Head Tracking with NintendoWiimote to view Terrain Maps

Mbulelo Ntlangu

A project report submitted to the Department of Electrical Engineering,
University of Cape Town, in partial fulfilment of the requirements
for the degree of Bachelor of Science in Engineering.

Cape Town, October 2008

Declaration

I declare that this research report is my own, unaided work. It is being submitted for the degree of Bachelor of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author .....

Cape Town

21 October 2008

Abstract

This research report aims to investigate the viability of creating an application for the purpose of interactively viewing 3d terrain maps with the motion of a user's head by performing headtracking using the Nintendo Wiimote.

This investigation begins by reviewing previous work done in the field of head tracking. It then details the theory of operation of the Nintendo Wiimote followed by a series of tests conducted on the Wiimote to verify its ability to reliably provide motion sensing data. A review and assessment of some of the various graphics programs and file formats used to render, view and store terrain map data is presented. Some of the available Wiimote api's are also reviewed here.

In conclusion it was found that the Wiimote doesn't necessarily outperform previous technologies used for head tracking, however it does incorporate functionality which can greatly simplify head tracking systems. Also the combination of GlovePIE scripts and MeshLab showed the most potential for the development of a head-tracking-terrain-map-viewing application.

Acknowledgements

I would like to thank my mother for every sacrifice that she has made to allow me this opportunity, for all the love and support she embraced me with and for believing for the both of us even when I'd lost faith.

I'd also like to thank Ms Bianca Valentine for her love and patience, and for all the home cooked meals which kept me alive while preparing this report.

Finally I'd like to thank Professor M.R. Inggs for his non-intrusive supervision and guidance and for providing me with all the materials necessary to prepare this project.

Contents

Declaration **i**

Abstract **ii**

Acknowledgements **iii**

1 Introduction **2**

1.1 Background 2

1.2 Problem Specification 3

1.3 Objectives 3

1.4 Scope and Limitations 3

1.5 Plan of Development 4

Bibliography **47**

Bibliography **47**

List of Figures

1.1 Photograph of the back and front views of the Nintendo Wiimote 7
1.2 Schematic of the ADXL330 Accelerometer IC 8
1.3 Flow chart showing the flow control provided by SDP 9
1.4 Coordinate System of the Wiimote's internal accelerometer 12
1.5 Wiimote Pendulum System 14
1.6 Comparison of measured acceleration against calculated acceleration for Y-Z plane . 16
1.7 Comparison of measured acceleration against calculated acceleration for X-Z plane . 17
1.8 Wiimote and Spring Balance Test System 19
1.9 Comparison of measured acceleration and calculated acceleration in the Wiimote-Spring system for X-axis 21
1.10 Comparison of measured acceleration and calculated acceleration in the Wiimote-Spring system for Y-axis 22
1.11 Comparison of measured acceleration and calculated acceleration in the Wiimote-Spring system for Z-axis 22
1.12 Comparison of Measured and Expected X coordinates for 20mm steps along the X-axis of the Wiimote's infrared camera. 27
1.13 Comparison of Measured and Expected Y coordinates for 20mm steps along the Y-axis of the Wiimote's infrared camera. 28
1.14 Comparison of Measured and Expected distance for 20mm steps along the distance-axis of the Wiimote's infrared camera. 28
1.15 Comparison of Measured and Expected distance for 20mm steps along the distance-axis of the Wiimote's infrared camera. 29
1.16 Comparison of Measured and Expected X coordinates for 20mm steps along the X-axis of the Wiimote's infrared camera. 29
1.17 Comparison of Measured and Expected Y coordinates for 20mm steps along the Y-axis of the Wiimote's infrared camera. 30

1.18 Comparison of Measured and Expected X coordinates for 20mm steps along the X-axis of the Wiimote's infrared camera. 31
1.19 Comparison of Measured and Expected distances for 20mm steps along the distance-axis of the Wiimote's infrared camera. 31
1.20 Comparison of Measured and Expected Y coordinates for 20mm steps along the Y-axis of the Wiimote's infrared camera. 32
1.21 Description of a Cube in the obj file format 36
1.22 A portion of the obj file format showing four of the vertex data types 37
1.23 Illustration of the Hierarchy of 3ds Chunk Types 38
1.24 Example of a ply file header 39

List of Tables

1.1 Wii Bluetooth HID Descriptor Block 10
1.2 Wiimote Button Bit Assignment 11
1.3 Results of the Wiimote Pendulum Test for the YZ plane 15
1.4 Results of Wiimote Pendulum Test for the XZ plane 16
1.5 Results of the Wiimote and Spring Balance system test for the X-axis 20
1.6 Results of the Wiimote and Spring Balance system test for the Y-axis 21
1.7 Results of the Wiimote and Spring Balance system test for the Z-axis 21
1.8 Results of the first test setup showing the change in x coordinates and distance corresponding to the change in angle applied to the infrared goggles. The results tabulated here were taken at a distance of 300 mm. 25
1.9 Change in x coordinates and distance corresponding to the change in angle applied to the infrared goggles at a distance of 1200 mm 25

Chapter 1

Introduction

The purpose of this report is to investigate the various capabilities of the Nintendo Wii and in particular its application in performing headtracking for the purpose of viewing 3D terrain maps. In this chapter a brief background to the topic at hand is provided within which the various aspects relevant to this project are indentified. This is followed by the setting out of the project objectives and the scope of work to be covered in this research report. Finally the structure of the remainder of the thesis is outlined.

1.1 Background

Head tracking has long been an area of interest within the research community. Many research groups who have sought to address this particular challenge have found the conception of a suitable stable and effective method of implementing head tracking highly illusive. In particular systems that have been developed in the past tend to suffer from inadequacies in resolution, accuracy, update rate and lag. Today there are numerous methods in existence used to perform head tracking most of which are designed for very specific tasks and applications and usually tend to trade off certain performance attributes in order to achieve the desired performance required by their respective functions/industries. While some implementations are reasonably effective, they are highly inaccessible and expensive. Thus there is still substantial room for improvement and development in this field.

The Wii is a gaming console developed by Nintendo which is used in conjunction with a wireless controller called the "Wiimote" in order to enable its users to control objects on a screen via the motion of the controller for the purpose of creating an intuitive and immersive game-play experience. The capabilities of the wiimote however go beyond the gaming realm. Its sensing capabilities along with its ability to communicate data wirelessly have opened up room for many other fields of application. Recently the Wiimote has been documented being used to achieve head tracking. The head tracking capabilities of the Wii are typically realized by using the Wii in reverse ie. by placing a source of infrared light (usually a sensor bar consisting of infra-red LED's) on the user's head and

using the Wiimote's built-in infrared camera to detect the position of the light source and thus of the user's head. The Wii remote in combination with its sensor bar are able to provide measurements of distance, height, latitude and tilt relative to each other. These parameters can be retreived and fed into software which produces an image on the screen corresponding to what a user is expected to see ie. the image on a screen can then be modified in order to mimic the change in view that one experiences in reality when looking through a window from different perspectives. Thus the user experiences the illusion of viewing objects in 3d.

1.2 Problem Specification

This research report examines the performance and limitations of the Nintendo Wiimote with particular regard to its headtracking capabilities. This evaluation includes the assessment of the api's and libraries currently available for the purpose of developing various applications which interact with the Wiimote. It then further investigates the possibility of harnessing the headtracking capabilities of the Nintendo Wiimote in order to view three dimensional terrain maps interactively.

1.3 Objectives

The objectives of this project are as follows:

- To outline the theory of head tracking and provide some examples of the various configurations and methods developed thus far in order to achieve this function.
- To replicate the hardware/software used to perform headtracking with the Wiimote and carry out tests and simulations on this technology.
- To detail the theory of operation and limitations of the Nintendo Wiimote in its head tracking capacity.
- To investigate possible methods of extending the above mentioned functionality of the Wiimote in order to facilitate the viewing of 3d terrain maps interactively.

1.4 Scope and Limitations

This report addresses the remote sensing and position tracking capabilities of the Nintendo Wiimote. It deals specifically with determining the accuracy, sensitivity and limititations of this device and its ability to reliably provide the measurements required in order to perform head tracking. For the purpose of retrieving the data provided by the Wiimote this project will only make use of Wiimote

api's that have already been released. Due to the amount of time available for this project it will not be viable to develop a new one from scratch. Subsequently, any error introduced by the software or code used to retreive the data and measurements from the wiimote will merely be noted, but not corrected. This project will be making use of the C++ and C# Wiimote api's as well as a few scripts which make use of the Python basedGlovePIE library. Api's written in other programming languages which potentially might perform more accurately and reliably are not covered. This is again due to the unavailability of the time required to learn other programming languages and their corresponding api's. This thesis does not adress the creation of terrain maps but rather assesses currently existing software and file formats for the purpose of determining which would be most convenient/suitable for implementation within the application of 3d terrain map viewing with the Wiimote. The software used and reviewed in this thesis is limited to open source software, this choice averted the time consuming and expensive exercise of obtaining licences for some of the other available software packages.

1.5 Plan of Development

The remainder of this report is presented as follows:

Chapter 2 begins by introducing the topic of head-tracking and outlining the problem at hand. The most commonly used technologies and techniques in previous work done in this field are reviewed and their merits discussed. Examples of some of the systems which have been developed in the past are then given and the implementation of some of these configurations is also briefly detailed.

Chapter 3 outlines the key aspects of the operation of the Nintendo Wiimote. Included in this is discussion is a breakdown of the Wiimote in terms of its its components and the technologies it makes use of.

Chapter 4 details the method and procedures that were followed in testing and verifying the various aspects of performance of the Wiimote. A summary of the results obtained is presented and then discussed. A basic outline of the programme used to interface with the Wiimote and retreive the data being transmitted by the Wiimote is also included here.

Chapter 5 reviews the various software aspects pertaining to this thesis including the Wiimote apis that were studied and the various 3d software packages and file formats experimented with in order to determine which were most suitable for the application of viewing 3d terrain maps with the Wiimote.

Chapter 6 is a consolidation of the report and brings together all the aspects discussed prior to it in order to make suggestions as to how an application that allows the viewing of terrains byhead motion can be developed. The merits and and various obstacles that are likely to be encountered in implementing these solutions are also mentioned in this discussion.

Chapter 7 is the final chapter of this research report consisting of the conclusions made based on the material presented and recommendations of future work that would be of interest to this topic.

Chapter 2 Previous Work

Head tracking has been an area of technical interest for a very long time and even today it remains a feat that is highly sought after. The ability to effectively track the position and orientation of a user's head has implications extending from virtual reality gaming, to flight simulation and pilot training, and even into the medical faculty. As such it is understandable that many an attempt has previously been made in order to achieve this function. Some of the most popular technological systems employed in this endeavor include mechanical, magnetic, ultrasonic and optical systems[28].

2.1 Mechanical Head Tracking Systems

Mechanical head tracking configurations make use of mechanical linkages in order to measure the position and orientation of a head mounted unit or the direction of gaze of the user wearing the unit[28]. While the accuracy, resolution and frequency response provided by these types of setup is reasonably good, the mechanical linkages place a restriction on the movement that can be made by the head and greatly limit the range over which operation is possible[28, 31]. Mechanical setups do however have the added benefit of not being sensitive to environmental disturbances which would otherwise distort signals or obstruct line of sight as is the case with magnetic and optical systems[31].

2.2 Magnetic Head Tracking Systems

Magnetic motion detectors typically make use of orthogonally arranged coils to produce and transmit electric fields that interrogate sensors/receivers ware hich consist of a similar setup. These electric fields are then isolated in order to obtain data regarding the [x, y, z] translational coordinates as well as pitch, roll, and yaw [y,p,r] rotational co-ordinates which collectively provide measurement with six degrees of freedom (6 dof) of movement[17]. Magnetic motion detectors are relatively small and cheap making them very popular for this type of application[28, 31]. In fact they are currently the most widely used motion tracking devices with Polhemus, a manufacturer of magnetic motion detection devices, claiming 70% of the motion detection market[17]. The main shortfalls of magnetic motion detectors are that they are susceptible to interference from ferrous materials,

radio signals and various other devices which emit/distort electromagnetic waves[31]. The accuracy experienced with magnetic systems is also adversely affected by increasing distance[28, 17].

2.3 Ultrasonic Head Tracking Systems

An ultrasonic configuration would typically consist of multiple ultrasonic transmitters attached to the head and the transmitted signals being received by receiver's placed on the frame of a screen. The coordinates and orientation of the head are then determined by the triangulation of the Time of Flight (tof) measurements. Tof measurements are simply measures of the time taken for are a particle, or in this case sound, to travel over a known distance. Variations in the speed of sound due to changes in air density as well as reflections of the sound waves are the main sources of error in most ultrasonic set ups[28].

2.4 Optical Head Tracking Systems

There is also an abundance of optical systems used to perform head tracking. These can even be broken down further according the source of light they make use of. Sources that have been used include visible light, laser light and infra-red[28]. Optical head tracking configurations also suffer from limited range of operability with the exception of laser light systems which have the ability to work over greater ranges. While optical head tracking systems are not susceptible to electromagnetic interference, they do require that line of sight be maintained between the receiving and transmitting parties. In addition they are also subject to error introduced by the presence of other light sources and reflections. In one particular optical configuration a camera is attached to a head band and views eight 3d reference points placed on a computer monitor. In order to track the user's head, an image processing board is then used to continuously calculate the camera's pose with respect to the eight reference points on the monitor[31]. Another configuration makes use of a ceiling which has been tiled with led's. The user then wears a helmet with photo-sensors/cameras attached to it. The images created by the led's seen by the sensors are used in conjunction with the known led positions to locate and track the position of the wearer's head. This is also known as a cellular head tracking system[28]. Many other variations of this theme and technology exist. Optical head tracking systems have enjoyed success particularly in their use in aircraft cockpits[28].

Chapter 3 Wiimote Theory of Operation

3.1 Breakdown of the Wiimote and its Components

The Nintendo Wiimote is a game controller or joystick designed to be used with the Nintendo Wii game console. This joystick is unique not only because of its unusual design, but also because it is connected wirelessly to the host console. As seen in figure 1, the Wiimote consists of a power button in the top left corner below which is a typical directional pad made up of four buttons arranged in a cross configuration. There is an ”A" button situated below the directional pad and a “B" button directly underneath on the reverse side of the controller. In addition there is a “home” button with “+" and “-" buttons on either side of it, a small speaker below which are the “1" and “2" buttons and four leds. The Wiimote has dimensions 148mm x 36.2mm x 30.8mm and is powered by 2 “AA” batteries. Internally the wiimote consists of an infrared camera, a Broadcom BCM2042 Advanced Wireless Keyboard/Mouse Bluetooth Chip, a ADXL330 3-axis iMEMS accelerometer integrated circuit, 128kbit I2C serial EEPROM chip, a MSOP-8pin Mobile Phone Speaker Driver and a small motor with an unbalanced weight attached to it creating the “rumble" vibrating effect of the controller[18, 19, 20].

Figure 1.1: Photograph of the back and front views of the Nintendo Wiimote

The BCM2042 is a single-chip bluetooth device complete with an on-board 8051 processor and

RAM/ROM memory as well as full Bluetooth stack and Human Interface Device (HID) profile[18, 19, 22]. The USB Human Interface Device class defines the features available and the procedures to be followed in order to facillitate communication between HIDs. These would typically be keyboards, mouses, pointing devices, joysticks and other gaming devices, and any other devices used by humans to interact with a computer. The USB HID device class definition essentially enables all hardware, which makes use of the USB HID class, to be operable under any software. The Bluetooth HID profile simply extends the USB HID class to enable a device with Bluetooth wireless communications to support USB HID services over the Bluetooth protocol stack[19, 21].

The ADXL330 is small lower power 3-axis accelerometer IC with a full-scale range of ±3 g. The user has the ability to set the bandwidth of the accelerometer by adjusting the capacitors at the Xout, Yout, and Zout pins (shown in Figure 1.2). The range that can be achieved for the X and Y axes is 0.5 Hz to 1600 Hz, while for the Z axis it is 0.5 Hz to 550 Hz[19, 1, 25].

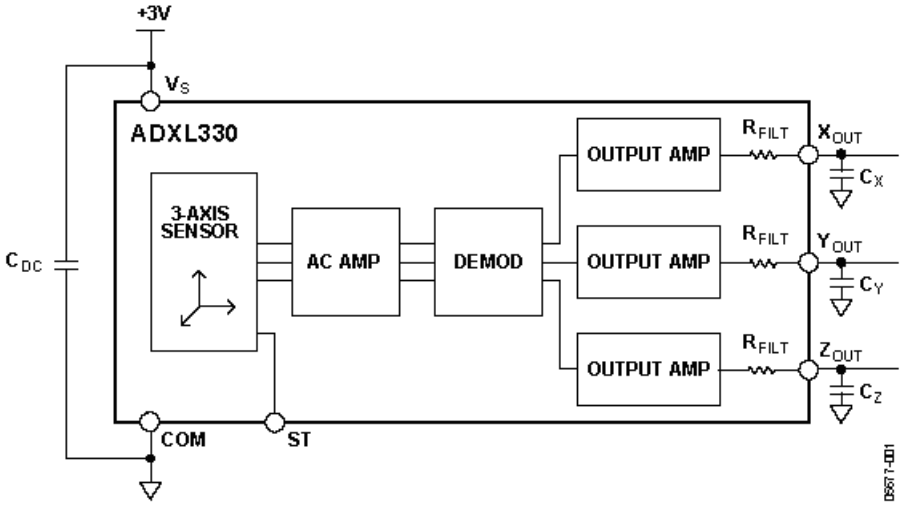

Figure 1.2: Schematic of the ADXL330 Accelerometer IC

3.2 Connecting and Communicating with the Wiimote

In order to connect to the wiimote, one has put it into “discoverable” mode and then interrogate it with the host Bluetooth HID driver. The Wiimote can be placed in “discoverable” mode by either pressing both the “1” and “2” buttons simultaneously or by pressing the small red sync button on the inside of the battery compartment. When in “discoverable” mode, the leds will blink for twenty seconds during which time the HID driver on the host should connect to the wiimote. If this period elapses without a connection having been established, the Wiimote will turn itself off again.

Data describing the Wiimote in terms of its mode of operation, state of its components and other related information is encapsulated in a HID descriptor block. This block consists of a catalog of

reports which can be sent at a frequency of up to 100 reports per second. In order to retreive the HID block of the Wiimote, the host must interrogate it with the Bluetooth Service Discovery Protocol (SDP)[18]. The function of SDP is to determine which services are available (ie. what services other Bluetooth devices are offering) and what the characteristics of these available services are. This is particularly important in the Bluetooth environment due to the dynamic nature of the set of available services. Service availability in the Bluetooth environment is largely determined and affected by RF proximity of devices in motion. SDP implements a request/response model that entails the exchange of one request protocol data unit (PDU) for every one response PDU. In the Bluetooth context SDP is usually implented over the L2CAP transport protocol[23].

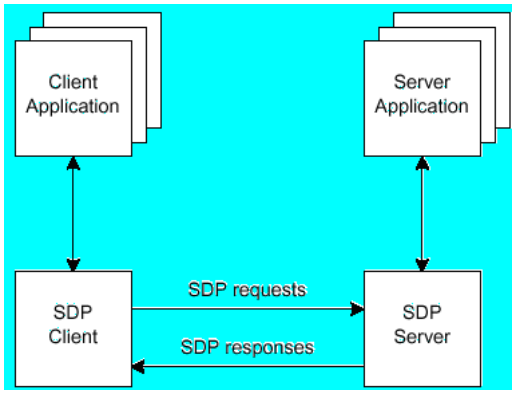

Figure 1.3: Flow chart showing the flow control provided by SDP

The reports residing in the HID descriptor block are similar in function to network ports. Each report is dedicated to a specific service, reports however are unidirectional. For each report the HID descriptor lists the direction and payload size of the report. Table 1.1 shows a human readable version of a typical Wii HID descriptor block. The output refers to packets sent from the host to the Wiimote, while the input refers to packets sent from the Wiimote to the host.

Table 1.1: Wii Bluetooth HID Descriptor Block

Output

Report ID	Payload Size	Known Functions
0x11	1	Player LEDs, Rumble
0x12	2	Report type / ID
0x13	1	IR Sensor Enable
0x14	1	Enable speaker
0x15	1	Controller status
0x16	21	Write data
0x17	6	Read data
0x18	21	Speaker data
0x19	1	Mute speaker
0x1a	1	IR Sensor Enable 2

Input

Report ID	Payload Size	Known Functions
0x20	6	Expansion Port
0x21	21	Read data
0x22	4	Write data
0x30	2	Buttons only
0x31	5	Buttons | Motion Sensing Report
0x32	16	Buttons | Expansion Port | IR??
0x33	17	Buttons | Motion Sensing Report
0x34	21	Buttons | Expansion Port | IR??
0x35	21	Buttons | Motion Sensing Report | Expansion Port
0x36	21	Buttons | Expansion Port | IR??
0x37	21	Buttons | Motion Sensing Report | Expansion Port
0x38	21	Buttons | Motion Sensing Report | IR
0x3d	21	Buttons | Expansion Port | IR??
0x3e	21	Buttons | Motion Sensing Report | IR??
0x3f	21	Buttons | Motion Sensing Report | IR??

Button presses on the Wiimote automatically generate an input report comprising of a 2-byte bitmask with the current state of all the buttons. The bit assignments for the buttons, arranged in big endian order, are shown in Table 1.2.

Table 1.2: Wiimote Button Bit Assignment

Button	Number (dec)	Value (hex)
Two	1	0x0001
One	2	0x0002
B	3	0x0004
A	4	0x0008
Minus	5	0x0010
?? Unknown (motion reports) ??; or Z Acceleration, bit 6 (report 3E) or bit 2 (report 3F)	6	0x0020
X Acceleration LSB (motion reports); or Z Acceleration, bit 7 (report 3E) or bit 3 (report 3F)	7	0x0040
Home	8	0x0080
Left	9	0x0100
Right	10	0x0200
Down	11	0x0400
Up	12	0x0800
Plus	13	0x1000
Y Acceleration LSB (motion reports); or Z Acceleration, bit 4 (report 3E) or bit 0 (report 3F)	14	0x2000
Z Acceleration LSB (motion reports); or Z Acceleration, bit 5 (report 3E) or bit 1 (report 3F)	15	0x4000
? Unknown ?	16	0x8000

3.3 Motion Sensing

The ability of the Wiimote to detect any motion which it undergoes is provided by the the ADXL330 3-axis linear accelerometer IC present in the architecture of the Wiimote. The ADXL330 is located slightly to the left of the “A" button and has a cordinate system as shown in figure 4. The Wiimote is able to detect motion in the 3 linear translation directions (X, Y, Z) and the 3 rotation angles (pitch, roll, yaw)[18]. When at rest the Wiimote experiences a force of 1G perpendicular to the surface upon which it rests as a result of the force that holds it up against gravity. Thus when properly calibrated the Wiimote should read zero on two of its axes and one on the other while at rest. All axes should ideally read zero when the Wiimote is in freefall.

The ADXL330 IC is made up of micro-mechanical structure or mass which rests on silicone springs. When the Wiimote experiences an acceleration the mass is forced to exert a force on its supporting springs. The subsequent displacement of the spring system causes a change in capacitance which in turn is measured as a proportional voltage. This signal is then digitized to an 8 bit unsigned integer and made available in the Wiimote’s status report[18, 1]. In order to access the Wiimote’s motion sensing data the host must send a “SET_REPORT” request to channel 0x12. This would typically look as follows:*(52) 12 00 31*

The hexadecimal number in parenthesis (ie. the first byte) is the report or channel ID. The second

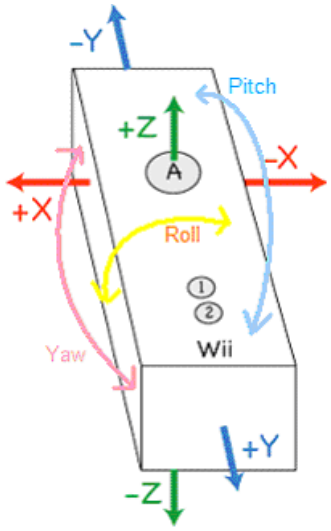

Figure 1.4: Coordinate System of the Wiimote's internal accelerometer

byte is the channel to which the request is going. The third byte specifies the channel mode. The bitmask x00 means that output is only generated when a button is pressed or released. The fourth byte specifies the channel to which the sensor output data must be sent. Upon receiving this request, the Wiimote responds by sending a stream of data input packets containing the accelerometer's X,Y and Z readings to the specified channel, in this case x31.

This would typically look as follows: *(a1) 31 40 20 86 8a a5*

In this stream bytes 3 and 4 are button values while x86, x8a and xa5 are the values measured on the X,Y and Z axes respectively. All the packet conventions can be found in the Bluetooth HID specification. For the purpose of this thesis it is worth noting some of the channel modes which are of particular interest. Mode 0x30 requests data regarding the state of only the buttons. Mode 0x31 requests motion sensor data. Mode 0x32 requests IR camera data, while mode 0x33 requests both IR camera and motion sensor data[18].

Chapter 4 Testing and Verification of Performance of the Wiimote

In order to assess the suitability of the wiimote for various applications such as motion detection and head tracking it is neccessary to conduct testing of the device. The testing performed on the wiimote is inteded to determine the limitations and boundaries of the performance of the wiimote. The main aspects of performance to be investigated in the wiimote are its infra-red position tracking capabilities, acceleration and tilt measurement.

4.1 The Pendulum Test

4.1.1 Aim

The aim of this test is to observe the output of the Wiimote's acceleration sensor in order to determine what this output represents, how accurate it is and how sensitive this is.

4.1.2 Apparatus and Setup

The setup consists of two wooden beams supporting a flat wooden board from which the wiimote is suspended by means of a wire. The wire is bent into in a "U" shape and the free ends are attached to the sides of the Wiimote. The bent segment of the wire is flattened and placed in a groove in the flat wooden board. The groove is then covered with a smal piece of paper and sealed with masking tape, creating a "hinge-like" mechanism which allows the Wiimote to swing freely while restricting its movement to a single plane.

Interaction with the Wiimote's sensors is facillitated by G-Force Logger, a Wiimote application written in C# which retrieves, displays and records/stores the current state of the wiimote in a text box. The user has the option of specifying the interval between Wiimote-state-data retrieval in milliseconds. The interval specified by the user is neccessarily required to be an integer greater than or equal to one. In order to begin the data retrieval, the "B" button on the Wiimote must be pressed. Once the user is satisfied with the data recorded, the "B" button can then be pressed again to stop the

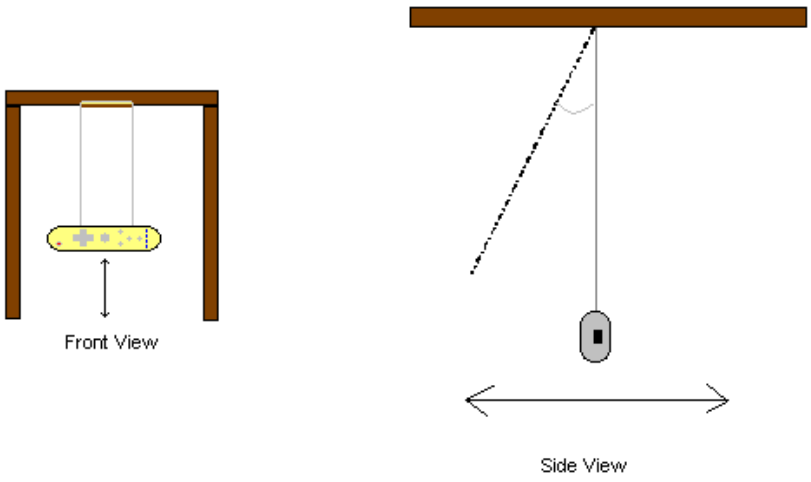

Figure 1.5: Wiimote Pendulum System

recording. For the purpose of analysing the data obtained, the user then has the option of copying the logs recorded to the “clipboard” from which it can be pasted into a data analysis package of the user’s choice.

4.1.3 Test Procedure

The test commenced by connecting the Wiimote to the computer (via Bluetooth) and then running the G-Force Logger application. While the Wiimote was at its rest position, it was verified that the Wiimote was correctly calibrated. This was done by observing the acceleration and angles measured on each axis, which are displayed in text labels on the application’s Graphic User Interface. The system was adjusted until these values reflected what was expected for the particular orientation of the Wiimote. The Wiimote was then displaced by a certain angle from its rest position. Once the value read by the G-Force Logger corresponded to the desired angle, the “B" button was pressed and the Wiimote released. The resulting oscillation of the pendulum system was then observed until it returned to rest. The “B" button was then pressed again in order to stop the recording of data and the recorded data was copied and pasted into Microsoft Excel.

This procedure was repeated for five different angles of displacement. After a set of angles had been observed, the orientation of the Wiimote was changed in order to vary the plane of oscillation and the entire test repeated. The planes in which pendulum motion was observed were the YZ and XZ planes.

4.1.4 Calculations and Results

The following is a summary of the calculations made and results obtained from the Wiimote pendulum test conducted.

Table 1.3: Results of the Wiimote Pendulum Test for the YZ plane

Angle (degrees)	Accel. Y-axis (G)	Accel. Z-axis (G)	Res. Accel. (G)	Res. Accel. (m/s^2)	Calc. Tang. Accel.
10	0.152	0.16	0.2207	2.1650	1.7035
16	0.23	0.2	0.3048	2.9900	2.7040
23	0.23	0.4	0.4614	4.5264	3.8331
40	0.384	0.64	0.7464	7.3218	6.3057
60	0.576	0.88	1.0517	10.3177	8.4957

In Table 3, Accel Y and Accel Z are the actual accelerations, in G, measured from the tests. Because Accel Y and Accel Z represent the acceleration of the Wiimote along the perpendicular Y and Z axes respectively they can be considered as the components of the resultant acceleration experienced by the Wiimote. This resultant acceleration, Res. Accel, is calculated according to equation 1.1

$$\sqrt{(AccelY)^2 + (AccelZ)^2} \tag{1.1}$$

At the point of release this can be considered to be the instantaneous tangentional acceleration experienced by the Wiimote. Calc. Tang Accel. is the calculated and expected tangentional acceleration obtained by the formula:

$$a_T = \theta'' \times r$$

Where θ'' is the angular acceleration which the Wiimote pendulum undergoes and r is the radius of the arc of the rotational motion. The angular acceleration is given by equation1.2,

$$\theta'' = \frac{G}{r} sin\theta \tag{1.2}$$

where G is the gravitational constant, and θ is the angle of displacement from the vertical/rest position.

The results of the tests conducted correspond to $G = 9.81 ms^{-2}$ and $r = 0.298 mm$.

Figure 1.6 shows the comparison between the calculated and measured accelerations for the relative angles of displacement.

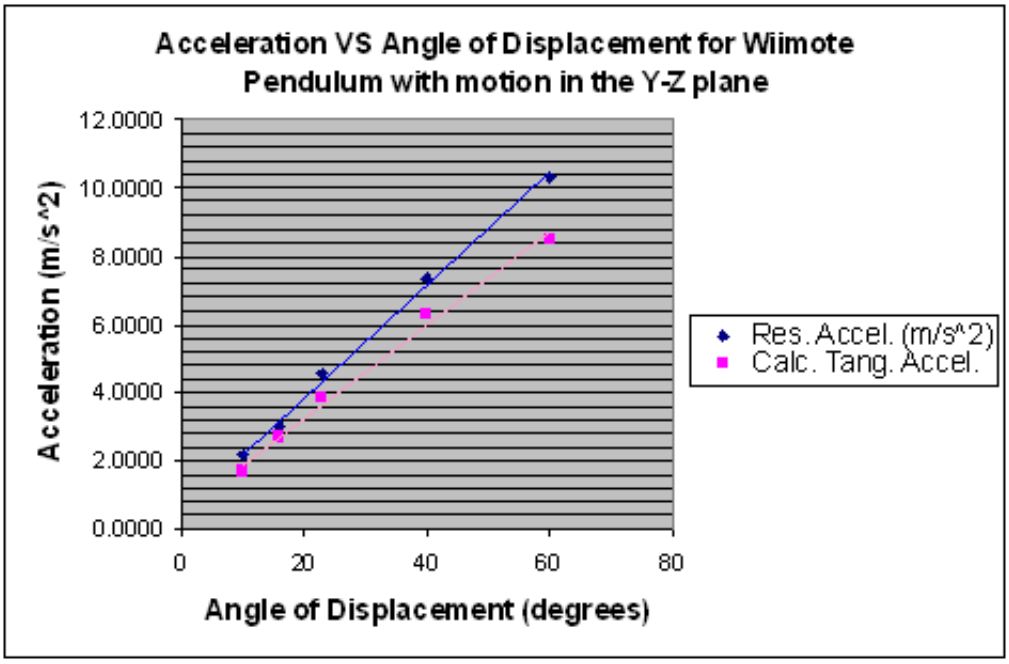

Figure 1.6: Comparison of measured acceleration against calculated acceleration for Y-Z plane

The following is the set of results obtained for the X-Z plane.

Table 1.4: Results of Wiimote Pendulum Test for the XZ plane

Angle (degrees)	Accel. Y-axis (G)	Accel. Z-axis (G)	Resultant Accel. (G)	Res.Accel. (m/s^2)	Calc. Tang. Accel
9	0.154	0.16	0.2221	2.1785	1.5346
16	0.269	0.08	0.2806	2.7531	2.7040
30	0.462	0.28	0.5402	5.2996	4.9050
36	0.577	0.24	0.6249	6.1305	5.7662
41	0.654	0.28	0.7114	6.9790	6.4359

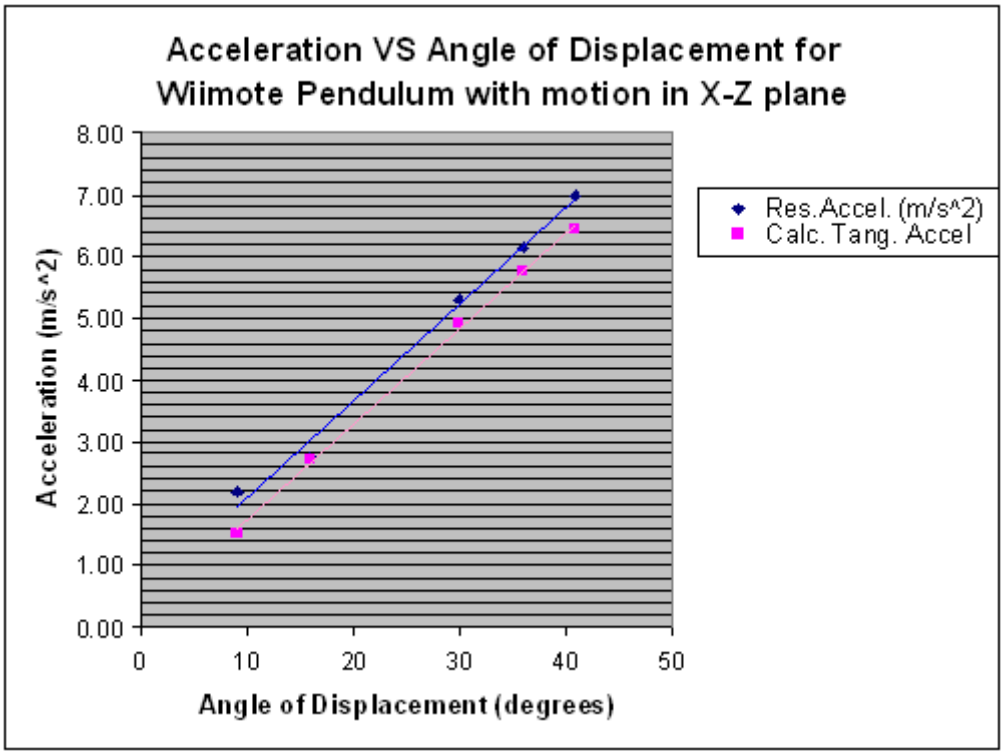

Figure 1.7: Comparison of measured acceleration against calculated acceleration for X-Z plane

4.1.5 Discussion and Analysis of Results

Analysis of the plotted graphs shows that while the acceleration measured by the Wiimote does not coincide exactly with the calculated acceleration, it does follow the same trend. This is particularly evident in the case of the Wiimote pendulum motion in the X-Z plane where the trend lines of the measured and expected acceleration have almost the same gradient. This suggests that the results are being offset by a small amount of error that may be the result of excessive friction in the hinge mechanism or inaccuracies in measurement of angle of displacement. It must also be taken into account that pendulum experiments can only provide reliable results for small angles of displacement, typically less than 15 degrees. The divergence of the trendlines of the measured and expected accelerations seen in Figure 1.6 beyond the 20 degree point may bear testament to this fact. From the raw data recorded during the oscillations of the Wiimote-pendulum it was also noted that no readings below 0.04G were registered despite the ongoing decline of motion prior to the pendulum's return to rest. This suggests that the Wiimote is incapable of registering acceleration measurements of a smaller magnitude than this. The Wiimote's ability to provide acceleration measurements is greatly limited by the fact that a maximum of only one sample is provided every 0.01 seconds. A substantial amount of motion data is subsequently lost during this interval. Possible ways in which this experiment could be improved are to construct a more stable and restrictive structure to ensure oscillation of the pendulum in a single plane. This particularly important as the Wiimote has an irregular shape

and a non-uniform mass distribution which causes it to have a natural tendency of swinging in other directions.

4.2 The Spring Test

4.2.1 Aim

The purpose of this test is to determine the limits of operation of the Wiimote's 3 axis accelerometer in terms of its sensitivity and maximum G-force that can be measured.

4.2.2 Apparatus and Setup

The apparatus for this experiment includes a spring balance, retort stand, masking tape, string and the Wiimote. Once again this experiment makes use of the G-Force Logger application in order to record the data being measured and transmitted by the Wiimote.

The spring balance was suspended from the retort stand. The ends of a piece of string were attached to either side of the Wiimote. This enabled the Wiimote to be hung from the hook of the spring balance.

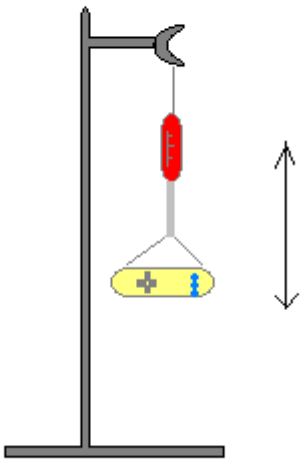

Figure 1.8: Wiimote and Spring Balance Test System

4.2.3 Test Procedure

The first step of the procedure followed was to determine the spring constant of the spring balance. This was done by placing a mass on the hook of the spring balance and observing the displacement from rest position. Knowing the force being exerted on the spring (ie. $mass \times gravity$) and the

subsequent displacement, the spring's constant was determined using equation 1.3

$$F = -kx \tag{1.3}$$

Once the spring constant had been obtained and the configuration as discussed above had been setup, The spring in the system with the Wiimote attached was extended by a certain distance. Once satisfied with the displacement, the “"B" button on the Wiimote was pressed to begin recording Wiimote status data and the Wiimote was then released. The subsequent oscillation of the spring system was then observed and the Wiimote's state data recorded from before the time of release to the point where the oscillations were no longer detectable. This was done a minimum of six times for a particular displacement before continuing to the next value of displacement. The six readings obtained for each value of displacement were averaged to obtain an acceleration corresponding to that specific displacement. This procedure was repeated for six different values of displacement for each of the X, Y and Z axes.

4.2.4 Calculations and Results

The following is a summary of the results and calculations involved in this experiment.

The spring constant, k, was determined to be 39.07 N/m.

The mass of the Wiimote including batteries was found to be 0.1354g.

The expected accelerations were calculated according to equation 1.4

$$a = \frac{kx}{m} \tag{1.4}$$

Tables 1.5, 1.6, and 1.7 show the results of the Spring test for the X, Y and Z axes respectively.

Table 1.5: Results of the Wiimote and Spring Balance system test for the X-axis

Displacement (mm)	Measured Accel (G)	Measured Accel (m/s^2)	Calc. Accel. (m/s^2)
1	0.04	0.3924	0.2886
4	0.078	0.76518	1.1542
9	0.232	2.27592	2.5970
14	0.367	3.60027	4.0397
20	0.505	4.95405	5.7710
25	0.587	5.75847	7.2138

Table 1.6: Results of the Wiimote and Spring Balance system test for the Y-axis

Displacement (mm)	Accel. Measured (G)	Accel Measured (m/s^2)	Accel Calc (m/s^2)
1	0.04	0.3924	0.2886
4	0.08	0.7848	1.1542
9	0.19	1.8639	2.5970
14	0.31	3.0411	4.0397
19	0.46	4.5126	5.4825
25	0.54	5.2974	7.2138

Table 1.7: Results of the Wiimote and Spring Balance system test for the Z-axis

Displacement (mm)	Measured Accel (G)	Measured Accel (m/s^2)	Calc. Accel. (m/s^2)
1	0.04	0.3924	0.2886
4	0.08	0.7848	1.1542
9	0.2	1.962	2.5970
14	0.4	3.924	4.0397
20	0.52	5.1012	5.7710
25	0.57	5.5917	7.2138

The following graphs show the comparison between the expected and actual measured acceleration for the various values of spring displacement.

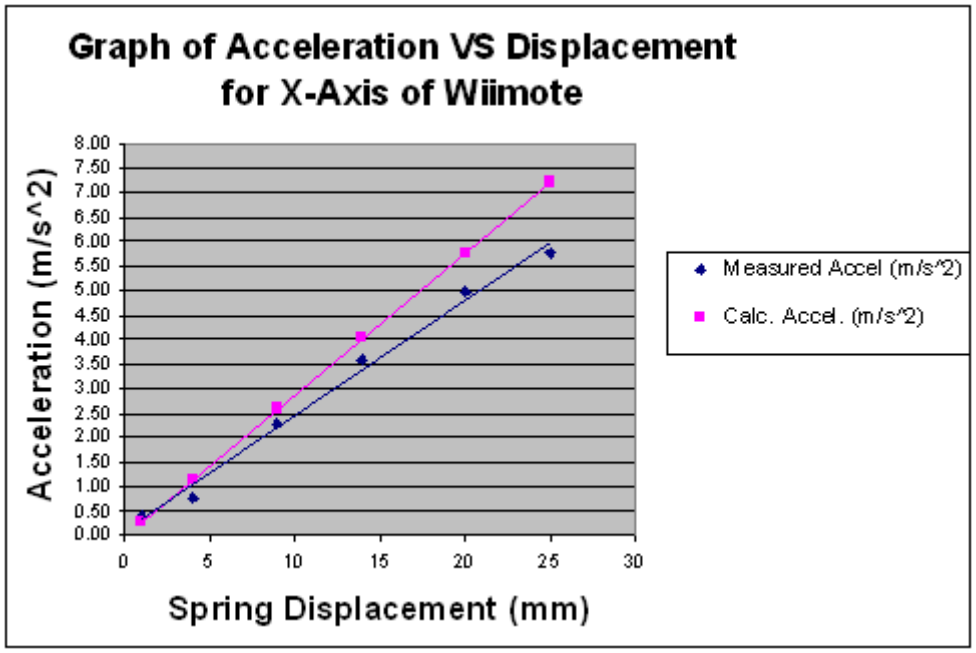

Figure 1.9: Comparison of measured acceleration and calculated acceleration in the Wiimote-Spring system for X-axis

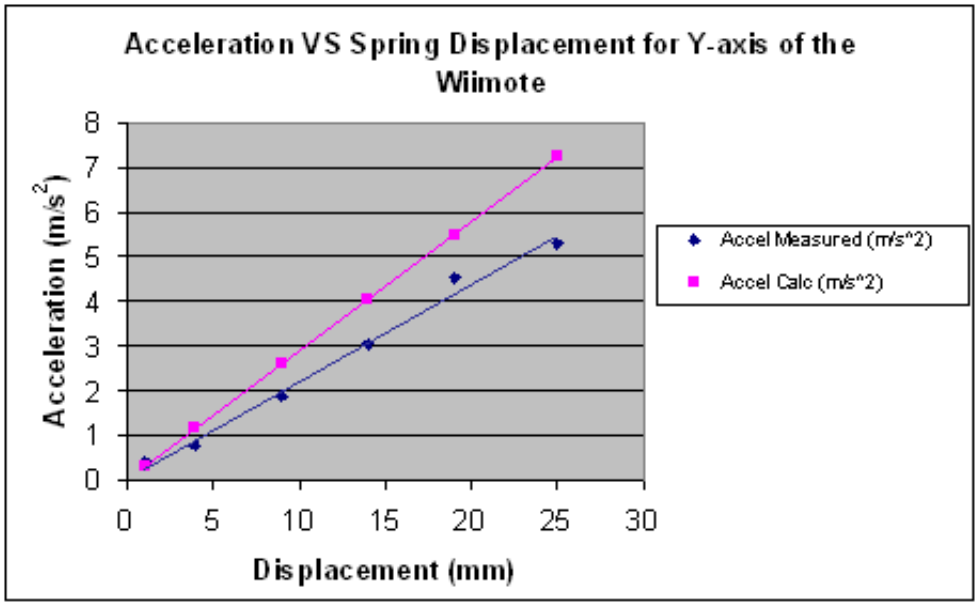

Figure 1.10: Comparison of measured acceleration and calculated acceleration in the Wiimote-Spring system for Y-axis

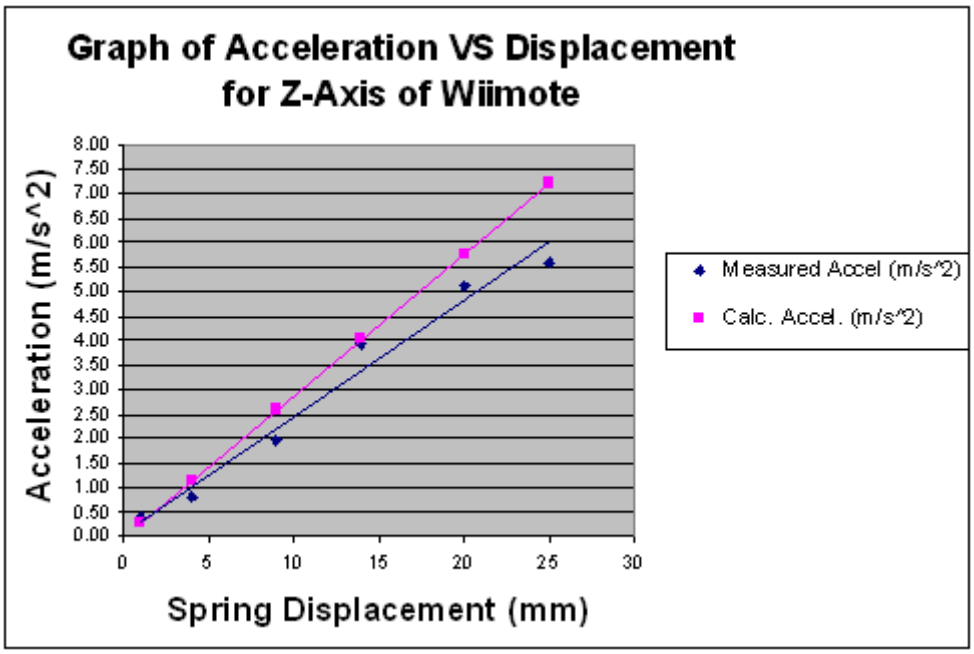

Figure 1.11: Comparison of measured acceleration and calculated acceleration in the Wiimote-Spring system for Z-axis

4.2.5 Discussion and Analysis of Results

From the results summarised above it is confirmed that the smallest acceleration which the Wiimote is able to detect is 0.04G or 0.392m/s^2. In order to verify the claimed +/-3G maximum detectable acceleration of the Wiimote's accelerometer, a stiffer spring was required, however in order to be able to verify the expected accelerations a spring balance was necessary. Unfortunately for this experiment a spring balance with a stiff enough spring constant could not be obtained. In all three plots of the measured and expected accelerations there is a noticable discretancy. It is very likely that this is due to the fact that when a spring is extended or compressed the subsequent force exerted on the object attached to the end of the spring, at the point of release, is not equal to the force generated by the compression of the spring by the relevant displacement. The unbalance caused by this fact together with other miscelleneous errors such as displacement measurement inaccuracies ultimately result in the apparent incongruence of the measured and calculated results. Despite this, one is still able to see that the results are somewhat correlated.

4.3 Wiimote Infrared Detection Tests

4.3.1 Aim

The aim of the tests documented in this section is to is to verify the operation of the Wiimote's infrared tracking capabilities. More specifically this experiment seeks to determine the limits and constraints of the Wiimote-IR system in terms of the viewing angle and the minimum and maximum distances at which operation can be achieved, the effect distance has on accuray and hence the optimal configuration for the purpose of achieving head tracking.

4.3.2 Apparatus and Setup

The apparatus used in this experiment includes a tape measure, protractor, two infrared leds, a pair of sunglasses, 9V battery, masking tape, a stack of paper and the Wiimote Desktop Virtual Reality Program. The Wiimote Desktop Virtual Reality Program is a C# application which was originally written to demonstrate the head tracking capability of the Wiimote. In this experiment it is used to retrieve the infrared measurements provided by the Wiimote. For the purpose of this experiment, a prototype virtual reality set of goggles was constructed in order to create a test configuration which most closely resembled the desired manner in which the devices would be used when viewing terrain maps. The goggles were constructed by soldering the positive leg of each infrared led to a 180ohm resistor and then further soldering these to lengths of wire which were then used to connect the leds in parallel and to the 9V battery. Once the electrical circuit had been completed, the leds were taped with masking tape onto either side of the sunglasses effectively recreating the Wii-sensor-bar as a

wearable device which is more suitable for head tracking. In order to get meaningful data from the software being use to retrieve the data provided by the Wiimote, the Wiimote Desktop Virtual Reality Program had to be reconfigured to match the hardware being use to conduct this experiment. The reconfiguration was done via the config file and the following parameters were altered as follows:

IR dot separation: 147 mm

Screen Height: 200 mm

Camera Position: Above Screen (True)

In this experiment there are three test setups. The first, used to determine the limits of the viewing angle of the Wiimote-IR as well as its sensitivity to small movement, makes use of the infrared goggles which are attached to a rotating protractor facing the infrared camera of the Wiimote. The second setup simply consists of the wiimote and the infrared goggles. This setup is used to determine the range limitations of the Wiimote-IR system. The final configuration is used to investigate the effect that distance has on the accuracy of the x, y and distance measurements provided by the Wiimote. This setup consists of a grid made up of 5mm squares as well as a platform upon which the infrared goggles are placed.

4.3.3 Method

For the first test, the infrared goggles, attached to the rotating protractor, were placed facing the infrared camera at the same height as the Wiimote. The infrared goggles were then slowly rotated to the left starting from their initial position of 90 degrees to the infrared camera until the point was reached where no further readings were obtained for further rotation of the goggles. The goggles were then repositioned to their intial position and slowly rotated to the right once again stopping immediately as soon as no further reading were being registered. Once results had been recorded the goggles were again repositioned at the starting point. The goggles were then rotated through a set of descrete values of angle and the corresponding change in distance and x coordinates was recorded.

The second was conducted by pointing the infrared goggles directly at the Wiimote infrared camera and approaching it until the software stopped rendering readings. Once the distance at which this occurred had been noted, the goggles were moved further and furthe backwards until either the Wiiimote was no longer capable of taking measurements or there was no more room to move within the test environment.

The final test was conducted by choosing three discrete distances at which each of the x, y, and distance coordinates would tested, measured and recorded. The coordinates of each axis were varied in 5 or 6 steps of 20mm each with measurements being taken at each step. The x and dist coordinates were varied by using the a grid made up of 5mm by 5mm cells. A pin was attached to the midpoint of the leds on the goggles and used as a marker to facillitate the changing the position of the goggles on the grid in known steps. The subsequent measurements provided by the Wiimote after each step

were then recorded. The y coordinate of the infrared goggles was varied by placing the goggles on a stack of papers of known thickness. The 20mm steps in the y axis were then obtained by removing the required number of sheets.The measurement obtained from the Wiimote were then plotted and compared to the calculated expected values.

4.3.4 Results and Calculations

The following is a summary of results obtained for the three tests conducted on the Wiimote Infrared Detection system.

The results of the first test were as follows:

The Wiimote was able to track the coordinates of the infrared goggles up to the point where the goggles had been rotated +/- 23 degrees left of their starting position. This result was also the case when the goggles were rotated to the right.

Tables 1.8 and 1.5 show the results obtained when using the same setup to test the sensitivity of the Wiimote to small changes in angle/position.

Table 1.8: Results of the first test setup showing the change in x coordinates and distance corresponding to the change in angle applied to the infrared goggles. The results tabulated here were taken at a distance of 300 mm.

angle (deg)	dist1(mm)	dist2(mm)	x1(mm)	x2(mm)	change in dist (mm)	change in x (mm)
0.5	299.06	299.55	3.44	3.33	0.49	0.11
1	299.06	299.53	1.83	1.49	0.47	0.34
2	299.06	300.55	1.83	0.807	1.49	1.023
3	299.53	301.047	1.723	0.693	1.517	1.03
5	300.03	302.59	2.76	0.348	2.56	2.412

Table 1.9: Change in x coordinates and distance corresponding to the change in angle applied to the infrared goggles at a distance of 1200 mm

angle (deg)	dist1(mm)	dist2(mm)	x1(mm)	x2(mm)	change in dist (mm)	change in x (mm)
0.5	1218.88	1226.71	92.93	93.995	7.83	-1.065
1	1226.71	1226.71	93.995	94.93	0	-0.935
2	1203.5	1218.88	94.52	94.93	15.38	-0.41
3	1203.5	1226.71	94.52	93.86	23.21	0.66
5	1211.32	1203	92.82	88.06	-8.32	4.76
10	1210.9	1203	95.56	88.07	-7.9	7.49

The results obtained for the second Wiimote-IR system test are as follows:

Distance at which the detection of motion by the Wiimote ceases due to the proximity of the infrared goggles: +/- 200 mm

Distance beyond which no further movement is detected: unknown, the infrared goggles were moved up to a distance of 4m away from the Wiimote before running out of space. The Wiimote was still able to detect some motion at this distance.

The following graphs represent the results of the third test conducted on Wiimote-IR system. The results show plots of the measured distance and x, y coordinates of the infrared goggles for 20 mm step changes along each axis in turn. The results are grouped according to the distance at which they were obtained. The first set of plot was taken at 3140.94 mm.

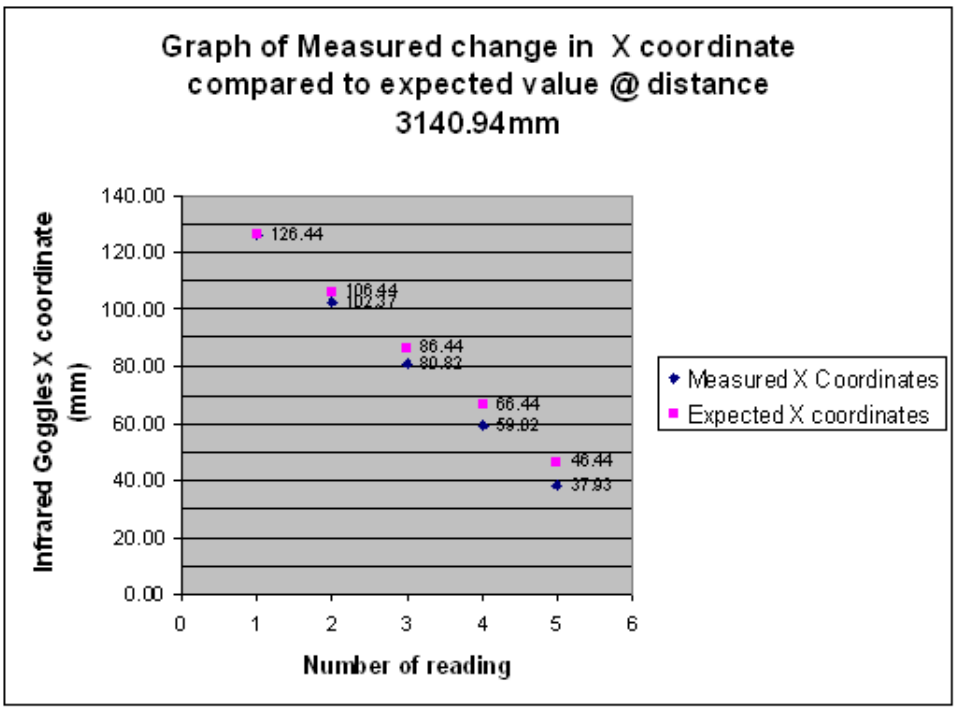

Figure 1.12: Comparison of Measured and Expected X coordinates for 20mm steps along the X-axis of the Wiimote's infrared camera.

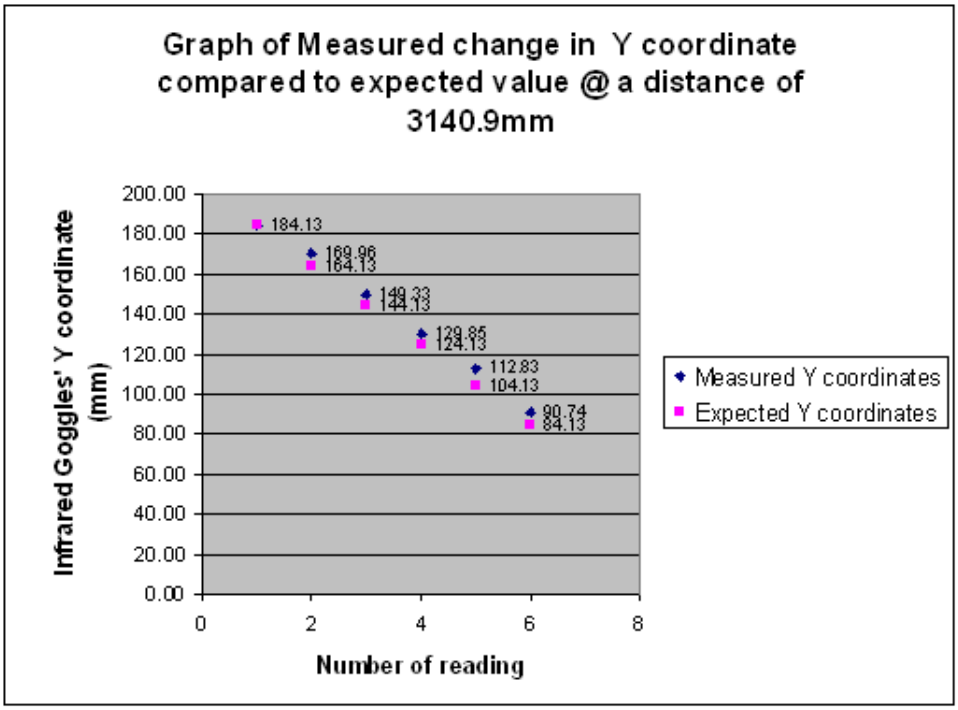

Figure 1.13: Comparison of Measured and Expected Y coordinates for 20mm steps along the Y-axis of the Wiimote's infrared camera.

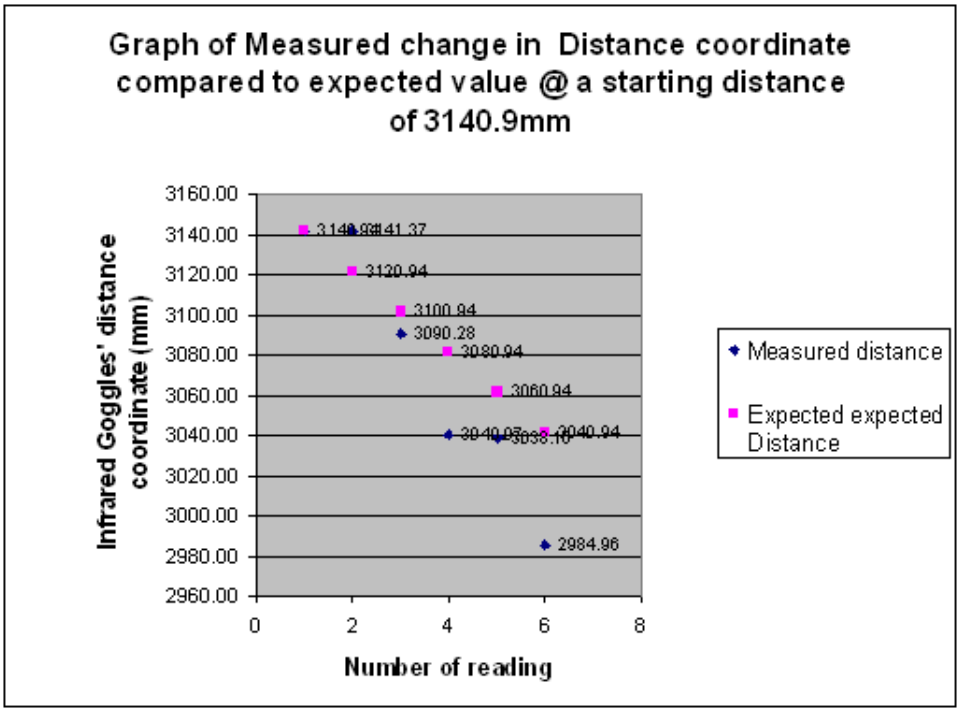

Figure 1.14: Comparison of Measured and Expected distance for 20mm steps along the distance-axis of the Wiimote's infrared camera.

The following set of plots represents the results obtained at a distance of 1250 mm

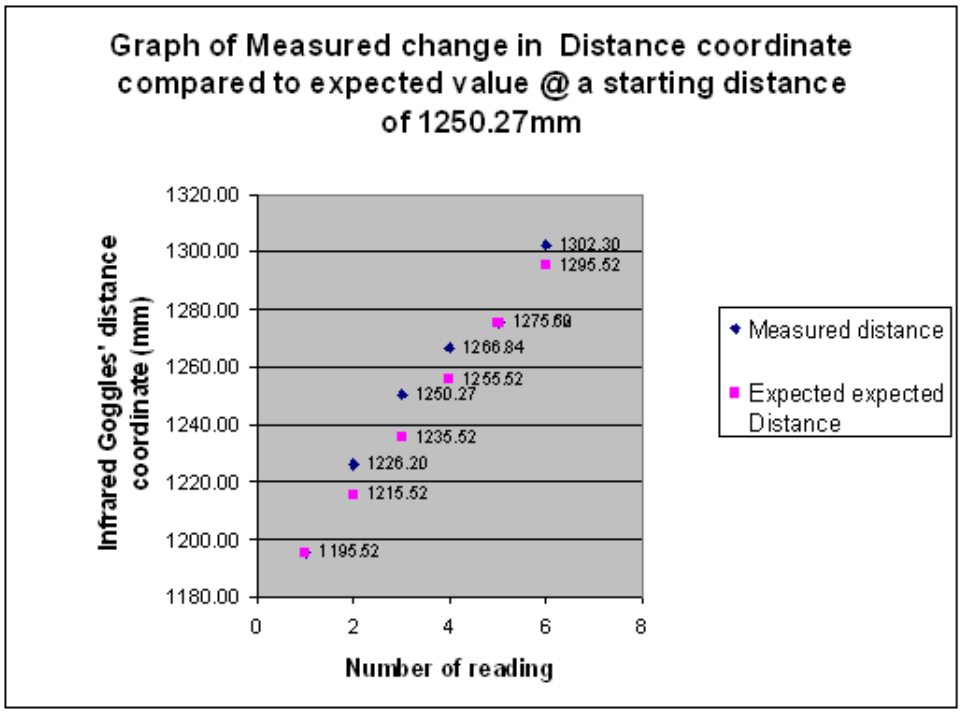

Figure 1.15: Comparison of Measured and Expected distance for 20mm steps along the distance-axis of the Wiimote's infrared camera.

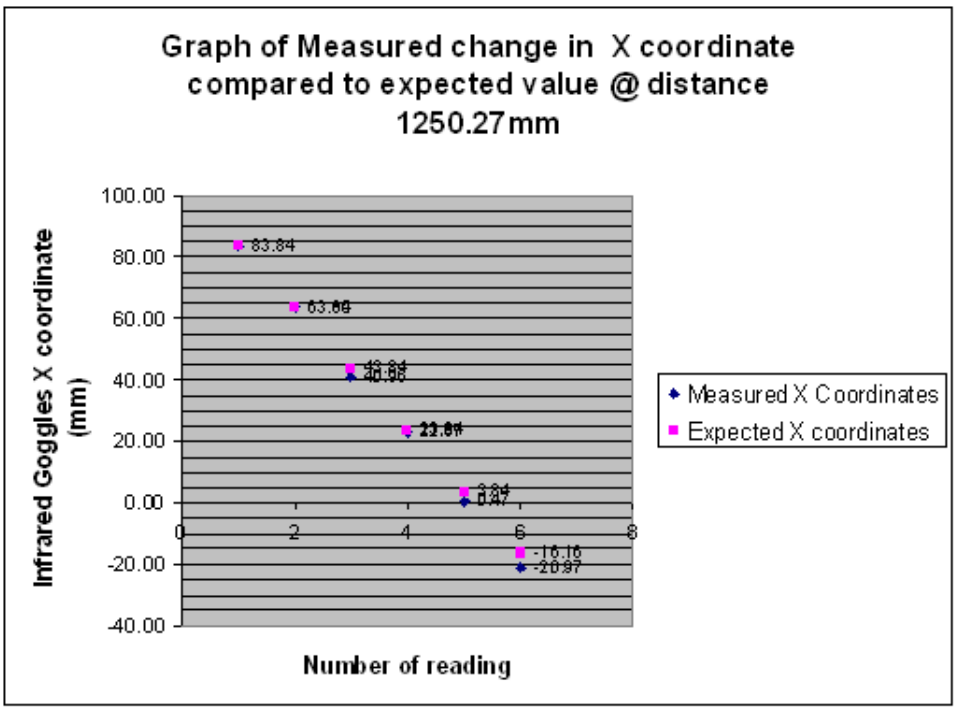

Figure 1.16: Comparison of Measured and Expected X coordinates for 20mm steps along the X-axis of the Wiimote's infrared camera.

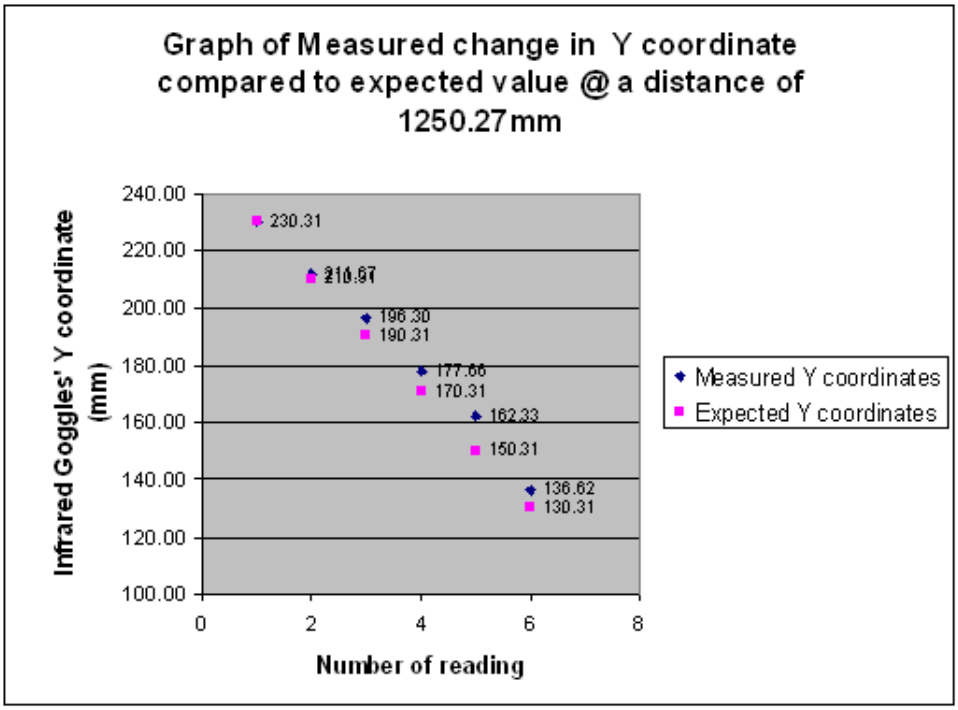

Figure 1.17: Comparison of Measured and Expected Y coordinates for 20mm steps along the Y-axis of the Wiimote's infrared camera.

This set of plots represents the results obtained at a distance of 507 mm

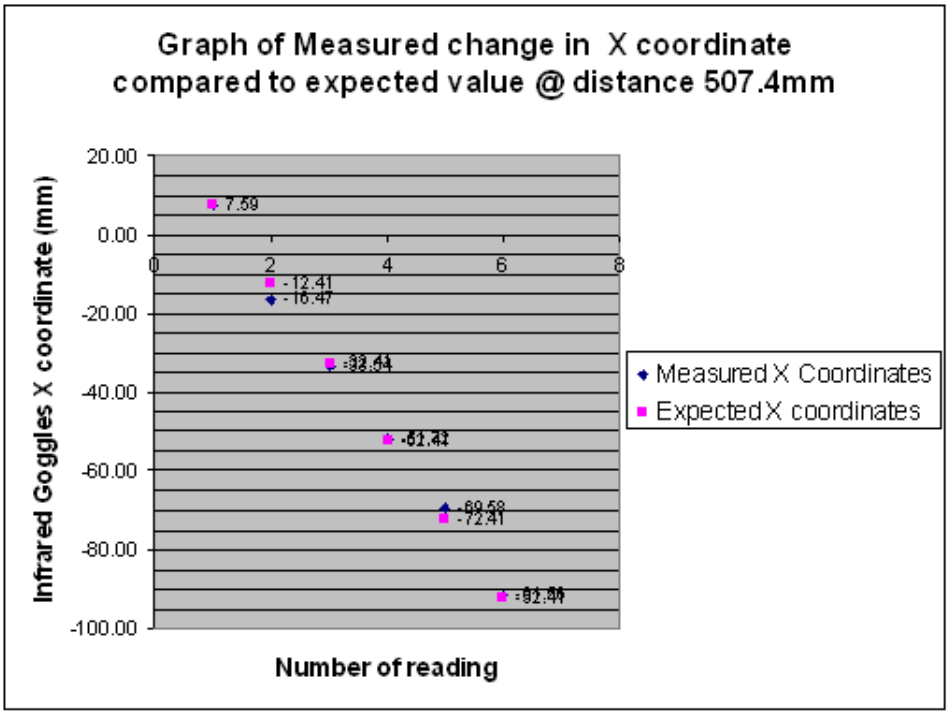

Figure 1.18: Comparison of Measured and Expected X coordinates for 20mm steps along the X-axis of the Wiimote's infrared camera.

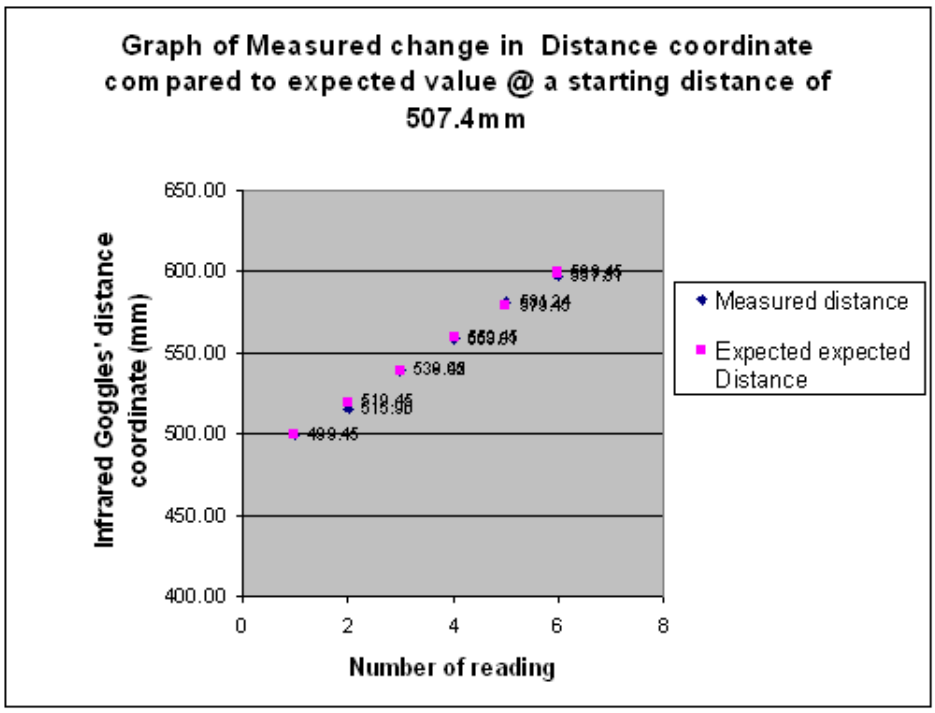

Figure 1.19: Comparison of Measured and Expected distances for 20mm steps along the distance-axis of the Wiimote's infrared camera.

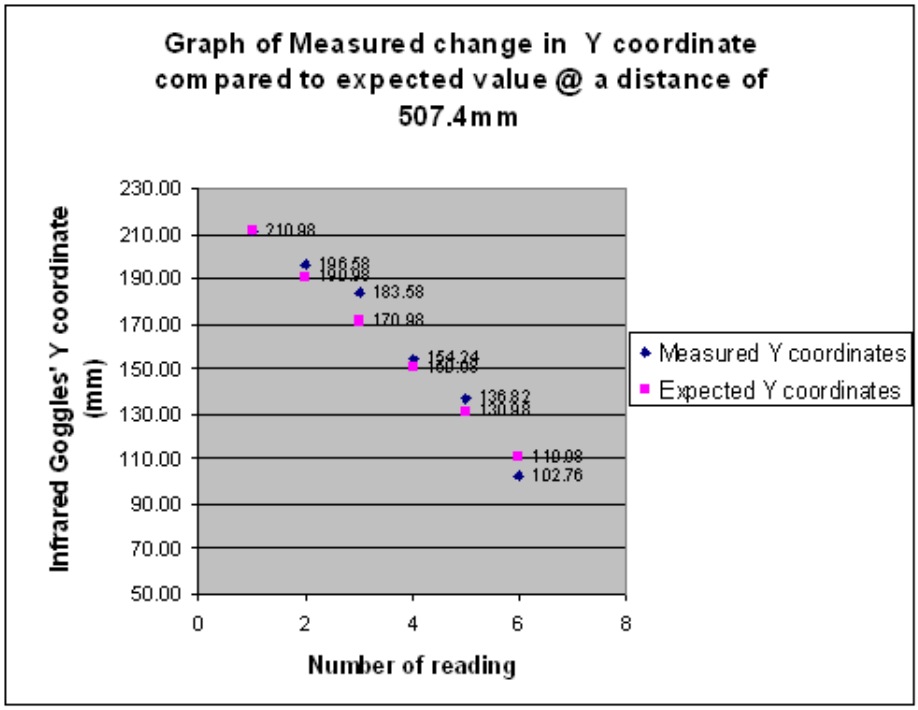

Figure 1.20: Comparison of Measured and Expected Y coordinates for 20mm steps along the Y-axis of the Wiimote's infrared camera.

4.3.5 Discussion and Analysis of Results

From the first test it can be concluded that the Wiimote infrared camera has view which is limited to 45 degrees within which it is capable of providing coordinate data. In terms of the Wiimote's sensitivity, it can be said that if placed in an optimal position, the Wiimote is capable of tracking motion even as minute as 0.5 degrees. This can be seen from Table 1.8 taken at 300 mm from the Wiimote. This figure may well be better than this, however it is impossible to determine with the naked eye and the instruments of measurement that were used to conduct this experiment. The sensitivity of the Wiimote to movement is highly dependant on the position of the infrared goggles, this is apparent when looking at the results obtained when the goggles were moved to a distance of 1200 mm. The changes in coordinates that result from the small changes in angle applied are unpredictable and don't even seem to correlate. This inaccuracy is particularly severe in the case of the distance measurement. The limit of proximity of the goggles to the Wiimote infrared camera is possibly due to the limitation that the Wiimote infrared camera has in terms its viewing angle. Once the goggles come too close to the infrared camera the separation between the infrared leds comes into play and the rays emitted by the leds no longer fall within the camera's viewing range. It can thus be concluded that the distance which the goggles are able to appraoch while maintaining the Wiimote's infrared detection operability is a function of the separation of the leds. As far as the furthest distance at which operability can still be achieved is concerned, the Wiimote exceeds the maximum distance

which might ever be required for the purpose of viewing terrain maps. Comparing the respective x, y and distance plots obtained at distance 3140 mm and 507 mm it can be seen that the measured points at 507 mm correspond more closely to the expected values than in the former case. Looking at each individual axis it can also be seen that the distance measurements provided by the Wiimote seem to contain the greatest amount of error as opposed to the X and Y axes measurements when compared to the expected coordinates.

The greatest source of error in these tests was the continual repositioning of the goggles and the platform upon which it was placed which had to be done in order vary the parameters and coordinates of interest. Great care had to be taken to ensure that this movement affected only the desired variable and didn't bias the results in any way. While every endeavor was made to ensure the accuracy of these tests, it cannot be ignored that they are subject to a great deal of human error, however the results obtained were clear enough to validate the integrity of the insights gained via these tests.

Chapter 5 Software Review

This chapter looks at the relevant available software for development with the Wiimote and three dimensional terrain maps. The topics of discussion in this regard are the Wiimote api's, open source 3D graphic development packages as well as the various file formats currently in use for the rendering of terrains.

5.1 Wiimote Api's

5.1.1 GlovePIE

GlovePIE or Glove Programmable Input Emulator is an application that was originally developed for the Essential Reality P5 Glove in order to enable the P5 Glove to emulate keyboard, mouse and joystick inputs. This functionality has since been extended to handle input from a host of other controller devices including the Nintendo Wiimote. GlovePIE borrows its syntax from a number of different programming languages including Java, Basic, C and C++. It is however not a linear programming language. PIE scripts operate by continuously looping through the entire script running any "if "statements encountered in the background and executing "loops" entirely at once. In order to make use of GlovePIE with the Wiimote, one needs to have GlovePIE version .22 or higher installed, a Bluetooth enabled PC (running Windows 2000, XP or Vista) with the compatible Bluetooth stack/drivers and DirectX 8.0 or higher[2, 29].

As mentioned above, the main function of the GlovePIE application is to enable users to assign certain keys, buttons and gestures to corresponding inputs from the Wiimote. In addition to this, the user has access to Wiimote motion sensing data from its 3-axis accelerometer and infrared camera. The user is also able to send and retrieve data reports as discussed in Chapter 3.2 as well as set and read bytes in the Wiimote's on-board memory[29]. GlovePIE also supports the Wiimote's extension devices such as the Nunchuck, WiiFit and WiiGuitar

A typical example of a GlovePIE command would be the assignment of a button to a certain key: *keyboard.b = Wiimote.A*.

Another would be: *wiimote.dot1x, wiimote.dot1y*. This command effectively reads the position of the infrared dots that the Wiimote can see[29].

GlovePIE provides the benefit of high level functionality with very simple and intuitive syntax. It is also not very demanding in terms of its dependencies and it can even be translated into other languages. Developers using GlovePIE also have a large repository scripts at their disposal which are especially useful for development with the Wiimote.

5.1.2 Wiimote.cpp

The Wiimote api written in the C++ language seems to exist in a number of forms with functionality varying according to the application being developed. A common thread between all the C++ Wiimote api's is the limited documentation or complete absence thereof. As a result a substantial amount of time is required in order to understand which functions are performed by which commands/methods and how to implement these methods. While using the C++ Wiimote api is not as tedious as bit bashing, it is still relatively low level. This does have the advantage of allowing a developer more options and flexibility in the development of their application, however for the purpose of this thesis this was unnecessary and time consuming. In addition to this the api has many dependencies which must be addressed first before development can commence. Subsequent to either the lack of documentation or the difficulty of use or both of these, there doesn't seem to be a great deal of development based on this api. What little source code there is available is very difficult to adapt. The benefit of using the C++ Wiimote api is that it is based on a comprehensive programming language (C++) thus it has the potential to be significantly extended and integrated into many other C++ based applications. The C++ Wiimote api typically provides the developer access to Wiimote IR, acceleration, button, calibration and other status data via the retrieval and manipulation of Wiimote Status Reports as discussed in Chapter 3. Support for most the Wiimote's extension devices is also provided.

5.1.3 The C# Based Wiimote Api

C# is a programming language that was developed from the C and C++ languages which are designed to be a low-level platform-neutral object-oriented programming languages. C# on the other hand differs from this paradigm in that it was designed to be a higher-level component-oriented language and it encourages the developer to allow the framework to manage the precise control of an application while the developer concentrates on the main aspects and issues at hand. The C# Wiimote api immediately reflects this school of thought and while most of the syntax would be very familiar to a C++ programmer this api is far more easier to learn and use. This is due in large part to the object browser facility which lists all the subclasses, methods and attributes of the Wiimote class in a tree like structure and further provides access to the various locations where these are defined. It is needless to say that this is an invaluable tool to any programmer seeking to understand a piece of code and it offsets the inconvenience caused by lack of documentation. The C# Wiimote api pro-

vides much of the same functionality as the C++ based api as well as a much larger base of source code and Wiimote based applications.

5.2 3D Graphics File Formats

5.2.1 The .obj file format

The obj file format describes the surface of a 3d object in terms of triangles or higher degree polygons. The obj file format was developed by Wavefront Technologies for its Advanced Visualizer animation package. It has since been incorporated into various 3d graphics software packages including MeshLab, 3D Studio Max and Blender[7]. A polygon is typically specified by describing all its vertices (denoted by “v") followed by its faces (denoted by “f"). The faces consist of a list of all the vertices contained by that face. All the vertices in a given face are listed as numbers which correspond to the order in which the points/vertices appear in the file. This is displayed in Figure 1.21 which describes a cube in terms of its faces and vertices[6, 8].

```
v 0.000000 2.000000 2.000000
v 0.000000 0.000000 2.000000
v 2.000000 0.000000 2.000000
v 2.000000 2.000000 2.000000
v 0.000000 2.000000 0.000000
v 0.000000 0.000000 0.000000
v 2.000000 0.000000 0.000000
v 2.000000 2.000000 0.000000
f 1 2 3 4
f 8 7 6 5
f 4 3 7 8
f 5 1 4 8
f 5 6 2 1
f 2 6 7 3
```

Figure 1.21: Description of a Cube in the obj file format

A description of a polygon face typically includes data relating to geometric vertex positions, texture coordinates associated with a vertex and the normal at a vertex[4]. In some releases of this format there are additional vertex data types which describe free-form geometric surfaces using curves and surfaces[6]. The .obj file format also specifies that each face described must be flat and convex[5]. Figure 1.22 is an example of some of the data found in a .obj file. In this particular example the four vertex data types shown are denoted as “v” for geometric vertex positions, “vt” for texture coordinates, “vn" for vertex normals and “vp" for parameter space vertices[6].

```
v      -5.000000      5.000000      0.000000
v      -5.000000     -5.000000      0.000000
v       5.000000     -5.000000      0.000000
v       5.000000      5.000000      0.000000
vt     -5.000000      5.000000      0.000000
vt     -5.000000     -5.000000      0.000000
vt      5.000000     -5.000000      0.000000
vt      5.000000      5.000000      0.000000
vn      0.000000      0.000000      1.000000
vn      0.000000      0.000000      1.000000
vn      0.000000      0.000000      1.000000
vn      0.000000      0.000000      1.000000
vp      0.210000      3.590000
vp      0.000000      0.000000
vp      1.000000      0.000000
vp      0.500000      0.500000
```

Figure 1.22: A portion of the obj file format showing four of the vertex data types

The “v" data type in .obj files describes geometric vertices in terms of their x, y, and z coordinates and sometimes a fourth coordinate, w, which is called the weight. The weight coordinate is required to describe rational curves and surfaces but can be left as one otherwise. It is also not necessary to include this coordinate if not needed as it defaults to one when not specified. The “vp" vertex data type specifies the first and second coordinates (u and v respectively) of the parameter space of a surface. In the case of a curve, only the u coordinate is required. The weight coordinate, w, may also be specified for rational trimmming curves. The “vn" vertex data type lists the i, j and k components of the vertex normal. The “vt" vertex data type consists of u, v and w coordinates where the values of u and v correspond to the horizontal and vertical directions of the texture respectively. The value of the w coordinate in this instance corresponds to the depth of the texture and defaults to 0 when not specified[6].

The obj file format does not place any limit on the number of vertices that can be used to specify a single polygon[5]. The .obj file format can also make reference to materials by means of the "usemtl" keyword. This enables characteristics such as color and light to be attributed to the desired vertices. The polygons in a obj file can also be grouped to form sub-objects by using the “g" keyword. This makes for easier editing and animation of models[8].

5.2.2 The .3ds file format

3ds is 3d object description file format developed by AutoDesk. The 3ds file format packages its data in chunks. All chunks start in the same manner providing information as to what the ID of the chunk is, and its location in the file. Within every 3ds file there exists a hierarchy as shown in Figure 1.23. The place of a specific chunk within this hierarchy is determined according to its ID. This ID, represented as a hexadecimal code, specifies the type of data to follow and its constituents. From this one can deduce the presence of other subchunks. The existence of subchunks may also be indicated

by the chunk length, which would then be larger than the expected length of the individual chunk. The very first chunk in any 3ds file is the Primary chunk which corresponds to the hexadecimal number 0x4D4D. The primary chunk would then typically be followed by either the start of object mesh data (3D3D) or the start of keyframer data (B000). These are examples of the main chunks which fall under the primary chunk. These main chunks would then be followed by their respective subordinate chunks and so on[32, 30].

```
MAIN3DS   (0x4D4D)
|
+--EDIT3DS   (0x3D3D)
|   |
|   +--EDIT_MATERIAL (0xAFFF)
|   |   |
|   |   +--MAT_NAME01 (0xA000) (See mli Doc)
|   |
|   +--EDIT_CONFIG1   (0x0100)
|   +--EDIT_CONFIG2   (0x3E3D)
|   +--EDIT_VIEW_P1   (0x7012)
|   |   |
|   |   +--TOP              (0x0001)
|   |   +--BOTTOM           (0x0002)
|   |   +--LEFT             (0x0003)
|   |   +--RIGHT            (0x0004)
|   |   +--FRONT            (0x0005)
|   |   +--BACK             (0x0006)
|   |   +--USER             (0x0007)
|   |   +--CAMERA           (0xFFFF)
|   |   +--LIGHT            (0x0009)
|   |   +--DISABLED         (0x0010)
|   |   +--BOGUS            (0x0011)
|   |
|   +--EDIT VIEW P2   (0x7011)
```

Figure 1.23: Illustration of the Hierarchy of 3ds Chunk Types

5.2.3 The .ply file format

The ply or Polygon file format is a 3d object description file format initially developed by Stanford University for the purpose of storing three dimensional data mainly obtained from 3D scanners. The structure of a typical PLY file is follows: Header, Vertex List, Face List and lists of other properties attached to their respective elements[10, 27]. The header found at the beginning of each ply file states the number vertices and polygons contained in the file as well as any additional properties associated with the vertices such as color, normals and texture coordinates. The header also specifies whether the file is in ASCII or Binary format. It is mandatory that the header begin with the string ”ply" , which serves to identify the file as a ply format file, and ends with "end_header". Each part of the header is preceded by a descriptive keyword as shown in Figure 1.24 with comments, which are not considered as part of the file, being preceded by the keyword “comment".

```
ply
format ascii 1.0          { ascii/binary, format version number }
comment made by Greg Turk  { comments keyword specified, like all
lines }
comment this file is a cube
element vertex 8           { define "vertex" element, 8 of them in file }
property float x           { vertex contains float "x" coordinate }
property float y           { y coordinate is also a vertex property }
property float z           { z coordinate, too }
element face 6             { there are 6 "face" elements in the file }
property list uchar int vertex_index { "vertex_indices" is a list of ints }
end_header                 { delimits the end of the header }
0 0 0                    { start of vertex list }
```

Figure 1.24: Example of a ply file header

Much like the obj file format discussed above, the ply format specifies the vertices of a polygon as (x,y,z) triples followed by faces which list the indices of the vertices they contain. In addition to this basic data, new element types and properties can be created and attached to the elements of the object specified in the file. This is allowed by the fact that newly created element types that are not recognised by programs can simply be disregarded by them and carry on without crashing. The ply format also allows different properties to be assigned to the back and front sides of a polygon.

5.2.4 The .ter file format

Ter format files are the output files of the Terragen Landscape Generator programme developed by Planetside Software. The ter file format makes use of a chunk-type architecture, however it is not as rigidly defined as is the case with the 3ds file format. A ter file is required to begin with an 8-byte “TERRAGEN " string followed by another 8-byte “TERRAIN " string including the space at the end of the string. This is then followed by a "Size” chunk, followed other data chunks and the file is concluded by a 4-byte “EOF" chunk. The following is a summary of the main chunks found in a ter file:

All of the chunk types that appear in the ter file format are identifiable by their 4-byte markers. "XPTS" and "YPTS" are the markers of chunks which contain 2-byte integer values representing the number of data points in the x and y directions respectively of the elevation image. They also contain a further 2-bytes of padding. These chunks along with all the other chunk types excluding the “ALTW" chunks and the “SIZE" are required to appear after the “SIZE” chunk and before any altitude data. The chunk with the“SIZE" marker contains a 2-byte integer equal to the number data points on the shortest side of an elevation image. It too contains 2-bytes of padding. The “SCAL” chunk is an optional chunk which contains three 4-byte floating point numbers representing the scale of the terrain in metres per terrain unit. These numbers correspond to the x, y and z coordinates and are required to be equal. “CRAD" is the marker of a chunk which specifies the radius of the world being rendered in kilometers. This value is presented as single 4-byte floating point number and is

also optional. The“CRVM" chunk is another optional chunk containing a single unsigned integer value which, when set to 0, causes a flat terrain to be rendered, while if set to 1 a spherical terrain results. Finally the “ALTW" marker which is followed by HeightScale, BaseHeight and Elevations all of which are sequences of 2-byte signed integers. This chunk contains elevation integers which are applied to the x and y points set previously in the “XPTS” and “YPTS" chunks[27].

5.3 3D Graphics Packages

5.3.1 Autodesk 3ds Max

3ds Max is a 3d graphic animation program developed by Autodesk Media and Entertainment primarily for the modeling, animation and rendering of 3d graphic objects. It is most commonly used by game developers, TV commercial studios, architectural visualization studios and has also been used to create movie effects and movie pre-visualization[11]. 3ds Max also has its own native file format, 3ds, which was discussed earlier in the text. In addition to 3ds, Autodesk 3ds Max is also capable of importing and exporting files of a variety of formats including polygon(.ply), object(.obj) and terrain(.ter). Other features provided by 3ds Max include normal map creation and rendering, global illumination, an intuitive and fully-customizable user interface, as well as its own scripting language[11].

5.3.2 MeshLab

MeshLab is a mesh processing system which is mainly aimed at facillitating the editing, cleaning, filtering and rendering of medium to large size unstructured 3d triangular meshes which typically arise from 3d scanning[12]. MeshLab is developed and maintained by a small group of researchers at the Visual Computing Lab of the Italian National Research Council as well as a larger community of contributing plugin developers[13]. MeshLab is capable of importing a variety of 3d file formats, however the terrain(.ter) is not one of these.

5.3.3 Blender

Blender is a graphics application that provides much of the same functionality provided by 3ds Max, however it has the added advantage of incorporating an embedded Python interpreter. This enables Blender to run Python scripts for creating tools, game logic, converting between file formats and various other functions which greatly expand the functionality that can be achieved with Blender[14]. Blender was originally developed and distributed by NeoGeo and Not a Number Technologies (NaN) as an in-house application, but later in 2002 it became open source software which today continues to be developed under the supervision of the Blender Foundation[15].

5.3.4 Terragen

Terragen is a scenery generation program designed for generating landscape images/visualization. Terragen is also able to import and export raw heightfield data which is typically used to generate terrains from externally created pictures. Terragen also makes use of a scripting system and plugins in order to extend its functionality. As previously mentioned, Terragen also has a file format of its own, the terrain (.ter) format[16].

5.3.5 Assessment of Software

Since this thesis is not really concerned with the modelling of terrain maps, the software described above is assessed according to its potential to facillitate the viewing of terrain maps interactively.

3ds Max is specifically geared for the modelling and animation of 3d objects and is not particularly suitable for the sole purpose viewing 3d objects interactively. MeshLab on the other hand is ideal for the viewing of 3d objects as it enables the user to maniplate their view with simple combinations of key strokes and mouse gestures. MeshLab's inability to read terrain files is somewhat of a disadvantage as the .ter file format is one of the prefered file formats for the rendering of terrain maps. This can however be overcome by converting any terrain files that one may want to view into another format which MeshLab recognises such as the .obj or .ply file formats. This is easily done with the aide of 3ds Max. Blender, as a result of its embedded python interpreter does have the potential for the development of an application which incorporates the viewing of terrain maps, however its shortfall is that it does not provide an easy enough interface for the interaction of the user with a 3d environment and it is a very difficult program to understand and navigate. Furthermore, Blender also doesn't seem to recognise any of the file formats discussed in the preceding text without the use of plugins or imported scripts. These reasons make Blender an overly complicated and unsuitable solution. Terragen is of particular interest due to the fact that it specialises in the generation of terrains and also seems to provide the prefered file format for storing terrain maps. Terragen may be useful for the editing of terrain maps in order to make them more suitable for viewing however, other than this most of the functionality provided is of no consequence for the purpose of this thesis. MeshLab thus emerges unrivaled in terms ease of use and ability to facilitate the viewing of terrain maps interactively with the Nintendo Wiimote. Due to the preference towards using MeshLab for the purpose of viewing terrain maps, it is desirable that a file format that is compatible with MeshLab be selected for this purpose. Thus the polygon (.ply) and object (.obj) file formats appear to be the most suitable formats in which to store terrain maps to be viewed interactively. Once again the option of converting from the terrain (.ter) format does exist should the need arise, this makes the use of MeshLab and the ply and obj file format perfectly viable.

Chatper 6 Merging the Wiimote with Terrain Maps

In order to achieve the function of viewing terrain maps interactively via head movements, the various capabilities of the Wiimote need to harnessed and incorporated into an application which is also capable of loading terrain maps and manipulating the view of these maps to provide the user with the view they desire. One method of doing this would be to “piggy back" on an application that has already been written for the Wiimote. An example of such an application which provides the desired functionality is the WiiDesktopVR application. This application could then be extended by writing a terrain map loader for one of the file formats discussed in Chapter 5. Writing a file format loader however is not an easy task. In addition to an in depth knowledge of the file format to be used, the developer would require expertise in either of the opengl, glut or directx graphics libraries. Another obstacle that the developer would encounter is integrating the Wiimote’s sensor data into the terrain file loader and using this information to render the correct view of the terrain. This method, while probably the best solution ultimately, requires lengthy period of time for development. The period of time required for the development of thiis application may further be augmented by the learning curb that an unexperienced developer would have to undergo.

Alternatively, an application which “piggy backs" on already existing programmes that allow the user to view 3d objects and terrain maps can be developed. An ideal candidate for such an application would be MeshLab. Since MeshLab already has all the possible views that a user may desire mapped to combinations of mouse and keyboard gestures, an application that allows the Wiimote to assume the functions of the mouse would then enable the user to manipulate their view with the Wiimote. This difficulty with this approach comes in when a specific manner of viewing requires both mouse and keyboard input at the same time. For instance, in MeshLab in order to move forward over a terrain the user is required to hold down the “shift” key while scrolling with the mouse. This obstacle can be overcome by usind two Wiimotes. One for the head tracking of the user while the other is used to generate the equivalent keystrokes required to view the terrain maps. With a script that enable the Wiimote to take over mouse actions, it then becomes possible to do many other things with the Wiimote. One such script can be found in the GlovePIE source code repository. In the interest of viewing terrain maps, one could use this application to view maps in Google Earth,

which for for a fee can provide an endless database of high quality terrain maps. Thus even the issue of file formats and relevant software packages is bypassed. Appendix A shows a typical example of a GlovePIE script which can be use to assign desired keystrokes and mouse gestures to the Wiimote. This particluar script simply moves the terrain backwards as the Wiimote is moved closer to the infrared leds. This movement is typically achieved in MeshLab by hold the shift key and the mouse left button while scrolling. The shift key and the left mouse button are assigned to the minus and plus button on the Wiimote in this script. Unfortunately this script does not implement headtracking as a second Wiimote is required to effectively use the headtracking to view the terrain..

Chapter 7

7.1 Conclusions

From the tests and experiments conducted on the Wiimote it can be concluded that the Wiimote is indeed capable of providing the data and measurements necessary to perform head tracking. The Wiimote is by no means the the best performing head tracking technology developed yet, however its unique combination of sensors (most notably its accelerometer and infrared camera) and wireless data transfer (via Bluetooth) are unrivalled in terms of head tracking and joystick devices. The capabilities presented by the Wiimote, though it may not be as accurate as some configurations and has various limitations on its operation, are a step forward within the head tracking faculty. Its ability to perform the functions of multiple systems presents the opportunity to greatly simplify head tracking configurations. The Wiimote facillitates the convergence towards the development of a compact, cheap, and easy to use head tracking system which may, in the future, become a standard device implemented in a wide spectrum of applications. In terms of the api's used to communicate with the Wiimote, the C# based Wiimote api was found to be the most complete and easily learnable of those discussed in this thesis. The GlovePie library was also found to be immensley useful for the purpose of development with the Wiimote. Not only does GlovePie provide an easy to learn high level api, but it also comes with a rich base of sample scripts written for the Wiimote among which are a few scripts written to control mouse actions with the Wiimote. These scripts used in combination with MeshLab provided the closest approximation to a viable solution for the viewing of terrain maps with Wiimote-head-tracking. Another alternative provided by this script is to use it in conjunction with Google Earth. This script however only allows for the movement of the mouse cursor. The required functionality to view terrain with head motion via the Wiimote could be added to this script given time, however it was found at an advanced stage of this thesis which did not allow for further development given the time constraint. With regards to the storage of terrain maps, the obj and ply file formats were found to be the most suitable as they are fairly universal file formats and are both recognised by MeshLab which is the graphics rendering software package that provided the most desirable terrain map viewing experience.

7.2 Future Work Recommendations

This project presents the opportunity for vast amounts of future. In particular for this topic, the most immediate work to be carried out would be to adapt the GlovePIE mouse control script to allow the Wiimote to control mouse gestures in such a way that is conducive to terrain map viewing. This script should include the generation of keystrokes and gestures by a second Wiimote and possibly some of the Wiimote's extension devices. One particular device that would be of interest would be the WiiFit, which could possibly be used to add "Walk Through" functionality to the viewing of 3d terrain maps. Beyond this, an all encapsulating application could be developed complete with Wiimote head tracking, terrain file format loading and "Walk through". Some other work may also be done in making Google Earth a Wiimote compliant application.

Appendix A

The following is a typical example of a GlovePIE scripts which can be use to assign desired keystrokes and mouse gestures to the Wiimote. This particluar script simply moves the terrain backwards as the wiimote is moved closer to the infrared leds. This movement is typically achieved in MeshLab by hold the shift key and the mouse left button while scrolling. The shift key and the left mouse button are assigned to the minus and plus button on the Wiimote in this script.

```
var.ButtonFreezeTime = 250ms
var.PointerBump=KeepDown(Pressed(wiimote.A),var.ButtonFreezeTime)or KeepDown(Pressed(wiimote.B),va
Wiimote.Led1 = true
// Mouse movement
if wiimote.PointerVisible but not var.PointerBump then
mouse.x = wiimote.PointerX
mouse.y = 1-wiimote.position end if
// Mouse Buttons
mouse.LeftButton = Wiimote.Plus and KeepDown(Wiimote.PointerVisible,0.5s)
key.Shift = Wiimote.Minus and KeepDown(Wiimote.PointerVisible,0.5s)
```

Bibliography

[1] *Small, Low Power, 3-Axis +/-3 g i MEMS Accelerometer*.

[2] Glovepie. http://www.wiili.org/index.php/GlovePIE, October 2008.

[3] http://www.c-sharpcorner.com/Articles/ArticleListing.aspx?SectionID=1&SubSectionID=144, October 2008.

[4] Obj files 3d object format. https://people.scs.fsu.edu/ burkardt/data/obj/obj.html, October 2008. Last revised on 27 September 2008.

[5] Wavefront and java3d .obj format. http://www.eg-models.de/formats/FormatObj.html, October 2008.

[6] Object files (.obj). https://people.scs.fsu.edu/burkardt/txt/objformat.txt, October 2008.

[7] Obj. http://en.wikipedia.org/wiki/Obj, October 2008.

[8] Wavefront obj file format summary. http://www.fileformat.info/format/wavefrontobj/, October 2008.

[9] The ply file format. http://www.cc.gatech.edu/projects/largemodels/ply.html, October 2008.

[10] Ply - polygon file format. http://local.wasp.uwa.edu.au/ pbourke/dataformats/ply/, October 2008.

[11] 3ds max. http://en.wikipedia.org/wiki/3dsMax, October 2008.

[12] Meshlab. http://www.ohloh.net/projects/5759, October 2008.

[13] Developers. http://meshlab.sourceforge.net/wiki/index.php/Developers, October 2008.

[14] Bundled scripts in blender 2.34. http://www.blender.org/education-help/python/, October 2008.

[15] Blender (software). http://en.wikipedia.org/wiki/Blender3D, October 2008.

[16] Terragen. http://www.planetside.co.uk/terragen/cando.shtml], October 2008.

[17] Magnetic tracking technology. http://www.cs.nps.navy.mil/people/faculty/capps/4473/projects/magtrack/full.html, October 2008.

[18] Wiimote. http://www.wiili.org/Wiimote, September 2008.

[19] Wii-mote guts. http://www.sparkfun.com/commerce/tutorialinfo.php?tutorialsid=43&page=, October 2008. Posted: December 19th, 2006.

[20] Wiimote hardware. http://www.wiili.org/index.php/Wiimote/Hardware, October 2008.

[21] *HUMAN INTERFACE DEVICE (HID) PROFILE.*

[22] Broadcom. Bcm2042 advanced wireless keyboard/mouse bluetooth® chip. http://www.broadcom.com/products/Bluetooth/Bluetooth-RF-Silicon-and-Software-Solutions/BCM2042, October 2008.

[23] Palowireless Bluetooth Resource Center. Service discovery protocol (sdp). http://www.palowireless.com/infotooth/tutorial/sdp.asp, October 2008.

[24] F. Panerai S. Hanneton J. Droulez V. Cornilleau-Peres. A 6-dof device to measure head movements in active vision experiments: geometric modeling and metric accuracy. *Journal of Neuroscience Methods*, 90:97, 1999.

[25] Analog Devices. Adxl330: Small, low power, 3-axis +/-3g imems accelerometer. http://www.analog.com/en/mems-and-sensors/imems-accelerometers/adxl330/products/product.html, October 2008.

[26] Yoshinobu Ebisawa. A pilot study on ultrasonic sensor-based measurement of head movement. *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT*, 51, NO.5:1109 – 1115, 2002.

[27] Matt Fairclough. Terragen terrain file specification. http://www.planetside.co.uk/terragen/dev/tgterrain.html, October 2008. Last modified 2003.10.30.

[28] Mark Ward Ronald Azuma Robert Bennett Stefan Gottschalk Henry Fuchs. A demonstrated optical tracker with scalable work area for head- mounted display systems. Technical report, University of North Carolina, 1992.

[29] Carl Kenner. Glovepie 0.30 documentation, December 2007.

[30] Jeff Lewis. The unofficial 3d studio 3ds file format. http://the-labs.com/Blender/3DS-details.html, October 2008.

[31] K. S. PARK and C. J. LIM. An efficient camera calibration method for vision-based head tracking. *Int. J. Human-Computer Studies*, 52:879–898, 2000.

[32] Jim Pitts. .3ds file format. http://www.whisqu.se/per/docs/graphics56.htm, October 2008. Posted: December 1994.