Real Time 3D motion tracking for interactive computer simulations

Brenton Bailey

Supervisor: Dr. Tony Field
Second Marker: Prof Alexander Wolf

June 2007
Abstract

Motion tracking has been employed by industry for many years but has remained an expensive and difficult process. Specialist equipment and the requirement for a highly controlled environment are obstacles stopping tracking from breaking into the consumer market.

This project aims to demonstrate that high accuracy optical tracking can be achieved using inexpensive video cameras in an unconstrained environment. The software developed here is used to evaluate the effectiveness of the tracking solution, showing that the idea of optical tracking with inexpensive cameras is a viable mechanism for computer interaction.
I would like to thank my supervisor Dr Tony Field for his invaluable assistance and enthusiasm. I would also like to thank Prof Alexander Wolf for agreeing to be my second marker. Lastly I would like to thank Steven Lovegrove for his brilliant Laser Tracking idea and constantly supplying me with cups of tea.
## Contents

1 Introduction ................................................. 6
   1.1 The Application ...................................... 7
   1.2 Contributions ...................................... 7

2 Background .................................................. 8
   2.1 Tracking ............................................. 8
      2.1.1 Magnetic Tracking ............................... 8
      2.1.2 Optical Tracking ............................... 9
      2.1.3 Passive optical tracking ...................... 9
      2.1.4 Temporal redundancy ......................... 11
      2.1.5 Model-based Tracking ......................... 12
   2.2 Computational Stereo ................................. 13
      2.2.1 Epipolar Geometry .............................. 13
   2.3 Commercial Tracking Solutions ...................... 13
      2.3.1 Hawk-eye ....................................... 13
      2.3.2 MotionStar® ..................................... 14
      2.3.3 A.R.T. System ................................. 15
   2.4 Software APIs ......................................... 15
      2.4.1 DirectShow Technology ......................... 15
      2.4.2 OpenCV .......................................... 15

3 Tracking ..................................................... 17
   3.1 Problem ............................................. 17
   3.2 Camera Calibration .................................. 17
      3.2.1 Lens Distortion ................................ 18
   3.3 Feature Extraction .................................. 20
   3.4 Triangulation ....................................... 21
      3.4.1 Intersection of two lines in 3D space ...... 22
      3.4.2 Coordinate Systems ............................ 23
   3.5 Correspondence ..................................... 25
      3.5.1 Fundamental Matrix ............................ 26
      3.5.2 Beyond Two Cameras ......................... 27
      3.5.3 Temporal Correspondence .................... 27

4 Hardware .................................................... 30
   4.1 Cameras ............................................. 30
      4.1.1 Triggering ..................................... 31
   4.2 IEEE 1394 - FireWire ............................... 32
4.2.1 Bandwidth ........................................ 33

5 Software ........................................... 35
5.1 Camera APIs ....................................... 35
5.2 Multi-Threading .................................... 36
5.3 Architecture ....................................... 37
  5.3.1 Image Rectification ............................. 37
  5.3.2 Feature Extraction ............................. 38
  5.3.3 3D Triangulation .............................. 39
  5.3.4 Correspondence ............................... 39
  5.3.5 Common Settings .............................. 41
  5.3.6 Extensibility ................................. 42
5.4 Camera Calibration ............................... 42

6 Evaluation ........................................... 45
6.1 A Small Taster .................................... 45
6.2 Latency ............................................ 46
  6.2.1 Reasons For Latency ......................... 47
  6.2.2 Latency Experiment ......................... 49
6.3 Static Accuracy ................................... 50
  6.3.1 Calibration Accuracy ....................... 51
  6.3.2 Two and Three camera accuracy ............ 52
6.4 Motion Accuracy .................................. 56
  6.4.1 Shutter Speed ................................ 56
  6.4.2 Synchronised Triggering .................... 58
  6.4.3 Juggling Balls .............................. 60

7 Conclusions .......................................... 62
  7.1 Further Work .................................... 63

A Example Tracker XML file ........................... 65

B Code Examples ....................................... 68
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Stereoscopic cameras viewing an object</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Epipolar geometry with stereoscopic cameras</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>Native camera image with straight line grid overlaid</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Rectified Camera image with straight line grid overlaid</td>
<td>19</td>
</tr>
<tr>
<td>3.3</td>
<td>Affect of lighting on colour based segmentation</td>
<td>20</td>
</tr>
<tr>
<td>3.4</td>
<td>Projection of a non-uniform shape</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>Three-dimensional triangulation of camera rays</td>
<td>22</td>
</tr>
<tr>
<td>3.6</td>
<td>Closest point of intersection for two rays</td>
<td>22</td>
</tr>
<tr>
<td>3.7</td>
<td>Standard camera perspective projection</td>
<td>25</td>
</tr>
<tr>
<td>3.8</td>
<td>Epipolar Geometry</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>Allied Vision Technologies Marlin F-046C</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>HIROSE I/O connector pin assignments [Marlin Technical Manual [2]]</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>Block diagram of the Marlin Camera image pipeline [Marlin Technical Manual [2]]</td>
<td>33</td>
</tr>
<tr>
<td>5.1</td>
<td>Image producers and consumers synchronisation using a Frame-Buffer</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>Sub-Pixel corner finding</td>
<td>38</td>
</tr>
<tr>
<td>5.3</td>
<td>Simplified software diagram</td>
<td>44</td>
</tr>
<tr>
<td>6.1</td>
<td>3D model obtained by tracking a laser pointer over a toy monkey</td>
<td>46</td>
</tr>
<tr>
<td>6.2</td>
<td>Raw point data for the monkey 3D mesh</td>
<td>47</td>
</tr>
<tr>
<td>6.3</td>
<td>Tracker Latency Graph</td>
<td>49</td>
</tr>
<tr>
<td>6.4</td>
<td>Establishing the world coordinate system</td>
<td>50</td>
</tr>
<tr>
<td>6.5</td>
<td>Static Checkerboard Pattern</td>
<td>51</td>
</tr>
<tr>
<td>6.6</td>
<td>Absolute Error distance for two and three cameras</td>
<td>54</td>
</tr>
<tr>
<td>6.7</td>
<td>Three camera stereo rig</td>
<td>55</td>
</tr>
<tr>
<td>6.8</td>
<td>Extended Three-Camera accuracy</td>
<td>55</td>
</tr>
<tr>
<td>6.9</td>
<td>The effect of shutter speed on “blob” elongation</td>
<td>57</td>
</tr>
<tr>
<td>6.10</td>
<td>Absolute error for triggered and non-triggered tracking</td>
<td>59</td>
</tr>
<tr>
<td>6.11</td>
<td>Tracking three juggling balls</td>
<td>61</td>
</tr>
<tr>
<td>B.1</td>
<td>Example feature extraction code</td>
<td>69</td>
</tr>
<tr>
<td>B.2</td>
<td>Code to initialise an OpenCV Kalman Filter</td>
<td>70</td>
</tr>
</tbody>
</table>
# List of Tables

4.1 Marlin parameters of interest ........................................ 31  
4.2 HIROSE connector pin functions ................................. 32  

6.1 Average Error from ten calibrations .......................... 52  
6.2 Checkerboard Model measurement drift ...................... 54  
6.3 Variance in two and three camera error values ............ 56  

Chapter 1

Introduction

Computer based simulations, be they computer games or virtual reality systems, are continually improving in sophistication and realism. However, methods for interacting with such systems have not shown the same rapid development. In order for a virtual reality system or a computer game to feel truly immersive it must not only look and sound convincing, there must be an easy and natural way to interact with it.

Traditional methods, such as the keyboard, mouse and joypad, aim to put all of the required controls at the user’s fingertips but are conceptually unrelated to the simulated action they are trying to manipulate. There are many scenarios where it would be desirable for the user to control the computer simulation by manipulating a physical object as if they were interacting with the real life scenario.

One possible way to achieve this is to use motion tracking to capture the user’s movements of a physical object and provide that as input information to a simulation.

The idea of motion tracking is well explored and currently utilised for many applications. For example the well known Hawk-Eye[17] system augments sports coverage with simulation based replays but uses expensive video cameras and does not work in real time. The film industry has been using motion capture for many years to produce realistic CGI characters but requires a highly controlled environment and is currently a very expensive and time consuming process. Whilst there are many successful tracking solutions available, none are oriented towards use with cheap hardware and in generic environments.

The aim of this project is to explore using video based motion tracking as a control mechanism for a computer simulation. The ideas investigated here might one day be used to build the next generation of user input devices. Before we can get to that stage there are several challenges that need to solved. In order to track the three dimensional position of an item we must first be able to accurately calibrate a set of stereo cameras. Camera calibration is discussed in Sections 3.2 and 5.4 with the effect it has on tracking accuracy explored in Section 6.3.1. Feature extraction is another key component of tracking but is computationally expensive and context sensitive. Some of the issues relating to feature extraction are described in Sections 3.3 and 5.3.2. There are also unanswered questions, such as which parameters affect the accuracy of a tracking system? This report will attempt to identify these parameters and investigates
the level impact they might have on tracking accuracy.

1.1 The Application

The objectives of this project are realised through the design and implementation of a general purpose optical tracking system. Once in place the general purpose solution is targeted at a particular tracking problem.

The purposed application being presented in this report is a Juggling tracker. Tracking the location of juggling balls and then providing the coordinate data as input for a juggling simulation presents many interesting problems. Juggling balls often move in non-uniform patterns, rapidly changing direction as they are thrown by the juggler. There are also occlusion problems that must be considered. Juggling balls may occlude each other as they cross in the air, also the balls are likely to disappear from the scene when they are clutched in the juggler’s hand. These difficulties as well as the sheer amount of movement in the scene make juggling a rigorous tracking problem.

1.2 Contributions

• A guide to the entire tracking process - The aspects of stereoscopic tracking are well understood but often presented in isolation. This report, in part acts a guide to the entire tracking process. Chapter 3 provides a theoretical overview of the tracking problem and Chapter 5 describes an implementation based view of a tracking system.

• A simple calibration process that can calibrate a set of stereo cameras in under two minutes (see Section 5.4).

• An extensible software framework that can be used to solve many optical tracking problems. The modular design of the software presented here is outlined in Chapter 5.

• A detailed account of the design and implementation of a practical tracking system based on Marlin F-046C[1] FireWire cameras (see Chapter 4).

• Exploring the effect of using more than two cameras to triangulate the position of an item with a view to improving accuracy. Section 3.5.2 describes different techniques for combining the information from more than two cameras whilst Section 6.3.2 considers the impact using three cameras has on tracking accuracy.

In addition, Chapter 2 of this report provides a detailed account of existing tracking techniques whilst critiquing various commercially available solutions.
Chapter 2

Background

The problem of motion tracking to provide input data for a computer simulation can be broken down into two sub-problems. The first is the general problem of tracking the movement of a given feature in free-space. The second is combining the movement of certain low-level features to form a model of the object that is being manipulated.

There have been countless systems proposed in the past for solving either or both of these problem and each one of these systems has its own strengths and weaknesses.

2.1 Tracking

The process of tracking involves identifying a salient feature at a point in time and then identifying the same feature at a future point in time. Many ways have been presented for identifying salient features in a scene.

2.1.1 Magnetic Tracking

Magnetic Tracking technology is based on the principle of magnetic induction. If a coiled wire is moved through a magnetic field, an electrical current will flow in the coil. The strength of this current is proportional to the distance between the source of the magnetic field and the coil. It is also proportional to the orientation of the coil in the magnetic field.

Using three orthogonal coils in the sensor and a transmitter that produces three orthogonal magnetic fields, the exact position and orientation of the sensor can be deduced. This will provide us with a point in 3D space that can reliably be detected at any time.

Magnetic tracking provides 6 degrees of freedom (DOF) for each sensor and is not affected by occlusions between objects and sensors. However the system is prone to magnetic and electrical interference. CRT monitors or electrical cables that emit a magnetic field cause interference, which means that this technology can only be used in an environment that has been shielded from magnetic radiation. Cabling for the sensors is also bulky, often leading to restricted movement.
2.1.2 Optical Tracking

Optical tracking is the process of trying to identify features in a video stream using image processing techniques. Much of the early work done in this area was concerned with tracking the position and orientation of a head mounted display for input into a virtual reality system, the idea being that the images shown by the display would change as the user moved their head.

One of the major problems faced by an optical tracking system is the computational overhead associated with extracting key features from a video stream. There have been a wide variety of different techniques proposed for solving this problem, which can broadly be categorised into two groups. The first imposes constraints on the environment to make feature detection easier. The Optical tracking system presented by Ronald Azuma and Mark Ward [8] uses an infrared LED matrix on the ceiling of the room and a Head Mounted Display (HMD) with four cameras pointed towards the ceiling. The benefit of this setup is that the environment is highly controlled. Detection of an infrared LED on a plain surface is straightforward: calculating the position and orientation of the camera becomes a problem of photogrammetry. Common points are identified on each image. A line of sight (or ray) can be constructed from the camera location to the point on the object. It is the intersection of these rays (triangulation) that determines the three-dimensional location of the point.[7].

Whilst this approach dramatically reduces the computational complexity of the problem, the requirement of a custom built environment reduces the usability of the system. A more practical solution, and one that is closer to the scope of this project, is to fix the position of the cameras and allow for a moving beacon. This approach has been used extensively in both commercial and research solutions.

Active Marker  Active Marking involves using unambiguous beacons in the scene as tracking points. This is similar to the infrared LED ceiling matrix presented by Azuma and Ward [8], except the Active Markers are expected to be moved around the capture space. Using unambiguous beacons removes the problem of detecting features within the scene, and as a result the number of tracking points in the system can be increased with little overhead.

The markers in a scene are made to be unique by only turning one on at a time. This means that the refresh rate falls as the number of markers increases. The commercial tracking system developed by PhaseSpace (US Patent 6,324,296 [21]) modulates the output of each active marker in the scene to produce a pulse train that is unique to that LED. This binary signature means that 32 markers can be on at once reducing the limitations of marker swapping.

Whilst active markers are a very popular solution to the tracking problem they are not ideal. Active markers have to be attached to all moving objects in the scene that are to be tracked. This often requires the user to wear custom suits or gloves and generally makes the systems less flexible.

2.1.3 Passive optical tracking

All of the systems discussed so far have required the tracking space to be embellished in one way or another to make the tracking problem simpler. Passive
optical tracking is concerned with using image processing techniques to detect natural features in a scene such as corners or blocks of colour.

**Region-based segmentation**

MIME (Mime Is Manual Expression) was a system presented by Heckenberg and Lovell [16], concerned with tracking finger movements and interpreting hand gestures. The system employed image segmentation based on colour histograms. The idea behind this is the assumption that the colour of a human hand will be significantly different to that of the background. Changes in illumination can be accommodated by using an Hue Saturation Luminance (HSL) colour space, in this case the system would only be concerned with a pixel’s Hue and Saturation. Variations in skin tone were dealt with using a statistical probability model generated from a range of different ethnic groups.

Once every pixel has been classified as skin or non-skin, regions need to be extracted. MIME [16] uses a contour following technique that results in fewer pixel tests than the more traditional scan-line based techniques. This was largely based on the work by Li and Shen [9].

Heckenberg and Lovell [16] concluded that the colour segmentation module achieved excellent tracking of a skin region under very few environmental constraints making their system a very successful passive optical tracking system.

**Edge-based segmentation**

Another common method for extracting features from a scene is to detect the presence of corners. Corners are simply the intersection of two edges and provide a strong feature to track. Corners are desirable over edges because they provide an anchor point in both the X and Y dimension.

The tracking system presented by Prewer and Kitchen [22] was primarily concerned with using computationally quick methods to extract edges in one frame and then match those with edges extracted in the next frame. The system is particularly desirable because it is presented as a general tracking solution. Systems such as [13, 20] which use model-based tracking (See section 2.1.5), are only applicable to the tracking problem they were designed for. The same can be said for MIME [16]. The ideology expressed by Prewer[22] is that it is better to do a simple task well rather than a complicated process badly. The system uses the standard Prewitt edge detector for feature detection. Structural matching across frames is performed using a slightly naive, but fast, method of matching lines with similar edge-intensity, length and gradient. This matching technique often leads to false positive matches, which the system handles by choosing the line with the least movement between frames.

The Prewer system[22] is a good example of the use of pragmatic fast techniques to maintain performance at the expense of accuracy. This raises an interesting point about the nature of the tracking system: there are many scenarios where it would be desirable to have a fast system that lacks accuracy, such as a system where the detection of movement is more important than the determination of that movement; an example is motion detection for a security system.
2.1.4 Temporal redundancy

Every method for passive optical tracking discussed so far is computationally expensive. One of the criteria for a good tracking system is high temporal resolution, whilst another is high spatial resolution. Spatial resolution is very much dependent on the quality of the capture hardware, whereas temporal resolution is largely based on the efficiency of the software and tracking method used.

Temporal redundancy can be exploited in a tracking system due to the fact that we have prior knowledge of where an object was in a scene. With this prior knowledge and an estimation of the maximum distance the object could have moved in between frames, the search space for a particular feature can be greatly reduced.

Kalman filtering A Kalman filter \[19\] is a set of mathematical equations that provide an efficient way of calculating the past, present and future state of a process in a way that minimises the amount of error introduced by the measuring process. The filter is a very powerful tool and has been widely used in vision and tracking systems.

One way of thinking about the Kalman Filter is to look at it as a recursive Bayesian estimation.

\[
P(X|Z) = \frac{P(Z|X)P(X)}{P(Z)}
\]

where

- \(P(X|Z)\) is the Posterior (current state of the system)
- \(P(Z|X)\) is the likelihood
- \(P(X)\) is the Prior (previous state of the system)
- \(P(Z)\) is the evidence

This is particularly relevant when we consider the tracking problem. If we consider the likelihood \(P(X|Z)\) as being a Gaussian probability distribution derived from knowledge of how far an object can possibly move in one time step. The evidence \(P(Z)\) as being a probability distribution based on our faith in the accuracy of the feature extraction algorithm and the Prior \(P(X)\) as being a measure of our belief that we know where the object currently is. This accurately describes the problem at hand providing a probability distribution for the posterior \(P(X|Z)\) that can be used to estimate where the object will be in the next time step. This estimate can be used to reduce the search space of the feature detection algorithm and increase the overall throughput of the system.

There is no single point when a tracking system is certain of the location of an object. Noise introduced by the capture equipment and pragmatic vision processing techniques introduce a level of uncertainty into the system. The presence of this uncertainty means that it is better to stop thinking about the system in absolute terms and start to reason about it as a set of beliefs. This type of reasoning is exactly what we see in the Kalman filter. Instead of knowing the exact position of an object in the scene, it can be represented as a Gaussian
distribution that demonstrates a belief about the location of the object. In other words $P(X)$ represents our belief about the current state of the system; the shape of this distribution conveys the level of certainty we have about the location of an object.

**Condensation** [18] is a technique developed by Isard and Blake that attempts to overcome the failings associated with the Kalman filter. Kalman filtering is often inadequate in a vision system because it is unimodal: being based on the Gaussian distribution it can only represent a single hypothesis. Condensation attempts to use learnt dynamic models and visual observations to propagate multiple possible interpretations over time.

Imagine trying to track an object over a cluttered background. Vision techniques may identify several possible features, that could form the object or could simply be part of the background. This would result in a possibility distribution with multiple peaks, each one of which could represent the object. It is in this situation that the Condensation algorithm is far more appropriate than Kalman filtering.

2.1.5 Model-based Tracking

**Model-based Tracking** involves using contextual information about the behaviour of an object to help with the tracking problem. In many ways model-based tracking is similar to the probabilistic inference algorithms discussed above, in that they both attempt to predict where an object will be based on knowledge of the typical patterns of motion that the object is capable of. Kalman filtering and Condensation rely on the physical property that an object cannot transport to another part of the scene. Models provide a much more sophisticated set of constraints for the movement of the object they are based on.

The framework presented by Rehg and Kanade[23] attempts to use a kinematic model of a human hand to predict occlusions that would otherwise impair the system’s ability to track finger movements. Occlusion will always be a problem when attempting to track articulated bodies such as fingers, standard template matching techniques cannot cope with partially occluded objects.

The idea of using a model to aid tracking is a good one. The model can predict occlusions as well as reduce the possible state space of the system (The model will have far more constraints on movement than a general tracking system. This means that impossible configurations of objects in the scene can be ignored). However, defining a good model can be problematic. The human hand has 27 degrees of freedom, some of which are related and constrained by others. Attempting to parameterise and model this is a difficult problem. There are also difficulties in associating the model with the 2D image data from the capture source. Gavrila and Davis[12] took a novel approach with their system. They approached the problem of pose recovery for articulated limbs as a search problem. The system entailed matching the synthesised appearance of a 3D human model with the captured image data. The pose parameters are stored in the model, reducing the problem to that of finding the most similar synthesised appearance.
2.2 Computational Stereo

Computational Stereo is the process of 3D reconstruction from multiple capture sources. The process is based on triangulation, if the relative position of the same feature is known in multiple sources, and the position and orientation of the cameras are known, the 3D coordinates of that feature can be recovered.

![Figure 2.1: Stereoscopic cameras viewing an object](image)

In Fig 2.1 the 3D coordinates of the point $P$ can be recovered by calculating the intersection of the two vectors $(pl - Cl)$ and $(pr - Cr)$.

2.2.1 Epipolar Geometry

Epipolar Geometry can be used as a technique for reducing the search space for image features in secondary capture sources. Once a feature has been identified in one source, everything apart from the depth is known. This means that the feature could lie at any depth on a line passing through the capturing camera’s image plane. This line will map to a different line in the coordinate system of a secondary camera.

In Fig 2.2 the points $Cl$, $P$ and $Cr$ form the epipolar plane and the blue line represents the epipolar line. Knowing that the feature must fall on the epipolar line in the secondary image plane greatly reduces the search space.

2.3 Commercial Tracking Solutions

2.3.1 Hawk-eye

Hawk-eye[17] is an optical tracking system that was developed in 2001 by Dr. Paul Hawkins whilst working for Roke Manor Research Limited. The
technology was turned into a commercial product by the spinoff company Hawk-Eye Innovations Ltd and has since been used to embellish the televised coverage of cricket and tennis, providing computer simulated replays following the path of a ball.

The exact details behind the Hawk-Eye system are a closely guarded secret. Six cameras are used to capture video data at up to 100 frames a second. The captured images are then processed by software to produce a 3D model of the ball. The future path of the ball is predicted using a parametric model with a claimed 5mm accuracy. The Hawk-Eye system is an excellent example of a passive optical tracking system that has been made commercially viable. It should be noted, however, that Hawk-Eye does not always produce a correct answer or at times, any answer. The system has the luxury of being a commentators tool rather than a mission-critical input device.

2.3.2 MotionStar®

MotionStar® [11] is a full body motion capture magnetic tracker. The system offers 6 degree of Freedom (DOF) for each sensor and will capture the position and orientation up to 144 times a second. As with all magnetic tracking solutions, the system is prone to interference from ferrous metals in the surrounding environment. The measurement accuracy decreases as the distance between the magnetic field transmitter and the sensors increases. At 1.52m the system has an accuracy of 0.08cm but doubling the distance causes the accuracy to fall to 0.25cm.

There are many advantages to the MotionStar® system if the application permits adding magnetic sensors to the scene. The high refresh rate and relatively good spatial resolution make it an accurate tracker. However at a price tag
of $25,000 it is only really accessible to professional motion capture companies.

2.3.3 A.R.T. System

The A.R.T System \[14\] is an infrared optical tracker. The system uses photo reflective markers and multiple infrared cameras to calculate position and pose of objects in the scene. Each camera, in turn produces an infrared flash that is reflected by the markers back towards the camera. Hardware processing on the camera unit determines the 2D image coordinates of each marker, which are then sent to a central computer for processing.

The central computer collects the 2D image coordinates from every active camera and calculates the 3D world coordinates of the marker. The orientation of an object in the scene is determined by the strictly controlled positions of a group of markers. Markers are placed on an object in such a way that the software knows the relative positions of those markers. If the orientation of the object changes, the absolute position of the markers will change but the relative position will remain the same, allowing the pose to be calculated.

The A.R.T. System runs at a maximum of 60 frames a second and has an absolute positional accuracy of 0.4mm and orientation accuracy of 0.12 degrees. The infrared cameras have between a 3m and 10m range depending on the size of the markers. The passive marker system could be seen as weakness in the product, when running in “accuracy mode” the system will only support three 6DOF targets\(^1\) compared with twenty in “fast mode”. However the A.R.T system is still a good accurate tracker and it is especially well suited to the field of human computer interaction.

2.4 Software APIs

2.4.1 DirectShow Technology

DirectShow is an API that Microsoft first released in 1998 and it has since grown to become the underpinning technology of almost all Windows based multimedia. DirectShow is distributed as part of the Microsoft Platform SDK\[^3\] and is freely available to download. Although it is widely considered to be one of the most complicated APIs Microsoft has ever released, DirectShow can offer several benefits to this tracking project. The API is an extensible filter-based framework that will provide the freedom to easily change the video capture source without modifying much sourcecode. This means that other cameras can easily be plugged in, or alternatively a video clip could be used as the input for the filter graph providing a platform for repeatable experiments.

2.4.2 OpenCV

Intel’s OpenCV \[^4\] is an open source library containing functions aimed at solving real time computer vision problems. There is a lot of community based information available at the OpenCV Wiki\[^5\]. The functions and algorithms implemented by the OpenCV library have been optimised to work especially

\(^1\)A six degrees of freedom target is an arrangement of markers such that position and orientation can be recovered.
well with Intel processors and it is recommended that anyone implementing this project looks to this when addressing the vision aspects of the system.
Chapter 3

Tracking

3.1 Problem

Optical tracking is a complicated problem that draws on many fields of computer vision. The literature on the subject is largely presented in isolation concentrating on various aspects of the problem without providing an overview of the entire process. The following chapter is intended as a conceptual guide to the entire tracking process. The problem has been broken down into five separate areas of concern, at each stage the main difficulties and techniques are discussed, providing a sound theoretical background from which to build a tracking system.

The five tracking problems are:

1. Camera Calibration - Extracting an accurate model of the cameras.
2. Image Rectification - Correcting lens distortion.
3. Feature Extraction - Identifying the item being tracked.
4. 3D Triangulation - Calculating the 3D coordinates of the tracked items.
5. Correspondence - Matching measurements with items being tracked.

The following sections describe these five concerns in further detail outlining the mathematics involved in designing a tracking system.

3.2 Camera Calibration

Camera Calibration is an essential part of a vision based tracking system where the intention is to retrieve three-dimensional information from a set of two-dimensional images. The calibration process provides three key pieces of information.

- Intrinsic Parameters - Parameters of the camera such as Focal length, pixel skew and the principal point.¹

¹The principal point is normally the center of the image.
• Extrinsic Parameters - The location and orientation of the camera in world space.

• Lens Distortion Coefficients - Radial and tangential coefficients introduced as an artefact of the camera’s lens.

Much work has been done to try and establish accurate calibration techniques. These can be categorised into two distinct areas. The first of which stem from the photogrammetry community and is concerned with measuring the distortions observed in photographs of objects whose 3D geometry is precisely known. Early work done by Brown[10] suggested using a series of plumb lines and analytically determining lens distortion by measuring variations in straightness. Whilst this approach can produce very accurate results it is impractical as it requires a precise calibration object and a highly accurate camera setup.

The second area of proposed calibration techniques is known as Self-calibration. It is named Self-calibration because a well known calibration object isn’t required. Finding correspondences between features in multiple views of the same scene provides enough information to calculate the camera’s internal parameters.

The calibration process implemented in OpenCV[4] and used for this project uses multiple views of a planar calibration rig. This process was explored by Zhang[24] and provides a flexible and convenient way to calibrate a set of stereo cameras.

3.2.1 Lens Distortion

Lens Distortion is an unavoidable artefact present in any camera image. Lens distortion can be described as a 4x1 vector containing both radial and tangential distortion coefficients. Figure 3.1 shows a raw camera image without correcting for lens distortion. The Marlin cameras (see Chapter 4) that were used to capture the images in Figures 3.1 and 3.2 are equipped with expensive lenses. However it is still visible that the checkerboard pattern around the edges of the image does not follow the straight lines of the grid overlay. Figure 3.2 is the same image only rectified against lens distortion. It is easy to see that the checkerboard pattern is a much closer fit to the straight lines of the grid overlay. The image is a more accurate representation of the world it is viewing.

The mapping between raw pixel coordinates and rectified pixel coordinates is given in Equation 3.1

\[
\begin{align*}
x' &= x * (1 + k_1 r^2 + k_2 r^4) + 2p_1 xy + p_2 (r^2 + 2x^2) \\
y' &= y * (1 + k_1 r^2 + k_2 r^4) + p_1 (r^2 + 2y^2) + 2p_2 xy
\end{align*}
\]

(3.1)

where

\[r^2 = x^2 + y^2; \ k_1, k_2 \text{ are radial distortion coefficients and } p_1, p_2 \text{ are tangential distortion coefficients.}\]
Figure 3.1: Native camera image with straight line grid overlaid

Figure 3.2: Rectified Camera image with straight line grid overlaid
3.3 Feature Extraction

**Feature Extraction** is an essential part of the tracking process. The accuracy of the 3D triangulation and hence the tracking system, is largely dependent on the ability to detect reliably the location of an object within the scene. There are several aspects to consider when designing a solution to the feature extraction problem.

- Reliably segmenting the tracked items from the rest of the scene. There have been numerous image segmentation techniques proposed for computer vision and it is largely domain specific as to which ones are most suitable.

  The juggling tracker example uses colour segmentation. Colour-based segmentation works relatively well however it does have some disadvantages, the main one being that it constrains the environment. In order for the segmentation to work correctly the juggling balls must be a unique colour that is not visible anywhere else in the scene. Colour segmentation also requires controlled lighting. Although a Hue, Saturation and Luminance colour model can theoretically be light invariant the reality is that the presence of non-white light introduces new colours to a scene altering the segmentation values. Figure 3.3 illustrates how changing lighting conditions can affect segmentation. The shape of the segmented “blob” changes as the direction of incident light changes. This will introduce uncertainty in the location of the extracted features.

![Figure 3.3: Affect of lighting on colour based segmentation](image)

- Reliably detecting the presence of an object does not guarantee an accurate triangulation. It is necessary to identify the coordinates of a single
point on the object. Using the example of tracking juggling balls, this single point is the center of mass for the segmented “blob”. It works well because a juggling ball is spherical meaning that it will be projected to the same shaped “blob” on the image plane of all the cameras despite them being located in different positions. However it may not be as straightforward for other tracking applications. Figure 3.4 illustrates how a non-uniform object will be projected onto different cameras. It is obvious that the centers of mass for these “blobs” do not correspond to the same point.

Figure 3.4: Projection of a non-uniform shape

- Computational speed. There is a tradeoff between performance and accuracy of the feature extraction. Performing image processing operations on multiple camera frames at high resolutions is going to be computationally expensive. This is a major concern for real time applications. There are techniques available that can direct the feature extraction reducing the search space, and consequently the computational time. These are discussed in Section 2.1.4. Benchmarks for the tracking system presented here can be seen in Chapter 6.

3.4 Triangulation

**Triangulation** is the process of determining three-dimensional world coordinates for an object given two-dimensional views of it from multiple cameras.

From the construction shown in Figure 3.5 the problem can be solved by calculating the intersection of two vectors in 3D space. In reality process noise and the quantisation of the image planes means that the two vectors will rarely intersect, as a result the problem now becomes one of calculating the midpoint of the minimum cord between the two vectors.
3.4.1 Intersection of two lines in 3D space

Figure 3.6 illustrates the problem more clearly. Expressing \( P_a \) and \( P_b \) as the vector equation of a line results in the following.

\[
P_a = P_1 + \mu_a(P_2 - P_1) \tag{3.2}
\]

\[
P_b = P_3 + \mu_b(P_4 - P_3) \tag{3.3}
\]

Thus finding the minimum cord between the two vectors is a matter of minimising the equation

\[
\|P_a - P_b\|^2 \tag{3.4}
\]
However this does not present a practical algorithm. A different formulation and one that is more conducive to computation is to notice that the minimum cord between two lines will in fact be perpendicular to both those lines.

Knowing that the dot product of two perpendicular vectors is zero leads to the following equations.

\[(P_a - P_b).(P_2 - P_1) = 0\]  \hspace{1cm} (3.5)
\[(P_a - P_b).(P_4 - P_3) = 0\]  \hspace{1cm} (3.6)

Expanding \(P_a\) and \(P_b\) gives

\[(P_1 - P_3 + \mu_a(P_2 - P_1) - \mu_b(P_4 - P_3)).(P_2 - P_1) = 0\]  \hspace{1cm} (3.7)
\[(P_1 - P_3 + \mu_a(P_2 - P_1) - \mu_b(P_4 - P_3)).(P_4 - P_3) = 0\]  \hspace{1cm} (3.8)

The two simultaneous equations can be used to solve \(\mu_a\) and \(\mu_b\) which in turn can be used to calculate the points \(P_a\) and \(P_b\). Once these two points have been obtained the closest point of interception \(P_t\) is simply the midpoint:

\[P_t = \frac{(P_a + P_b)}{2}\]  \hspace{1cm} (3.9)

### 3.4.2 Coordinate Systems

The techniques described in section 3.4.1 for finding an object’s 3D location are only valid when all of the vectors used are in the same world coordinate system. This presents a problem because the location of an extracted feature is only known in the image coordinate system.

In total there are three coordinate systems involved in stereoscopic reconstruction.

1. **Camera Coordinates**
   Camera coordinates are three dimensional coordinates relative to the location of the camera. This means that the camera is at the origin of this coordinate system. A camera will typically map from camera coordinates to image coordinates through a perspective projection.

2. **Image Coordinates**
   Image coordinates are two dimensional and the result of a perspective projection. In general depth information cannot be recovered from a set of image coordinates because the perspective transformation cannot be inverted. This is why other techniques such as stereoscopic reconstruction are employed.

3. **World Coordinates**
   World coordinates are three dimensional coordinates relative to an arbitrary origin. Given that the location of a camera will be expressed in world coordinates, a rigid body transformation can be calculated to map from world coordinates to camera coordinates.
The camera projection matrix is a mapping from world coordinates to image coordinates and as such provides all of the information needed to calculate the vectors used in the triangulation.

The camera projection matrix is a 3x4 matrix that projects from Euclidean three dimensional space to image space providing a mapping from the world coordinate system to the image coordinate system. It is defined as follows.

\[ P = C[R|T] \]  

(3.10)

Where

\[ C = I \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \]

\( C \) is the camera matrix, which is formed from the intrinsic matrix \( I \), produced during the calibration process and a perspective projection matrix. \([R|T]\) is the extrinsic matrix that represents the transformation from world space to camera space.

Fully expanding Equation 3.10 gives:

\[ x = \begin{bmatrix} \mu \\ u \\ 1 \end{bmatrix} = I \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \]

(3.11)

\[ x = \begin{bmatrix} \mu \\ u \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \]

(3.12)

It should be noted that the camera projection matrix \( P \) cannot be inverted because of the perspective projection matrix

\[ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = 0 \]

However both the intrinsic and extrinsic matrices are invertible. Multiplying the image point by the inverse of the intrinsic matrix partially transforms the coordinate from image-space to camera-space. The transformation is only partial because the perspective transformation has not been inverted also.

\[ I^{-1} \begin{bmatrix} \mu \\ u \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \]

(3.13)

The consequence of this is that the three dimensional coordinates of the object cannot be recovered because nothing is known about the depth component \( Z_c \). However the coordinates must lie on a line from the object to the camera that passes through the image plane.
Knowing that the image plane is at depth $f$, where $f$ is the focal length of the camera and that the camera itself is at the origin we can define a ray that passes through the object as:

$$r = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \mu \begin{bmatrix} f\alpha \\ f\beta \\ f \end{bmatrix}$$ (3.14)

Multiplying through by the focal length follows logically from the situation depicted in Figure 3.7 but is not entirely necessary; $f$ simply acts as a scaling factor that is divided out when the image coordinates are homogenised.

Now that the ray has been defined in the camera coordinate system, it is a simple matter to apply the inverse of the extrinsic matrix to transform the entire ray into world space, where it can be used to successfully triangulate the 3D coordinates of the object.

### 3.5 Correspondence

**Correspondence** is concerned with matching pieces of data that are logically connected in some way. For the problem of tracking there are two circumstances in which correspondence may be considered. Firstly measurements from multiple sources can be combined using the epipolar properties present in stereo camera systems (see Section 2.2.1). This can be considered as spatial correspondence. Secondly measurements need to be linked with items that are being tracked. This is simple for a single object but can become more difficult if the system is attempting to track multiple objects, such as the juggling application. I have termed this *temporal correspondence*.

The discussion of epipolar geometry in Section 2.2.1 explains the relationship between stereo cameras but does not present an algorithm for testing spatial cor-
respondence. As a reminder Figure 3.8 outlines the construction of an epipolar line.

![Epipolar Geometry](image)

Figure 3.8: Epipolar Geometry

It should be clear that the epipolar line is the intersection between the right camera’s image plane and the plane formed by points \( C_l, C_r \) and \( P \). This plane is often denoted by \( \pi \) and is refered to as the epipolar plane. An algorithm could be written that derives the equation of the epipolar line using the plane intersection method and then tests if a feature point lies on that line. Whilst this is a perfectly acceptable solution it does not provide a quick and easy way for testing for spatial correspondences.

### 3.5.1 Fundamental Matrix

The **Fundamental Matrix** is the algebraic representation of the epipolar geometry. It is a 3x3 matrix which has the property that two points from different cameras are in correspondence if:

\[
    p_l^T F p_r = 0 \quad (3.15)
\]

where:
- \( p_l \) is a point in the left camera’s image plane
- \( p_r \) is a point in the right camera’s image plane

Equation 3.15 shows that once computed the fundamental matrix can be used to test quickly for spatial correspondence. It also shows that the fundamental matrix \( F \) can be derived from image point correspondences alone with no knowledge of the intrinsic camera parameters of the image sources. The mathematics behind this are described in detail in chapter 11 of Zisserman [15]. Intuitively one can see that given enough values for \( p_l \) and \( p_r \) and knowing equation 3.15 the unknown matrix \( F \) can be estimated.

Using the fundamental matrix method provides benefits other than speed and convenience. The matrix has some useful properties.
• If $F$ is the fundamental matrix between $C_l$ and $C_r$ then the transpose $F^T$ is the fundamental matrix between $C_r$ and $C_l$.

• The equation of the epipolar lines can easily be derived using Equation 3.15.

$$e_r = F p_l$$
$$e_l = F^T p_r$$

Where $e_r$ and $e_l$ are the epipolar lines in the right and left image plane respectively.

### 3.5.2 Beyond Two Cameras

The fundamental matrix described in Section 3.5.1 illustrates the epipolar geometry of two cameras, but what can be used for finding spatial correspondences in three or more cameras?

Mathematic constructions that encapsulate the geometric relationships between three and more cameras exist but they become progressively more complicated and impractical.

The trifocal tensor for three cameras is analogous to the fundamental matrix described for two cameras (see Section 3.5.1). Like the fundamental matrix it can be derived from either the camera’s intrinsic parameters or simply using known feature point correspondences. Zisserman\[15\] describes the construction and computation of the trifocal tensor in Chapter 15 of his book and goes on to discuss the quadrifocal tensor for four cameras (this is stated for reference only).

The solution employed by tracking system presented in this report is to use several fundamental matrices and to check for correspondence between each pair of cameras in the system. Two points, that are found in the image planes of different cameras are in correspondence if they follow the fundamental matrix equation (Equation 3.15). In reality the relationship imposed by Equation 3.15 has to be relaxed. Inaccuracies introduced as a result of floating point rounding errors and the fact that the fundamental matrix is numerically estimated mean that the equation should be treated as:

$$p_l^T F p_r \approx 0 \quad (3.16)$$

This method has some practical benefits that more sophisticated methods lack. Using fundamental matrices between pairs of cameras allows a camera to be removed from the system without needing to recalibrate it. This would not be possible with a tensor solution because the tensor can only be defined with three cameras. This flexibility helps to facilitate the testing described in Chapter 6.

### 3.5.3 Temporal Correspondence

Establishing spatial correspondence is important for an accurate triangulation (see Section 3.4) and would be good enough for a static scene. However once the tracking of motion is introduced it is essential that consecutive measurements of an object’s position be associated with that object. Without temporal
/*
 * pl = feature coordinates in the left image plane.
 * pr = feature coordinates in the right image plane.
 * c1.fundamental(c2) is the fundamental matrix relating camera c1 to
 * camera c2.
 * CheckCorrespondence() is a function that tests if the two given points
 * obey the fundamental matrix equation.
 */

bool inCorrespondence = true;

foreach(Camera c1 in cameras)
{
    foreach(Camera c2 in cameras)
    {
        if (c1 != c2)
        {
            if (!CheckCorrespondence(c1.fundamental(c2), pl, pr))
            {
                inCorrespondence = false;
                break;
            }
        }
    }
}

correspondence the output from the tracker would fluctuate based on the order
in which the feature extraction algorithm detects feature points.

This is often posed as the quintessential tracking problem and many methods
have been proposed for solving it. Section 2.1.4 describes the Kalman Filter[19]
and the Condensation Algorithm[18], both of these ideas can be used as prediction
filters. The notion of accurately predicting the location of where an object
will be at some future time point is an important one. Most literature on the
subject presents prediction filters as a way of reducing the search space for a fea-
ture extraction algorithm, however for the purpose of temporal correspondence
the predictions can be used for associating measurements with items.

At this point it is important to be clear about the distinction between items
and features. A tracked item is a real world object that the system wishes
to track. A feature is the result of the feature extraction algorithm and pro-
vides evidence as to the location of the tracked item. Given a prediction of
the tracked item’s location and the coordinates of the extracted features the
temporal correspondence problem become a minimisation problem.

$$ S = \sum_{i=0}^{n} \sum_{j=0}^{n} \|P_i - E_j\|^2 $$

(3.17)

where $P_i$ is the predicted location of the $i^{th}$ item and $E_j$ is the location of
the $j^{th}$ extracted feature.
Minimising Equation 3.17 produces the globally minimal assignment of features to items. Equation 3.17 is an expression of the least squares problem. It should be noted that this is a computationally expensive process and does not scale up well to tracking a large number of objects.

It is interesting to note that a prediction filter can make use of a physical model to make the predictions about the item’s location as accurate as possible. Using the example of a juggling tracker, velocity and position can be unknown variables where the velocity changes due to gravitational influence.
Chapter 4

Hardware

4.1 Cameras

An important aspect of any successful vision project is to understand the technical and optical capabilities of the cameras being used.

This project has been developed using three Marlin F-046C[1] cameras manufactured by Allied Vision Technologies. The cameras support the IEEE 1394 FireWire digital interface allowing for easy connectivity and portability. A full reference of the camera’s capabilities can be found in the Marlin Technical Manual[2], Table 4.1 lists the parameters that are most relevant to this project.

![Figure 4.1: Allied Vision Technologies Marlin F-046C](image)

It is important to note some restrictions that the Marlin cameras have that are not illustrated in Table 4.1.

- The cameras can operate at up to 60 frames a second but the highest frame rates are only available in the Mono8\(^1\) colour mode.
- The highest resolution is 780x582, but this is only available in Format7 mode which is a non default mode of operation.

\(^1\)Mono8 represents each pixel as a single byte. This means an image can only be expressed in 256 gray levels.
Table 4.1: Marlin parameters of interest

<table>
<thead>
<tr>
<th>Colour Modes</th>
<th>RGB8, YUV422, YUV411, Mono8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Resolutions</td>
<td>(320x240, 640x480) Format0 (780x582) Format7</td>
</tr>
<tr>
<td>Frame Rates</td>
<td>3.75, 7.5, 15, 30, 60 FPS</td>
</tr>
<tr>
<td>CCD</td>
<td>1/2” (Diag 8mm) type progressive scan Sony IT CCD</td>
</tr>
<tr>
<td>Transfer Rates</td>
<td>100, 200, 400 Mb/s</td>
</tr>
<tr>
<td>Shutter Speed</td>
<td>11us/67s</td>
</tr>
</tbody>
</table>

• Setting the frame rate does not adjust the shutter speed accordingly. The cameras cannot run at the high frame rates if the shutter time is longer than the frame interval time. The camera API allows for manually configuring the shutter speed but only accepts values in the range of 1 to 4095. Equation 4.1 gives the shutter value to shutter time conversion and Equation 4.2 gives the required maximum shutter value needed to achieve the given frame rate.

\[
\text{shutterTime} = \text{shutterValue} \times \text{timeBase} + \text{offset}
\]

Where the default values are:

\[
\text{timeBase} = 20 \mu s \\
\text{offset} = 24 \mu s
\]

\[
\text{shutterValue} = \left\lfloor \frac{1}{\text{framerate}} \right\rfloor - \frac{24 \times 10^{-6}}{20 \times 10^{-6}}
\]

Figure 4.3 illustrates the camera’s imaging pipeline and is included to give a better intuition about the capabilities of the Marlin Cameras.

4.1.1 Triggering

The Marlin cameras support two modes of external triggering that can be used to synchronise image acquisition.

1. Software Triggering - The Marlin cameras can be made to start their image capture cycle (see Figure 4.3) whenever they receive a specific signal through the FireWire bus. This signal is sent to every camera connected to the bus by means of a broadcast packet. Using this broadcast packet, all cameras on the same bus can be synchronised. However there is no similar mechanism available for synchronising cameras on different FireWire buses.

2. Hardware Triggering - The Marlin cameras are equipped with an I/O connector designed to allow external devices to control certain functions of the camera. Figure 4.2 is a diagram of the I/O connector with table 4.2 describing the functionality of each pin.

Pin four is the default external trigger but the other input pins can be configured to act as triggers. All inputs configured as triggers are linked...
Figure 4.2: HIROSE I/O connector pin assignments [Marlin Technical Manual [2]]

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>External GND</td>
<td>Ground for RS232 and external power</td>
</tr>
<tr>
<td>2</td>
<td>Power IN (CCD models only)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>GP Input 1 (default trigger)</td>
<td>TTL rising or falling edge</td>
</tr>
<tr>
<td>4</td>
<td>GP Input GND</td>
<td>Common ground for inputs</td>
</tr>
<tr>
<td>5</td>
<td>RS232 RxD</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>RS232 TxD</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>GP Input 2</td>
<td>TTL rising or falling edge</td>
</tr>
<tr>
<td>8</td>
<td>GP Output 1 (default IntEna)</td>
<td>Open emitter</td>
</tr>
<tr>
<td>9</td>
<td>GP Output 2</td>
<td>Open emitter</td>
</tr>
</tbody>
</table>

Table 4.2: HIROSE connector pin functions

...together using AND logic. Furthermore it is possible to control the polarity of the triggering pulse by writing accordingly to the appropriate DCAM register. Full details of the hardware trigger can be found in the Marlin technical manual[2].

4.2 IEEE 1394 - FireWire

FireWire is a well established digital video and audio serial bus interface that offers high speed data transmission between most digital camera devices and computers. The Marlin cameras described in Section 4.1 provide a standard 6-Pin FireWire interface supporting FireWire 400 which can offer data rates of up to $400 \text{Mbs}^{-1}$. 
4.2.1 Bandwidth

In order to realise fully the imaging potential of the Marlin cameras we need to consider the bandwidth requirements for each camera. Let us consider the situation where a camera is running in the highest resolution, frame rate and colour depth available. For these specific cameras that would be (780x582) with a colour format of RGB8 running at 30 frames per second.

A single frame would require:

\[ frame = 780 \times 582 \times 3 = 1361880 \text{B} \]  \hspace{1cm} (4.3)

Therefore the total bandwidth required would be:

\[ \text{Bandwidth} = 1361880 \times 30 \times 8 = 311.71 \text{Mbs}^{-1} \]  \hspace{1cm} (4.4)

FireWire 400 is stated to run at 400Mbs\(^{-1}\) but measured throughput is never any higher than 393.216Mbs\(^{-1}\). Equation 4.4 shows us that the cameras can be used at their highest settings but we will need to limit the load to one
camera per FireWire bus. With this in mind this project has been developed using three Connectland PCI IEEE 1394A expansion cards.
Chapter 5

Software

5.1 Camera APIs

Allied Vision Technologies provide several software interfaces for communicating with their cameras that they package into an overall Software Development Kit (SDK) called FirePackage. The SDK is hierarchical and provides different levels of abstraction. It is important to determine which interface is the most appropriate and what level of control is needed over the camera’s functions.

- FireDrv - The FireWire driver, this driver replaces the stock Microsoft driver and restricts bus access.
- FireCtrl - A legacy C style API that deals with low-level calls to the device driver. This interface is now considered to be deprecated.
- FireStack - A C style API for bus access that deals with low-level calls to the driver. This interface provides full control over the FireWire bus.
- FireClass - A C++ class based API that encapsulates the functionality of FireStack. This interface is particularly useful if low-level FireWire bus access is required.
- FireGrab - A high level C++ API that presents the user with a simplified view of the camera’s functions. This interface is considered to be the end-user interface and is the easiest way to work with the Marlin cameras.

AVT also provide a second SDK called AVT Direct FirePackage that is built on Microsoft DirectShow Technology. This SDK uses a DirectShow compatible driver which means that the cameras can be used as the input for other multimedia applications on the Windows platform. Initially the tracking system presented here was developed around the Direct FirePackage API. There are many advantages gained by using a component based multimedia pipeline such as that offered by DirectShow. The modular pipeline means that different image sources such as WebCams or video files could easily take the place of the FireWire cameras in the system. However instabilities in the DirectShow compatible driver resulted in regular system crashes. At the time of writing Allied Vision Tech had no information available about an update to the DirectShow
driver making this solution unfeasible. A consequence of this is that development of the tracking system switched to using the native APIs, specifically the FireGrab API. FireGrab was chosen for its ease of use and simplified interface.

5.2 Multi-Threading

There are several architectural challenges that must be resolved in order to build a robust tracking system. Multiple cameras will produce image frames concurrently, executing as different threads of control. The feature extraction and tracking code will consume the images for a given frame once every camera in the system has produced one. This situation leads to concurrency issues inherent to multi-threaded programming.

The solution presented here uses a standard producer/consumer design to synchronise the executing threads. A \textit{FrameBuffer} class was created to act as a critical region. The consumer thread calls the frame buffer requesting an array of frames for processing. This request blocks until each producer has placed an image in the buffer. Figure 5.1 illustrates this principle more clearly.

![FrameBuffer](image.png)

Figure 5.1: Image producers and consumers synchronisation using a FrameBuffer
5.3 Architecture

Chapter 3 details a theoretical overview of the processes and problems involved in stereoscopic tracking. Subsequently the design of the motion tracking software presented here shares many of the same conceptual elements that were covered in that chapter. Once the technical details of concurrently producing images (see Section 5.2) have been solved there are four stages of processing that need to be completed:

1. Image rectification - Correct for lens distortion
2. Feature Extraction - Identify what the system is trying to track in the given images
3. 3D triangulation - Calculate the 3D coordinates of the items being tracked.
4. Correspondence - Matching measurements with items being tracked.

Chapter 3 also discusses camera calibration which is a key component of stereoscopic tracking. Implementation details associated with calibration are considered in Section 5.4. Figure 5.3 illustrates a simplified class diagram of the tracker with key methods listed.

5.3.1 Image Rectification

Correcting for lens distortion is conceptually straightforward once the distortion parameters are known. Section 3.2.1 describes the transformation that maps raw pixel coordinates onto rectified pixel coordinates. As a reminder Equation 5.1 lists the transformation again.

\[
\begin{align*}
x' &= x \times (1 + k_1 r^2 + k_2 r^4) + 2 p_1 x y + p_2 (r^2 + 2 x^2) \\
y' &= y \times (1 + k_1 r^2 + k_2 r^4) + p_1 (r^2 + 2 y^2) + 2 p_2 x y
\end{align*}
\]

(5.1)

where \( r^2 = x^2 + y^2 \), \( k_1, k_2 \) are radial distortion coefficients and \( p_1, p_2 \) are tangential distortion coefficients.

OpenCV[4] provides the function \texttt{cvUndistort2()} which is a direct implementation of Equation 5.1. However image rectification is a process that must be performed on every frame produced by every camera. As a result it is more efficient to use Equation 5.1 to calculate the mapping for each pixel and store it in memory than it is to repeatedly use the formula on each pixel. The lens distortion never changes so rectification is simply a matter of applying the stored mapping. OpenCV provides two functions that can perform this task: \texttt{cvInitUndistortMap()} creates the mapping and \texttt{cvRemap()} applies the mapping to an image.

Image rectification is a function of the \textit{Camera} class. The camera captures the image, applies the undistortion mapping and then places the image into the \textit{FrameBuffer} (see Section 5.2).
5.3.2 Feature Extraction

Several image processing classes were written that are primarily responsible for performing the feature extraction routine. The abstract class \textit{BaseImageProcessor} is the superclass of all feature extraction implementations. The abstract class handles the consumer threading and synchronisation issues described in Section 5.2, allowing the concrete implementations to deal exclusively with the problem at hand. Concrete image processor implementations include the following classes:

1. \textit{ColourTracker} - Performs feature extraction using colour-based image segmentation (see Section 2.1.3). As an example, this class is used to segment juggling balls from the background as part of the juggling application described in Chapter 1.

The \textit{ColourTracker} segments the image based on Hue and Saturation thresholding. Once the “blobs” have been identified the OpenCV \texttt{cvFindContours()} algorithm can be used to transform the groups of pixels into more meaningful contours. Finding contours is essentially a graph search where neighbouring pixels are either included or excluded from the contour based on their binary colour values. An example of the code to perform this is given in Appendix B.

2. \textit{Calibrator} - Performs corner extraction on images of the calibration rig to facilitate camera calibration. Initially corners are identified through edge-based segmentation (see Section 2.1.3) and then further refined to sub-pixel accuracy.

The OpenCV function \texttt{cvFindCornerSubPix()} is an iterative algorithm that takes an estimate of the corner locations and refines the coordinates. The algorithm is based on the observation that every vector from a corner point \( q \) to a pixel \( p \) in the neighbourhood of \( q \) is orthogonal to the image gradient at \( p \) (see Figure 5.2).

![Figure 5.2: Sub-Pixel corner finding](image)

Thus, the optimal corner location is obtained by minimising Equation 5.2.

\[
\epsilon_i = \Delta I_{p_i}(q - p_i)
\]

(5.2)

where \( \Delta I_{p_i} \) is the image gradient at point \( p_i \). In a perfectly accurate image \( \epsilon_i \) will be zero.
3. **CornerTracker** - Very similar to the **Calibrator** class with the exception that it does not have all of the calibration logic in it and it submits the extracted features for triangulation and tracking. This class was used to facilitate the static accuracy experiments described in Section 6.3.

4. **LaserTracker** - Like the **ColourTracker** this class uses colour-based segmentation to identify points of laser light. The class uses the center of mass of the segmented “blob” as the coordinates of the extracted feature. The main difference between this class and the **ColourTracker** is that this class only expects to extract a single “Blob” as it is designed to work in a controlled lighting situation.

### 5.3.3 3D Triangulation

There are several important classes that handle the triangulation process. Section 3.4 outlines the mathematics involved with the intersection of lines in 3D space. The various equations are realised by the two classes; **CRay** and **CVec3**. 

**CVec3** encapsulates a vector in three dimensional space and provides common vector operations such as cross and dot products. **CRay** is a class that represents a vector line between two points. It is the `intersection()` method of this class that calculates the intersection point of two 3D rays.

The **Camera** class supports the transformation of points between the three different coordinate systems as described in Section 3.4.2. The **Camera** class encapsulates the model of the physical camera in that it provides access to the intrinsic and extrinsic parameters that are calculated during the calibration. Knowing the model parameters the **Camera** class is able to provide two essential functions:

- **FromWorldToImage(CVec3* p)** - Apply the 3x4 camera projection matrix to a three dimensional world point. This maps a world point onto an image point which is the principal duty of all cameras. It is a useful function for testing the validity of the camera model.

- **FromImageToWorld(CvPoint2D32f p)** - Convert the point from image coordinates to world coordinates. This function inverts the intrinsic and extrinsic camera matrices to calculate the location of a world point. This is a direct implementation of the equations listed in Section 3.4.2.

### 5.3.4 Correspondence

The abstract class **Tracker** and its concrete implementations are responsible for establishing spatial and temporal correspondence. The concrete class **TemporalTracker** uses a prediction filter to establish temporal correspondence between measurements and tracked items as well as checking for spatial correspondence. The simpler implementation **SimpleTracker** only establishes spatial correspondence.

**Spatial Correspondence**

All applications of a tracking system will need to establish spatial correspondence between two or more extracted features. Section 3.5.1 outlines the steps
that are involved. In a practical system the fundamental matrix equation 3.15
needs to be relaxed to account for numerical inaccuracies introduced through the
estimation of the fundamental matrix and floating point rounding issues. As a
reminder Equation 5.3 lists the relaxed fundamental matrix equation previously
seen in Section 3.5.1.

\[ p_l^T F p_r \approx 0 \] (5.3)

where \( p_l \) and \( p_r \) are points in the left and right camera’s image planes respectively.

Directly coding the relaxed fundamental equation provides a quick and easy
way of establishing whether two extracted features are in spatial correspondence.

**Temporal Correspondence**

Temporal correspondence is essential for matching measurements with tracked
items, when the items in question are moving. In order to guess the optimal
union of measurements and tracked items the system needs to be able to pre-
dict the future location of an item that is being tracked. The abstract class
PredictionFilter provides an interface that a prediction class must adhere to. A
concrete implementation of this is the KalmanFilter class.

OpenCV provides a number of functions that implement the Kalman filter.
For the purpose of motion tracking we want a Kalman filter to predict the
location and velocity of an item. The state of the Kalman filter will be updated
with the known location of an item. This means that the Kalman filter needs
to be initialised to have four dynamic parameters \((P_x, P_y, V_x, V_y)\) and two
measured parameters \((X, Y)\) where \(P_x, P_y\) are the predicted location, \(V_x, V_y\)
are the predicted velocity and \(X, Y\) are the measured location.

Initialising a Kalman filter consists of constructing a model transition matrix,
that is, a matrix that will move a vector of the parameters being modelled onto
a new vector of parameters.

\[ x_k = Ax_{k-1} + w_{k-1} \] (5.4)

where, \(x_{k-1}\) and \(x_k\) are respectively the previous and current model parameter
vectors, \(A\) is the model transition matrix and \(w_{k-1}\) represents process noise.

The KalmanFilter class uses a simple motion model described by Equations
5.5 and 5.6.

\[ P' = P + V \] (5.5)

where, \(P'\) is the new position, \(P\) is the old position and \(V\) is the velocity.

\[ V' = V \] (5.6)

where, \(V'\) is the new velocity and \(V\) is the old velocity. That is to say the
velocity of the ball remains constant.

It should be noted that this is a naive model which can be made far more
effective by including the influence of external forces such as gravity. However,
the optimal motion model is context sensitive. As an example, the juggling
tracker would benefit from including gravity in the model but there are many scenarios where it would not be sensible. The motion model implemented by the *KalmanFilter* class was designed to be sufficiently generic as to be useful in any tracking application.

Expressing the motion model described by Equations 5.5 and 5.6 in matrix form gives the following transition matrix:

\[
\begin{bmatrix}
P'_x \\
P'_y \\
V'_x \\
V'_y \\
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
P_x \\
P_y \\
V_x \\
V_y \\
\end{bmatrix}
\]

(5.7)

Appendix B includes a code example, demonstrating how to initialise the OpenCV Kalman filter using the motion model described here.

Given an estimate of the tracked items location generated by the *PredictionFilter*, the *TemporalTracker* class can attempt to match measurements with items being tracked, in a globally minimal way. The principles behind this idea are explained fully in Section 3.5.3. As a reminder the globally minimal solution is obtained by minimising the following equation:

\[
S = \sum_{i=0}^{n} \sum_{j=0}^{n} \| P_i - E_j \|^2
\]

(5.8)

where \( P_i \) is the predicted location of the \( i^{th} \) item and \( E_j \) is the location of the \( j^{th} \) extracted feature.

The implementation of Equation 5.8 in the *TemporalTracker* class results in an order \( N \) factorial recursive algorithm. The computational demands of such an algorithm mean that it does not scale up to tracking large numbers of items. For example, the static accuracy experiments detailed in Chapter 6 track 320 corners on the static checkerboard pattern. This would result in \( 320! \) operations which is intractable.

The *SimpleTracker* class was developed to offset the computational demands of tracking large numbers of items with the *TemporalTracker*. The class uses a simple in-order algorithm to associate measurements with tracked items. This naive method matches features with items based on the order in which the feature extraction algorithm detects them. That is to say, the first feature detected is matched with the first tracked item, the second feature with the second item and so on. The matching algorithm assumes that the feature extraction algorithm always detects the same number of features and always detects them in the same order. This is a flawed assumption for most tracking applications. However it is applicable to the static checkerboard tracking used in the experiments described in Chapter 6 because the corners never move and the feature extraction algorithm is deterministic.

### 5.3.5 Common Settings

The tracker loads a settings file on startup that determines the behaviour of certain elements in the system. These settings mainly refer to visual output but can also be used to control the feature extraction and hence the tracking context of the system.
5.3.6 Extensibility

The important design elements pertaining to extensibility are the three abstract classes:

- `BaseImageProcessor`
- `Tracker`
- `PredictionFilter`

Each of these classes allows for different concrete implementations that can be used to extend the tracking system. For example, different implementations of `BaseImageProcessor` use various feature extraction techniques to provide calibration, laser tracking, and colour tracking functionality. `BaseImageProcessor` implementations can provide a unique identifier that, if present in the `Settings.xml` file, causes them to be used in the executing tracker.

Concrete implementations of the `Tracker` class can be used to determine different ways in which measurements are associated with items being tracked. Implementations of the `PredictionFilter` class control the tracker’s ability to predict the future location of items being tracked. A possible extension to this class would be to use Condensation (see Section 2.1.4) as the prediction mechanism.

5.4 Camera Calibration

The tracking system presented in this report provides an easy to use calibration routine. A set of up to ten cameras can be calibrated in under two minutes which is extremely convenient if the cameras are regularly being moved.

Calibration is performed by moving a checkerboard pattern, the calibration rig, around in such a way that it is visible in all cameras. Snapshots of the calibration rig where the corners of the checkerboard pattern can be identified are captured every second. In order to make the process as easy as possible, the output from the corner finding algorithm is displayed as an overlay on the video stream. When all of the corners can be identified, a highly visible grid is overlaid to inform the user that the algorithm is working properly.

The only constraint on the calibration process is that a single view is captured where the checkerboard corners are identified in all cameras. This view is then used to establish the world coordinate system (see Section 3.4.2). Once at least thirty snapshots have been captured on each camera, the parameters for each camera will be calculated and can then be saved to an XML file (Appendix A contains an example tracker XML file).

OpenCV provides several functions that will perform camera calibration. The two most important functions are `cvCalibrateCamera2()` and `cvFindFundamentalMat()`. The `cvCalibrateCamera2()` function takes the following as input parameters:

- `object_points` - A 3xN matrix of object points which represent the real world coordinates of the corners detected in the calibration rig. (N is the total number of corners in all views of the calibration rig)
image_points - A 2xN matrix of corresponding image coordinates representing the detected corners

points_count - A 1xM matrix where M is the total number of views of the calibration rig. Each matrix element represents the number of corners detected for that particular view.

image_size - The resolution of the calibration images.

intrinsic_matrix - A pointer to the output 3x3 intrinsic matrix that will be calculated.

distortion_coeffs - A pointer to an output 4x1 vector of distortion coefficients $[k_1, k_2, p_1, p_2]$.

rotation_vectors - A pointer to an output 3xM matrix of rotation vectors. This matrix will contain the rotation part of the extrinsic transformation generated for each view.

translation_vectors - A pointer to an output 3xM matrix of translation vectors. This matrix will contain the translation part of the extrinsic transformation generated for each view of the calibration rig.

Once this function has been computed the model parameters of the camera are fully known.

The cvFindFundamentalMat() function takes as input two 2xN matrices containing the coordinates of corresponding points (the corners of the calibration rig). The function outputs the fundamental matrix relating to the two cameras that generated the coordinates for the input matrices.

With the camera model parameters and the fundamental matrices known the tracking system has all of the information needed to track the three dimensional world coordinates of an item.
Figure 5.3: Simplified software diagram
Chapter 6

Evaluation

The criteria for a successful tracking system can be considered under the following headings:

- Latency - How much delay is there between movement and the detection of that movement?
- Static Accuracy - How accurately can the tracker pin-point the location of a static item?
- Motion Accuracy - How accurately can the tracker follow the path of a moving object?

The following sections detail the experiments performed to try and determine the success of the tracker for each of the evaluation criteria listed above. Section 6.2.2 describes the experiment designed to explore the amount of latency in the system for both two and three cameras at resolutions of 320x240 and 640x480. Section 6.3.1 elucidates the experiment designed to measure errors in static accuracy introduced as a result of the calibration process. The experiments that were designed to establish the affect of introducing a third camera on tracking accuracy are outlined in Section 6.3.2. Finally Sections 6.4.1 and 6.4.2 investigate motion accuracy by exploring the affects of camera shutter speed and capture synchronisation.

All experiments were carried out on a machine with the following specification.

- Processor: Intel Core 2 Duo E6400 @ 2.13GHz
- Memory: 2048MB DDR2
- Operating System: Microsoft Windows XP Service Pack 2

6.1 A Small Taster

In general extracting features in large or complicated shapes is a difficult process. The task can be made considerably easier by using a point light source such as a Light Emitting Diode (LED) or a laser. Segmenting a laser point from
the background image is straightforward and offers a cheap and easy feature extraction algorithm. Using structured lighting in this way presents a novel tracking application. The stream of coordinates for a tracked laser point can be recorded and used to generate a 3D model of an object.

The results of the laser-tracking idea are illustrated in Figure 6.1. The left and right images are respectively the textured mesh and wire-frame view of the 3D model generated using this technique. A point of laser light was methodically passed over a toy monkey and the raw point cloud of three dimensional coordinates was saved to a file. The recorded data was then rendered using the Visualisation Toolkit VTK[6]. Figure 6.2 depicts the raw point cloud, rendered first with and then without a texture. The toy monkey is clearly recognisable, especially in the un-textured side view. The side profile view also shows a smooth outline, suggesting that any inaccuracies in the calculated depth of the points are negligible.

The 3D mesh seen in Figure 6.1 was created by applying a delaunay triangulation to the raw point cloud. The delaunay process essentially operates on 2D point vectors. When it is applied to 3D vectors one of the dimensions is ignored, this is why the wire frame view looks as if the mesh has been stretched down over the monkey’s head. The texture is a snapshot taken by one of the cameras doing the tracking. Calculating the texture coordinates in the model is simply a matter of taking the ratio between the extracted feature coordinates and the image width and height.

Figure 6.1: 3D model obtained by tracking a laser pointer over a toy monkey

6.2 Latency

The latency of a tracking system is defined as the time between a movement and the detection of that movement. The importance of latency is largely dependent on the context of the tracking system. The effectiveness of interactive applications such as the proposed juggling tracker is highly sensitive to the latency in the system. However non real-time systems can compensate for latency and still produce valid data. The Hawk Eye[17] system is an example of a non real-time optical tracking system that has proved to be very effective.
6.2.1 Reasons For Latency

Latency is an unavoidable consequence of using digital cameras. There are several known sources of latency in the tracking system being presented here.

- Charge Coupled Device (CCD) Jitter - The Marlin cameras[1] use an Interline Transfer CCD sensor. This design means that odd stripes on the wafer are light sensitive and even stripes are storage areas. Image capture is treated as an interleaved sequence of exposing the CCD to light and reading out the image using a rolling shutter principal.

  Expose Image → Output Image → Expose Image

In order for the camera to capture a new frame the CCD sensor must be completely reset. As a result of the interleaved capture - read out cycle there is a level of uncertainty in the time taken to reset the CCD. The Marlin technical manual[2] distinguishes two CCD states and quotes the following times for CCD jitter:

1. Camera Idle - The sensor is ready to capture a new image (33.33ns).
2. Camera Busy - The sensor is reading out an image and cannot be reset (32.17µs).

- Frame memory and deferred image transport - The Marlin cameras[1] are equipped with 8 MB of internal RAM. This allows the camera to buffer up to 13 frames in a FIFO image queue before transferring them to the IEEE 1394 FireWire bus. The worst case latency may therefore be as much as:

\[
\frac{1}{\text{framerate}} \times 13 = \frac{1}{30} \times 13 = 0.43s
\]

- FireWire transfer times - FireWire devices allow for two types of data packet transfer.
  - Isochronous - The devices transfer data within fixed time slots.
- Asynchronous - Devices are allowed to transmit data at any time but are not guaranteed to have sole access to the bus. A consequence of this is that asynchronous devices are often required to arbitrate for bus access.

The Marlin cameras\(^1\) are isochronous devices that transfer data packets on a standard FireWire bus in \(125\mu s\) windows. Given a 640x480 image with a colour format RGB8 \(^1\) we can expect the camera to transfer \(640 \times 480 \times 3 = 921600\) bytes per frame. The Marlin cameras use a standard packet size of 3840 bytes which means that transferring an entire frame takes:

\[
\frac{921600}{3840} = 240 \text{ packets} \tag{6.1}
\]

\[240 \times 125\mu s = 0.03\text{s}\]

At the lower resolution of 320x240 the Marlin cameras\(^1\) will only run with a colour format YUV422. \(^2\)

A frame will be \(320 \times 240 \times 2 = 153600\) bytes which would take:

\[
\frac{153600}{3840} \times 125\mu s = 5\text{ms}
\]

- Software buffering - The software implements two forms of buffering. Firstly incoming images get put in a frame buffer until every camera has provided an image for this frame. Once all of the images have arrived they can be further processed by the tracking code. The second form of buffering does not strictly affect the latency. The previous frame is always buffered and used for visual output for the tracker. This means that the video display will be a frame behind the tracking process.

- Discrete Framing - The cameras are capturing images at a discrete interval, namely the frame rate. This introduces an inter-sample latency, which is the time between an external action (such as the screen turning white) and a new sample point. Given that a camera takes a sample once every frame the average inter-sample latency would be half the sample time:

\[
\frac{1}{2 \times \text{framerate}} = \frac{1}{2 \times 30} = 0.016\text{s}
\]

However, as more cameras are added to the system, the processing overhead increases and the frame rate consequently decreases. With three cameras and an average frame rate of 15FPS the inter-sample latency would be:

\[
\frac{1}{2 \times 15} = 0.033\text{s}
\]

\(^1\)RGB8 represents a 24bit pixel value where each Red, Green and Blue channel is an 8bit number.

\(^2\)YUV422 represents a 16bit pixel value. Chrominance is sub-sampled meaning that 4 bytes describes two pixels. In the following sequence u, y1, v, y2 the two pixels would be (y1, u, v) and (y2, u, v).
6.2.2 Latency Experiment

The latency of the tracking system presented here was measured by pointing one, two and three Marlin cameras at a black monitor and timing the lapse between flashing the monitor white and recording the detection of white pixels. The same colour based segmentation that is used for detecting juggling balls was used here to detect the white pixels (see Section 5.3.2 for details about the ColourTracker). A large white rectangle (1/4 the size of the screen) was displayed in the center of the screen, appearing to the ColourTracker as a single “blob”. The significance of this is that the processing overhead of feature extraction contributes to the latency figures. Although knowing the latency of the hardware alone would be useful, the experiment was designed this way in order to obtain the latency figures that we are likely to see in a real tracking scenario. The experiment was repeated thirty times at camera resolutions of 640x480(RGB8) and 320x240(YUV422) pixels.

![Figure 6.3: Tracker Latency Graph](image)

Figure 6.3 shows a fairly constant value of approximately 0.1s for all parameters apart from three cameras at 640x480 which is significantly higher. Each camera in the experiment was attached to a separate FireWire controller so bus contention is not an issue. One explanation of this artefact could be that the processing overhead of feature extraction for three cameras at 640x480 is a bottleneck in the system. The result of this bottleneck is that the tracker runs at a lower frame rate than the cameras are capturing at, it is possible that this imbalance is causing frames to be internally buffered by the cameras (see Section 6.2.1).

The lower frame rate would also be responsible for a larger inter-sample latency, although this is not significant enough to explain such a large artefact. Another explanation may be found by looking at the specification of the testing computer. The machine used for testing (see the beginning of this chapter) has an Intel Core 2 Duo processor, which has two independent processing cores. It is possible that the introduction of a third camera and hence a third compu-
tionally demanding thread of execution, is causing additional contention for processor time that was previously masked for two cameras due to the second processor core. If this is a contributing factor we would expect to see similar increases for four and more cameras. Unfortunately at the time of testing there were only three cameras available for use.

### 6.3 Static Accuracy

Figure 6.2 gives an intuition that the static accuracy of the tracking system is good. However without measurements in a real world-coordinate system we cannot be sure that the model is a faithful representation of the item being scanned. The following experiments aim to provide a quantitative measure of accuracy based on comparing recorded point coordinates with known real world coordinates.

The camera calibration process establishes the world coordinate system by choosing a view of the calibration rig that is visible in all cameras and locking the origin to a corner on the calibration rig. The principal X and Y axes are aligned along the two perpendicular edges of the checkerboard and the Z axis is set to be orthogonal to the other two (see Figure 6.4). Consequently it is very difficult to determine where the absolute coordinates of a physical item will be in the arbitrary world coordinate system.

![Figure 6.4: Establishing the world coordinate system](image)

Static accuracy was measured by tracking the location of three hundred and twenty corners on a 17x21 checkerboard pattern. Each individual square was 28mm by 28mm. It should be noted that this checkerboard pattern is different from the pattern used to calibrate the system. The calibration rig is a smaller checkerboard pattern that is free to be moved around, this pattern is much larger and fixed to a wall. To avoid any confusion the pattern used for these experiments will be referred to as the static checkerboard pattern.

In order to over come the difficulties associated with measuring the location of physical items in an arbitrary coordinate systems, the accuracy experiments
were designed to measure the relative distance of a point from a designated origin. This is illustrated in Figure 6.5. The relative distances can be compared with the real world distance between two points on the static checkerboard pattern because the exact size of the pattern is known.

![Figure 6.5: Static Checkerboard Pattern](image)

6.3.1 Calibration Accuracy

A difficulty of trying to evaluate which parameters effect tracking accuracy is that every time the cameras are moved the system needs to be recalibrated. Section 3.2 outlines the calibration process. Moving the cameras invalidates the extrinsic matrix which describes the relative location and pose of the camera in the world coordinate system. In theory the intrinsic matrix only needs to be calculated once as it will never change. However my calibration implementation is designed to estimate every parameter needed for stereoscopic triangulation.

The calibration algorithm essentially has to guess the camera parameters based on the spatial relationships between corners on the checkerboard calibration rig. In order to compute a correct transformation matrix the calibration routine uses an iterative algorithm that converges on a transformation and minimises back projection error. The fact that the calibration is a numerical estimation means that one can identify both good and bad calibrations that in themselves can affect the accuracy of the tracker. Table 6.1 lists the average error for ten typical calibrations. The key points of this experiment are outlined as follows:

- The average error for this experiment is the average difference between a measured distance from the origin and a known distance, based on the static calibration checkerboard described in Section 6.3.
- A typical calibration cannot reasonably be quantified. Here we take it to mean a calibration that an ordinary user might perform; where no attempt is made to “break” the calibration routine. At least thirty different views of the calibration rig are used to compute the camera parameters. Most literature on the subject suggests at least 10-15 views should be used.

- All point locations are taken as the average coordinates for 240 frames in order to try and eliminate process noise.

- The stereo cameras are 280mm apart and approximately 3m from the checkerboard pattern.

Table 6.1 shows us that the calibration process introduces a significant amount of variance. The variance of the ten error values is 41.34712. Table 6.1 also lists re-projection error. During the calibration routine, the coordinates of the corners in the calibration rig are associated with arbitrary object coordinates; this is how the world coordinate system is established. Knowing the corner measurements and the object coordinates, the algorithm can estimate the camera’s parameters. One way of checking the computed parameters is to use the generated projection matrix (see Section 3.4.2) to re-project the object points onto the image plane. The re-projection error is the mean of the sum of squared differences between the measured corner coordinates and the re-projected object coordinates for each calibration view.

We can see that there is a correlation between re-projection error and average error in Table 6.1. This provides evidence that the variance is a direct consequence of the calibration uncertainty and not introduced as an artefact of feature extraction or triangulation.

An implication of these results is that further experiments designed to elucidate the factors that affect tracking accuracy should be calibration invariant in order to ensure a scientifically rigorous investigation.

<table>
<thead>
<tr>
<th>Calibration Attempt</th>
<th>Average Error (mm)</th>
<th>Re-projection Error (pixels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.270278</td>
<td>0.120904</td>
</tr>
<tr>
<td>2</td>
<td>13.06237</td>
<td>0.158461</td>
</tr>
<tr>
<td>3</td>
<td>10.06936</td>
<td>0.156002</td>
</tr>
<tr>
<td>4</td>
<td>3.368272</td>
<td>0.139261</td>
</tr>
<tr>
<td>5</td>
<td>14.42697</td>
<td>0.152314</td>
</tr>
<tr>
<td>6</td>
<td>0.481943</td>
<td>0.126102</td>
</tr>
<tr>
<td>7</td>
<td>1.305083</td>
<td>0.093768</td>
</tr>
<tr>
<td>8</td>
<td>9.385898</td>
<td>0.197874</td>
</tr>
<tr>
<td>9</td>
<td>18.73769</td>
<td>0.136084</td>
</tr>
<tr>
<td>10</td>
<td>2.524335</td>
<td>0.153662</td>
</tr>
</tbody>
</table>

Table 6.1: Average Error from ten calibrations

6.3.2 Two and Three camera accuracy

The two and three camera static accuracy experiments were designed to investigate the affect of introducing a third camera into the tracking process. Section 3.5.2 describes how three camera triangulation is implemented in this tracking
system. The question that the following experiments are trying to answer is whether or not a third camera increases the spatial accuracy. The experiments are based on the same static checkerboard pattern used in Section 6.3.1.

The details of this experiment are outlined as follows:

- Knowing that calibration variance can affect the tracking accuracy (see Section 6.3.1) this experiment was designed to be calibration invariant. The system was calibrated once using three cameras. Ordinarily implementation details mean that removing a camera from the system would require recalibration. The fundamental matrices that define the relationships between the different cameras, are represented using two indices that identify which cameras the matrix is defined between. Removing a camera changes which index represents which camera. In order to remain calibration invariant the various indices in the calibration file were manually changed to facilitate the two camera setup (see Appendix A for an example tracker XML file). This means that both the two and three camera setups were calibrated using the same calibration views.

- Absolute error is defined as the distance between a measured coordinate and the designated origin based on the checkerboard pattern.

- All values listed are the average of 240 frames, ensuring that the effect of any process noise is minimised.

Figure 6.6 shows the absolute error for both two and three cameras and illustrates two interesting features.

1. Three cameras appear to be significantly more accurate than two. The average error distance for two cameras is 5.5752mm whereas it is only 0.4577mm for three cameras. This is an interesting consequence of the triangulation process described in Section 3.4. Recall that three cameras are treated as three pairs of cameras that produce three independent triangulations. These triangulations are merged into an estimate of the tracked item’s location by taking the average. The fact that three cameras are more accurate suggests that the three independent triangulations must be evenly spread around the item’s true location.

2. The results for two cameras show an upward trend. The absolute error increases as the distance from the origin increases. Given what we know about the stereoscopic reconstruction process, we would expect there to be an error ellipse associated with the estimate for the location of each point which would suggest a horizontal trend. However the presence of an upward trend may be evidence of a cumulative error. Imperfections in the checkerboard pattern may be responsible for some of the error we are seeing. When comparing the measured distances with the real distances, an idealised model of the checkerboard is used. Two assumptions are made about the model.

(a) The pattern is planar - In reality this is not the case, the pattern is a large printed sheet attached at the corners to a flat surface. The result of this is a slight bubbling effect that could introduce error.

53
(b) The squares are $28\text{mm}^2$ - Inaccuracies in the printing process mean the squares are only approximately $28\text{mm}^2$. This is not an issue for short distances but results in an accumulating drift for larger distances. To illustrate this point more clearly the checkerboard pattern was measured with a tape measure. Table 6.2 compared the width, height and diagonal measured distances with those assumed by the model.

We can see from Table 6.2 that the largest measured diagonal distance on the checkerboard pattern differs by $8\text{mm}$ from the value assumed by the model. Unfortunately the model cannot be replaced with actual measured values because the measuring equipment available cannot resolve distances smaller than $1\text{mm}$.

<table>
<thead>
<tr>
<th></th>
<th>Horizontal</th>
<th>Vertical</th>
<th>Diagonal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Distances</td>
<td>420.00</td>
<td>532.00</td>
<td>677.80</td>
</tr>
<tr>
<td>Actual Distance</td>
<td>424.00</td>
<td>538.00</td>
<td>685.00</td>
</tr>
<tr>
<td>Drift (mm)</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 6.2: Checkerboard Model measurement drift

Figure 6.6 illustrates an interesting relationship between two and three cameras that deserves further investigation. Knowing the effect of calibration variance on two cameras (see Section 6.3.1) it is possible that different camera pairs affect the three-camera accuracy in different ways.

The extended three camera accuracy experiment was performed under the same conditions as the first with the only difference being that three, two-camera calibration files were created to represent each pair of cameras in the system.

Extended Three Camera Accuracy

Figure 6.6 illustrates an interesting relationship between two and three cameras that deserves further investigation. Knowing the effect of calibration variance on two cameras (see Section 6.3.1) it is possible that different camera pairs affect the three-camera accuracy in different ways.

The extended three camera accuracy experiment was performed under the same conditions as the first with the only difference being that three, two-camera calibration files were created to represent each pair of cameras in the system.
The three cameras were arranged in a similar way to Figure 6.7. The naming of the camera pairs in Figure 6.8 is based on the numbers seen here. Pair Two.1.0 represents that cameras 1 and 0 are in use.

![Figure 6.7: Three camera stereo rig](image)

Figure 6.7: Three camera stereo rig

Figure 6.8 confirms the results seen in figure 6.6 where three cameras appear to be more accurate than two. However the extra information provided by this experiment shows that one of the camera pairs was actually far more accurate than the other two. This would reduce the average and lower the error seen in three cameras, which suggests that using three cameras has the added benefit of reducing the affect of calibration variance.

Another result seen in Figure 6.8 is that there is far less variance in the errors for three cameras than there are for two. Table 6.3 illustrates this more precisely. Low variance is a useful property as it makes it possible to predict the amount of error for a given point. Depending on the nature of the error it may even be possible to compensate for it producing even more accurate results.
6.4 Motion Accuracy

When determining the affect that motion has on tracking accuracy we first need to establish the difference between a static scene and an animated scene from the perspective of the tracker. There are two assumptions that we can make which will reduce the motion problem to a static scene.

- Cameras can take an instantaneous snapshot - Reducing the time taken for a camera to capture an image to zero is effectively the same as capturing a static scene. The mechanism available for controlling the time taken to capture an image on the Marlin\[1\] cameras is the shutter speed.

- The cameras operate in synchrony - Taking simultaneous snapshots of a moving item is analogous to taking sequential snapshots of a static item. This assumption removes the uncertainty introduced by the motion between one camera capturing the scene and another camera capturing the same scene. The Marlin\[1\] cameras provide both a hardware and a software trigger mechanism that can be used for synchronisation.

6.4.1 Shutter Speed

A camera’s shutter is responsible for controlling the exposure time of the CCD. Long exposure allows more incident light to fall on the CCD, producing a brighter picture. However if the scene changes during the shutter interval the camera will start to expose the new scene resulting in motion blur. Short exposure reduces motion blur but also produces darker images.

The amount of motion blur an item has in a camera snapshot is directly comparable with the amount uncertainty we have in the position of that item. It is this uncertainty that leads to tracking error.

The shutter speed experiment was designed to try and establish a relationship between exposure time and uncertainty. The key details about the experiment are outlined as follows:

- A camera was setup such that the entire image plane was occupied by a computer screen.

- A small white dot was repeatedly scanned horizontally across the computer screen reappearing on the left hand side after disappearing from the right.

- The second moments for the white “blob” captured by the camera were recorded.

- In order to eliminate process noise 450 frames worth of values were captured and averaged.
**Spatial Moments**

Moments can be used to describe the elongation of a shape and are defined in a similar way to mechanical moments:

\[
m_{pq} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (x - x_c)^p(y - y_c)^q f(x, y)dxdy \tag{6.2}
\]

where \((x_c, y_c)\) is the center of mass.

Since we are operating over pixels which are a discrete medium we can replace the integrals with summations. The function \(f(x, y)\) can also be treated as the gray level value of the pixel.

\[
m_{pq} = \sum_{i=1}^{Xres} \sum_{j=1}^{Yres} (i - x_c)^p(j - y_c)^q f(i, j) \tag{6.3}
\]

For a perfectly round “blob” we would expect the x and y second moments to equal. Therefore:

\[
\frac{m_{20}}{m_{02}} = 1
\]

However, as the “blob” becomes elongated in the x direction (motion blur) we would expect the ratio of the second moments to fall:

\[
\frac{m_{20}}{m_{02}} < 1
\]

Figure 6.9: The effect of shutter speed on “blob” elongation

Figure 6.9 does show the expected downward trend, however the results are not as pronounced as we might have first thought. This could be an artefact of the feature extraction algorithm. In order to successfully separate the background from the item of interest some heuristics must be employed. The heuristic used for this experiment is that the foreground “blob” is a sufficiently
different colour to the background. The segmentation process involves thresholding the foreground colour in order to find a contour. It is possible that the thresholding is eliminating some of the darker edge pixels that we would expect to see as a result of motion blur. There is an inherent tradeoff between successfully segmenting the item of interest and allowing for blurred edges.

### 6.4.2 Synchronised Triggering

The stereoscopic tracking process works by using two or more views of an object at a given time point to estimate the three-dimensional coordinates of that object. It is convenient to assume that multiple cameras viewing the scene will simultaneously capture the item, which is what we have done up to this point. In reality the cameras capture the scene at a discrete interval, i.e. the frame rate.

\[
P' = P + \Delta t V
\]  

Equation 6.4 expresses the position of an object as a simple function of time. If we consider \( \Delta t \) to be the time lapse between the first and second cameras capturing the scene we can expect the item to have moved by \( \Delta t V \).

The Marlin[1] cameras provide two triggering mechanisms that can be used to synchronise image capture (see Section 4.1.1).

- The hardware trigger will capture the scene on either the rising or falling edge of an input pulse.
- The software trigger will transmit a broadcast FireWire packet that will cause all cameras on the bus to capture the scene.

The synchronisation experiment was designed to establish the effect that capture synchronisation has on tracking accuracy. The key points of the experiment are outlined as follows:

- Two cameras were positioned such that they were both facing a computer screen.
- Two red circles and a black background were drawn on the screen. One of the circles was stationary at the center of the screen to act as an origin, the other circle orbited the first at a fixed radius.
- The distance between the orbiting circle and the origin circle was calculated and compared with the known distance between the two.
- The known distance between the origin and orbiting circles was calculated using the following equation:

\[
D = \sqrt{\left( \frac{X_p - X_c}{DpiX} \right)^2 + \left( \frac{Y_p - Y_c}{DpiY} \right)^2} + 25.4
\]

where

- \((X_p, Y_p)\) is the position of the orbiting circle in pixels
- \((X_c, Y_c)\) is the position of the origin circle in pixels
- \(DpiX\) and \(DpiY\) are the dot per inch values for the specific display device
• Two hundred and fifty values were recorded to try and reduce process noise.

• The software trigger was broadcast on every bus with an active camera.

Figure 6.10: Absolute error for triggered and non-triggered tracking

Figure 6.10 illustrates the results from the synchronisation experiment. There is no discernible difference between triggered and non-triggered results, which is unexpected. The results in Figure 6.10 are difficult to explain without further investigation but we can hypothesise about some possible reasons.

The software trigger mechanism is designed to be simultaneously sent to all cameras on the FireWire bus by means of a broadcast packet. Due to technical implementation details, the tracking system uses an individual FireWire bus per camera. In order to overcome this limitation the software sequentially sends the broadcast packet on each bus.

```c
foreach (Camera cam in cameras)
{
    cam->Trigger();
}
```

Whilst the intention of the above code was to send the trigger to all buses at once there are several factors that could have delayed transmission. The process could have been preempted halfway through execution or the idiosyncrasies of FireWire could mean that the packet gets buffered until an appropriate time in the transmission cycle. Given these factors that cast doubt on the validity of this experiment it would be interesting to compare these results with results obtained using the hardware trigger. Unfortunately time constraints mean that we are unable to make this comparison.

The criticism of this experiment is largely based on the premise that we would expect to see a difference between the triggered and non-triggered results. This premise assumes that the non-triggered results are not synchronised
which may in itself be flawed. The software configures the cameras to use a consistent frame rate and then simultaneously starts their capture cycles. If the independent cameras can maintain a constant capture rate we can argue that they would remain in synchrony.

Both case argued here provide possible explanations for the results seen in Figure 6.10 but are simply speculation at this stage. It would require a more in depth analysis to determine which, if any is the correct explanation.

### 6.4.3 Juggling Balls

The proposed application for the tracking system presented in this report is to track the movement of juggling balls. The juggling example will incorporate the sources of error described in Section 6.4 as the balls are in constant motion. Another potential source of tracking error is described in Section 3.3. The direction of light incident upon a ball will change as that ball moves through the air. The changing light causes the shape of the “blob” segmented by the feature extraction algorithm to change, which in turn affects the level of uncertainty in the calculated center of mass of that “blob”.

Figure 6.11 demonstrates the tracker determining and highlighting the location of the three juggling balls. The colour segmentation used by the tracker works best when juggling a three ball cascade. A standard juggling cascade has balls moving in a figure of eight pattern across the juggler’s chest. The timing of the throws is such that the airborne balls rarely occlude each other. In this situation the tracker works well and is able to reliably detect the position of each ball, even if a ball is partially occluded by clutched fingers as it is caught. However the difficulties in determining the exact center of the balls (as described in Section 3.3) and the errors introduced by motion (see Section 6.4) mean that the triangulated coordinates fluctuate.

To try and determine the level fluctuations present, the output from the tracker was used to render a juggling visualisation. Based on the movement of a juggling ball through the air we would expect the calculated depth coordinates to be planar. However it was visible from the visualisation that the depth component was fluctuating. Any error in the X and Y coordinates was less obvious and cannot be identified using this subjective measure.

---

3A sequential processor cannot simultaneously execute code but we can consider it to be simultaneous for the purpose of this discussion.
Figure 6.11: Tracking three juggling balls
Chapter 7

Conclusions

The primary objective of this project was to produce a system capable of three dimensional motion tracking using inexpensive cameras and to investigate its feasibility as a method of human computer interaction.

Chapter 6 describes the experiments performed to try and measure the accuracy of the tracking system. In order for the tracker to be applicable as a form of user input it must be reactive. The latency experiment described in Section 6.2 shows us that this is an issue for real time applications. The latency value of 0.1 seconds for two cameras at 640x480 is passable but using three cameras at 640x480 produces a time lag of 0.8 seconds which may be unacceptable in some applications.

The second area under investigation was the spatial accuracy of the tracking system. Section 6.3 describes the steps taken to try and establish the static accuracy and Section 6.4 describes how accurately the system can track a moving object. The static accuracy results are very promising, demonstrating that the tracker can resolve spatial coordinates of an item to within a few millimeters of the item’s true location (see Section 6.3).

The laser scanning application described in Section 6.1 demonstrates a novel use for an optical tracking system. Tracking a point of laser light as it was scanned over a toy monkey produced excellent results. The point cloud in Figure 6.2 illustrates that the monkey is clearly depicted by the locations of the tracked points. The diagram shows us that the tracker can accurately identify the spatial relationships between respective points but tells us nothing about the distances between them. For this we can look to Section 6.3. The static checkerboard experiments demonstrate that the distances between multiple tracked points are homologous to the real world distances. The average error for the ten calibrations shown in Table 6.1 is 7.56mm over a distance of 577mm with the camera rig 3m away.

Motion accuracy proved to be much more difficult to measure. One of the factors identified as affecting motion accuracy is shutter speed. Section 6.4.1 gives a detailed description about how shutter speed affects the uncertainty in the position of the extracted feature. The Marlin cameras used here have a controllable shutter which can minimise the effect of motion blur, however cheaper cameras will not have this functionality. This could be an issue if the tracking idea were ever extended for use with cheaper hardware.

One of the stated project objectives was to investigate which parameters af-
fect the accuracy and stability of the tracking system. An unexpected outcome of the experiments described in Chapter 6 was to discover that calibration variance can have such a large effect on tracking accuracy. The calibration variance problem restricts the range of parameters that can be investigated because the results will only be meaningful if the experiments are calibration invariant. For example, it would have been interesting to measure the effect that camera separation has on tracking accuracy. However, moving the cameras would require recalibration and invalidates any results that are obtained.

The usefulness of tracking as a form of user input depends on the application. It is easy to imagine a scenario, such as tracking the motion of a scalpel during remote surgery where an error of 7.56mm is unacceptable. Having said that the system has proved to be more accurate over short distances. For less critical applications the level of accuracy may be adequate. Section 6.4.3 describes using the software developed here to track juggling balls. Although the triangulated coordinates are known to fluctuate in the juggling example it does demonstrate that tracking has the potential to control a computer simulation.

7.1 Further Work

The investigation presented here has gone some of the way to achieving the original aims of the project as outlined in Chapter 1. However the analysis that was performed has raised several other questions.

One piece of further work may be to determine if a tracking system built with standard web-cams can achieve a similar level of accuracy. The tracking system presented here uses relatively inexpensive cameras when compared to commercial tracking solutions. However if optical tracking is to be used in a domestic environment it will require cheaper hardware such as standard web-cams.

An investigation into whether a Field-Programmable Gate Array (FPGA) or other custom hardware can offset the computational burden of optical tracking would be interesting. The tracking implementation presented in this report is computationally expensive. The feature extraction algorithm is a bottleneck in the system that restricts the operational frame rate. It is conceivable that this step in the process can be offloaded to a custom processor.

Performance and computational cost are real concerns to the applicability of tracking as a user interface device. Modern computer games already tax the available processing resources in order to produce realistic virtual scenes. The difficulty that would need to be addressed by this work is implementing a feature extraction algorithm in hardware that is generic enough to be applied to different tracking applications but still robust enough to reliably detect all of the required features.

The tracker presented here has largely concentrated on accuracy rather than execution speed. Possible future work might be to optimise the algorithms used, in order to increase the frame rate. It would be possible to use the predictions for the future location of the tracked items, to restrict the search space of the feature extraction algorithm. It would be interesting to investigate what amount of speed increase can be obtained from doing this.

Lastly, an area that this report has neglected is whether or not tracking
is conducive to human computer interaction. The work presented here has demonstrated that the technical aspects of tracking for computer interaction can be achieved, but is it a productive method of interaction? It would be interesting to see a usability study done to determine if tracking can replace more conventional input devices.
Appendix A

Example Tracker XML file

```xml
<?xml version="1.0"?>
<opencv_storage>
<cameras>2</cameras>
<orientation>2</orientation>
<Camera>
<resolution>2</resolution>
<colour/>4</colour>
<framerate>4</framerate>
<coverage>0.90000000000000000</coverage>
<intrinsic_matrix type_id="opencv_matrix">
<rows>3</rows>
<cols>3</cols>
<data>
1.946841613529077e+000 0.31950000000000000 0.
1.94835678904372e+000 50000000000000000 0. 0. 1.0</data>
</intrinsic_matrix>
<distortion_coeffs type_id="opencv_matrix">
<rows>1</rows>
<cols>4</cols>
<data>
-0.433546329753779 0.6882170969494367 0.823024251629637e+005
2.52067239159164e+000</data>
</distortion_coeffs>
<translation_matrix type_id="opencv_matrix">
<rows>4</rows>
<cols>4</cols>
<data>
1.0 0.0 53.971312393378910 1.0 -179.2608506347562200 0. 0.
1.2.052561779126500e+003 0. 0. 0. 1.0</data>
</translation_matrix>
<rotation_matrix type_id="opencv_matrix">
<rows>4</rows>
<cols>4</cols>
<data>
</data>
```
0.9919307030348710.1172694651579357-0.04977100139866340.
-0.16669751707379220.990060272552960.01477733806389810.
0.0201653690655399-8.3679877930066666e-0030.99970067834964130.
0.0.0.1.</data></rotation_matrix>

<Camera>
<resolution>2</resolution>
<colour>4</colour>
<framerate>4</framerate>
<coverage>1</coverage>

<intinsic_matrix type_id="opencv_matrix"/>

</Camera>

<distortion_coeffs type_id="opencv_matrix"/>

</distortion_coeffs>

<translation_matrix type_id="opencv_matrix"/>

</translation_matrix>

<rotation_matrix type_id="opencv_matrix"/>

</rotation_matrix>

66
0.0342905037105983 0.034275182504790 0.9998138857974243 0.0 0.0 0.

0.1.\textless/data\textgreater</rotation_matrix>
\textless/Fundamentals\textgreater
\textless/fundamental_count\textgreater1</fundamental_count>
\textgreater/

\textless/index\textgreater0</index>
\textless/fundamental_matrix type_id="opencv_matrix"\textgreater
  \textless/rows\textgreater3</rows>
  \textless/cols\textgreater3</cols>
  \textless/dt\textgreater3</dt>
  \textless/data\textgreater
-7.485644295133761e-008 2.9613688877309447e-006
-0.0365101196221111-8.3333333333105369e-007
1.1312566792304985e-007-1.1885282220621994e-003
0.03597164772489690-7.7711664606778917e-004 1.\textless/data\textgreater</fundamental_matrix\textgreater</f0</Fundamentals</Camera>
\textless/opencv_storage\textgreater
Appendix B

Code Examples
// Convert the image to a HSV image
IplImage* hsv = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 3);
cvtColor(image, hsv, CV_BGR2HSV);

// Select the hue channel
IplImage* imgHue = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 1);
cvSetImageCOI(hsv, 1);
cvCopy(hsv, imgHue);

// Threshold the hue values
cvInRangeS(imgHue, minHue, maxHue, imgHue);

// Select the saturation channel
IplImage* imgSat = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 1);
cvSetImageCOI(hsv, 2);
cvCopy(hsv, imgSat);

// Threshold the saturation values
cvInRangeS(imgSat, minSaturation, maxSaturation, imgSat);

// Do a bit wise AND to get the parts with the right hue AND saturation
cvAnd(imgHue, imgSat, imgHue);

// Find the contours
CvSeq* contours;

// Create a place to keep the contours
CvMemStorage* storage = cvCreateMemStorage(0);

cvFindContours(
    imgHue, 
    storage, 
    &contours, 
    sizeof(CvContour), 
    CV_RETR_LIST, 
    CV_CHAIN_APPROX_SIMPLE, 
    cvPoint(0,0)
);

Figure B.1: Example feature extraction code
kalman = cvCreateKalman(4, 2);

// Initialise the kalman matrices
const float A[4][4] =
{
    {1, 0, 1, 0},
    {0, 1, 0, 1},
    {0, 0, 1, 0},
    {0, 0, 0, 1}
};

memcpy(kalman->transition_matrix->data.fl, A, sizeof(A));
cvSetIdentity(kalman->measurement_matrix);

// Introduce uniform process noise of 0.5
cvSetIdentity(kalman->process_noise_cov, cvRealScalar(0.5));
cvSetIdentity(kalman->measurement_noise_cov);
cvSetIdentity(kalman->error_cov_post);

Figure B.2: Code to initialise an OpenCV Kalman Filter
Bibliography


[21] PhaseSpace. Distributed-processing motion tracking system for tracking individually modulated light points.


72