rijksuniversiteit groningen

2008-2009

Low-cost eye-tracking-like control of a PC:
Communication and Filtering

Yuri Meiburg

BACHELOR PROJECT

Supervised by: dr. T. Isenberg and dr. M. Wilkinson

15 / 07 / 2009

Yuri Meiburg
1634518

Preface

After having followed the course *Innovative Interactive Systems*, given by Isenberg at the Rijksuniversiteit Groningen, I became very interested in using the Wiimote as a controller for the PC. During the course we had to create something innovative and interactive. We (Jan Kazemier and I) chose an image viewer which we have built in Java, and controlling the application would happen solely with the head. As the end of the course came nearer we came to the conclusion that head tracking definitely had potential, but needed much more refinement. The application we delivered was far from what we planned at the beginning of the course, leaving us a bit disappointed. Then at the end of our third year we had to decide what to do for our bachelor project. None of the given possibilities seemed appealing, and it did not take long before we tried to use head tracking for our bachelor project. Now, half a year later, things are finally rounded up, and we are still very pleased with the subject we had. Still, we did not get to where we thought we would be (just as with the image viewer), but we did get much nearer to our goal. This time we made a functional application, and we have demonstrated its usage by playing the game *Portal*, from Valve Software.

All in all this was a really interesting experience and I want to thank the following people, for greatly helping me write this thesis:

Many thanks to **Jan Kazemier**, for being a friend, offering moral support and for working on this Bachelor Project together with me.
Many thanks to **Samantha Kalwij**, for being a great supportive girlfriend, really getting me motivated at times where needed, and showing tremendous amounts of patience whenever needed.
Many thanks to **Dr. Tobias Isenberg**, for being our first supervisor, for supplying materials and numerous great ideas and for helping greatly us with the project and theses.
Also thanks to **Dr. Michael Wilkinson**, for being our second supervisor.

Yuri Meiburg

Abstract

The traditional way of controlling a computer is to use a keyboard and a mouse. However, there are circumstances in which this traditional way of controlling a computer is not sufficient, or not preferred. For instance, people may have a disability preventing them from using the computer. Alternatives are available at the time of writing, but are mostly very expensive. So far we have found solutions starting from $150, up to $25,000. These will be described in the section *Existing Methods*. We aim to create a cheap alternative to the computer mouse. We will discuss some alternatives currently commercially available and why they do not meet the standards we have set for our project.

The choice of hardware for this project is simple. Using only a Wiimote, and glasses containing infrared LEDs, we will provide the software to control a PC. By making people wear the glasses, the Wiimote will track the points of the glasses, corresponding to the head of the actor. These movements are then filtered and translated to the mouse controls. There are also some gestures which can be used to trigger certain actions (e.g. mouse clicks). In order to achieve these goals, I discuss the communication with the Wiimote, how to manipulate preprocessed data to make mathematical filters more reliable, how to filter the data, and the structure of the program.

Contents

1 Introduction **4**

2 Related Work **6**
- 2.1 Non-optical methods . . . 6
- 2.2 Optical Methods . . . 7
  - 2.2.1 Passive Markers . . . 7
  - 2.2.2 Active Markers . . . 8
  - 2.2.3 Markerless Systems . . . 9
  - 2.2.4 Eye tracking . . . 9
  - 2.2.5 Head tracking by Lee . . . 10
- 2.3 Summary . . . 10

3 Design Approach **12**
- 3.1 Wiimote . . . 12
- 3.2 Glasses . . . 13
- 3.3 Software . . . 14
  - 3.3.1 Library WiiUseJ . . . 14
- 3.4 Conceptual Software Design . . . 16
- 3.5 Summary . . . 16

4 Software Implementation **18**
- 4.1 Structure . . . 18
- 4.2 Communication with the Wiimote . . . 19
- 4.3 Filters . . . 20
- 4.4 Interpreters . . . 21
- 4.5 Configuration Manager . . . 22
- 4.6 Summary . . . 22

5 Evaluation **23**

6 Conclusion **24**

Bibliography **25**

Chapter 1

Introduction

Head tracking, or capturing the motion of the head, can be done using a wide range of methods, and has been used for research purposes since the late 1970s. Apart from research, head tracking can have numerous applications, such as capturing movements for movies, or providing additional input in games [12], [6]. Even though games continue to become more realistic, head tracking is still not implemented by default. Current head tracking solutions for personal use are quite scarce and expensive. If head tracking is easily available for everyone and would cost less this could be the next step in gaming.

Lee, a researcher working at Microsoft, previously demonstrated a new head tracking implementation, using the Wiimote from Nintendo [10]. The Wiimote is the controller from Nintendo's gaming console Wii. This controller features an infrared camera, specifically designed to track infrared points. By holding the Wiimote and pointing at stationary infrared LEDs, the Wiimote tracks these points and calculates its location.

In the setup of Johnny Lee the Wiimote is used the other way around (see Figure 1.1. An actor wears a pair of glasses, equipped with infrared LEDs at the side, and the Wiimote is put stationary so the actor is visible. Instead of measuring the movement of the Wiimote, the Wiimote now tracks the motion of the glasses, and thus tracks the motion of the actors head.

Figure 1.1: A screenshot of a video by Lee, demonstratating head tracking using the Wiimote. Source: `http://johnnylee.net/projects/wii/`

Our bachelor thesis is about head tracking, inspired by the set up used by Johnny Lee. The goal of this joint project was to develop a head tracking method, capable of tracking 6 degrees of freedom, that is low cost and that easily integrates in other applications. Besides the additional 6 degrees of freedom, we have also developed gesture based control. This resulted in the development of a new application in Java, using the Wiimote stationary on top of the screen, and glasses having infrared LEDs in 4 corners.[1] While Jan Kazemier focused on constructing the glasses and creating the interpreters for the data, this thesis focuses on the general structure of our application, communication with the Wiimote, and the preprocessing of the data.

[1] As a demonstration, we used our application to partially control the game Portal from Valve Software. A video can be found at: `http://www.yurimeiburg.nl/thesis/absmov.wmv`, and: `http://www.yurimeiburg.nl/thesis/relmov.wmv`

Chapter 2

Related Work

Motion tracking (or motion capture) dates back to around the late 1970s, when it was used as an analysis tool for biomechanic researchers. Nowadays, there are quite a few variants. In this section we will discuss the various methods existing to perform head tracking. We will start off discussing the non-optical methods, and then discuss the more common optical methods. In the optical methods we've added a section dedicated to Lee's variant, as it is really similar to our approach. Please note that whenever we reference to an 'actor', we reference to the person which is undergoing motion capture.

2.1 Non-optical methods

Non-optical methods do not require any light or its reflection to measure the position of a point. There are various techniques which work without optical feedback, of which we will discuss three common techniques. The first is using so called "inertial" methods. This means that instead of using light, gyroscopes are used to measure the positions [20]. Usually the resulting data is transmitted wireless to a nearby computer, which is able to transform the data to a skeleton (if used as full body tracking). This method is really accurate for relative movements, as the gyroscopes are able to measure up to a precision of roughly 1° [13]. The disadvantage is that this method is less suitable to measure exact locations, as it suffers from the "drift problem". This means that without external correction measurements could drift off from the truth due to noise and offsets [22]. Due to accelerometers being lightweight and compacht, they are often accomodated in portable services, such as mobile devices [22].

Another method is the use of mechanical systems, and is only usable for certain types of motion tracking. Typically, an exoskeleton is used for an actor, where measurements are performed along the joints. Now when an actor moves (and thus moves joints), so will the exoskeleton with mechanical joints [11]. This data can be used to measure relative movement. This method nowadays is wireless as well, so using a PC as receiving end ensures a large data capacity. The big advantages of this type of system is that it is free of occlusion. The disadvantage is that the exoskeleton that needs to be worn by the actor might hinder his natural movement. An example is shown in Figure 2.1.

Figure 2.1: An exo-skeleton suit, courtesy of Gypsy Gyro Motion Capture System [5]

A third common technique is using magnetic systems. The system is capable of measuring 6 degrees of freedom by measuring the relative magnetic flux of three orthogonal coils on both the transmitter and each receiver. One of the mayor drawbacks of this system is that it is highly influencable by magnets or large pieces of metal [13]. Aside from that the measurement is non-linear, especially near the edges, and the wiring of the sensor often prevents extreme movements. These drawbacks require extensive calibration [22].

2.2 Optical Methods

Using infrared light there are various techniques which are capable of tracking motions of an actor. This section covers these different techniques.

2.2.1 Passive Markers

Passive markers are the easiest to carry for the actor. Passive markers are no more than reflective dots which are placed on a suit of the actor. Using infrared light emitters and special infrared cameras, the reflective points will show up as bright points for the camera, while all other parts remain black. These dots are then connected to each other, forming a skeleton.(see figure 2.2). Usually there are between 1 and 24 cameras available to get a full 360° model [4]. One of the problems which could occur is marker swapping, a phenomenon which occurs because there is no way to distinguish different markers, apart from their location [20]. One of the solutions is to use more cameras. The main

advantage of passive markers compared to active markers is the lack of wires on the suit, as the dots do not have to emit light, but merely reflect [20].

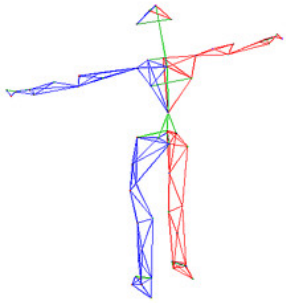


Figure 2.2: A skeleton reconstructed from an actor wearing markers.

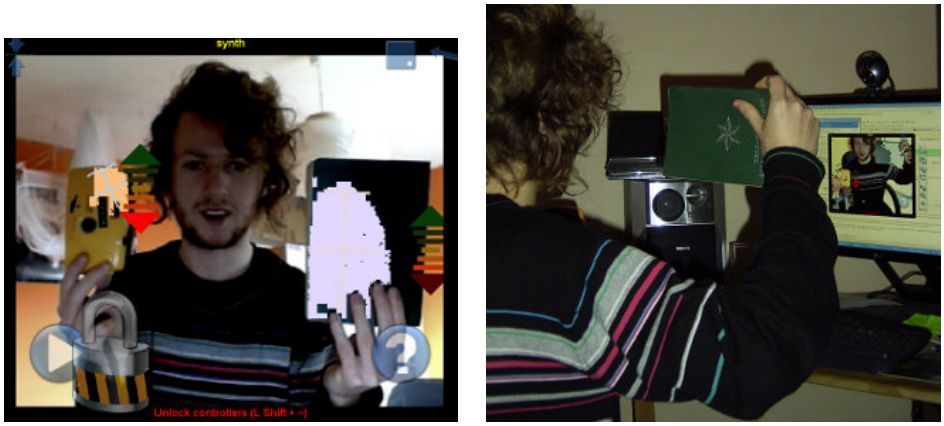
2.2.2 Active Markers

An active marker system uses LEDs which light up rapidly one after another (or multiple at once), in order to simplify the distinguishing of different points. The corresponding software then uses triangulation to calculate the 3D position of the point [1]. Using synchronization to control the LEDs the marker swapping problem (which occurs with passive markers) can be greatly reduced [4]. Another advantage is that because active markers emit light, an actor can stand further away from the cameras than an actor with passive markers. Unintended additional reflections are also brought to a minimum, which could, for instance, arise when an actor wears glasses. The disadvantage is that each point also brings additional cables, as the LEDs need to be controlled and need power. We use active markers for this project, as it provides a more reliable beam of light than passive markers, and the additional cables are not a problem, as shown in Figure 2.3.

Figure 2.3: The glasses we use for our head tracking project.

2.2.3 Markerless Systems

Another category of motion tracking systems are markerless systems. These systems usually do not require any kind of suit, but use algorithms to distinguish a person or object from the background. There are several solutions commercially available using markerless systems, requiring only the software and a webcam. An example of a free version is CamSpace.[1] CamSpace works by first calibrating the software, so it knows what to recognize. To achieve this, the software should know what object should be tracked. This is done by showing the actor the input CamSpace retrieves, along with the outline of a square in the center of the image. The actor must place the object within that center, so CamSpace can analyze the object and track it. If the calibration succeeds, the object will be lit up and be tracked. CamSpace will then output coordinates of the object, along with some extra information such as rotation. These coordinates can then be mapped to alternative actions by the end user, for instance to a MIDI-composer, as shown in Figure 2.4.[2] For an extensive survey of various object tracking methods, see [21]



(a) Camposer screenshot (b) Camposer set up

Figure 2.4: Camposer in action

2.2.4 Eye tracking

A special case of motion capturing is eye tracking, which works by putting some device in front of the eye of an actor. This can be done using a helmet, or by making the actor look into a device. This device then measures the position of the iris and with that information it is able to capture where this person is looking at. Published work

[1] The CamSpace software is available at: `http://www.camspace.com/`

[2] The MIDI-Composer is a project from W. van Ackooij and T. Klein. More information on this project is available at: `http://www.cs.rug.nl/~isenberg/interaction/pmwiki.php/ProjectGallery/2008-2009-Camposer`.

of eye tracking studies date back to the second half of the 1800's, such as [7]. These studies were made using direct observations. The first system capable of performing eye tracking was created around 1930, by Tinker (see [15] for a thorough review of this work). This system worked by shooting beams of light at the pupil which reflected and were captured on film. Since then eye tracking has evolved a great deal, as shown in figure 2.5. An example is EyeLink[3], which is commercially available, and is capable of performing extremely accurate eye tracking. This device is able to measure the position of the iris at up to 2000Hz. Unfortunately, this product costs around £45.000.

Figure 2.5: Historical versus modern head tracking.

2.2.5 Head tracking by Lee

Lee, a researcher working at Microsoft, showed a new method to perform head tracking, using low cost hardware [10]. The set up from Lee makes use of the Nintendo Wiimote. The controller of Nintendo's game console, the 'Wii'. The Wiimote is equipped with an infrared camera, capable of capturing up to 4 infrared points at 100Hz. The Wiimote is placed at the bottom of a screen, in such a way that it points towards the user. By wearing glasses equipped with infrared LEDs the Wiimote picks up the infrared points on the glasses. This means that the movement of the glasses is tracked, and thus the movement of the head. This technique is an example of a marker system, using low cost hardware components. It can be executed using either passive [14] or active markers [19]. The setup of Lee is not able to capture all 6 degrees of freedom, because it uses only 2 LEDs. The application Lee developed uses the rule that the actor always looks at the screen, so 3 degrees of freedom are obtained.

2.3 Summary

There are 2 types of head tracking. Systems using markers and markerless systems. Markerless systems are either really expensive, such as the EyeLink at £45.000, or not

[3]More information on EyeLink is available at `http://www.sr-research.com`.

very suitable for head tracking. Systems using markers show more potential for head tracking. We only need a few points to track the position of the head, enabling us to choose from active and passive markers. Active markers have the advantage of a higher reliability, while passive markers have the advantage of being wireless. Lee used an active marker system, using low cost hardware, which shows the most potential. This system, in its current state, is not capable of tracking 6 degrees of freedom, but by using more LEDs we will be able to track all 6 degrees of freedom.

Chapter 3

Design Approach

For this project we chose to work with the Nintendo Wiimote, combined with infrared LEDs which we have mounted on a pair of glasses. We have mounted the Wiimote on top of our monitor, so that instead of monitoring the movement of the Wiimote, it now monitors the movement of the glasses.

3.1 Wiimote

Figure 3.1: The Wiimote, as released by Nintendo in 2006.

The Wiimote (figure 3.1) is one of Nintendo's controllers released in November, 2006. This controller is the first controller to primarily rely on motion detection. The Wiimote is the primary controller for the "Wii" console, and has since its release received much attention due to its innovative features. It has also received a lot of attention from people who aim to use it as a controller for non Wii-related devices. The Wiimote connects through blue-tooth and communicates using the standard HID-profile (The HID V1.11 specification can be found at [16]). This advantage lead to the development of many libraries, to simplify the connection and communication with the Wiimote.

The big advantage of the Wiimote is the price/performance ratio. The Wiimote sells for around €30, but offers a high speed infrared camera and gyroscopes to measure absolute and relative movements. The infrared camera is able to track up to four points at a rate of 100 Hertz, with a resolution of 1024×768. The price of the Wiimote, along with the specifications of the camera and the great library support, are the reasons we decided to work with this device.

One of the disadvantages of the Wiimote is the fact that none of the libraries available are from Nintendo. This means that all knowledge of the Wiimote is reverse-engineered, and they are far from perfect. In practice this means that the Wiimote sometimes refuses to connect until a full reboot of the computer has been issued, or it refuses to send the position of the infrared LEDs until the project has been restarted. This unreliable behavior restrains developers from commercializing their Wiimote based software, as the frequency of these occurrences is absolutely intolerable for commercial applications. It is, however, very reliable on the Wii itself, leading to believe that this is a maturity issue of the libraries.

3.2 Glasses

The Wiimote is able to track up to 4 points, but there are 10 LEDs in the original sensor bar. They are placed as shown in figure 3.2. The reason for this is that when the LEDs are close enough, the Wiimote interprets them as one bigger point. Because of this feature we can place multiple LEDs next to each other to ensure a broader field of view, which figure 3.2 also demonstrates.

Figure 3.2: Representation of the sensor bar, two corners, 5 LEDs each, on the left an example of the beam of light of 5 LEDs compared to 1 LED.

We decided to mimic this, and used 12 LED's on our glasses. The glasses consist of a frame of a pair of sunglasses, and 3 LEDs in each corner. The LEDs are all placed in a different angle, just like the sensor bar. We used this feature because the beam of light emitted by the LEDs we have is very narrow. On the first pair of glasses we made we only used 4 LEDs. The result was unworkable because of the narrow beam of light. Our solution was to grind the LEDs, to diffuse the light. This works really well, but when we remade our glasses and built the final version, we decided to add multiple LEDs *and* grind them. This way we can be reasonably sure each corner will be constantly visible, which is necessary for our algorithms to work as expected. Both the glasses are shown in figure 3.3 (for a more detailed explanation see [8]).

The advantage of using 4 corners instead of 2 is that we can now calculate additional

Figure 3.3: left: Our old glasses, right: Our new glasses

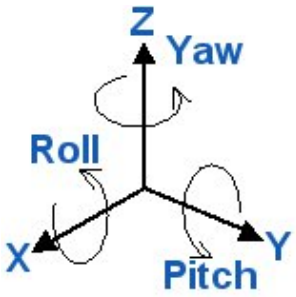


Figure 3.4: 6 Degrees of Freedom: X, Y and Z coordinates, along with Pitch, Yaw and Roll.

degrees of freedom. 4 LEDs enables us to distinguish 6 degrees of freedom, without the need of assumptions, such as "The user should always look towards the screen". The 6 degrees of freedom we can now track are shown in figure 3.4. How the positions of the LEDs change with the respect to the different degrees of freedom is shown in figure 3.5.

3.3 Software

The language we chose for our application is Java. We chose this language because it is an object oriented programming language, which helps us to create a clear structure with separate modules. This enables us to perform rapid prototyping, and simplifies programming simultaneously. Java also helps us to write a more stable program, for example by removing the ability to write/read arbitrary memory locations.

3.3.1 Library WiiUseJ

Another motivation for us to use Java which I have not previously mentioned is the existence of WiiUseJ. WiiUseJ is a layer on top of WiiUse, porting WiiUse from C to Java.[1] WiiUse (and thus WiiUseJ) are libraries which handle the communication with

[1]The official WiiUse site is available at `http://www.wiiuse.net/`, WiiUseJ can be found at `http://code.google.com/p/wiiusej/`.

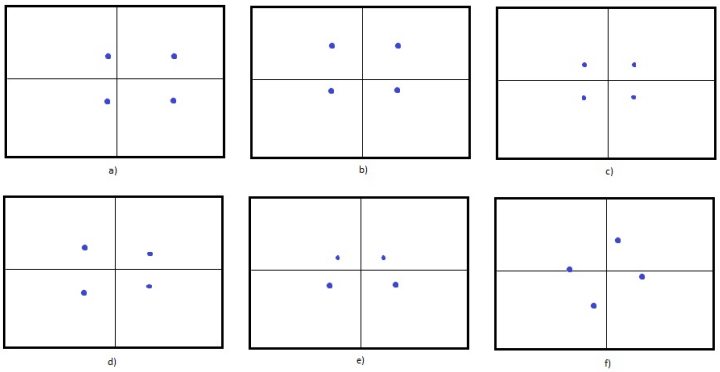


Figure 3.5: 6 Degrees of freedom, accomplished with 4 LEDs: a) X-axis, b): Y-axis, c): Z-axis, d): Yaw, e):Pitch, f): Roll

the Wiimote on the lowest level. To provide an example of the abstraction WiiUseJ offers, setting the player LEDs at the bottom of the Wiimote at the lowest level means writing an integer value to a specific memory address at the Wiimote. With the use of WiiUseJ this is simplified to the command `wiimote.setLED(led1, led2, led3, led4)`, where `led1` to `led4` are boolean values. This abstraction layer enables us to communicate with the Wiimote on a higher level, minimizing communication errors. A sample application to demonstrate the functionality of WiiUseJ is shown below.

```
1  import wiiusej.wiiusejevents.wiiuseapievents.DisconnectionEvent;
2  import wiiusej.wiiusejevents.wiiuseapievents.NunchukInsertedEvent;
3  import wiiusej.wiiusejevents.wiiuseapievents.NunchukRemovedEvent;
4  import wiiusej.wiiusejevents.wiiuseapievents.StatusEvent;
5
6  public class MyClass implements WiimoteListener{
7
8      public void onButtonsEvent(WiimoteButtonsEvent arg0) {
9      /* Shutdown if a is pressed */
10         System.out.println(arg0);
11         if (arg0.isButtonAPressed()){
12             WiiUseApiManager.shutdown();
13         }
14     }
15
16     public void onIrEvent(IREvent arg0) {
17     /* New inrared point(s) detected */
18         System.out.println(arg0);
19     }
20
21     /* Other required event handlers here ... */
22
23     public static void main(String[] args) {
24         Wiimote[] Wiimotes = WiiUseApiManager.getWiimotes(1, true);
25         Wiimote Wiimote = Wiimotes[0];
```

```
26            Wiimote.activateIRTRacking();
27            Wiimote.activateMotionSensing();
28            Wiimote.addWiimoteEventListeners(new MyClass());
29        }
30    }
```

The above code shows how to connect to the Wiimote, enable infrared tracking and the gyroscopes, and perform some simple actions. In this case when a new infrared point (or more) is detected, it prints all the information captured in the `IREvent`. And when the A-button is pressed, the connection with the Wiimote is closed. This shows that WiiUseJ provides an abstraction of the connection layer, enabling the programmer to focus solely on the incoming data from the Wiimote.

3.4 Conceptual Software Design

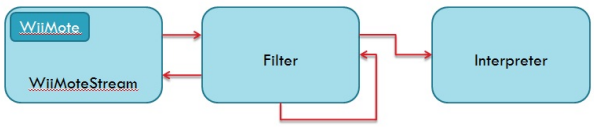


Figure 3.6: The 3 stages our application consists of.

Our software will consist of 3 major stages. Stage one is a further abstraction of the Wiimote, sending the newest points at a regular interval to our filters, which will preprocess the data. When done, it can send the data through another filter, to further preprocess the data. Once preprocessing is done, the resulting data is sent to the interpreter, which performs actions based on the incoming data. Dividing the application in these 3 stages simplifies expansion for future purposes, enables us to do rapid prototyping, as we can update each stage individually (or simultaneously), and provides the freedom of multiple filters and interpreters.

3.5 Summary

We will be using an infrared camera, installed stationary on top of a screen. An actor will wear glasses, containing 4 infrared points, 1 in each corner. The infrared camera we have chosen is the one equipped on the Wiimote, which is the primary controller of Nintendo's game-console 'Wii'. The Wiimote offers an infrared camera, capable of tracking up to four points at a rate of 100 Hertz, with a resolution of 1024×768, while remaining low cost. The glasses we have built contain 12 LEDs, 3 in each corner, enabling us to track X, Y and Z coordinates, and rotation along the 3 axes (roll, pitch and yaw). Because of the narrow beam of light emitted by the infrared LEDs, we have ground them, to create a more diffuse beam. The communication with the Wiimote will be done through the use of a Java library called WiiUseJ. WiiUseJ provides an abstraction of the communication

layer, helping us focus on the incoming data rather than the communication process. Our software will consist of 3 stages, communication with the Wiimote, preprocessing data, and interpreting data. The application is written in Java, because it improves the stability of our application, and the object oriented language helps us to program simultaneously.

Chapter 4

Software Implementation

4.1 Structure

The design of this software is very modular. A simplified class diagram is shown in Figure 4.1.

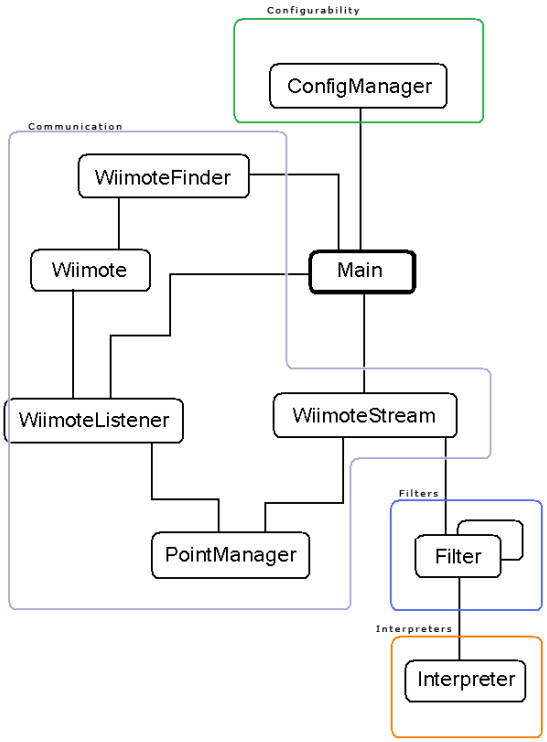


Figure 4.1: Structure of our program (simplified)

The software is divided in 4 sections, `Communication`, `Filters`, `Configurability` and `Interpreters`. The `Main` class couples the various components to one application. By using the structure as shown in figure 4.1 we get the freedom of piping filters, and using multiple interpreters.

Once the communication is set up, the Wiimote starts sending data, which is sent on to one or more filters. Once preprocessing is done, the data is sent to interpreters. The distinction of filters and interpreters is because it enables us to test various filters, while retaining the same functionality of the program. This also provides the ability to quickly change the interpreter, providing different functionality with the same data. Multiple interpreters are also possible, so we can monitor the data while the application is functioning. When working on the program, it became apparent that we changed a lot of variables all the time. As these variables were scattered throughout the program, the need for a central configuration arised. Therefore, another component called the `ConfigManager` was created. This functions as a central place where all classes obtain their parameter settings.

All in all this structure definitely cost more time to set-up, but proved worthy in the end, as adding, modifying and deleting filters and interpreters was very common, but thankfully quite easy.

4.2 Communication with the Wiimote

The Wiimote connects via bluetooth to the computer, which is all handled by WiiUseJ. Once the application is running with the Wiimote connected, it starts sending the location of infrared points. The Wiimote has the "feature" that it only sends new infrared points when locations change, up to a maximum of 100Hz. So it has varying intervals. The filters we use, however, solely rely on the location of the dots and not on the interval between two dots. These properties can give some odd results when combined, which lead to an abstraction of the Wiimote, called the `WiimoteStream` (as shown in figure 4.1). The `WiimoteStream` class is a thread which basically emulates the infrared part of the Wiimote, but with regular intervals. It retrieves the latest point as sent by the normal wiimote, sends it through the filter, and sends the result to the interpreter. It then checks how long it took, and sleeps for the rest of the interval, which is also set in the Configuration Manager.

The `PointManager` class, which is also shown in figure 4.1 is an object which does no more than provide an easy way to store and retrieve the latest data points. This intermediate class was created, because the behavior was unclear when the `WiimoteListener` would directly push the points to the `WiimoteStream` while it was sleeping. This has close to no overhead, but makes it easier to predict the behavior of the application on multiple platforms.

4.3 Filters

Filters need to extend the Filter interface, in order to ensure proper integration. The interface at this point requires a mere 2 functions, but for all purposes it has shown to suffice. The current interface looks like this:

```
1 public interface IFilter {
2
3     public void insertPoints(IRSource[] points);
4     public IRSource[] getPoints();
5 }
```

An interface must have a method to insert new points, and a method to retrieve processed results. We started out with the most basic filter, the filter which applies:

$$F(x) = x \tag{4.1}$$

Because we started off researching, we needed the raw, unfiltered data. The result was very noisy. Even if you try to keep your head at the same place, the data would still show a lot of movements. In order to filter this data we wrote a simple unweighted n-points average filter. This filter compares each of the points it receives with previous gathered data, and for each of the corners it separately calculates an average with the following simple formula:

$$F(x) = \sum_{i=0}^{N} \frac{history[i][x]}{N}, \tag{4.2}$$

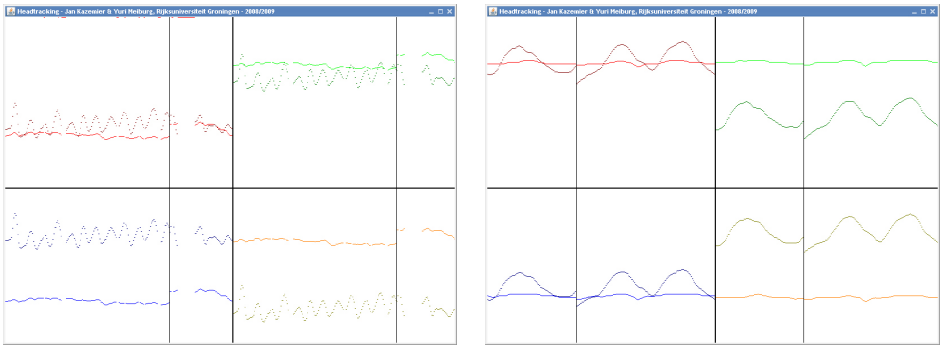
where x is the corner and N is the number of points to go back in the history. We took $N = 5$ in our program, as this is roughly the minimum for which the noise is filtered out. Taking higher values for N will decrease the latency up to a point where it becomes unworkable (at around $N = 50$). An example of the filtered data versus the raw data is shown in figure 4.2.

The filter alters each point individually, and sends the resulting points to an interpreter. This means that an interpreter receives up to 4 data points at a time. The position of the head is calculated in the interpreter, because doing so in a filter means removing data which could prove useful for debugging (seečitekazemier).

The next thing to accomplish is to make a filter which signals when it detects a gesture from the head. We started with 1 gesture, a fast vertical movement with the head. After each iteration of creating a suitable filter, we noticed there were a few requirements in order for it to work pleasantly:

1. Relative movements mean you need to keep track of previous points, so we had to create a history.

2. Fast movement detection means that we need to calculate Δy between each point. This Δy should be compared to a threshold.



(a) The raw data (b) N points averager (N=10)

Figure 4.2: Analyzing the incoming data.

3. Not all points seem to pass the threshold on a click, so we need to keep a history of passed items and apply a threshold on that value as well.

4. Detection is not fast enough, so you need a third value pointing out that some values might become a click.

5. If an x amount of intervals was a positive click in a history set, set the most recent interval to 'click' as well.

After having implemented these properties, the result was that the click-detection worked quite accurately, but too slow. A signal would be generated a few moments after an actor made the gesture. This made clicking and moving the mouse at once impossible on normal interfaces. Using this filter for gestures where the current mouse location was irrelevant, such as jumping in a game, this filter functions quite well. As the normal navigation with the head was quite slow, the high-pass filter created a signal in almost all of the cases when needed, while giving hardly any false positives.

To use this filter as a gesture detection where the location of the mouse *is* important, additional calculations would be necessary. The time between the actual gesture and the detection could be measured, and then be used to compute the average distance the mouse has traveled by the time the gesture was detected. This data could than be used to estimate the location of where the click was intended.

4.4 Interpreters

After having filtered the data, we still need to process it, which is the task of the interpreter. The interpreter defines the goal of the application. As it gets the preprocessed data, it can for instance visualize the incoming data, or map the data to certain actions. By default, an interpreter is an extension of a `Canvas`, because with all of our research we needed to visualize data. After having gathered enough data to work on the filters, it

was time to create an interpreter capable of moving the mouse, using the infrared sources as input. For a full explanation of interpreters I would like to refer to the bachelor thesis of Jan Kazemier [8], as the development of suitable interpreters was one of his sections of our bachelor project.

4.5 Configuration Manager

In our goals we also mentioned configurability as a requirement for our project. To achieve this we used a separate object, which we've called the Configuration Manager (or CM). This is a separate object which is maintained by the `Main` class. The Configuration Manager retrieves its settings by parsing and interpreting an initialization (`INI`) file. These settings are then stored in a Hash table, so other objects can retrieve them. For a full explanation of the Configuration Manager I would like to refer to the bachelor thesis of Jan Kazemier [8], as the development of the CM was one of his sections of our bachelor project.

4.6 Summary

The endresult is a modular system, consisting of 4 components: `Communication`, `Filters`, `Configurability` and `Interpreters`. The system allows us to add multiple filters and interpreters. Communication with the wiimote happens through an abstraction called the `WiimoteStream`, which sends data at regular intervals. We have used an unweighted moving average filter to filter out noise.

Chapter 5

Evaluation

At this stage we have a working prototype, capable of replacing the movements of the mouse using the head. Besides that it also has the capability of sensing relative movements (gestures), to perform other actions. We've let various people test our setup and the conclusion is that the absolute positioning of the mouse with head movements works very natural and feels intuitive. The gestures work reasonably well, in the sense that gestures do get recognized, but get recognized roughly half a second too late, making it unsuitable for accurate tasks.

Our application demonstrates the possible functions of head tracking in commercial applications. It could, for instance, be implemented as an optional feature in a game. Moving your head in real life would correspond to the movement of your head in a game. This way you could look around corners, under cars, etc. These are all features which would really improve the realisticness of games, and can be implemented without too much work. Because this is just an additional input device, meaning that it could actually be implemented as an extra feature, which would not be required to play the game. Glasses can be made for even under 1 U.S. Dollar (all you need is an old frame plus 4 infrared LEDs), so they could even be shipped in a game the same way as 3D glasses are supplied with 3D movies. This project should show the ease at which this could be integrated in other systems. The only setback so far is the reliability of the Wiimote, but that should only be a matter of (little) time.

Another interesting improvement could be done in our hardware. Using a normal webcam, covered with a filter which would only let infrared light pass through, we could build our own version of the Wiimote. Even though the refresh rate of the data points would be reduced from 100 times per second to 30 times per second, it does have a lot of advantages. By using the raw images from the webcam, more points could be tracked. Because the shape of the points is not important, and they should be much brighter than their surroundings, recognition should be easy and efficient. Another big advantage is that it should become more platform-independent. The Wiimote is quite specific in which hard- and software it can connect to. Web-cams are more often supported by platforms, and usually require an USB connection, rather than blue-tooth pairing. This should enable head tracking on more systems, thus lowering the threshold even further.

Chapter 6

Conclusion

For this project a head tracking system, which is capable of tracking all 6 degrees of freedom, was created, using the Wiimote and a pair of glasses fitted with infrared LEDs. This together costs roughly €35. Considering the fact that the Wiimote takes up around €30 we could say that this system is very cheap, especially when compared to other head tracking systems currently on the market.

The modular design and ability to add multiple filters and interpreters, enabled us to rapidly test new filters and interpreters. It also enabled us to work simultaneously, without having to worry about one another.

By far the biggest problem we have encountered was the unreliability of the Wiimote, when connected to the computer. Even after having written a threaded Wiimote searcher and using a library which takes care of all blue-tooth stack types, we have encountered a lot of problems. One of the problems which occurred was that the Wiimote failed to pair with the computer. This could happen if it was paired when the computer was put in sleep mode. Upon awakening the computer it would show up as paired, but in fact was not. The only solution was to restart the computer and try again.

When it was connected it would still sometimes refuse to function properly. Sometimes on execution of our Java application, the Wiimote failed to send any points to our application. The source of this problem is still unknown to us, it could be in the library, or at a lower level. Upon restarting the application it would usually work.

The end result enabled us to play the game Portal, from Valve Software. This puzzle game was, partly because of the slow pace of the game, a suitable game to learn head tracking applied in gaming. We have let several other people play the game, and all of them were surprised by the intuitiveness of our set up.

Future work includes the improvement of our gesture recognition. It should either be faster, or calculate an approximation of where the click was intended.

Bibliography

[1] Budiman, R. A motion capture implementation for 3d cartoon movies. Master's thesis, The University of Western Australia, 2005.

[2] Bull, J. M., Smith, L. A., Pottage, L., and Freeman, R. Benchmarking java against c and fortran for scientific applications. *Java Grande Conference* (2001), 97 – 105.

[3] Duche, G. The wiiusej official project page, 2009. http://code.google.com/p/wiiusej/.

[4] Gleicher, M. Animation from observation: Motion capture and motion editing. *ACM SIGGRAPH Computer Graphics 33* (1999), 51 – 54.

[5] Gypsy Gyro. Gypsy gyro motion capture system. http://www.metamotion.com/gypsy/gypsy-motion-capture-system.htm.

[6] Insight VR. Wiimote head tracking 3d demo, 2009. http://insightvr.com/.

[7] Javal, E. Essai sur la physiologie de la lecture. *Annales d'Oculistique 79* (1879), 97–117, 155–167, 240–274.

[8] Kazemier, J. J. Low-cost eye-tracking-like control of a pc: Hardware realisation and interpreting data. Bachelor thesis, Rijksuniversiteit Groningen, 2009. Rijksuniversiteit Groningen.

[9] Kazemier, J. J., and Meiburg, Y. Wiiview, 2009. Online available at: http://www.cs.rug.nl/~isenberg/interaction/pmwiki.php/ProjectGallery/2008-2009-WiiView.

[10] Lee, J. Wii remote projects, 2008. http://johnnylee.net/projects/wii/.

[11] Lee, J., Chai, J., Reitsma, P., Hodgins, J., and Pollard, N. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics (SIGGRAPH 2002) 21*, 3 (2002), 491500.

[12] Natural Point. Trackir, 2009. http://www.naturalpoint.com/trackir/.

[13] Roetenberg, D. *Inertial and Magnetic Sensing of Human Motion.* PhD thesis, Universiteit Twente, 2006. Online available at: http://www.xsens.com/images/stories/PDF/InertialandMagneticSensingofHumanMotion.pdf.

[14] Smit, J. Affordable headtracking. Bachelor thesis, Rijksuniversiteit Groningen, 2008.

[15] Tinker, M. A. *Legibility of print.* Iowa State University Press. Ames., 1963.

[16] USB Implementers' Forum. Device class definition for human interface devices (hid), 2001. http://www.usb.org/developers/devclass_docs/HID1_11.pdf.

[17] Valve. Portal, the game, 2007. http://www.whatistheorangebox.com/portal.html.

[18] van Ackooy, W., and Klein, T. Camposer, 2009. http://www.cs.rug.nl/~isenberg/interaction/pmwiki.php/ProjectGallery/2008-2009-Camposer.

[19] Vlaming, L. Human interfaces - finger tracking applications. Bachelor thesis, Rijksuniversiteit Groningen, 2008.

[20] Welch, G., and Foxlin, E. Motion tracking: No silver bullet, but a respectable arsenal. *IEEE Computer Graphics and Applications 22*, 6 (2002), 24–38.

[21] Yilmaz, A., Javed, O., and Shah, M. Object tracking: A survey. *ACM Computing Surveys (CSUR) 38* (2006), article 13.

[22] Zhoua, H., and Hu, H. Human motion tracking for rehabilitationa survey. *Biomedical Signal Processing and Control 3* (2008), 1–18.