University of Dublin

TRINITY COLLEGE

PS3 Type Motion Controller

Emma Louise Lynch
B.A.I. Engineering
Final Year Project April 2011
Supervisor: Kenneth Dawson Howe

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland
DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

__________________________________________  ______________________
Name                                          Date
I would like to thank my supervisor Kenneth Dawson Howe for his advice and guidance during this project.

I would also like to thank my family and friends for their continued support and advice.
Abstract

Motion controllers and alternative human-computer interface devices are becoming more widely used as viable alternatives to traditional interface devices. The use of motion controllers has become particularly widespread in the gaming industry in recent years, with each main console manufacturer having released some commercial motion controller device.

The technology and hardware used in these devices is expensive and specialized. Motion controllers offer a more intuitive and immersive way of interacting with computers. The aim of this project is to emulate the “Playstation Move” motion controller using only computer vision techniques and a webcam.
# Table of Contents

1. Introduction ................................................................................................................ 7  
   1.1 Aims ....................................................................................................................... 7  
   1.2 Motivation ............................................................................................................. 8  
   1.3 Problem Overview ............................................................................................... 8  

2. Background .............................................................................................................. 9  
   2.1 Human Computer Interaction (HCI) ................................................................. 9  
   2.2 Normans Model of HCI ....................................................................................... 10  
   2.3 Input Devices ....................................................................................................... 12  
   2.4 Matching Tasks and Input Devices ...................................................................... 13  
   2.5 Direct Manipulation ............................................................................................ 16  
   2.6 Controllers in Interactive Entertainment ............................................................ 17  
   2.7 The History of Motion Controllers ....................................................................... 19  
   2.8 Modern Motion Controllers in Interactive Entertainment .................................. 19  
   2.9 PlayStation Move ............................................................................................... 20  
   2.10 Adaptations of Motion Controllers ................................................................... 23  
   2.11 Object Tracking ................................................................................................ 24  
      2.11.1 3D Object Tracking with Hardware ......................................................... 24  
      2.11.2 3D Object Tracking with Software ......................................................... 25  
      2.11.2.1 Detection ................................................................................................. 25  
      2.11.2.1.1 Colour Based Object Segmentation .................................................. 26  
      2.11.2.2 Tracking ................................................................................................. 26  
      2.11.2.3 Analysis .................................................................................................. 27  
      2.12 OpenCV ........................................................................................................... 27  

3. Controller Design .................................................................................................... 29  

4. Sample Data & Histogram ...................................................................................... 33  
   4.1 RGB vs HSV ...................................................................................................... 33  
   4.2 Statistical Alteration of Data ............................................................................ 37  

5. Object Detection and segmentation ....................................................................... 41  
   5.1 Colour Classification ......................................................................................... 42  
   5.2 Gaussian Smoothing .......................................................................................... 44  
   5.3 Mathematical Morphology ................................................................................ 45  
   5.4 Connected Components ..................................................................................... 47
6. Pose and Orientation Recovery ................................................................. 49
   6.1 XY Position ......................................................................................... 50
   6.2 Z Position ......................................................................................... 51
   6.3 Roll ................................................................................................. 52
   6.4 Pitch and Yaw ................................................................................. 54
7. Evaluation ............................................................................................... 56
   7.1 Results ............................................................................................. 56
      7.1.1 Screenshots ............................................................................... 56
      7.1.2 Look-Up Table Values ............................................................. 59
      7.1.3 Real Time Calculations ............................................................ 60
   7.2 Successes .......................................................................................... 60
   7.3 Difficulties ....................................................................................... 62
      7.3.1 Lighting Issues ......................................................................... 62
      7.3.2 Problems with Orientation and Pose Recovery ...................... 63
      7.3.3 Noise ....................................................................................... 64
      7.3.4 Real Time Calculations ........................................................... 64
      7.3.5 Other Difficulties .................................................................... 65
   7.4 Future Work and Improvements ....................................................... 66
      7.4.1 Improving Pose and Orientation Recovery .............................. 66
      7.4.2 Improving Performance Under Varying Lighting Conditions .... 66
      7.4.3 API and Error and Status Reporting ....................................... 67
      7.4.4 Integration with a Game ........................................................... 67
      7.4.5 Accuracy Comparison and Testing ........................................... 67
8. Conclusion .............................................................................................. 69
References ................................................................................................. 70
Appendices ............................................................................................... 73
   Appendix A: Electronic Resources ....................................................... 73
1. Introduction

This project intends to prove that computer vision can offer a simple alternative to complex hardware based solutions to problems. It will show that it is possible to emulate the functionality of modern motion controllers with simple items and with the use of computer vision principles and methods.

The tracking software was written in OpenCV and C++. The tracking software should run in real-time and run on a standard computer. The software was written and tested on a 2.26 GHz Intel Core 2 Duo MacBook Pro with 2GB of RAM. The webcam used was the MacBook Pros internal “iSight” camera, which runs at a maximum of 30fps. The tracking software is demonstrated on a video which runs at 15fps.

This report aims to outline the motivation and background to the project. It will also detail how each component of the project works, and the approach that was taken to the problem. Finally, it will discuss the results that were obtained, the difficulties that were encountered and future improvements that could be made to the project.

1.1 Aims

The aim of this project is to emulate the “PS3 Move” motion controller and interface. To achieve this, a controller was designed, and a standard webcam was used to track it. The tracking software was able to detect, segment, track and analyze the controller in real-time and provide six degrees of freedom.

The controller should aim to be responsive and intuitive as an interface device. The tracking software should provide information about the pose and orientation of the controller. The data provided should contain values for the position of the controller in three dimensions, as well as the roll pitch and yaw, giving a full six degrees of freedom.

The tracking software should perform all the processing in real time so that it is a feasible solution as a motion controller.
1.2 Motivation

Motion controllers have become popular and commercially successful over the past number of years. Motion controllers have experienced particular widespread use in the area of interactive entertainment. However, there is growing demand for these controllers to be used as general input devices (section 2.10). Additionally, there has been a demand for a new way of interacting with technology. Computers have become more widespread and more powerful, so now can support new types of interface device. Most interface devices that are commonly used are somewhat archaic.

1.3 Problem Overview

This section will provide the reader with a general outline of the problems to this project and the work that was done for it. Detailed descriptions of how this work was carried out will be provided in sections 3-6.

The project can be broken down into two main constituents: the controller, it’s design and the tracking software. The software can again be further broken down into: object segmentation, tracking, and analysis and pose recovery.

The controller had to be designed in a way that allowed it to be easily locatable (i.e. distinct from the background in some manner) and that allowed the pose and orientation information to be recovered. The software had to perform the tasks of object detection and segmentation to locate the object, tracking the object from frame to frame and the determination of the position and orientation of the object.
2. Background

This section of the report will discuss human-computer interaction, input devices, direct manipulation and matching input devices to tasks, motion controllers, and a background of the methods used in the implementation of the project.

Traditionally, alternatives to generic input devices (such as keyboards, or gamepads) were considered a novelty, or a solution for people who had difficulties using such devices. In recent years however, there has been a movement away from traditional input devices in favor of more intuitive, user-friendly ones. This is particularly true for video-game consoles and controllers. The traditional input devices for video games are gamepads, joysticks, mice and keyboards. There has been more research and emphasis on non-traditional input devices and controllers, particularly since the introduction of the Nintendo Wii in late 2006. Although the Wii Remote has not been widely used as an input device for computers, there has been work in this area and the Wii Remote is one of the most common input devices in the world (Chung, 2009). Before this there were some motion controllers available for consoles, but the Wii proved that motion controllers were a viable alternative to their traditional counterpart. The move away from traditional input devices has led to the increase in popularity of motion controllers. Motion controllers are generally considered to be more intuitive and immersive, yet still offer accuracy, reliability and functionality.

2.1 Human Computer Interaction (HCI)

Human-computer interaction is the field of study concerned with the design, use and development of how humans interact with and use computers. The area has experienced huge growth and interest in the past 20 years. This is due to the increase in computing power and to the more widespread use of computers as an entertainment medium. HCI investigates the range of possibilities in hardware and software in order to provide the most appropriate interface between humans and computers. The interaction between humans (users) and computers occurs at the user interface, which can be hardware and/or software. Currently, traditional input devices, such as keyboards, limit human computer interaction. There is a growing demand for input devices
to be more intuitive and suited to their task. It is important for the control structure of the device to match the perceptual structure of the task. This will be discussed further in section 2.4.

The way in which users interact with computers has been evolving rapidly in the past 20 years. Decreasing hardware costs have led to larger memory and faster systems, meaning that computers are capable of more complex calculations and tasks than they were 20 years ago. This means that more computationally expensive interfaces can be utilized more widely than they may have been in the past. Portability is also large factor in the evolution of human-computer interaction. The reduction in size and price of hardware components coupled with the reduction in power requirements has led to increased portability of computers. This has impacted how humans interact with computers. A good example of this is the use of trackpads on laptops as an alternative to the computer mouse. The increased accessibility of computers however is arguably the largest factor in the shift in human-computer interaction.

2.2 Norman’s Model of HCI

Norman’s model of interaction is one of the most influential models in human-computer interaction. Proposed by Norman(1988) this model can be considered a logical simplification of an execution-evaluation cycle. It describes the interaction between humans and computers in a way that is close to our own intuitive understanding of how humans and computers interact.

The model is divided into two main stages: evaluation and execution. Each of these stages can be subdivided into further stages. The stages of the model are as follows:

- Evaluation
  - perceiving the system state
  - interpreting the system state
  - evaluating the system state with respect to goals and intentions
• Execution
  • establishing the goal
  • forming the intention
  • specifying the sequence of action
  • executing the action

This model is used by Norman (1988) to explain how poor interface design can be troublesome for its users. Problems can be explained in the terms of the gulf of execution and the gulf of evaluation. The gulf of execution describes the gap between the users’ formulation of the actions required to reach some goal and the actions that are supported or understood by the system. The gulf of evaluation describes how the computer or application represents the system state and how easily accessible and useful this information is to the user. "The gulf is small when the system provides information about its state in a form that is easy to get, is easy to interpret, and matches the way the person thinks of the system" (Norman 1988: p. 51). A graphical representation of Norman’s model of interaction can be seen above in fig. 2.1. The gulfs of evaluation and of execution refer to the mismatch between our internal goals on the one side, and, on the other side, the
expectations and the availability of information specifying the state of the world (or an artifact) and how me may change it.

Norman’s model can be used when choosing input devices and interfaces for applications. By choosing an appropriate interface device, the gulf of execution can be reduced. By reducing the gulf of execution, the ease of use of the interface device and indeed the application is increased.

Motion controllers aim to translate the physical motion of the user into data that can be used by some application. By using the direct data from the motion of the user, the gulf of execution can be reduced. According to Norman’s model of interaction, this should improve the users experience and interaction with the application. This idea has been used in interactive entertainment in particular, as reducing the gulf of execution in this case can lead to a greater overall level of immersion and enjoyment. This will be further discussed in section 2.5.

2.3 Input Devices

This section will define what an input device is, and discuss the merits of many traditional input devices as well as the advantages and disadvantages of some newer alternative input devices, namely motion controllers. J. Preece (1994: 212) defines an input device as “a device that, together with appropriate software, transforms information from the user into the data that a computer application can process.”

For the purpose of this project, the input device should encode motion into data that can be read by a computer. The computer will interpret this data into an action that controls or interacts with the computer in some way. This data should accurately represent what the user intended to do. Traditionally, the design of input devices was limited by what was technologically feasible on a large scale, as opposed to what devices might be more intuitive or suited to specific tasks.

Input devices can be defined by several different characteristics. One of the most important characteristics of input devices is the degrees of freedom (DOF) that it can represent. A degree of freedom can be described as “a particular, independent way that a
body moves in space” (Bowman, 2004: 88). The Playstation Move controller represents six DOF consisting of three position values (x, y and z) and three orientation values (roll, pitch and yaw). The devices DOF can be seen as a representation of how complex the device is and its adaptability to different interaction techniques (Bowman, 2004). Input devices can also be classified by the type of data they provide and by the action that is required by the user to generate this data.

Data can be provided from the input device in either a discrete or continuous manner, or a combination of both. Discrete information is usually generated by some boolean which is activated by the users actions. Continuous information consists of multiple data values that are generated in response to the users action. In some cases (e.g. motion controllers) this information is generated regardless of what the user is doing.

Devices can be either purely passive or purely active. This describes the action required by the user to generate data. Purely passive devices generate data without any physical input from the user. For example, a tracking system will get the coordinates of the object it is tracking, whether this object is stationary or not. Purely active devices require the user to perform some physical action before any data can be generated, e.g. pushing a button to get a character on screen to perform an action.

The properties of different input devices is a very important consideration in choosing the optimal tool for the task. The importance of choosing an appropriate input device will be discussed in more detail in the next section.

2.4 Matching Tasks and Input Devices

J. Preece (1994: 212) states that an input device should be “appropriate for the tasks that are to be performed”. In the past, input devices were not chosen in light of their suitability to the task that they needed to perform, but rather to match the computational limits of the technology at the time. This led to mice and keyboards becoming the norm for personal computers, and controllers or to a lesser extent, joysticks becoming the input device of choice for video games and interactive entertainment consoles.
Tasks, as well as input devices can have multiple dimensions. Take, for example, the task of focusing on a particular point on a screen. To focus on a particular point, the user must adjust the values for the x, y and z axes. The task has two main components: zooming (altering values on the z-axis) and panning (altering values on the xy-plane). It is clear that this task is three-dimensional. The most common solution to this is that the user will use a mouse to perform this task, using the mouse to pan, and a separate control to zoom. A mouse is a common two-dimensional device and is often adapted to complete three-dimensional tasks. The user typically does not think of zooming and panning as separate actions, but thinks of them as one action that focuses on a specific point. Creating two actions to complete a task which the user thinks of as one action is counter-intuitive and shows the disadvantages of using a two-dimensional input device for a three-dimensional task.

Different devices that may both operate multidimensionally are often considered equivalent, even though the user experience can vary between the two. This in counter-intuitive and can negatively affect performance and user experience. No device is definitively better than another, but certain devices do have advantages over others with regards to certain tasks.

Buxton (1986) argues that almost any property of any input device can be advantageous in one situation but disadvantageous in another. Buxton (1986) uses three scenarios to explain his argument.

The input devices considered for these scenarios are all pointing devices. The first scenario involves panning over a large graphical surface. The second scenario extends the task to panning and zooming. The third scenario proposes that the objects must be located by panning, and then manipulated by twisting without letting the position in the xy-plane change. Consider the use of a trackball and a joystick for each scenario.

In the first scenario, Buxton (1986) suggests that the trackball is best suited to this task because the movement of the ball is mapped directly to motion across the xy-plane. When simultaneous zooming and panning is required, as in the second scenario, Buxton(1986) claims that the joystick is better suited to the task. This is because it is easier to zoom and pan by twisting and moving the joystick respectively than it is to do the same with the trackball. In the final scenario, the trackball would prevail as the more suitable input
device. Buxton(1986) suggests that this would be easier with the trackball. Although this scenario is quite similar to the last, where the joystick had the advantage, the added restriction that the xy-position much not drift means that the trackball would be better suited. Twisting a spring-loaded joystick whilst holding its position steady would be more difficult than performing the same action with a trackball.

Using a motion controller, such as the Playstation Move or the Wii-mote to navigate a three-dimensional world is arguably better than using an analogue stick or joystick. The standard Sony Playstation 3 controller can be seen in fig. 2.2 above. In order to obtain six DOF, two analogue sticks are required. In line with Buxtons(1986) conclusions in the example above, using a motion controller which maps naturally to six DOF should be better suited to the task of traveling through a three-dimensional world and should provide a better user experience because of this.
2.5 Direct Manipulation

According to Nelson (1980) direct manipulation is a user interface technique where objects and actions are represented by a model of reality. An example of this in the real world would be driving a car. The user turns the steering wheel, and receives visual feedback in the change of scenery. This is different from a command-oriented approach, which would result in the driver issuing a command, such as “go left”, and then issuing a command to show the orientation of the car. It is obvious that direct manipulation is a much more intuitive way of controlling or interacting with something. Video games and interactive entertainment make extensive use of direct manipulation.

Rutowski (1982) noted that an important part of direct manipulation is the principle of transparency, where attention shifts from issuing commands to observing results conveyed from feedback. At its most basic understanding, this means that the user isn’t aware of the tool itself. The user is solving the task as opposed to using the tool to solve the task. It is also apparent that with direct manipulation, it is not necessary for the user to have a great knowledge of computer-related semantic knowledge and syntactic knowledge. Task-related semantics should be more essential than computer-related semantics. To achieve maximum effect of direct manipulation, computer-related semantics would need to be replaced by task-related semantics.

This creates an obvious argument in favor of motion controllers over traditional controllers and joysticks in video games. Sony claims that the Playstation Move has full motion capture, meaning that it is capable of mapping the users actions directly to whatever the user is controlling. Using a traditional video game controller, like that in fig. 2.2, computer-related semantic and syntactic knowledge is necessary. In order for the user to complete certain movements, a combination of button presses and manipulation of the analogue sticks may be required. However, using a controller than can map the users movements rather than needing the user to remember different combinations of button presses for different tasks minimizes the need for computer-related semantic knowledge and instead, focuses on task-related semantics.
The use of direct manipulation in conjunction with motion controllers should be a good combination when considering Normans model of interaction and the concept of the gulfs of execution and evaluation. Recall that reducing the gulf of execution and the gulf of evaluation should lead to a better and more intuitive user experience. The gulf of evaluation is minimized by the use of direct manipulation. Using direct manipulation as a user interface provides very clear visual feedback about the model to the user. By reducing the complexity of the controller and matching the controller to the task, the gulf of execution is also reduced. This would, in theory, make interfaces that exploit direct manipulation much more intuitive and immersive than alternatives. It can be argued that motion controllers add to the overall experience of direct manipulation and in doing so add to the overall immersiveness of the application and in turn, make the experience more enjoyable for the user.

2.6 Controllers in Interactive Entertainment

Motion controllers have applications in various aspects of human-computer interaction but they are most widely used in interactive entertainment at present. They have been utilized extensively in both video games and virtual reality, with the former having commercial success.

Viola (2010) argues that since traditional game controllers often have abstract control schemes, with unnatural “mappings between control mechanisms that are easy to perform naturally and spatially (running, jumping, punching, kicking, and so on) and a series of button presses,” casual gamers have become more alienated. This is due to the fact that game controllers are often quite unintuitive. 3D spatial interaction should be able to reclaim the casual gamer and improve overall game play for all gamers (Viola, 2010).

The growth in complexity of modern computer games has not been reflected in the design of interface devices. In the past 30 years, video games have improved in terms of graphics, sound, artificial intelligence and storytelling (Viola, 2010). This has inevitably increased the complexity of the gameplay itself. However, input devices have remained relatively constant until recent years. Although game controllers have certainly changed in the past 30 years, they have not necessarily become better or more suited to their purpose.
Often, with the added complexity of gameplay, the complexity of the game controller would increase. An example of this is the Sony Playstation game controller. The original game controller can be seen in fig. 2.3 above. When compared to fig. 2.2, the added complexity of the controller is apparent. Two analogue sticks have been added, as well as a “PS” button which can be seen in the centre of the controller. Two additional shoulder buttons have been added, which cannot be seen in the images. These extra buttons and controls have been added to reflect the increased complexity in gameplay. However, the newer controller (fig. 2.2), although more complex than its ancestor, effectively provides the same type of data to the application and does not change the interaction between the user and the computer. It does not decrease Normans gulf of execution as one may have hoped.

Viola(2010) suggests that although this added complexity allows for greater expression, it had made game controllers less intuitive, which has alienated casual gamers.
2.7 The History of Motion Controllers

Motion controllers have been used in video games consoles since the late 1980s and into the early 1990s, with devices such as the Nintendo UForce, Mattel PowerGlove, and Sega 3D glasses. Due to a number of factors (including poor technology and general lack of adoption) these devices were largely unsuccessful. However, alternatives to traditional motion controllers re-emerged in the early 2000s and have become widely successful since.

Sony released the Playstation EyeToy in 2003. It was a webcam that was used to allow players to use their bodies to interact with games. It operated at 60Hz to ensure that there was no perceivable delay between the user moving and the result being seen on-screen. This was twice the framerate as the majority of webcams at the time. It also had a 56 degree diagonal field of view so that it could capture the entire torso and outstretched arms of the player from a reasonable distance. The EyeToy was a simple adaptation of a common technology to provide a new interaction method for Playstation games. The software aspect of the interface device relied mainly on motion detection to distinguish the user from the background.

The Nintendo Wii was the next console to implement motion controllers. However, unlike previous consoles, the Wii would use motion controllers as it’s primary control system. The Wii uses a mixture of infrared technology in conjunction with an accelerometer (and gyroscopes in the Wii MotionPlus extension) to represent a full six DOF.

2.8 Modern Motion Controllers in Interactive Entertainment

The most popular modern device to implement motion controls is the Nintendo Wii. It was first released on the 19th November, 2006. Nintendo claims that the Wii targets a larger demographic than rival consoles, Microsoft’s Xbox 360 and Sony’s Playstation 3. This is supported by the fact that the Nintendo Wii is the best-selling current generation console. Because of this commercial success, the Wii Remote is one of the most commonly used input devices in the world. This has led to the popularity of motion controllers and reduced the somewhat novel aspect of motion controllers in interactive entertainment.
The success of the Wii has prompted a response of both its rival console manufacturers, with both companies releasing alternatives to the Wii for their respective consoles. Microsoft’s response to the Nintendo Wii is the “Kinect” which was released for Xbox 360 in November 2010. It does not make use of a controller, instead tracking the users actions with a camera, and obtaining control data from those motions.

Sony’s answer to the Wii is the Playstation “Move”. It is a motion controller that is used in conjunction with a webcam, the Playstation “Eye”. The PS Move was released in September 2010. It implements motion control using an array of hardware devices as well as some computer vision methods. The webcam tracks a coloured glowing sphere. This input data is supplemented by data from magnemeters, gyroscopes and accelerometers. The design and operation of the controller will be further discussed in section 2.9.

The popularity of these devices is clear from their sales figures. The Wii console has sold over 84 million units\(^1\) since its release in 2006. Microsoft’s “Kinect” has become the fastest selling consumer electronic device of all history and has sold 10 million units as of March 2011\(^2\). The Playstation Move has sold 4.1 million units as of December 2010\(^3\).

### 2.9 Playstation Move Controller

This section aims to outline how the Playstation Move controller works and it’s advantages and disadvantages over it’s rival controllers.

The Playstation Move is intended to be an alternative to the Playstation dual-shock controller (fig. 2.2) rather than a replacement (Viola, 2010). The Move aims to address many of the issues with Sony’s older motion controller peripheral (the EyeToy) by using webcam tracking in combination with motion sensing and buttons.

---

\(^1\) Wii sales source: [http://www.nintendo.co.jp/ir/library/historical_data/pdf/consolidated_sales_e1012.pdf](http://www.nintendo.co.jp/ir/library/historical_data/pdf/consolidated_sales_e1012.pdf)


\(^3\) PS MOVE sales source: [http://news.cnet.com/8301-13506_3-20024165-17.html](http://news.cnet.com/8301-13506_3-20024165-17.html)
The Playstation Move system consists of the Playstation Eye and at least one Move controller. However, it can also use up to four controllers with or without a sub-controller. The wireless controller is hand-held and has various buttons to provide greater versatility.

The system works through a combination of software and hardware object tracking to fully represent six DOF and to provide full motion capture. The hardware includes a three-axis accelerometer, a three-axis gyroscope and a terrestrial magnetic field sensor. This array of hardware components is used to find the roll, pitch and yaw of the controller. It also helps to represent the x, y and z position of the controller when the sphere is obscured or out of frame.

Perhaps the most distinctive feature is the coloured sphere at the end of the controller. The colour of this sphere is controlled by three LEDs which can change colour independently.
The LEDs provide a full 24-bit colour range. The colour of the sphere is determined by software and is based on a number of criteria. Depending on the application, the colour of the sphere can be changed to give visual feedback. This follows Normans model of interaction (Section 2.2) and reduces the gulf of evaluation. An example of this feature is when the colour of the sphere changes momentarily to represent the muzzle flash of a gun in a first-person shooter game. However, the primary purpose of the sphere is to help the Playstation Eye detect the absolute position of the controller in three-dimensional space. The colour is chosen by the software to be purposefully distinct from the background to facilitate the colour segmentation and tracking. The controller must be calibrated each time it is used. This calibration involves initialization of the sensors, but also allows the software to analyze the colours in the background. Once the background colours have been determined, the sphere is given an optimal colour for tracking, which ideally, would be unique in the scene.

The Move has a number of advantages over rival motion controllers and indeed over traditional game controllers. The Move offers full 1:1 motion capture, meaning that it directly maps the users actions. It exploits haptic technology, with a vibration feature, which provides another form of feedback. It also provides visual feedback as discussed above. The Move operates in real time. This is due to the webcam running at 60Hz and the fact that the spherical shape allows the position to be calculated with minimal latency.

Additionally, the Move only requires and extra 1-2MB of system memory, according to co-designer Anton Mikhailov\(^4\) and the raw data can be calculated and translated to the game in “under a frame”.

Reviews of the controller have been mostly positive. The Guardian claims that the Move is “streets ahead of the Wii in terms of control and accuracy”\(^5\). It was awarded the “Most immersive game controller” by Popular Science in 2010\(^6\).


2.10 Adaptations of Motion Controllers

Recently, there has been a lot of independent work in adapting commercial gaming peripherals to work with PCs. This has been a growing area of interest since the release of the Wii, Kinect and Move.

The Wii Remote is the oldest of the current generation motion controllers, so there has been arguably more progress achieved in adapting it for PC. Johnny Chung has done a lot of research in this area and has adapted the Wii Remote for several different applications. Chung has created a finger tracking system using the Wii infrared camera. Using LEDs and reflective tape, Chung has made an application which can track objects in 2D space. This application is similar to the computer interface seen in the movie “Minority Report". Chung has also adapted the Wii Remote for use with interactive whiteboards and for head tracking as part of a desktop virtual reality display.

fig. 2.5 - The Move sphere lit up as different colours

23
Shortly after the release of the Microsoft Kinect, the controller had been modified to work on a PC (KinectMan2, 2010). A video was released showing data from the Kinect being displayed in a simple GUI on a Windows PC. Since then, numerous “Kinect hacks” have been released and the controller has been used for everything from a gaming motion controller to controlling robots. In response to this, Microsoft have decided to release a non-commercial Kinect SDK which can be used by enthusiasts and academics\(^7\). On March 20th 2011, a video was released showcasing the use of the Kinect on the Playstation 3 (shantzg001, 2011).

Enthusiasts have also investigated adapting the Playstation Move to work on PC. A Google Code project was created shortly after the Moves release in 2010. Although progress has been made, the controller cannot yet work fully on a PC as an input device. However, Sony have now said that a developers kit, called Move.me, will be released in Spring 2011\(^8\). It is unsure if this will allow the Move to be used as an input device solely for the PC, but it will allow amateur developers to use the controller for different applications.

### 2.11 Object Tracking

Cantarella (2010) defines object tracking as “calculating the trajectory of an object in the image plane, as it changes around a scene.” 3D object tracking can be achieved by software or hardware, or a combination of both. There are arguments for using either hardware or software to track objects in three-dimensions, which will be discussed in this section.

#### 2.11.1 3D Object Tracking with Hardware

Until recently, real-time object tracking was done mostly in hardware, using devices such as accelerometers and gyroscopes. This was due to the computational expense involved in trying to extract data using image processing methods and the limitations of the hardware that was readily available, in particular, cameras. Achieving accurate pose and orientation data from hardware is often much easier than getting equivalent data from

---

\(^7\) Source: [http://www.siliconrepublic.com/innovation/item/20533-microsoft-research-releases](http://www.siliconrepublic.com/innovation/item/20533-microsoft-research-releases)

software. Hardware methods also tend to be much more robust than software methods. However, the sensors required to obtain this data can be expensive and delicate.

Obtaining the x, y and z pose information from both software and hardware methods is relatively easy, but when it comes to obtaining angular data, hardware is often the preferred method. Three-axis gyroscopes and accelerometers can be utilized in order to obtain this information. Using these devices, it is possible to get an object’s angular data as well as the linear acceleration.

Both gyroscopes and accelerometers contain information about the orientation of the sensors. Luinge (1999) suggests that combining the inclination information found from an accelerometer with the gyroscope information can provide a more accurate estimation of orientation than from either device alone.

The pitch can be estimated from the gravitational vector found by the accelerometer. Although this estimation is not very accurate, it does not suffer from “integration drift” (Luinge, 1999). By fusing the pitch information obtained from the accelerometer with the pitch information from the gyroscope, a more accurate estimation is found.

This shows the strength of the use of several sensors to obtain pose and orientation data. Although there is an overhead involved in combining the data and a cost involved in the extra hardware, the results are very accurate.

2.11.2 3D Object Tracking with Software

Cantarella (2010) discusses many algorithms and methods for object tracking in three-dimensions using software and claims that there are three main steps in video tracking. They are detection, tracking and analysis and will be discussed individually below.

2.11.2.1 Detection

There are a number of methods for detecting the objects of interest in a scene. The application may need to detect generic objects, motion, or specific objects. The methods used for detection in each of these cases differ. To detect generic
objects, one of the methods that can be used is the generalized Hough transform. Template matching is also an option for detection. Another method of detection is colour segmentation, which will be further discussed in section 2.11.2.1.1.

Another important aspect of object detection is choosing which features to track. This can be more of a design problem than a technical one. Choosing appropriate features can be critical to the success of the programme. Generally, features are chosen by their uniqueness. This may be a unique shape, colour, or method of motion. For the purpose of this project, the main feature that was used in object detection and segmentation was colour. A multi-coloured dodecahedron was ultimately chosen to be the main controller object. This design, along with its advantages and disadvantages, will be further discussed in section 3.

2.11.2.1 Colour Based Object Segmentation

Colour can be used to detect an object in a scene. Colour based object segmentation is used by the Playstation Move in order to extract the controller from the background scene (section 2.9). Colour can be used to make an object unique in comparison to its background. Extracting colour information from an image is relatively simple and fast. Using OpenCV (see section 2.13) images can be stored as an IplImage. This gives access to colour information for each pixel. Colour can be represented by several different schemes. The most common representation of colour is in RGB. Each of the three channels stores a value between 0 and 255 which represents either red, green or blue, giving full 24-bit colour when each of these channels is 8bits. There are a number of disadvantages involved with using this representation including illumination issues. Alternative colour spaces (e.g. normalised RGB, HSV) do solve some of these problems. The choice of colour space used in this project will be further discussed in section 4.1.

2.11.2.2 Tracking

The actual tracking of the object involves following the object from frame to frame. Again, there are a number of different methods for doing this, including using a Kalman filter, or other probabilistic methods. There are a number of difficulties associated with tracking. These include, but are not limited to loss of information
due to the two-dimensional representation of a three-dimensional world, noise, illumination changes, object occlusion and real-time processing problems. These issues are usually solved by imposing constraints on the scene. However, the methods for solving these problems often depend on the context and environment, so there is no definitive way to overcome these problems. The difficulties that were experienced in the course of this project, as well as the methods used to try and overcome these difficulties will be discussed in detail in section 5 and section 7.3.

2.11.2.3 Analysis

Analysis is concerned with obtaining useful data from the object once it has been found. This data is representative of the objects behavior and can be used for many applications. Depending on the aim of the application, the pose and orientation of the object may be desired. This information is obtained in the analysis stage. Again, there are many different methods for extracting information from a scene and the choice of which is dependent on a number of factors. The analysis aspect of this project is concerned with obtaining the pose and orientation of the controller to try and mimic the Playstation Move. The methods used in this project will be discussed in section 6.

2.12 OpenCV

OpenCV is an open source, cross-platform programming language used in computer vision. It was used to write the software for this project.

The OpenCV project was first launched in 1999 as an Intel Research initiative. Since 2008, OpenCV has gotten corporate support from Willow Garage. It was originally written in C, but recent releases (since version 2.0, 2009) some of the new methods were written in C++. OpenCV runs on Android, FreeBSD, iOS, Linux, Mac OS and Windows operating systems (OpenCV, 2011).
OpenCV contains a number of data structures and methods that are useful for computer vision. These include data types to represent images (e.g. IplImage) and some of the most commonly used computer vision algorithms which can be used for general image processing, machine learning, segmentation and tracking to name but a few.

OpenCV 2.1 was used in this project. This project uses various OpenCV data structures and methods. Some of the methods used include CvMoments, CvFindContours, CvQueryFrame. These will be further explained in sections 4 and 5.
The next three chapters will discuss the problems that needed to be solved and the approach that was taken to solve them. Section 3 discusses the design of the controller and the reasons behind this design. Section 4 will discuss how the software knew what to look for in the scene to identify the controller. Section 5 aims to explain exactly how the controller was found and extracted from the background scene. Finally, section 6 will detail how the pose and orientation information was recovered from the controller.

3. Controller Design

The first step in emulating the Playstation Move controller was designing an object to be used as the controller (i.e. the object to be tracked). There were a number of considerations involved in choosing an object. As Cantarella(2010) states, the object to be tracked must be distinct from the background somehow, and be unique in the scene, yet it should still allow recovery of pose and orientation information.

A number of designs were considered. Each will be explained below, outlining their advantages and disadvantages.

The Playstation Move controller is tracked by the glowing coloured sphere on the end of the controller. So, initially, a spherical design for the tracking object was considered. This had a number of advantages and disadvantages. Using a spherical object would allow calculations about the pose of the controller to be calculated very efficiently. However, using a plain coloured sphere would make it impossible to find information about orientation just using a webcam.

A design consisting of multiple spheres was then proposed. A representation of this design can be seen in fig. 3.1. By using two differently coloured spheres, it would be possible to get orientation information from the controller. The relative positions and sizes of the two spheres would provide information for pose as well as orientation. This design is particularly advantageous for finding the z position (or scale) of the controller. An absolute value for z could be obtained by calculating the ratio of the size of the two spheres. Finding an absolute value for z was troublesome with many other designs. Pitch and yaw can be calculated by analyzing the positions of both spheres in conjunction with the distance between them.
However, this design was prone to occlusion problems. Finding any values other than the x and y positions would be impossible if just one of the spheres was occluded. Self-occlusion is also a problem with this design. It would be possible for one of the spheres to occlude the other and therefore make the retrieval of pose and orientation information considerably more difficult, or impossible in some cases. Getting values for roll would also be troublesome except for under very precise circumstances.

The next design that was considered involved using patterns to determine pose and orientation information. A sphere with a pattern could provide enough data. By analyzing the changes in the pattern and the relative sizes of certain parts of the pattern, the pose and orientation information could be obtained. However, this design was ultimately dismissed as the analysis required to get the data would be too complicated and unreliable. This design would also be error prone and unreliable, as certain parts of the pattern might be lost due to illumination conditions or other external factors.

fig. 3.1 - Proposed controller design with two different coloured spheres. Orientation data can be found by analyzing the sizes and positions of both spheres.
The final design that was ultimately chosen was a multicoloured dodecahedron. This design was chosen because it facilitated more robust pose and orientation recovery than the alternatives using colour and shape analysis. There are two main components to this design. The first is the coloured faces and the other is the shape of the object. The object that was used as the controller can be seen in fig. 3.2 and a representation of it showing more of the coloured faces can be seen in fig. 3.3.
In total, 6 colours were used for the controller. This is the minimum number of colours possible to ensure that no adjoining faces are the same. Using a larger number of colours was not feasible, due to the camera’s interpretation of colour. This is demonstrated in fig. 4.3 on page 33. From the histogram, it is clear that using more colours would start to create problems with overlap. The different coloured faces facilitated the detection and segmentation of the object and of the individual faces. The entire image was filtered for certain colours. This made it possible to get the positions and the areas of multiple faces. With this information, it is possible to get pose and orientation information.

The dodecahedral shape was chosen because it allowed simultaneous detection of a minimum of three faces at a time. In a dodecahedron, each face has five neighbouring faces. Depending on the angle, some of these may not be visible. The minimum number of faces that can be seen at any one time is three. It is necessary to be able to detect at least two faces at any time in order to get pose and orientation information. An object with fewer than 12 faces would make this more difficult and alternative methods of information retrieval would have to be implemented.

It is plausible that not every face that is visible to the camera will be detected every time. This can be due to adverse lighting conditions, the colour of the face being present in the background or by processing which would result in the face being lost. This will be discussed in section 7.3. By having multiple colours, more than one face should be detected at any time. This means that pose and orientation information should still be retrievable. Further detail about the use of colour for object detection as well as how the controller’s shape was used for pose and orientation recovery will be discussed in chapters 4-5 and 6 respectively.
4. Sample Data & Histogram

The method for detecting the controller is completely colour-based. The image is filtered to discard everything except certain colours. An example of this can be seen in fig. 4.1 where the filter is discarding all pixel values that do not correspond to yellow. If a pixel is yellow, a white pixel is placed in another image. This can be seen in the top left corner of fig. 4.1. The top right of fig. 4.1 shows the regions of pixels which are yellow. This will be further explained in section 5.4.

4.1 RGB vs. HSV

As mentioned previously in section 2.11.2.1 colour information can be represented in a number of different schemes. The colour space that is perhaps the most widely recognized is RGB. In this colour representation, there are three colour channels, and each channel stores a value between 0 and 255 to represent red, green or blue. In this colour representation scheme, the colour information is combined with the brightness information.
This means that the same object under one lighting condition would have different RGB values than in another lighting condition (this is explained further in section 7.3.1). This does not lend itself well to object segmentation by colour information. There are a number of alternative colour representation schemes that separate colour information from lightness information. These colour schemes lend themselves to being used for object detection as they are more robust to varying lighting conditions.

The colour scheme that was chosen for this project is HSV where the colour information is represented in three channels; hue, saturation and value (or lightness). Hue represents the colour (e.g. red or green) and has a value between 0 and 179. It is important to note that hue has a cylindrical representation. This means that reddish values range from about a hue of 170-179 and 0 -10. Saturation represents the colourfulness of the colour with a value between 0 and 255. Finally, value is a measure of the brightness of the colour and like saturation, has a value between 0 and 255. A visual representation of HSV colour space can be seen above in fig. 4.2. HSV is simply a transformation of RGB. This means that an RGB image can be converted to HSV relatively easily. OpenCV has a built-in method to do this.

![fig. 4.2 - An image showing HSV colour space on the left and RGB on the right. Note how in HSV red can be seen on the left and right, showing the circular structure of the colour space. Also note that the top of the HSV image is black. This shows how the brightness information is separate to the colour information when compared to the RGB image. Zevan Rosser, (2011), HSV [ONLINE]. Available at: http://www.actionsnippet.com/imgs/hsv.jpg [Accessed 30 March 11].](image-url)
Another advantage to working in HSV colour space is that it provided more criteria to distinguish colours. Typically, if one were to try to specify a colour in RGB, they would need to provide values for all three channels. This means that when searching for the colour in the images, all three values would be examined. However, by using HSV, the colour can be chosen with the value for hue, and the values can be further specified by saturation and value.

To ensure that the object particularly distinct and unique, highly saturated colours were chosen for the object. This is based on the assumption that most people will not have a background with several highly saturated colours of the specific hues that were chosen. By choosing colours based on their hue and saturation, the object should be distinct in the scene.

The value component of the colour information was more or less disregarded. Since the object needed to be tracked under different lighting conditions, almost all brightness values were considered. A small restriction was placed on this, however. If the brightness was very high or very low, the assumption was made that that pixel was not of interest.

To make the tracker more robust, and to make it more likely to work under different lighting conditions, a histogram of valid colour pairs was constructed from a sample set of images. A sample image for each colour was provided. For each pixel in the sample image, the hue and saturation values were recorded. This created a valid hue-saturation pair. These values were then saved to a look-up table. This was repeated for each sample image. This approach assumes that there is no overlap between colours, i.e. a valid hue-saturation pair for one colour will not be found for another. Each hue-saturation pair can only map to one colour for the tracker to work, so this is a reasonable assumption. Once this was completed, a histogram with all valid hue-saturation pairs and the colour they mapped to was created. This histogram was used as a look-up table when searching for the controller. This will be further discussed in chapter 5. The histogram can be seen below in fig. 4.3.
It is clear from fig.4.3 that there are a number of issues with the data that was obtained from the sample images. There are a number of outliers for every colour, but in particular for blue. Some of the values are also a lower saturation than is desired. In order to create a better look-up table for hue-saturation pairs, this data needed to be adjusted somehow.

The following (simplified) code demonstrates how pixel values are extracted from the image, and how they are stored in the look-up table (sample_data[][]):

```c
void Tracker::getImageInfo(IplImage* source, int colour){
    //copy the source image and convert to HSV
    IplImage* image = cvCreateImage(cvGetSize(source), IPL_DEPTH_8U, 3);
    cvCvtColor(source, image, CV_BGR2HSV);

    int width_step=image->widthStep;
    int pixel_step=image->widthStep/image->width;
    unsigned char* curr_point;

    for (row=0; row < image->height; row++)
        for (col=0; col < image->width; col++)
            {
                curr_point = GETPIXELPTRMACRO( image, col, row, width_step, pixel_step );

                val = (int)curr_point[RED_CH];
                sat = (int)curr_point[GREEN_CH];
                hue = (int)curr_point[BLUE_CH];

                sample_data[hue][sat]=colour;
            }
}
```

fig. 4.3 - Histogram containing all valid hue-saturation pairs found from samples of each colour
4.2 Statistical Alteration of Data

To create a more usable data set, statistical operations were performed on the original data obtained from the sample images. One of the main purposes of this work was to remove the outliers which can be seen in fig. 4.3. The data was adjusted using values for the standard deviation of both hue and saturation for each colour.

For each colour in the histogram, the average values for hue and saturation were found. These values were then used to find the standard deviation for each colour. The results of this operation can be seen below in fig. 4.4. The data was then altered based on the values for standard deviation. This method provided a simple solution which was also computationally inexpensive. It can easily be adjusted to include fewer or more pair values.

<table>
<thead>
<tr>
<th>Colour</th>
<th>Average Hue</th>
<th>Standard Deviation Hue</th>
<th>Average Saturation</th>
<th>Standard Dev. Saturation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>114</td>
<td>15</td>
<td>98</td>
<td>55</td>
</tr>
<tr>
<td>Green</td>
<td>50</td>
<td>9</td>
<td>150</td>
<td>36</td>
</tr>
<tr>
<td>Red</td>
<td>16</td>
<td>47</td>
<td>192</td>
<td>19</td>
</tr>
<tr>
<td>Pink</td>
<td>153</td>
<td>55</td>
<td>210</td>
<td>18</td>
</tr>
<tr>
<td>Yellow</td>
<td>27</td>
<td>1</td>
<td>127</td>
<td>22</td>
</tr>
<tr>
<td>Turquoise</td>
<td>95</td>
<td>4</td>
<td>142</td>
<td>36</td>
</tr>
</tbody>
</table>

fig. 4.4 - Table outlining the values obtained for average and standard deviation of hue and saturation for each colour
The standard deviation is calculated using the following formula:

\[
\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2}
\]

where \( \sigma \) is the standard deviation,

\( n \) is the number of samples (i.e. the number of hue-saturation pairs for that colour),

\( x_i \) is the current sample,

\( \bar{x} \) is the average.

The standard deviation is essentially a measure of the variability of the data. To ensure that the colours in the histogram result in a reasonable detection of the object, it is necessary to remove outliers from the data set. The standard deviation provides a means of doing this, as described above. In the following two figures, the results of this operation can be seen.

The code below demonstrates how the standard deviation was calculated.

```c
void standardDeviation(int colour, int elem, int n){
    int total_hue=0, temp_hue=0, total_sat=0,temp_sat=0;
    for (int i=0; i<HUE_RANGE; i++) {
        for (int j=0; j<SAT_RANGE; j++) {
            if (data[i][j]==elem) {
                //get sdev for hue
                temp_hue = (i - averages[colour][HUE]);
                temp_hue = pow(temp_hue, 2);
                total_hue = total_hue + temp_hue;
                //get sdev for sat
                temp_sat = (j - averages[colour][SAT]);
                temp_sat = pow(temp_sat, 2);
                total_sat = total_sat + temp_sat;
            }
        }
    }
    sdev[colour][HUE] = sqrt(total_hue/n);
    sdev[colour][SAT] = sqrt(total_sat/n);
}
```
fig. 4.5 - Altered histogram. The black points represent those that have been discarded. These results were obtained by discarding any point whose hue or saturation was greater than the standard deviation from the average.

fig. 4.6 - Altered histogram. As in fig. 4.5, the black points represent those that have been discarded. This histogram was obtained by discarding any values whose hue or saturation were greater than two standard deviations from the average.
This (simplified) code gives an example of how the standard deviation is used to alter the values in the look-up table. The variable \texttt{scale} is the number of standard deviations that the hue or saturation cannot exceed without being discarded.

```c
for (int i=0; i<HUE_RANGE; i++) {
    for (int j=0; j<SAT_RANGE; j++) {
        switch (data[i][j]) {
            case YELLOW:
                //check the hue
                if (fabs(i-averages[0][0])>(sdev[0][0]*scale)) {
                    //discard the point
                    data[i][j]=-2;
                }
                if (fabs(j-averages[0][1])>(sdev[0][1]*scale)) {
                    //discard the point
                    data[i][j]=-2;
                }
                break;
        }
    }
}
```

By replacing the value in the look-up table (\texttt{data[ ] [ ]}) with -2, the point is discarded and the colour of that point in the histogram is changed to black. Only certain values which correspond to each of the 6 colours are used as valid data in the look-up table. Any value in the look-up table which does not correspond to these values is not used in the classification.
5. Object Detection and Segmentation

The object detection and segmentation is performed by utilizing the data in the histogram. The histogram of valid hue-saturation pairs is used to classify each pixel in the image. By classifying each pixel, the software will detect the controller in the scene which will ultimately allow the pose and orientation information to be found. This section details how the histogram is used for object segmentation.

To track the object, a video feed must be established. This is done using OpenCV methods. By creating a CvCapture object and using CvQueryFrame to get the current frame from the webcam, a live video feed is established. In the demonstration video, the frame is retrieved from a pre-recorded video. However, the method for extracting frames is the same. The code below shows how this is done in OpenCV and C++.

```cpp
CvCapture* capture = 0;
IplImage* frame = 0;
cvNamedWindow("Camera", 1);

capture = cvCaptureFromCAM(0);
if (!capture) {
    return -1;
}
frame = cvCreateImage(size, IPL_DEPTH_8U, 3);//create a blank frame

while (true) {//continuously get new frames
    //get a new frame from video capture
    frame = cvQueryFrame(capture);
    ...
    ...
    cvShowImage("Camera", frame);//display the frame in a window
}
```

Once the frame has been extracted, classification can begin. Each pixel in the image is analyzed and then classified based on the values in the look-up table. As discussed previously, only hue and saturation values are considered. The next section will explain the colour classification method in further detail.
5.1 Colour Classification

Colour classification is performed by comparing the pixel values from the current frame with the known valid values in the look-up table. For each pixel in the image, the hue and saturation values are obtained. If this combination of hue and saturation lies within the look-up table, that pixel is assigned the appropriate colour and placed in the result image. If that hue-saturation pair is not in the look-up table, no value is placed in the result image at that pixel location. An example of the results of this colour classification can be seen in fig. 5.1.

In fig.5.1, it can be seen that some pixels are being incorrectly classified as part of the controller. It is not feasible to create a perfect look-up table that would correctly classify every pixel. However, something must be done to try to reduce the false positives in the result image. A number of methods are used in combination to try to get a clean result. First, the image is smoothed to try to reduce the noise. Then some morphological operations are performed on the image to further reduce noise. Then regions of the same coloured pixels are found, and these regions are filtered by size to eliminate remaining falsely classified pixels.
fig.5.2 - Images demonstrating different smoothing filters. The top image shows the result of a Gaussian filter with a 9x9 kernel. The middle shows a median filter with a 9x9 aperture. The bottom image shows the result of using a Gaussian filter with a 27x27 sized kernel.
The smoothing operation is executed by using OpenCV's `cvSmooth` method. This method facilitates the use of several different filters including Gaussian and Bi-Lateral smoothing filters. The filter that was used was a Gaussian with 9x9 kernel. The images on the preceding page in fig. 5.2 show the results of Gaussian smoothing with different sized kernels and an example of median filtering for contrast. Gaussian filtering was chosen because it does not create artifacts, it is rotationally symmetric and it has a simple and intuitive relationship between the kernel size and the smoothing.

The 9x9 Gaussian filter removed some noise while still keeping enough detail to properly identify and track the controller. This can be seen when comparing the top image in fig. 5.2 and the unsmoothed image in fig. 5.1. The details of how Gaussian Smoothing works will be explained in the following section.

### 5.2 Gaussian Smoothing

Gaussian smoothing is used to blur images and remove detail. In this project, it is used to reduce the amount of noise in the background. By blurring and removing some detail, it is possible to reduce the amount of incorrectly classified pixels, thereby improving the overall performance of the tracking software.

The Gaussian filter is a two dimensional convolution operator and works in a somewhat similar manner to a mean filter, but using a different kernel which is shaped like a bell curve. The 2-D Gaussian is defined by:

\[
G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}
\]

Where \( \sigma \) is the standard deviation.

The Gaussian is efficient to implement because it is separable. This means that the x and y components can be calculated separately. The 2-D convolution can be achieved by first convolving with a 1-D Gaussian in the x direction, and then repeating in the y direction (Fisher 2003).
The Gaussian provides a weighted average of the pixel values, as opposed to the mean filter which provides a regular average of their values. The pixels in the centre of the Gaussian are given a greater weighting. The weighting is dependent on the kernel and is shaped like a 2-D Gaussian distribution, as seen in fig. 5.3. This results in more gradual blurring and preserves more edge information.

5.3 Mathematical Morphology

Mathematical morphology differs from the majority of image processing in the fact that it makes use of non-linear algebra as opposed to the mathematics that is used in computer vision which is something that is very close to calculus. (Sonka, 2007). Morphological operations can be used for a number of purposes, but here they are used as part of object segmentation and noise removal.
Erosion and dilation are used to remove noise and retain the overall size of the controllers' faces.

Erosion typically decreases the size of objects and removes small anomalies by subtracting objects with a smaller radius than the structuring element. It also removes perimeter pixels from large objects. While it is advantageous that this method removes small anomalies, the removal of perimeter pixels from larger objects is undesirable.

fig.5.4 - The result of morphological operations. Note the reduction in noise when compared to fig.5.2
To compensate for this, the image is dilated. A graphical representation of dilation is given above. Erosion works in a similar fashion, but places zero values instead of non-zero values.

Dilation usually increases the size of objects. It fills in holes and broken areas and connects areas that are separated by spaces smaller than the structuring element. Although this is not perfect in restoring the original shape of the controller faces, the benefits of noise removal outweigh this sacrifice.

5.4 Connected Components

The next step in object detection is to identify the individual faces of the controller. This was done by creating regions of pixels that are the same colour. This also facilitated further noise and false positive removal.
These regions were formed by using OpenCV’s built-in method, CvFindContours. The contours are found using Suzuki and Be’s topological structural analysis of digitized binary images by border following algorithm. This algorithm determines the surroundness relations around the borders of the binary image (Suzuki et al, 1985). The contours are then filled in to form components. The code that finds the components is as follows:

```c
cvFindContours( binary_image,
storage,
&contours,
sizeof(CvContour),
CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE );
```

```c
if (result) {
    cvZero( result );
    for(CvSeq* contour = contours ; contour != 0; 
        contour = contour->h_next ) {
    area = cvContourArea(contour, CV_WHOLE_SEQ);
    if (area>100) {//filter small components
        //colour inside of the contour boundaries
        cvDrawContours( result, contour, color, color, -1,
                        CV_FILLED, 8 );
    }
}
```

CV_CHAIN_APPROX_SIMPLE describes the algorithm that is used to find the components. Here it means that just the outer borders of objects are found. Using different values here can also find “holes” inside of components, or interior borders.

A number of methods can be carried out to find out more information about the objects in the image. The area of the components can be found, simply by counting the number of non-zero pixels in the image. This figure is then used in a further noise removal step. Any component whose size is not greater than a given area is discarded, as it is most likely some noise or other anomaly.

This resulted in the image in fig.5.5. Each visible face in the colour classification result image (bottom right) has its own component representation in the appropriate window. These representations of the faces of the controller are then used to obtain pose and orientation information. The methods used to retrieve this information will be outlined in the following section.
6. Pose and Orientation Recovery

This section will outline the methods that were used in the analysis portion of the tracking software. Using the component representations of the faces of the controller, a number of properties of the object can be determined. By analyzing the size and positions of the faces, pose and orientation information can be recovered. The methods that were used were able to indicate changes in pose and orientation information, but were unable to provide absolute values for all six pose and orientation values. The methods used and the reasons behind their use will be outlined below.

To fully emulate the Playstation Move controller, six degrees of freedom should be represented by the controller. These need to be representative of the x, y and z positions and the roll, pitch and yaw of the controller. The x and y position of the controller can be found very easily, as it is just a matter of getting the pixel coordinates of the controller in the image. Determining the z position (or the scale) of the controller is less trivial, as it involves trying to get three-dimensional data from a two-dimensional image. This problem is continued whilst getting the roll, pitch and yaw values. Getting orientation information using just computer vision and what is basically just colour information is inherently difficult. This is perhaps why commercial motion controllers make use of gyroscopes and angular accelerometers to obtain this information accurately.

As stated above, x, y and z values represent the position of the controller in three-dimensional space. The roll, pitch and yaw are simply rotations about each axis. As demonstrated in fig. 6.1, the roll is a measure of the rotation about the x axis, the pitch is a measure of the rotation about the y axis, and the yaw is a measure of rotation about the z axis.
6.1 XY Position

The x and y positions were the most easily obtained coordinates from the controller. This is because the values for x and y in the real world correspond to the pixel coordinates of the controller in the frame.

The pixel coordinates of each face were found using OpenCVs CvGetSpatialMoments method and cvMoments data structure. The x and y coordinates for each face detected are found, and an average of these are obtained. This average represents the overall x and y coordinates of the controller. However, there is a slight issue with this method which will be further discussed in section 7.

CvGetSpatialMoments retrieves the spatial moment from the motion state structure, cvMoments. The function retrieves the spatial moment which is defined as:

\[
M_{x\_order, y\_order} = \sum_{x,y} (I(x, y) \cdot x^{x\_order} \cdot y^{y\_order})
\]

where \(I(x, y)\) is the intensity of the pixel at \((x, y)\).
The code for obtaining the spatial moments of each component is show below:

```c
void
getMoments(IplImage* source)
{
    moments = (CvMoments*)malloc(sizeof(CvMoments));
    if (source==NULL) {
        return;
    }
    cvMoments(source, moments);
    // The actual moment values
    double moment_x = cvGetSpatialMoment(moments, 1, 0);
    double moment_y = cvGetSpatialMoment(moments, 0, 1);
    double area = cvGetCentralMoment(moments, 0, 0);
    posX = moment_x/area;
    posY = moment_y/area;
}
```

It should be noted that the x and y positions are only updated if they are greater than the previous x and y positions by a certain constant. This is to try to stabilize the values and to remove jittery and small movements and errors due to noise. This error checking is performed when updating any of the six degrees of freedom.

6.2 Z Position

The z position of the controller is found using the same general method as that which is used to find the x and y positions. Finding the z position involves extracting three-dimensional information from a two-dimensional image. The area of the controller in the image is representative of the z position of the controller. It is not the absolute position of the controller on this access, but it does give a representation of the z value. If the area of the controller is large, it means that the controller is close to the camera. When the area is smaller, it indicates that the controller is further away from the camera.

The information is determined from the cvMoments data structure using the function cvGetCentralMoment. This gives a measure of the size of the component. This is then scaled down to a number that is more manageable. This will indicate the relative position and movement along the z axis. There are a number of issues and difficulties associated with retrieving the z position, which will be discussed further in section 7.
6.3 Roll

The roll is a measure of rotation around the x axis. Measuring the roll accurately and precisely proved to be quite difficult. It is determined using the slope between two faces and a predefined model of the controller.

The object detection part of the tracking software produces a number of images, each containing a face of the controller if it has been found. To calculate the roll, the information from two faces is used. If more than two faces are found, the faces to be used in the calculations are chosen arbitrarily. The centre points of these faces are obtained (as previously discussed in section 6.1). The next step in obtaining the roll information is calculating the slope between these two points. There is a predefined condition for each face pair to describe when the roll is 0, i.e. the controller is upright. For each possible pair of faces, there is a pre-defined positional relationship which indicates when the roll should be zero.
In fig. 6.2, the blue face is assigned the front face. The other face that was found is the green face. The slope between the two centre points of these faces is calculated. The orientation depicted below shows the upright model. The slope is then offset depending on the relative positions of the faces. The upright model when the front face is blue describes a roll of 0 degrees to have the turquoise face at 0 degrees. Since the slope between blue and turquoise is not known, the slope that has been calculated between blue and green must be offset to compensate for this.

The following code demonstrates how the roll is calculated:

```java
switch (front) {
    case YELLOW:
        switch (other_face) {
            case GREEN:
                roll = slope;
                break;
            case RED:
                roll = slope+OFFSET1;
                break;
            case PINK:
                roll = slope+OFFSET2;
                break;
            case TURQUOISE:
                roll = slope+OFFSET3;
                break;
            case BLUE:
                roll = slope+OFFSET4;
                break;
        }
    break;
}
```

Where front is the “front” or largest face and the variable `slope` is calculated by:

```java
double getSlope(CvPoint a, CvPoint b){
    double y_diff = b.y-a.y;
    double x_diff = b.x-a.x;
    if (((y_diff==0)||(x_diff==0)) {
        return 0;
    }
    return y_diff/x_diff;
}
```

`CvPoint a` and `CvPoint b` represent the x and y coordinates of two faces.
This is just part of the code demonstrating the model for when the yellow face is forwards. There is code for each face which continues in this manner.

The method that was used provides an indication of the change, if any, in rotation about the x axis. However, this method does not obtain the absolute value or indeed the magnitude of the change. It is a rough approximation to the roll and the disadvantages and advantages of this approach and alternative methods for obtaining the roll will be discussed in section 7.

6.4 Pitch & Yaw

The pitch and yaw are both calculated using the same data as what is used to calculate the roll. However, it is used in a different manner to get a representation of the pitch and the yaw. As discussed previously, the pitch and yaw are a measure of the rotation about the y axis and the z axis respectively. Again, with the Playstation Move controller, this information is calculated using an angular accelerometer and a gyroscope. Very precise values are found for both pitch and yaw. Getting this level of precision and accuracy using the information available in this project is nontrivial and the values that were ultimately found provided an indication of the change in pitch and yaw rather than the absolute values.

As in the method for calculating roll, a “front face” is assigned. Then, if present, an upper/lower face and a left/right face are assigned. These are assigned based on their coordinates in relation to the front face. If a face has an x coordinate which is roughly equal to that of the front face, and it’s y coordinate is greater than that of the front face, it is classified as the upper face. The left/right face is assigned in a similar manner.

The pitch is then calculated as the distance between the centre point of the front face and the upper face. As the controller is tilted upwards, this distance will decrease and vice versa for the downwards direction. The yaw is calculated in the same manner, but using the left or right face.
The following code is used to calculate the pitch:

```java
if (slope > 0.7) { // if an upper or lower face has been found
    if (fabs(last_pitch - length) < JITTER) {
        return;
    }
    if (fabs(last_pitch - length) > 20) { // if the change is too great,
        // do not update
        return;
    }
    last_pitch = pitch;
    pitch = length;
}
```

The following code is used to calculate yaw:

```java
if (slope < 0.3) { // if a left or right face has been found
    if (fabs(last_yaw - length) < JITTER) {
        return;
    }
    last_yaw = yaw;
    yaw = length;
}
```

Where length is the distance between the two faces.

Again, this provides an indication of changes in yaw and pitch. The problems with this approach will be further discussed in section 7.

fig.6.3 - Graphical representation of pitch and yaw retrieval method. The distance between the “front face” and the left/right face gives the yaw. The distance between the “front face” and the upper or lower face gives the pitch.
7. Evaluation

This section aims to review the results of the project. It will discuss the overall successes and failures of the project. It will discuss the difficulties and disadvantages of the approach that was taken. It will then go on to propose solutions to any problems that were encountered and outline possible improvements and additions that could be made to the project.

7.1 Results

The software was tested primarily on a prerecorded video. This was to ensure that the lighting conditions were fully known and so the sample images could be chosen appropriately. The following screenshots show the results of the tracking software. A model was drawn to represent the position of the controller. The position and size of the cube are altered to correspond to the motion of the controller itself. The full pose and orientation values can be seen in the top left hand corner of the “Model” window. The demonstration video is included in a CD which is attached to this report. The following screenshots aim to be representative of this video.

7.1.1 Screenshots

The following screenshots demonstrate the operation of the tracking software. fig.7.1 demonstrates the successes of the tracking software. The first image shows working pose recovery. The second image displays colour classification with noise removal techniques enabled. The third image shows the component classification without the noise removal techniques enabled. Some of the problems associated with the tracking software can be seen in fig. 7.2. The first image shows a case where the colour classification fails. The yellow face is not detected. This can be seen again in the second image, as can some incorrectly classified noise. There is a large area below the controller which has been wrongly classified as the red face. The final image demonstrates an example of the pose recovery failing despite successful object detection. Further screenshots, along with demonstration videos are attached to this project in appendix A.
fig.7.1 - Images demonstrating successful classification, tracking and pose recovery
fig. 7.2 - Images demonstrating incorrect classification and errors in pose recovery.
7.1.2 Look-up Table Values

The values that were obtained for the average hue and saturation and standard deviation of each colour provide some insight on the validity of the idea to use highly saturated colours to make the controller distinct in the scene. The table below details these values.

<table>
<thead>
<tr>
<th>Colour</th>
<th>Average Hue</th>
<th>Standard Deviation Hue</th>
<th>Average Saturation</th>
<th>Standard Dev. Saturation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>114</td>
<td>15</td>
<td>98</td>
<td>55</td>
</tr>
<tr>
<td>Green</td>
<td>50</td>
<td>9</td>
<td>150</td>
<td>36</td>
</tr>
<tr>
<td>Red</td>
<td>16</td>
<td>47</td>
<td>192</td>
<td>19</td>
</tr>
<tr>
<td>Pink</td>
<td>153</td>
<td>55</td>
<td>210</td>
<td>18</td>
</tr>
<tr>
<td>Yellow</td>
<td>27</td>
<td>1</td>
<td>127</td>
<td>22</td>
</tr>
<tr>
<td>Turquoise</td>
<td>95</td>
<td>4</td>
<td>142</td>
<td>36</td>
</tr>
</tbody>
</table>

A number of features of these results should be noted. The values for standard deviation are not consistent for each colour. From the table above, it can be seen that the standard deviation for hue and saturation differ greatly for each colour. With the exception of pink and red, the standard deviation for hue values is quite low. This indicates that the values that were found from the sample images were mostly consistent. A low standard deviation for a colour signals that there should be fewer false classifications as the colour is distinct and consistent. This is supported by the fact that the vast majority of the noise seen in the demonstration video and screenshots is classified as red, which has the highest standard deviation for its hue values. The standard deviation of the saturation values for each colour is larger than that for the hue. The average standard deviation for hue is 22 whereas the average standard deviation for the saturation is 31. This shows that the concept of using saturation to create a unique object may be flawed. More testing with a larger variety of colours would be necessary to prove or indeed disprove the validity of this concept.
7.1.3 Real Time Calculations

One of the aims of this project was to ensure that the object tracking software operated in real time. The demonstration video (Appendix A), with no processing performed ran at a framerate of 14.6815Hz. The tracking software using the same video also ran at 14.6815Hz. This is a good indication of the feasibility of the use of the controller as an input device.

Testing the framerate for the object tracking software with a live camera feed was more difficult. This was due to the webcam that was used. The webcam performed a number of processing steps itself (e.g. automatic exposure correction) and so did not always run at its maximum 30Hz. The webcam only operated at 30Hz under very strict lighting conditions. However, it was observed that the object tracking software had no effect on the framerate up to 25Hz. This was the maximum framerate that was achieved when testing the tracking software and was the same the framerate for the webcam with no processing under those lighting conditions.

7.2 Successes

Overall, the project was somewhat successful. A number of components of the system worked well, whilst others did not. The programme was successful as a proof of concept. It was able to locate the controller in a controlled environment and in finding its x, y and, to a lesser extent, z positions. There were a number of issues with the values that were found for the z position which will be discussed below in section 7.3.2. However, while the position of the controller was found relatively successfully, the orientation was not found and proved to be quite difficult to get any values for.

The method used for locating the object (i.e. the look-up table and colour classification) was quite successful in a constrained environment. Although a number of steps were taken to try to make the tracking software more resilient to adverse lighting conditions, namely the look-up table and working in HSV colour space, the tracker ultimately only worked under the same lighting conditions that the sample images were taken in. There
are methods which could be implemented to try to overcome this problem more successfully which will be discussed in section 7.3.1.

The x and y positions of the controller was found quite successfully. The method for finding the x and y positions of the controller itself worked well. Any inaccurate x and y values were due to failings of the classification method or negative effects of the morphological operations.

Another success of the project is the fact that all of the calculations and tracking can be done in real time, which is essential in trying to use this as a viable controller. The Playstation Move operates at 60Hz. The tracking software in this project operates at 14.68Hz in the demonstration video and at a maximum of 30Hz when working from the live camera feed. However, in both cases, it is not the software which limits the speed. The video in the demonstration was recorded at approximately 15Hz, and the webcam used operates at a maximum of 30Hz under optimal lighting conditions.
7.3 Difficulties

The project experienced a number of difficulties and setbacks throughout its development. These were due to a number of factors and will be discussed individually below in detail.

7.3.1 Lighting Issues

One of the biggest problems that was encountered was the issue of colour constancy. Colour constancy describes the human eyes ability to perceive colours as constant under different lighting conditions. However, a webcam cannot do this. This created a problem when trying to identify the controller. As stated previously, to try and overcome this problem, a look-up table of valid hue and saturation pairs was generated on a sample set of images and HSV colour space was used instead of RGB. However, it was found that even with these precautions, it was still difficult to reliably identify the controller under different lighting conditions.

fig.7.4 - Specular effect on the controller. The change is colour is apparent between the left-hand and right-hand image. The green face appears to have changed colour, while the other colours remain constant.
Specular light also causes a problem with colour classification. If bright light shines on a face, it makes the colour very washed out, even though the colour is highly saturated. This effect can be seen clearly in fig. 7.2 above. Under this condition, faces are completely lost. This can be seen in the demonstration video, in particular with the classification of the yellow face. The yellow face was particularly susceptible to incorrect classification due to specular effects. This effect could be minimized by the choice of a different colour.

7.3.2 Problems with Orientation and Pose Recovery

In an attempt to try to remove errors, the accuracy and reliability of the tracker was reduced. It was assumed that any movements that were small were not valid movements, and were just some factor of noise. This made it difficult to detect very small changes in motion. Another factor that contributed to this was when face were lost or not detected. This could be seen when a face was angled away from the camera, making it smaller than the threshold for component filtering. This means that the average position would be inaccurate. This could be solved by inferring the locations and sizes of faces that should be present based on the known positions and sizes of the faces and adjusting the positional data accordingly.

Calculating an absolute value for the z position is inherently difficult. It is relatively easy to get a representation of the position along the z axis, using the information about the size or scale of the object. Getting an absolute value is more difficult because it involves getting z values from an image which only has two dimensions. This is best solved by using the initial position of the object as the zero value for z. From there, it is possible to calculate if the object moves in a positive or negative direction along the z axis and by how much.

Finding the orientation information proved to be very difficult. The values that were found were unreliable and not a very good indication of the movement of the controller. This is partly due to the classification and filtering issues that were discussed above, but it is mainly due to the actual method that the roll pitch and yaw values were calculated with. Using the same information about size and position of the faces, a more robust and accurate method of obtaining these values could be used. By comparing the areas and positions of three different faces, it should be possible to obtain the rotational information. For example, using the same orientation as that in fig. 6.3, if the pink, yellow and blue
faces are found, by getting the ratio of the faces to one another and combining that information with the positions of each face, it would be possible to obtain the angular information for each axis.

7.3.3 Noise

The system is susceptible to noise. Because the values were calculated as an average of the positions of the present components, if there was a component found that was in fact not a face, it could influence the positional values. This can be seen in fig. 7.1 above. The image on the left shows how the x and y positions have been corrupted by the area of noise towards the bottom of the image. The image of the right shows how a small amount of noise does not affect the accuracy of the positional data.

This problem could be solved by implementing some validation phase. By comparing the positions of all the components, anything that deviated too much from the majority of components could be discarded as noise. In the case of fig. 7.1, the position of the falsely classified components is far away from the controller itself. This should be then be re-classified as noise. The validation phase could also see if the components are close to other components that are a different colour, since it is known that the object has several neighbouring coloured faces.

Another fault in the object detection software is due to the noise filtering methods (i.e. component filtering and morphological operations). These operations cause faces to be lost when they are too small. This occurs when the face is angled away from the camera. This has repercussions on the overall accuracy of the positional and orientation values. Again, there a number of possible solutions to this problem. From the faces that are found, and from information about the faces that have been found recently, the software could be expanded to infer the location of faces.

7.3.4 Real Time Calculations

In an early version of the tracking software, the method that was used to identify the controller was different to what was ultimately used. The OpenCV method cvInRange was used to classify pixels. This method worked as a binary filter. The colours were
specified as parameters of the method. The method returned an image with non-zero values in the pixels whose values were within the limits specified. However, this method proved to be too costly and the application then was subject to lag and could not longer operate in real time. A similar issue was met when OpenCVs histogram data structure was used. Originally, an array of cvHistogram types (one for each face’s colour) was used to store the look-up table information. However, searching through these histograms during the colour classification stage increased the processing time and the programme no longer operated in real time. Ultimately, both of these problems were solved by writing specialized methods for performing these tasks.

7.3.5 Other Difficulties

The framerate of the webcam unfortunately could not be controlled or limited. This was due to a compatibility issue with the webcam itself and OpenCV. Although OpenCV has a method for controlling the framerate, the drivers for the iSight camera do not allow this. The framerate was therefore determined by the iSights drivers, and was dependent on lighting conditions.

The iSight webcam that was used during development also may have contributed to some of the lighting problems. It has a feature to automatically correct the exposure in the image. This may have contributed to the issues with specular effects.

Initially, in order to clean up the data in the look-up table, kernel density estimation was implemented. Kernel density estimation is a non-parametric data smoothing technique (Duong, 2007). However, the attempts to write a method for this technique were ultimately unsuccessful. The data that was obtained was not useable. This was most likely due to the implementation of the method itself. A week was spent developing this code and trying to debug it and it was eventually replaced by using the standard deviation.
7.4 Future Work and Improvements

The current weaknesses of the tracking software indicate areas which could use more attention and development. There are also a number of features which could be added to the software. The controller should also be demonstrated with some application as a proof of concept of its usefulness.

7.4.1 Improving Pose and Orientation Recovery

One of the most important improvements that could be made is to obtain absolute and reliable full pose and orientation information. This would require the classification to be made more robust as well as the pose and orientation calculations to be altered. However, most of the information required for this already exists in the software.

7.4.2 Improving Performance Under Varying Lighting Conditions

Improving the performance of the object tracking software under different lighting conditions is another improvement that could be made in the future. There are several possibilities that could be implemented to try and improve the colour classification under various lighting conditions. The first of these would be to use a larger sample set of images to generate the look-up table with varying lighting conditions. Rather than having just one look-up table containing sample data for varying lighting conditions, which could lead to a lot of falsely classified points, using multiple histograms and finding the best one depending on the lighting would be a good solution. This could be done when the programme is started during a calibration phase.

An alternative to this would be to alter the colour information to try and compensate for the changed lighting condition. By applying a uniform colour filter to the entire image, the colours could be altered to be more similar to those that are in the sample set of images.
7.4.3 API and Error and Status Reporting

There are a number of features which would be desirable to add to the programme. These are the creation of an API so that the application can be easily integrated with other applications as a controller. It would also be useful to have some status and error reporting. This would be particularly useful for when too few faces are found or a particularly large area of noise is found.

7.4.4 Integration with a Game

For a further proof of concept, the tracker could be integrated with a game. It would hopefully work as a viable alternative to other input systems. It would most likely have to be used in conjunction with some other device, perhaps a keyboard. The Playstation Move has a number of buttons which allow for greater input variations. Because there are no sensors in the emulated controller, use of some other input device in conjunction with the controller would be necessary to achieve a similar performance and usability as that of the Playstation Move.

7.4.5 Accuracy Comparison and Testing

A comparison of accuracy and usability is another future expansion on this project. The improvements outlined above would be necessary before any decent accuracy tests could be carried out. The Playstation Move offers full motion capture and is very accurate. This is largely due to the sensors and hardware that it implements. Assessing the accuracy of the emulated controller and the Playstation Move and comparing the two would give a more concrete measure of the viability of “copy-cat” controllers such as this. Although it is unlikely that the emulated controller would be as accurate as the Playstation Move, purely because of the nature of computer vision and the inherent difficulty of pose and orientation recovery, it would be possible to assess whether or not the accuracy of the emulated controller was accurate enough to be used as an input device.

A comparison between the controller and other input devices would also be useful in assessing its practicality. A test could be designed to time how long it takes a user to complete a number of tasks using the motion controller and a traditional input device, such
as a mouse. To prove the idea that motion controllers are more intuitive than the traditional input devices, the user should take less or at least the same time to complete. This test would have to be designed carefully, and it would be necessary to ensure that the tasks weren’t biased to either input device, but it would be a very useful measure of the viability of this technology as an interface device.
8. Conclusion

Current input devices, in particular those used on PC are still very primitive in their design. Although much research has been done in the field of human-computer interaction, input devices have not reflected their findings.

One of the largest barriers to the widespread use of motion controllers in the past was the performance of computers and the high cost of specialist hardware. This project has proven that it is possible to track an object in real time on a standard laptop using open source libraries and easily obtainable objects. This overcomes both of these issues.

Although the pose and orientation information that was obtained was not as accurate or reliable as that of the Move, with further development, the performance of the tracking software could be improved. The tracking software was able to find the object in constrained environments. With refinement, the reliability of the detection of the faces of the controller could be improved.
References


**Electronic Sources:**


Videos:

Appendices

Appendix A: Electronic Sources

The attached CD contains all of the source code for the object tracking software. It also includes a full set of sample images and a test video. The source code has only been tested on Mac OSX. Some alterations to the code are necessary to compile it on a different operation system. Directions on how to install and run OpenCV can be found at: http://opencv.willowgarage.com/wiki/.

Additionally, the demonstration videos and screenshots of the application are available on the CD. The videos are in .mov format and will play on a computer that has quicktime installed.