Engineering:*

Convergence in Graphics and Vision (Glasgow, Scotland, April 2005), pp. 347–352.

[48] Patten, J., Recht, B., and Ishii, H. Interaction techniques for musical performance with tabletop tangible interfaces. In *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology* (Los Angeles, USA, June 2006), ACM Press, p. 27.

[49] Pilu, M., Fitzgibbon, A., and Fisher, R. Ellipse-specific direct least-square fitting. In *The IEEE International Conference on Image Processing* (Lausanne, Switzerland, September 1996).

[50] Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, 1992.

[51] QR Code. `http://www.denso-wave.com/qrcode/index-e.html`. Retrieved December 2006.

[52] Qt, Cross-Platform Library. `http://www.trolltech.com/products/qt`. Retrieved December 2006.

[53] Quek, F., Mysliwiec, T., and Zhao, M. FingerMouse: A Freehand Pointing Interface. In *International Workshop on Automatic Face and Gesture Recognition* (Zurich, Switzerland, June 1995), pp. 372–377.

[54] Raskar, R., Welch, G., Cutts, M., Lake, A., Stesin, L., and Fuchs, H. The office of the future: a unified approach to image-based modeling and spatially immersive displays. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (Orlando, Florida, USA, July 1998), ACM Press, pp. 179–188.

[55] Raskar, R., Welch, G., and Fuchs, H. Seamless projection overlaps using image warping and intensity blending. In *Fourth International*

Conference on Virtual Systems and Multimedia (Gifu, Japan, November 1998).

[56] Ratti, C., Wang, Y., Ishii, H., Piper, B., and Frenchman, D. Tangible User Interfaces (TUIs): A Novel Paradigm for GIS. *Transactions in GIS 8*, 4 (October 2004), 407–421.

[57] Rice, A. C., Cain, C. B., and Fawcett, J. K. Dependable coding of fiducial tags. In *2nd International Symposium on Ubiquitous Computing Systems* (Tokyo, Japan, November 2004), pp. 259–274.

[58] Robinson, J. NEAT - Negative Exponential/Adaptive Threshold Edge Detector. Internal research report, Department of Electronics, University of York, 2006.

[59] Robinson, J. A., and Robertson, C. The livepaper system: augmenting paper on an enhanced tabletop. *Computers & Graphics 25*, 5 (2001), 731–743.

[60] Robinson, P., Sheppard, D., Watts, R., Harding, R., and Lay, S. A framework for interacting with paper. In *EUROGRAPHICS* (Budapest, Hungary, September 1997), vol. 16, pp. 329–334.

[61] Salvador, E., Cavallaro, A., and Ebrahimi, T. Cast shadow segmentation using invariant color features. *Computer Vision and Image Understanding 95*, 2 (2004), 238–259.

[62] Sato, Y., Kobayashi, Y., and Koike, H. Fast tracking of hands and fingertips in infrared images for augmented desk interface. In *FG '00: Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000* (Grenoble, France, March 2000), IEEE Computer Society, p. 462.

[63] ShotCode, Offline Weblinks. `http://www.shotcode.com`. Retrieved December 2006.

[64] Stafford-Fraser, Q., and Robinson, P. Brightboard: a video-augmented environment. In *CHI '96: Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (Vancouver, Canada, April 1996), ACM Press, pp. 134–141.

[65] Studierstube Augmented Reality Project. `http://studierstube.icg.tu-graz.ac.at/`. Retrieved December 2006.

[66] Tappen, M. F., Freeman, W. T., and Adelson, E. H. Recovering intrinsic images from a single image. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 9 (September 2005), 1459–1472.

[67] Thomas, G. A., Jin, J., Niblett, T., and Urquhart, C. A versatile camera position measurement system for virtual reality tv production. In *International Broadcasting Convention* (Amsterdam, Holland, September 1997), pp. 284–289.

[68] Uchiyama, S., Takemoto, K., Satoh, K., Yamamoto, H., and Tamura, H. Mr platform: A basic body on which mixed reality applications are built. In *ISMAR '02: Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR'02)* (Darmstadt, Germany, September 2002), IEEE Computer Society, p. 246.

[69] Underkoffler, J., Ullmer, B., and Ishii, H. Emancipated pixels: real-world graphics in the luminous room. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (Los Angeles, California, USA, August 1999), ACM Press/Addison-Wesley Publishing Co., pp. 385–392.

[70] Vincze, M. Robust tracking of ellipses at frame rate. *Pattern Recognition 34*, 2 (2001), 487–498.

[71] von Hardenberg, C., and Bérard, F. Bare-hand human-computer interaction. In *PUI '01: Proceedings of the 2001 workshop on Perceptive User Interfaces* (Orlando, Florida, USA, November 2001), ACM Press, pp. 1–8.

[72] Weiss, Y. Deriving intrinsic images from image sequences. In *The Eigth IEEE International Conference on Computer Vision* (Vancouver, Canada, July 2001), vol. 2, pp. 68–75.

[73] Wellner, P. The digitaldesk calculator: tangible manipulation on a desk top display. In *UIST '91: Proceedings of the 4th annual ACM symposium on User Interface Software and Technology* (South Carolina, USA, November 1991), ACM Press, pp. 27–33.

[74] Wellner, P. Interacting with paper on the digitaldesk. *Communications of the ACM 36*, 7 (1993), 87–96.

[75] Woods, E., Mason, P., and Billinghurst, M. Magicmouse: an inexpensive 6-degree-of-freedom mouse. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia* (Melbourne, Australia, February 2003), ACM Press, pp. 285–286.

[76] wxWidgets: Cross-Platform GUI Library. `http://www.wxwidgets.org/`. Retrieved December 2006.

[77] Xie, Y., and Ji, Q. A new efficient ellipse detection method. In *Proceedings of the 16th International Conference on Pattern Recognition* (Quebec, Canada, August 2002), vol. 2, pp. 957–960.

[78] Yoon, J. J., Koch, C., and Ellis, T. J. Shadowflash: an approach for shadow removal in an active illumination environment. In *Proceedings of the British Machine Vision Conference* (Cardiff, UK, September 2002).

[79] Zhao, Y., and Robinson, J. A. Design and evaluation of multilevel fiducials for augmented reality. In *BMVA Symposium on Vision, Video and Graphics* (London, UK, July 2004).