Low-Cost Eye-Tracking-like Control of a PC: Hardware Realization and Interpreting Data

Jan Kazemier (S1609238),
j.kazemier@gmail.com

January 20, 2010

university of groningen

Low-Cost Eye-Tracking-like Control of a PC: Hardware Realization and Interpreting Data
Bachelor Thesis in Computing Science
Author: Jan Kazemier
j.kazemier@gmail.com
Supervisors:
Dr. Tobias Isenberg
Dr. Michael Wilkinson

Abstract

Head tracking is used in research: in many studies, the position of the eyes is measured, for instance in usability studies [3]. Other research shows that head tracking has applications in distance education and video conferencing [6]. Another application of head tracking is in Virtual Reality, where the position of the head influences the image the user is looking at [13, 5]. Another head tracking application is in games, where the position of the head is used to control parts of the games, most likely, but not limited to, the view of the user (i.e., looking around corners, looking left and right and looking up and down) [7]. Head tracking is also used in animation [4].

As a Bachelor project we made a low-cost solution for head tracking for the PC using a WiiMote and self-made glasses with IR-LEDs, using a Bluetooth connection. We will explain the choice for this hardware in the chapter Materials and Hardware Realization. Normally a PC is controlled using a keyboard and a mouse. We aim to make a low-cost alternative to the use of the mouse. This alternative may help people with a disability to their arms or hands to control a PC. Also, our project could be used in one of the applications mentioned above.

In this project we made the software which retrieves the data perceived by IR-sensors in the WiiMote, based on the position of the IR-LEDs. This data is then filtered and interpreted.

This thesis will concentrate on the hardware components, the realization and implementation of the hardware components and technical details of the hardware. On the software side, this thesis will concentrate on the interpretation of the filtered points. Existing solutions for head tracking have several disadvantages: we will discuss a few commercially available products and look at the advantages and disadvantages in detail.

As a case study we have decided to play a game with our head. The game we were able to play was Portal, by Valve Software [15].

Contents

0.1 Preface

This document is my Bachelor thesis, made in the third and fourth period of my third year at the Computing Science Bachelor, University of Groningen. The project was done together with Yuri Meiburg.

I would like to thank the following people, as they were of great help for writing my thesis and doing my Bachelor project.

Many thanks to Yuri Meiburg, for being a friend, offering moral support and for working on this Bachelor project together with me.
Many thanks to Dr. Tobias Isenberg, for being our first supervisor, for supplying materials and numerous great ideas, for helping greatly us with the project and theses, and for being very patient.
Also thanks to Dr. Michael Wilkinson, for being our second supervisor.

Chapter 1

Introduction

Applications of head tracking include, but are not limited to, research, gaming, virtual reality, and, as in our main case, the control of a PC. Head tracking is used in research: in many studies, the position of the eyes is measured, for instance in usability studies, where the eye movement is tracked to understand how humans react to certain situations [3]. Other research shows that head tracking has applications in distance education and video conferencing [6]. Another application of head tracking is in Virtual Reality, where the position of the head influences the image the user is looking at [13]. Yet another head tracking application is in games, where the position of the head is used to control parts of the games, most likely, but not limited to, the view of the user (i.e., looking around corners, looking left and right and looking up and down) [7]. Head tracking is also used in animation [4].

The commercially available products have several disadvantages:

- Commercially available products are considered expensive: prices vary from €150 to more than €25000. One of our goals is to make a cheap alternative to the commercially available products.
- Some commercial products are limited in use, i.e., you are not allowed to use them for (for example) research under certain circumstances, like faceAPI. Our program is freely available with no usage restrictions.
- Not all products are able to track movements in more than 2 degrees of freedom. Our program is able to support the detection of movements of the head in multiple degrees of freedom. This enabled us to be able to control the mouse of a PC with our heads and play a computer game with it.

We have seen some movies of Lee, who does some interesting tracking of fingers and the head, using just a WiiMote and IR-LEDs [10]. A Wii-Mote is relatively cheap (around €40) and IR-LEDs are less than €0.20 per piece. Before working on this project, we have done a smaller similar project, called WiiView [8]. In that project we already did some work on head-positioning using glasses with IR-LEDs and a WiiMote. This project only took the absolute position of the head and mapped this to certain actions (i.e., next image, rotate image clockwise, etc.) in an image viewer. Our goal is to support movements in all three directions (i.e., movement along all three axis), as well as rotation around the three axis. Those movements are the six degrees of freedom.

In this project we create an implementation of head tracking using the WiiMote and IR-LEDs. Using a Java library called WiiUseJ, we are able to track up to four points in 2D space, as seen by the WiiMote. Our software is able to get the tracked data from the WiiMote. After filtering this data, we interpret it. We built a framework and software to be able to track movements of the head in all six degrees of freedom. After detecting these movements they are interpreted and mapped to mouse movements, in order to replace the mouse as an input device for a PC. In the end we were able to play a game (Portal, by Valve Software) with our head tracking solution. This thesis concentrates on the hardware components, the realization and implementation of the hardware components and technical details of the hardware. On the software side, this thesis will concentrate on the interpretation of the filtered points.

The main goal of this project is to deliver a low-cost eye-tracking-like (i.e. head-tracking) way to control a PC. The tracking could also be used in research where the position of the head is important. It could be used too in research for people who cannot use their arms, or otherwise would like to use their head as an input device for the computer. Other applications as named before are gaming, animation and virtual reality [7, 4, 13]. In this project we built a framework and/or program for using a WiiMote and IR-LEDs to do cool and useful things with the PC. In the first place we look at replacing the mouse by a head tracking device. The framework should be flexible and extra functionality should be added with ease. The program should be easily adaptable, robust and precise. Using only a WiiMote and self-built glasses wit IR-LEDs, we already guarantee it will be relatively cheap.

Chapter 2

Related Work

Motion tracking (or motion capture) has been derived from rotoscoping, a technique to copy realistic motion onto cartoon characters [12]. It has been performed since 1970 [12], when it was used as a tool for biomechanic researchers. In this section we will discuss some of the existing variants, along with some examples of commercially available implementations of these variants at this time.

As said before, before working on this project we have done a much smaller project also using a WiiMote and IR-LEDs in the Innovative Interactive Systems course at the University of Groningen. Yuri Meiburg and I made an image viewer which is able to react to the position of the head, performing actions (i.e., next image, rotate image clockwise, etc.) based on the position of the head. The project is called WiiView and it allows the user to use her head as an input device for the image viewer. With some adaptations it is possible to use it for any program, or to control the cursor, based only on the averaged perceived x and y position of your head. Based on this and other related work we started working on a better version of head tracking using a WiiMote and IR-LEDs. This means being able to track more than two degrees of freedom, in order to be able control a PC. The related work is described below.

2.1 Commercially available products for head-tracking

For different applications there already are some products commercially available. A few relevant products are set out beneath.

2.1.1 Reflective IR-patches

One way of tracking is to emit IR-light and have this reflected by reflective patches on the head . An example of this is the TrackIR (see Figure 2.1). The TrackIR is commercially available for around €150, which is quite low cost, and specifically meant for gaming. TrackIR (see figure 2.1) is a commercially available product of the NaturalPoint company, mainly aimed at gamers. It uses IR-light to track a users face. It allows tracking for six degrees of freedom, linking your head-movements in 3D-space to your InGame-view. Our

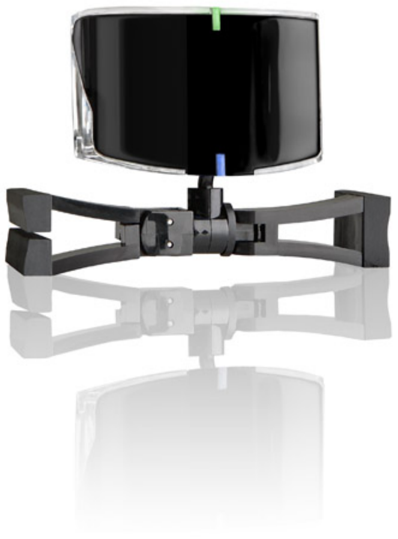

Figure 2.1: the TrackIR, source: *http://www.naturalpoint.com/trackir/*

program will act quite a lot like this: track multiple degrees of freedom and link this to (in this case games) input. We might want to investigate how well reflective patches work, compared to sending IR-light directly from the head. The big advantage of patches is that they weigh less, because you do not need batteries on the head. TrackIR places the patches on a headset or if you do not want to use a headset, you can use a baseball cap. On the headset are three reflective patches. Their demo[1] shows that they are able to track all degrees of freedom, using only three reflective patches. We will look into methods to extract all three degrees of freedom by using all three or four LEDs.

2.1.2 Eye tracking

Eye tracking is a technique where the position of the iris is measured. Although not completely the same, it has some similarities with head tracking. An example of eye tracking is the Eyelink 2, from the company SR Research.

According to their own website, their eye-tracking products have already lead to 1134 peer-reviewed publications [1].
The Eyelink II itself (without any options) costs 25,535 GBP, so that is a quite expensive solution. The EyeLink II system consists of three miniature cameras mounted on a headband, two cameras are pointed at the eyes. The software recognizes the iris and derives the direction the eye is looking. The third camera is used to track the orientation of the head. The Eyelink II is used mainly for research, in projects where the position of the eyes

[1]Available at their website: `http://www.naturalpoint.com/trackir/products/trackir5/#5`

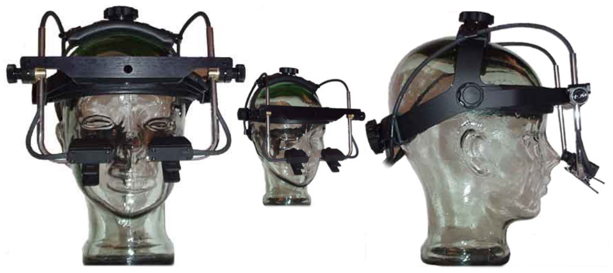

Figure 2.2: the Eyelink 2, source: *http://www.sr-research.com/products/*

is important. Our project might be used when the position of the head is important.

2.1.3 faceAPI

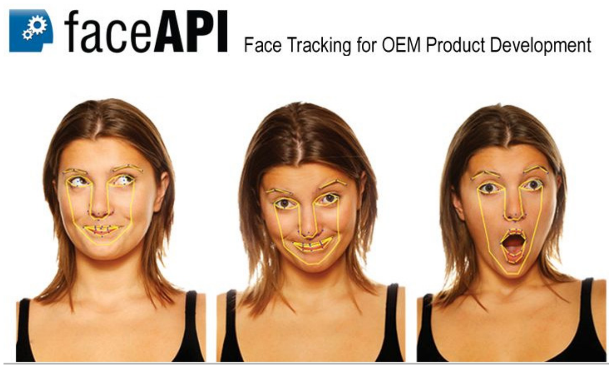

Figure 2.3: faceAPI, source: *http://www.seeingmachines.com/product/faceapi/*

FaceAPI (Figure 2.3) is an interesting head tracking solution, as it has no need for anything attached to the head. It works using a web cam, recognizing the shape of the head, and thus determining the position of the head. As the figures 2.3 and 2.4 show it does his job quite well. Others also already have done some work based on this program [14]. Unfortunately, the free version of the program is crippled, so we have had no experience using this program. The idea however is a nice alternative to our approach of using IR-LEDs, results of both projects could be compared. The faceAPI does not need the user to apply anything to her head, while still able to track six degrees of freedom. As this is a completely different approach than ours we will leave this field for others as future work.

2.2 Using a WiiMote

Our attention was drawn to the projects of Lee. He already has done some very nice projects with a WiiMote and IR-LEDs. The most relevant is the Head Tracking for Desk-

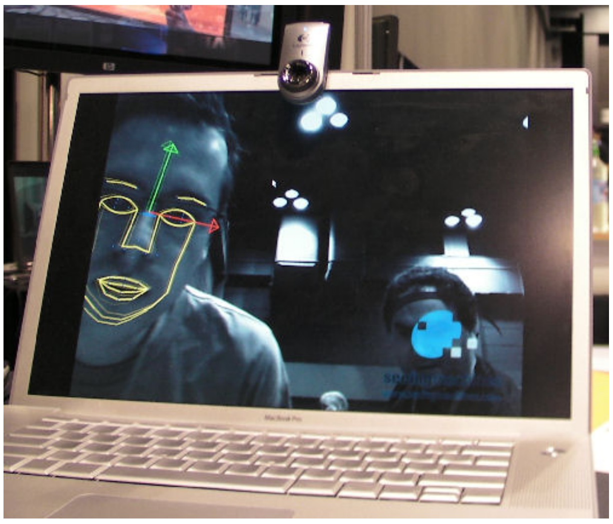

Figure 2.4: faceAPI in action, source: *http://www.seeingmachines.com/product/faceapi/*

Figure 2.5: Lee with his WiiMote project, source: *http://johnnylee.net/projects/wii/*

top VR Displays using the Wii Remote (see figure 2.5). Lee uses active markers to track objects, however, reflective patches could be used too.

Using the IR-camera in the Wii remote and a head mounted sensor bar with two IR-LEDs, he can accurately track the location of the head and render view-dependent images on the screen [9]. This effectively transforms a display or TV into a sort of portal to a virtual world. The display properly reacts to head and body movement as if it were a real window, creating a realistic illusion of depth and space. This software was written in C# with DirectX, the source code is provided at the website [10]. What he has done here is quite similar to what we try to achieve: using 2 LEDs, he is able to track the position and orientation in x, y and z position. Also, rotation around z and y axis can be measured using 2 LEDs. For more robustness and eventually being able to track rotation around the x-axis, we might want to use three or four LEDs.

2.3 Summary

There are commercially available products to perform head tracking in six degrees of freedom. A big disadvantage of eye-tracking solutions is that they are relatively expensive. FaceAPI has a complete different approach while still able to track six degrees of freedom, which could be used to compare results. Lee has good results tracking the head using two LEDs and a WiiMote. We base part of our work on his, extending it to four LEDs, but we will build our own framework.

Chapter 3

Concept and Design

3.1 Introduction

In order to create a low-cost solution to control a PC we want to track the head. This might replace the mouse, though it might be added to the conventional keyboard/mouse combination. We will only use PC with Bluetooth, a WiiMote and a pair of glasses with IR-LEDs. The head tracking will consist of a few important steps:

- Communicate between WiiMote and PC via Bluetooth (Bluetooth communication)
- Make a platform-independent framework
- Create the ability to filter raw data from the WiiMote (filters)
- Interpret data to take control of a PC, using only your head (interpreting)
- Give some sort of output

Ideally, we should not need any more hardware. It might however also be used besides the mouse and keyboard. For full control of a PC with only the head tracking as an input device, one should also implement a keyboard (or something similar). Implementing a keyboard to be used with your head is a whole different project, so we will not look into that; we will only concentrate on replacing the mouse of the PC by a head tracking solution.

What we want to achieve is a solid and expandable platform-independent framework. To achieve platform-independence we will use Java. In our previous project WiiView, we have found that the WiiUseJ library is very flexible and usable to receive data in Java from the WiiMote. WiiUseJ takes care of all Bluetooth issues, allowing us to fully concentrate on the filters and interpreters. This thesis will specifically concentrate on the interpreters, Yuri Meiburg concentrates on the filtering of the data [11].

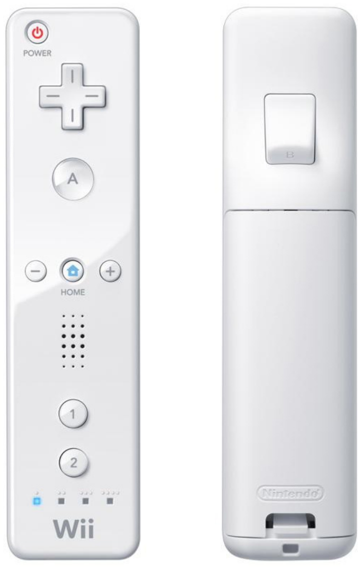

Figure 3.1: The Nintendo Wii Remote™

3.2 WiiMote

In figure 3.1 the Nintendo Wii Remote (WiiMote) is shown. The WiiMote is the remote controller for Nintendo's game-console Wii. On the Wii it is used as the only input-device for playing games, although there are a few extensions available and sold separately for use in specific gaming environments (e.g. a steering wheel, a table tennis bat, etc.). The WiiMote has some interesting sensors: it has the ability to sense movement along the three axis using an accelerometer. The more interesting sensor for our project is the Infra Red sensor (IR-sensor), a sensor capable of tracking up to four infra red points on a resolution of 1024×768 "pixels". This sensor enables us to use any device, apply a few IR-LEDs to them and use them in our project. We will place the WiiMote on top of the monitor, while the IR LEDs will be on our glasses (see figure 6.1).

Chapter 4

Materials and Hardware Realization

4.1 Introduction

In this chapter we will especially discuss what materials we are going to need.

The general setup is as follows: the user puts the glasses on her head, the WiiMote is placed upon the monitor. The glasses will have four LEDs. When the software is enabled, the WiiMote should pick up the points (position of the 4 LEDs in 2D-space) and communicate via Bluetooth with the PC. Our software should receive the data and filter the results. After filtering the data has to be interpreted. The concept behind the interpreting has been handled in the next chapter.

4.2 Glasses

Figure 4.1: The original Wii Sensor Bar

The glasses and other hardware were part of my “specialization”. We chose to have the IR-LEDs attached to our head, because we believe that this will give the best results as you get all light from the LEDs to the WiiMote without reflecting or specifically pointing it in a certain angle at the WiiMote. In further research one could of course try to apply reflecting patches to his head, or even try to see the eyes reflecting to the WiiMote. We did not look into this ourselves.

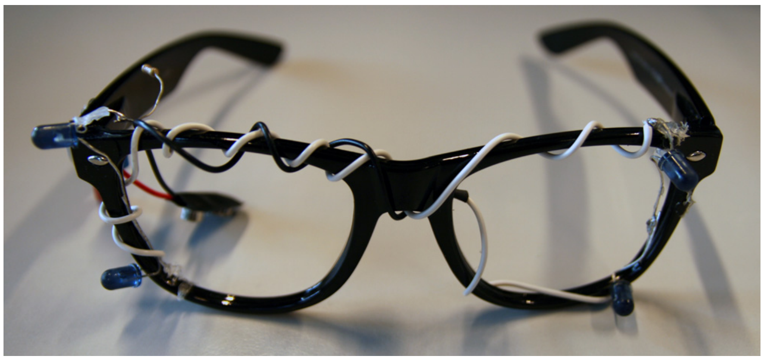

Figure 4.2: Our first pair of glasses

Normally, a IR-LED bar (see figure 4.1) is packaged with your Wii. A possibility is to tape such a bar to a baseball cap, like Lee has done in his project [10]. However this would be highly impractical, as the bar is quite wide, and the WiiMote will only distinguish two points (the original bar contains 2 groups of 5 LEDs). For more robustness we built glasses using 4 IR-LEDs (see figure 4.2).

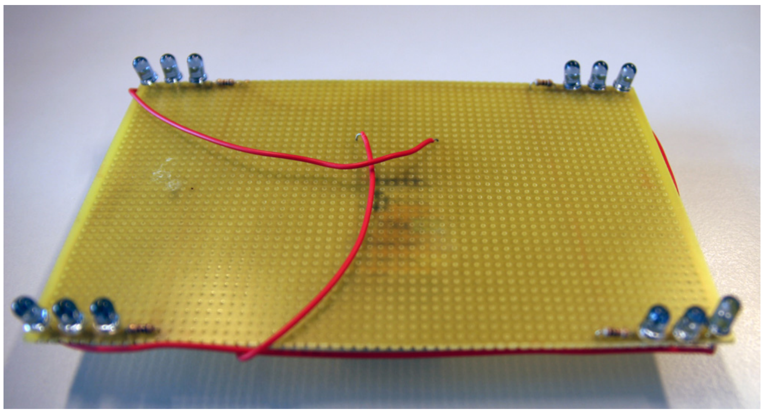

Figure 4.3: Our model breadboard

Because in the end we wanted to try different types of glasses we built a model for glasses on a breadboard (see figure 4.3). We used this to test different setups of different amounts of LEDs. The best board contains four groups of three IR-LEDs. We tried different amounts of LEDs as well as different angles of the LEDs.

We slightly bent the leads of the LEDs so they made a little angle, while still overlapping the beam. This way we have quite a lot of freedom in turning the head while the Wii-Mote still keeps registering the points at about the same place.

In the end we built glasses (see figure 4.5) based on this model, again using four groups of three LEDs. This gave quite a robust feeling to our program. The glasses overlap in beam and work quite well. Building these glasses does not cost a lot, the LEDs are about

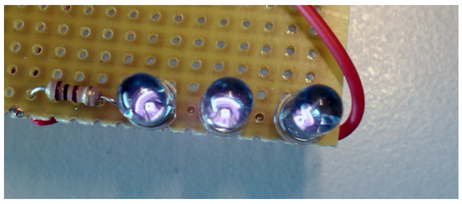
Figure 4.4: A close-up of the IR-LEDs

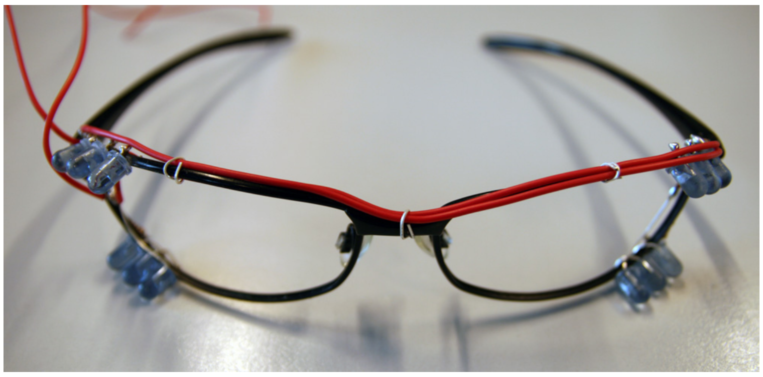
Figure 4.5: Our second pair of glasses

€0.20 These glasses turned out to work very robustly and we even were able to play a game with it in the end.

A problem we had was that the LEDs produced a glare. This sometimes confused our program. To diffuse the LEDs a little bit we ground the LEDs with sand-paper: this resulted in less glare and a bit diffused beam. Diffusing the LEDs resolved most of the problems. Another way of overcoming this problem is to buy diffused LEDs.

4.3 Software

4.3.1 Language and Library

We chose to use Java as programming language as we have quite a lot of experience in programming in Java. Using Java also enables us to program platform independent as well as make a platform independent application.

The WiiMote communicates via Bluetooth with the Wii. Luckily for us Windows automatically detects the WiiMote as an input device. For our project we use a library called WiiUseJ, which is a Java wrapper class for the C library WiiUse [2]. This library enables us to use Java to communicate between the PC and the WiiMote. The library has many built-in functions for all buttons and sensors. IR-LED source data is delivered raw: it gives us the numbers corresponding with up to four points of IR-light it sees at that specific time.

4.4 Detecting Degrees of Freedom

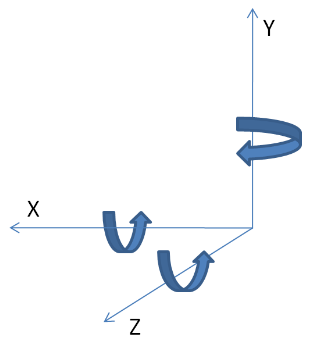

Figure 4.6: 6 degrees of freedom

The degree of freedom is a mathematical concept. Degrees of freedom (DOF) are the set of independent displacements and/or rotations that specify the displaced or deformed position and orientation of the body or system. In this thesis we will talk about degrees of freedom as being movement in our everyday 3D space:

- Movement along the x-axis, moving left and right, also called swaying
- Movement along the y-axis, moving up and down, also called heaving
- Movement along the z-axis, moving forward and backward, also called surging
- Rotation around the x-axis, tilting forward and backward, also called pitching
- Rotation around the y-axis, turning left and right, also called yawing
- Rotation around the z-axis, tilting side to side, also called rolling

Those are the six degrees of freedom we will try to be able to track using the WiiMote and IR-LEDs.

Our goal is to be able to measure as many degrees of freedom as possible. Since the WiiMote only sees an x and y position of the IR-points some smart interpreting of the points will have to be done, however, the related work shows us that three points should be enough for all six degrees of freedom. Having an easily expandable framework helps achieve this goal a lot easier as you can test small implementations first and go to more difficult and extensive implementations later.

Also one has to take in account that there is a big difference in absolute and relative movement. Positioning the mouse can easily be done by reading the x and y position of the head and mapping this to the screen, while gestures are complex relative movements, sometimes involving multiple degrees of freedom. We will try to implement the movement of the mouse by moving your head, as well as emulate clicking by detecting nodding of the head. If needed, the software can be extended easily to track other gestures.

4.5 Interpreting Four 2D Points to Six Degrees of Freedom

Since Yuri Meiburg concentrates on the filtering, we will not handle the filtering extensively. In the following part we assume the data is filtered (i.e. smoothed and in a constant stream).

In order to translate the four points to the six degrees of freedom we distinguish two cases: absolute and relative movement. Absolute movement can be a 1-to-1 mapping between the resolution of the WiiMote (1024×768 "pixels") and the resolution of the screen. Hence the formula for swaying $p(x) = c \times a \times x$, where $p(x)$ is the horizontal position of the cursor, c is a constant acceleration factor, a is the factor between the horizontal resolution of the screen and the Wiimote it's horizontal resolution. x is the measured absolute horizontal position of the (averaged and filtered) input points.

Likewise, the formula for heaving is $p(y) = c \times b \times y$, where $p(x)$ is the vertical position of the cursor, c is a constant acceleration factor, b is the factor between the vertical resolution of the screen and the WiiMote it's vertical resolution. x is the measured absolute vertical position of the (averaged and filtered) input points. The other four degrees

Detecting movement along the z-axis can be quite tricky, as you have little depth information (the WiiMote does not have a very usable intensity detection). Therefore we will implement gesture detection by just looking at change of the position of the LEDs in the x and y-axis.

In order to detect (fast) gestures, like nodding and shaking, we need to look at relative movements. A nod (a quick pitching movement) equals a quick movement of all four points along the y-axis. Therefore, we constantly measure a fixed number of previous points and check whether the last movement has been a nodding movement (i.e. a fast downwards movement, followed by a fast upwards movement).

Detection of movements in these directions can be coupled easily to output, for instance using the `java.awt.Robot` class, to mimic cursor movements and/or keyboard keystrokes. Implementation of the interpreters can be found below in the Implementation and Choices chapter.

In figure 4.7 you can see how the 6 degrees of freedom can be seen by the WiiMote:

4.5.1 Swaying

Swaying is movement along the x-axis, i.e., moving left and right. As figure 4.7A shows, swaying can be detected by looking at all four points; when all four points move left (or right) at the same time, you detect swaying. Since all four points are moving in the same direction, you could also look at the average position of all four points, and check whether they are all moving left or right.

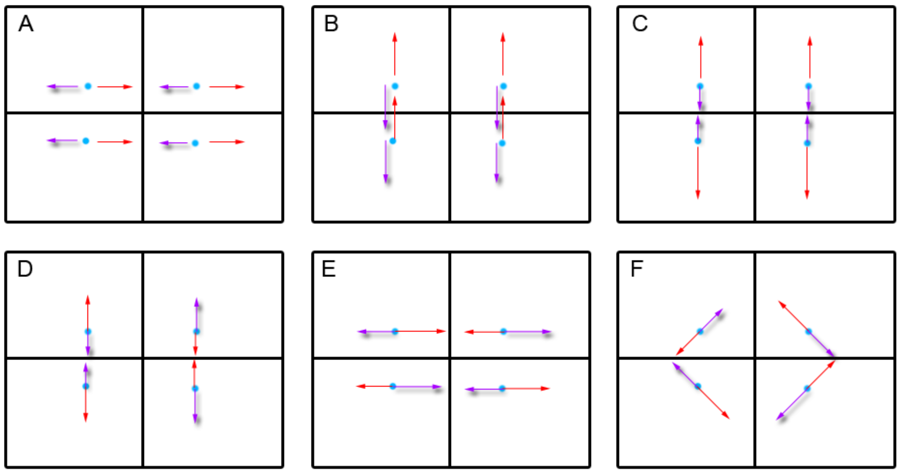

Figure 4.7: 6 degrees of freedom as seen by the WiiMote

4.5.2 Heaving

Heaving is movement along the y-axis, i.e., moving up and down. Heaving can be detected in the same way as swaying, only this time in the vertical direction in stead of the horizontal direction: when all four points move up (or down) at the same time, you detect heaving (see figure 4.7B). The same holds for heaving: since all four points are moving in the same direction, you could also look at the average position of all four points, and check whether they are all moving left or right.

4.5.3 Surging

Figure 4.7C shows surging: movement along the z-axis, i.e., moving forward and backward. This movement can be detected by looking at the distance of the upper two points relative to the lower two points, and the distance between the two left points and the two right points. When moving forward (i.e. towards the WiiMote) the distance will become larger, that is, the points will move away from each other. Likewise, when moving backwards (i.e. away from the WiiMote), the distance will become smaller.

4.5.4 Pitching

Pitching is rotation around the x-axis, i.e., tilting forward and backward, or nodding. The effect of nodding on the received position of the four points is that the points will all move up (or down) while also decreasing the distance between the upper and lower two points (see figure 4.7D). This is quite hard to detect, since it is hard to see the difference with surging combined with heaving. Also, the difference in distance between the upper two and the lower two points is small and hard to measure. Therefore we only detect fast nodding as being rotation around the x-axis. We measure the speed of the movement; slow movement is perceived as swaying, while fast movement is perceived as pitching.

4.5.5 Yawning

Yawning is rotation around the y-axis, i.e., turning left and right. This rotation can be detected by looking at the distance between the two left points and the two right points, combined with the position of all four points (see figure 4.7E). When the points move towards each other and overall moving to the left or right, you are detecting yawning. Just like pithing, only fast movements will be detected as yawning, slow movements will be detected as heaving.

4.5.6 Rolling

Rolling is the rotation around the z-axis, i.e., tilting side to side. To detect this movement you have to look at all four points and see if the positions of the four points correspond to either rolling to the left or rolling to the right. When rolling to the left, the upper left point will go towards the lower left, the upper right point to the upper left, the lower right point to the upper right and the lower left point to the lower right. When rolling to the right, the upper left point will go towards the upper right, the upper right point to the lower right, the lower right point to the lower left and the lower left point to the upper left.

4.6 Summary

To do head tracking we need to:

- Communicate through Bluetooth with a WiiMote to get the input data from the WiiMote
- Filter the data
- Interpret the data
- Give output

Detection of six degrees of freedom is possible with four IR-points.

Chapter 5

Implementation and choices

As we have to write different theses, Yuri and I have both “specialized” in making different parts of the software. This chapter will describe what we have done together and what parts Yuri specialized in, and which parts I did.

5.1 Structure

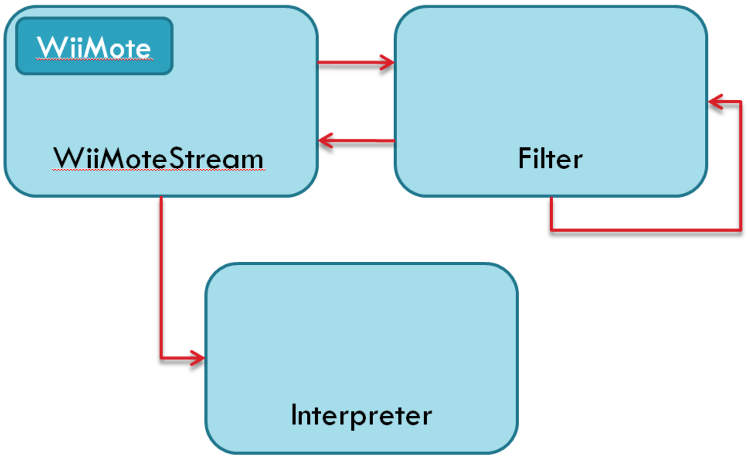

Figure 5.1: The global structure of our program

In figure 5.1 the global structure of our program is shown. It works as follows: The WiiMote keeps sending the detected points to the WiiMoteStream. The program is then split into two big parts: the filters and the interpreters. Yuri concentrated on the filters whereas I concentrated on the interpreters.

The WiiMoteListener listens to packets received over Bluetooth and gives these to the WiiMoteStream. The WiiMoteStream supplies us with a constant (i.e. always available) stream of points. If the WiiMote cannot find any points, the last saved position of the points will be sent instead. This guarantees that for the interpreters and filters that a stream is always available. Interpreters interpret the points in different ways. We made an interface for interpreters, so we know that some functions will always be available for interpreters.

Filters get points from the stream and modify them in some way. There also is an interface available for the filters. Implementation of interpreters and filters is explained below.

As we supply interfaces for the filters and interfaces, new filters and interfaces can be easily added tot the program. The framework is highly extensible and different ways of interpreting can easily be implemented and attached to the program.

5.2 Filters

For the filters, we built an interface providing the needed (yet to be implemented) functions. Every filter inherits from this interface. Because this is Yuri his specialization, we will only briefly discuss the filters here.

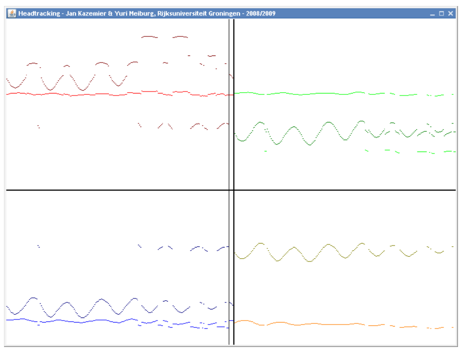
Headtracking - Jan Kazemier & Yuri Meiburg, Rijksuniversiteit Groningen - 2008/2009

Figure 5.2: SinglePointFilter output

Our first filter is called `SinglePointFilter` and just takes the stream and gives it directly to the interpreter (i.e., it does not modify the stream). Also see figure 5.2.

The second, little bit more interesting filter is the `NPointsAverager`, also known as a Kalman Filter [16]. This filter averages n points before giving it to the stream. This, however, needs a small buffer. On the other hand this does smooth the mouse quite a lot, see figure 5.3. We experienced the best results when using about fifteen to twenty samples.

We thought about implementing some more difficult ways to filter the points, but the `NPointsAverager` already gave a quite acceptable result.

In order to emulate mouse clicks Yuri wrote the `HighPassFilter`. This filter is able to detect the nodding of the head or other gestures. We mapped the nodding-movement of the head to a mouse click (in an interpreter). The `MouseFilter` acts upon moving just

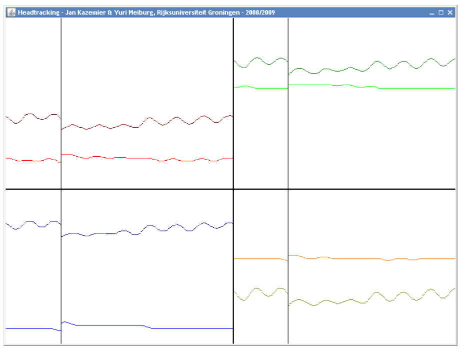
Headtracking - Jan Kazemier & Yuri Meiburg, Rijksuniversiteit Groningen - 2008/2009

Figure 5.3: NPointsAverager output

a bit in the way of nodding. When a nod is suspected the `MouseFilter` gives a corresponding output to the interpreters. Note that gestures are relative movements whereas the normal mouse movement are absolute movements (with respect to the screen or the WiiMote).

5.3 Interpreters

After filtering the input of the WiiMote, the interpreters will receive the filtered data and will interpret the data in order to control the PC. In a first attempt to see everything was working correctly we have created a simple `Monitor`-interpreter, which plots the positions of the first IR-point the interpreter gets from the filters to the screen.

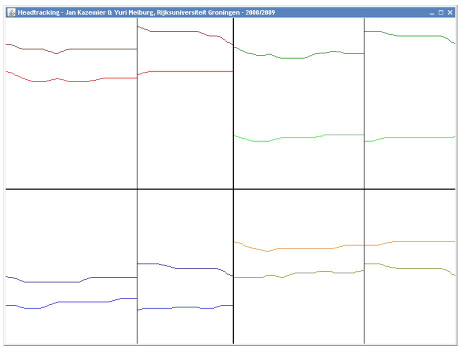
Headtracking - Jan Kazemier & Yuri Meiburg, Rijksuniversiteit Groningen - 2008/2009

Figure 5.4: The QuadMonitor

Later on we have created a bit more complicated monitor called `QuadMonitor`. This interpreter takes the input from the filter(s) and plots the position of of all four points to the screen (see figure 5.4). This QuadMonitor enables us to see what the values of the data are when the IR-points are on a certain position.

The implementation of swaying and heaving was fairly simple, the output (a moving cursor in this case) was just a simple mapping of the absolute x and y position of the average of the four measured points. The formulas are described in the Concept chapter.

To be able to click, we implemented the nodding gesture. A high-pass filter (called `MouseFilter`) is used to detect fast vertical movement. This information is then passed to the interpreter. To test the `MouseFilter` we created a `ClickMonitor`-interpreter. When the filter sees a click, the monitor shows that a click was recognized. Further testing showed that nodding can work just fine as a clicking mechanism. A problem we had though, was that nodding would also move the mouse away from the place you wanted to click. Therefore we first go back to the position of the supposed start of the nodding, *freezing* the movement and click on the spot where the click started.

Since our framework can be easily extended, detection of other gestures can be implemented fast. In the Detecting Six Degrees of Freedom chapter you can see what movements you should track to detect gestures in that Degree of Freedom. Since we only wanted to replace the mouse, we have not implemented any other gestures than nodding, however, implementing other gestures can be done easily by adapting the detection of the nodding gesture.

5.4 Configuration

A tiny part of my specialization was to make a configuration manager. A configuration file, `config.ini`, holds various settings for the program: you can set which interpreter to use, which filter(s) to use, and some interpreter-specific settings as well. These settings are then read by the `ConfigurationManager` and used in the program. When no values are set, defaults are used.

A simple example of `config.ini`

```
[Window]
monitortype = mouse
width = 800
height = 600
```

Chapter 6

Case: Playing a Game

When we finished creating a working mouse-replacing head tracking system, we decided to try it in a game. We chose the game Portal, by Valve Software [15], because it has a low-pace action, while still walking in 3D.

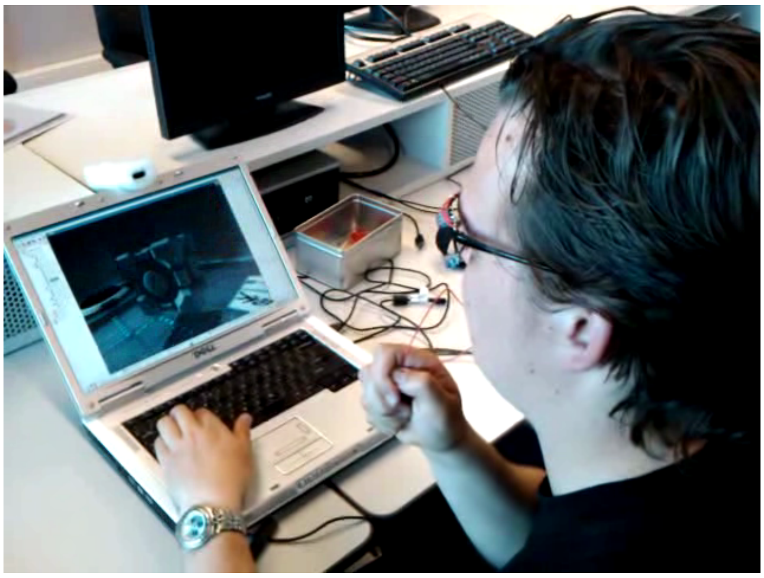

Figure 6.1: A picture of me playing Portal with my head

Of course, it is not necessary to play the game with your head, however, the game was very playable. As a demo we showed the game at our Bachelor Presentation about this project. The game does not need any adaptations, all you have to do is use your head instead of the mouse. The sensitivity of the mouse can be changed in the in-game menu; it is advised to put it to a very low sensitivity, although depending on the settings of our software.

Moving your head to a certain direction, or rotating your head to that direction was mapped to looking in that direction in the game. In the game you can pick things up by pressing a pre-configured button. We mapped nodding of the head to this button (i.e. nodding your head resulted in picking up an object). Playing the game this way is very much fun, and does not make it very much harder. We think playing faster paced action games will be too hard, as you lose some fast precision.

Playing Portal with your head is a lot of fun. To have a less biased person try it, we have asked some classmates to play the game, without explaining what to do. They immediately were able to play the game, just as well as me. Therefore, we presume the head tracking input for a game like Portal is quite intuitive and can be much fun.

Chapter 7

Future work

We have implemented a head tracking solution in order to be able to control the PC with your head. In this system the head replaces the mouse as an input device. Clicking is done by nodding. Of course this project can be extended for other uses besides controlling a PC. A few examples of work that can be done based on our work:

- Implement more gestures besides nodding, this can be done by simply adapting the nodding-detection to detect gestures in different degrees of freedom
- Create a data-logger to be able to use the head-tracking data in research
- Adapt the program to be able to create a Virtual Reality, for instance the window-like implementation Johnny C. Lee has done
- Create an on-screen keyboard to also be able to type with your head

Chapter 8

Evaluation and Conclusion

In this project we have had a few problems. The problems we had, could be resolved: one of the problems was a wrong detection of the position due to glare of the LEDs. This problem was mostly resolved by grinding the LEDs. Another solution would be to use (more) diffuse LEDs. Even better results were obtained using twelve LEDs in stead of four, as this gave us a larger angle in which the LEDs could be seen by the WiiMote. Another solution would be to use LEDs with a larger angle.

Another problem was that we did not know how to measure relative movement. We found that relative movement can be measured using a small data-history.

One of our goals was to work platform-independant. We think using Java was a fine choice as this allowed us to use the WiiUseJ library and it also enables us to work platform-independent.

We think our system would be suitable to be shipped with games or used in other applications like research, virtual reality or full-control of a PC (including replacing the keyboard) with only small adaptations. Improvements to our software can be made by adding more gestures or implementing an on-screen keyboard.

We wanted to make a low-cost head tracking solution to control a PC with your head. Also we wanted a platform independent, extendable framework. We think we succeeded in that, our program is written in Java (using the WiiUseJ library), therefore it is platform-independent. Also, our framework is easily extendable. Another goal was to make it low cost. Our solution is relatively cheap to the commercially available products. As we have shown, our software is able to replace the mouse of the PC by detecting the position of the head, while clicking can be performed by nodding. Other actions can easily be added by mapping gestures or movements to certain functions in programs.

As we were able to play a game with our head, we think we have shown that creating a low-cost eye-tracking-like way to control a PC can be very well done by using a WiiMote and some glasses with IR-LEDs. Also, using your head to play a game is a lot of fun and is a nice result of this project. Using a Wii-Mote and cheap IR-LEDs on glasses could work just as well as other commercially available head-tracking product.

Working on this project was a lot of fun, although it took quite a lot of time. Thanks again to our supervisors for helping us greatly with this project.

Source code is available at request.

Bibliography

[1] SR Publications website, August 2009. URL `http://www.sr-research.com/publications.html`.

[2] Guilhem Duché. The WiiUseJ library website, August 2009. URL `http://code.google.com/p/wiiusej/`.

[3] Hu Fengpei. The studies of eye tracking and usability test. In *Computer-Aided Industrial Design and Conceptual Design, 2006. CAIDCD '06. 7th International Conference on*, pages 1–5, Hangzhou, Nov. 2006. IEEE. ISBN 1-4244-0683-8. doi: 10.1109/CAIDCD.2006.329343.

[4] Michael Gleicher. Animation from observation: Motion capture and motion editing. *SIGGRAPH Comput. Graph.*, 33(4):51–54, 2000. ISSN 0097-8930. doi: 10.1145/345370.345409.

[5] George Hsu and J. James. A sourceless, low-cost head tracker for virtual reality head mounted displays. In *WESCON/'95. Conference record. 'Microelectronics Communications Technology Producing Quality Products Mobile and Portable Power Emerging Technologies'*, pages 706–708, San Francisco, CA, USA, Nov 1995. IEEE Computer Society. doi: 10.1109/WESCON.1995.485487.

[6] Sami Huttunen and Janne Heikkila. An active head tracking system for distance education and videoconferencing applications. In *Video and Signal Based Surveillance, 2006. AVSS '06. IEEE International Conference on*, volume 0, page 30, Los Alamitos, CA, USA, Nov. 2006. IEEE Computer Society. ISBN 0-7695-2688-8. doi: 10.1109/AVSS.2006.19.

[7] J. Jacobson and M. Lewis. Game engine virtual reality with CaveUT. *Computer*, 38(4):79–82, April 2005. ISSN 0018-9162. doi: 10.1109/MC.2005.126.

[8] Jan Kazemier and Yuri Meiburg. WiiView project website, 2008-2009. URL `http://www.cs.rug.nl/~isenberg/interaction/pmwiki.php/ProjectGallery/2008-2009-WiiView`.

[9] J.C. Lee. Hacking the Nintendo Wii remote. *Pervasive Computing, IEEE*, 7(3):39–45, July-Sept. 2008. ISSN 1536-1268. doi: 10.1109/MPRV.2008.53.

[10] J.C. Lee. Wii projects website, August 2009. URL `http://johnnylee.net/projects/wii/`.

[11] Yuri Meiburg. Low-cost eye-tracking-like control of a PC: Communication and filtering. Bachelor thesis, Rijksuniversiteit Groningen, the Netherlands, Oktober 2009.

[12] Alberto Menache. *Understanding Motion Capture for Computer Animation and Video Games*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, October 1999. ISBN 0124906303.

[13] J. Rekimoto. A vision-based head tracker for fish tank virtual reality-vr without head gear. In *Virtual Reality Annual International Symposium, 1995. Proceedings.*, pages 94–100, Los Alamitos, CA, USA, Mar 1995. IEEE. doi: 10.1109/VRAIS.1995.512484.

[14] Oliver Smith. An investigation into the use of a webcam for use as an HID for desktop virtual reality applications. Master's thesis, Northumbria University, 2009.

[15] Valve Software. Portal, the game. website, August 2009. URL `http://www.whatistheorangebox.com/portal.html`.

[16] M. St-Pierre and D. Gingras. Comparison between the unscented Kalman filter and the extended Kalman filter for the position estimation module of an integrated navigation information system. In *Intelligent Vehicles Symposium, 2004, IEEE*, pages 831–835, June 2004. doi: 10.1109/IVS.2004.1336492.