# Leiden University

# Bachelor Computer Science

Interacting with BigEye

Using gesture-based input methods

to control applications on a videowall


| | |
|---|---|
| Name: | Jacob Jonkman |
| Date: | March 9, 2017 |
| 1st supervisor: | Fons Verbeek |
| 2nd supervisor: | Kristian Rietveld |


BACHELOR THESIS


Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

## Abstract

In computing, the usage of multiple monitors is no new practice. When a single system composed of twelve full-HD monitors in a 3 by 4 grid, called BigEye, is introduced, the challenges of such a system become too big to ignore, with mouse and keyboard input becoming cumbersome, time-consuming and undesirable in general. We propose a novel interaction method for a high-resolution image viewer on BigEye featuring gesture-based interaction using a Leap Motion. Multiple gesture sets are described, and a final gesture set is constructed using qualitative measures found through a user test.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In this chapter, we will introduce some of the key concepts we have used for our research and the challenges that led us to carry out our research. We will end the chapter by formulating a research question and formulating the structure of the rest of this thesis.

## 1.1 BigEye

In the Snellius building of the University of Leiden, a computer system was constructed unlike most others. While regular computers use one, or maybe up to three monitors to display their information, this system, called BigEye, features a setup of twelve individual screens, all of which are controlled by a single computer to form one big videowall. This setup can be seen in Figure 1.1. These screens are positioned in a grid with 3 columns and 4 rows, with each screen being 42 inches diagonally. The monitors have a resolution of 1920 by 1080 pixels, and a Windows 7 program called Actual Multiple Monitors is used to spread the content of a single screen over all twelve monitors, creating the illusion of a single, immense screen with a total resolution of 5760 by 4320 pixels. Internally, the system features three video cards which each control one column of four screens.

All of this results in a massive screen, but this is not necessarily useful for regular desktop applications. There are several issues that prevent BigEye from being as useful for standard computing as one might think. For example, the user interface of programs are generally kept the same size when using Actual Multiple Monitors. This means that when a user wants to browse the web using BigEye, everything that is not part of the actual web page is displayed just as small as it would be on a single screen. Because of this, the user has to click on tiny buttons to control the browser, which is not a very user-friendly experience. Moreover, using the mouse can be a frustrating experience. Because of the large number of pixels, a simple activity like moving the cursor to the top right corner suddenly becomes a task that takes several seconds.

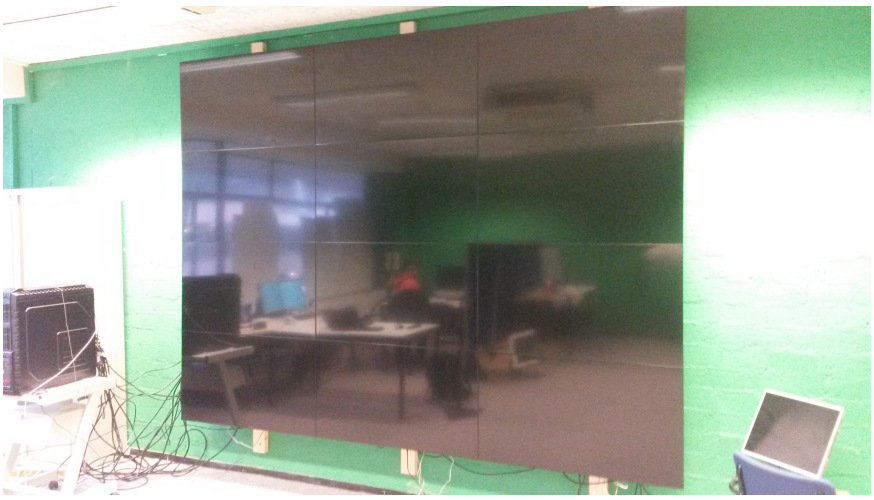


Figure 1.1: BigEye and its twelve joined monitors

However, there is a clear category of applications which do thrive on BigEye, such as applications that center around the viewing of high resolution images. Some examples would be Petri nets, graphs, big data visualization tools and viewing high resolution images. The sheer size of the joined screen means that a user can view more information at the same time, without losing anything in terms of resolution or sense of detail. We will introduce two of these applications below.

### 1.1.1 High Resolution Microscopy Images

The first application features a viewer for high resolution microscopy images, using HTML-canvasses in combination with the open-source, JavaScript-based OpenSeadragon framework [ope16]. An example of this program can be seen in Figure 1.2. The application is currently mainly used for the viewing of scans of blood vessels, but it could be used for all kinds of high resolution and deep-zoom images. Some examples would be telescopic images of galaxies and microscopy images of wood. Users can zoom in and out of the image, rotate the image and move the current viewport over the canvas. The high resolution of the images (about 60.000 by 80.000 pixels) makes the size of the video wall's joined screen a valuable asset. Originally, no special interaction techniques were implemented for this application, mouse and keyboard were the only way to use it.

### 1.1.2 Paintings

The second application, seen in Figure 1.3, divides the screen in a 4 by 4 grid of photos of paintings, with the middle 4 tiles starting empty. All the other tiles are filled with a single painting. When a painting is clicked, the middle four tiles become an enlarged version of the painting which was clicked. All the other tiles become paintings that are similar to the clicked image, with a mention of how similar they are to the central image.

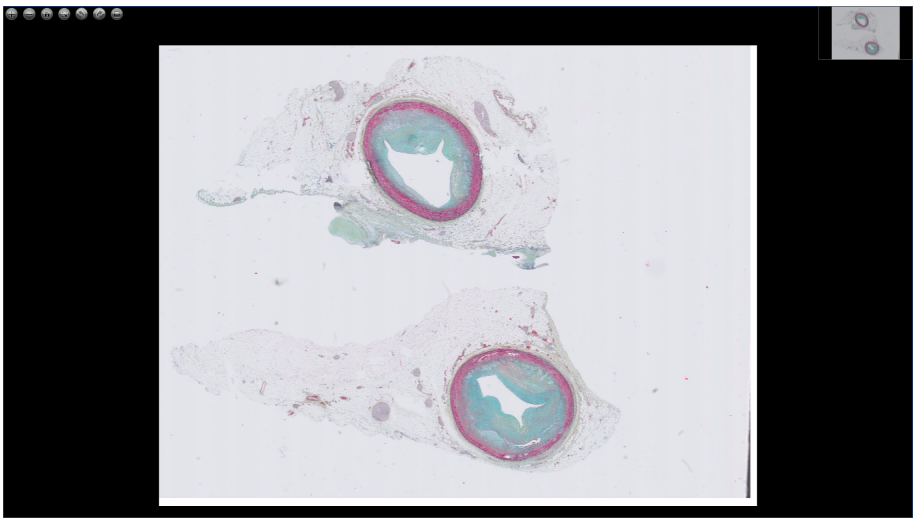


Figure 1.2: Example of a blood vessel in the High Resolution Microscopy Images application

In this manner, a user can keep clicking on, each time finding new paintings that are similar to the painting that was last clicked. The photos of the paintings come from the 19th Century Portrait Database [SH97].

Although a regular mouse and keyboard setup can be used, this application was designed to work with a Wii Remote. A program called GlovePIE was used to interpret Wii Remote input as mouse and keyboard input. One issue with this application comes from the fact that it was developed on an old version of the system, back when BigEye had sixteen screens instead of twelve [BH14]. While previously every painting had its own screen, this is no longer the case. This causes the edges between the individual paintings to be in the middle of a screen instead of being exactly between separate monitors. This might have to be optimised for the new setup, but we chose to focus on the interaction.

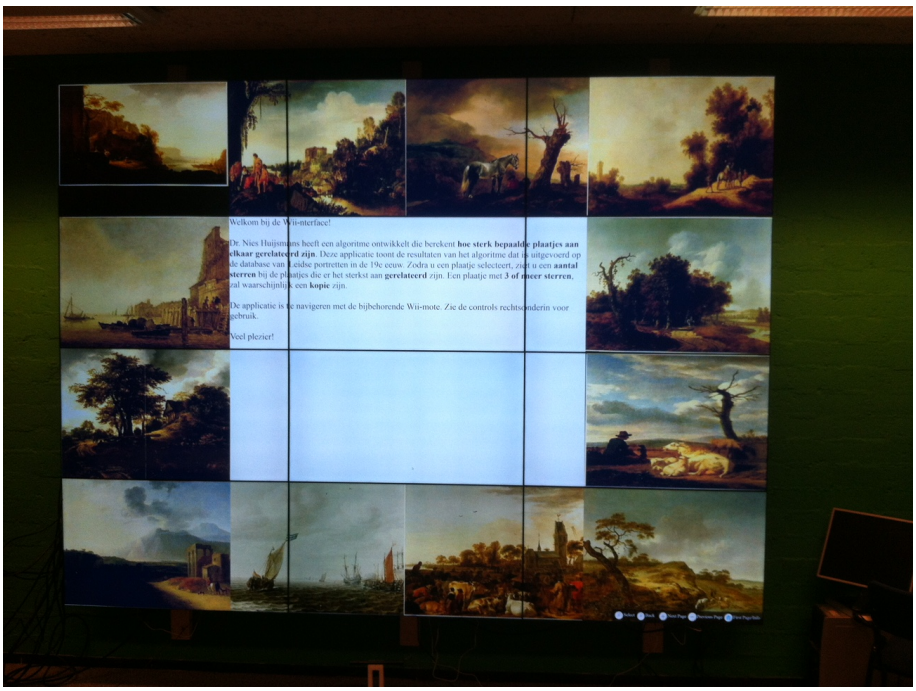



Figure 1.3: The Paintings application

## 1.2 Gesture-based input

Gesture-based input is considered to be any kind of input that revolves around the more natural interaction methods of gestures instead of input methods like mouse and keyboard. Several gesture-based input devices have are available, which will be covered in the next section. The classic mouse and keyboard setup does not possess the same kind of intuitiveness as the kinds of gestures humans commonly use in day to day life. Research has shown that a system focusing on gestures can be more efficient in terms of the speed of task completion [CV14]. The two main challenges of gesture-based interaction are considered to be the problem of computers having to recognize and understand gestures [PGS09], as well as the challenge of designing an intuitive gesture vocabulary for the user.

Little to no research has been done regarding human computer interaction on a video wall like BigEye, let alone focusing on gesture-based input. However, there has been a lot of research concentrating on gesture-based input in general. For example, systems have been developed for physical rehabilitation [FAB10], 3D concept design [RAL17], helping children with autism [GF17], web browsing [leac] and many more. Using gesture-based input has been proven to have merit, since research pointed out that pointing tasks are performed more quickly using a three dimensional input device [CV14]. More difficult tasks were performed more quickly using a mouse, which was thought to be due to limitations of the device, which in this research was a Leap Motion.

In 2008, a study was done in which participants were asked to express interactions with a multimedia device using only hand gestures [EVV08]. Examples of these actions were pause, resume and fast forward. The goal of this experiment was to gain some insight in what kind of gestures people would personally use to express a certain interaction with an application, hoping to identify trends like similar gestures used for similar interactions. In some cases, a pattern was indeed discovered. Actions which involved a notion of direction, like fast forward, were often described with metaphoric gestures in combination with deictic (pointing) or positioning gestures. However, most other actions showed a more diverse mix of gestures. Most actions did have some sort of typical gesture, but for a single person there was no pattern in the kind of gestures he or she used over multiple interactions. Another interesting result was that participants preferred using a single hand instead of two. For devices like Leap Motion, it is generally computationally easier to track one hand instead of two. This means that more 'favorable' gestures are also easier to track.

## 1.3 Gesture-recognition devices

For gesture-based input, several different gesture-recognition devices have been developed which can track the movements of a user. Each device does this in unique ways, and understanding the differences between

them helps determine when a certain device is more viable than another. Following is a short description of the devices which were most popular at the time of writing.

### 1.3.1 Leap Motion

Leap Motion (Figure 1.5a) is a device for Windows, Linux and Mac computers. The Leap is connected to the computer through a USB cable and is typically placed on a flat surface, facing upward. Using two infrared cameras, combined with three infrared lights, the Leap Motion tries to construct a three-dimensional image of the area directly above and around it. The field of view expands upwards in a conical shape of 150 degrees on the long side and 120 degrees on the short side, starting at a distance of 25 millimeters, and up to a distance of about 60 centimeters, which is visualized in Figure 1.4. Using a proprietary API, the Leap Motion constructs a model of any hands that are inside its field of view and tracks their movement, among other things. More than one hand can be used at the same time, but for the best results, at most two hands should be used, because the tracking becomes a lot less reliable when tracking three or more hands [leab].

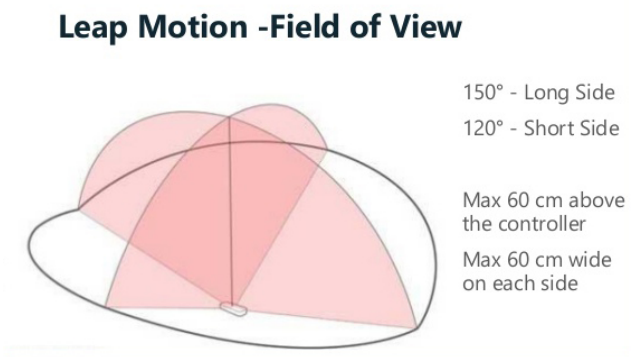



Figure 1.4: The Leap Motion's field of view with the angles and distances. Source: `http://www.slideshare.net/tinux/dev09-la-battaglia-del-touchless`

Since its release, Leap Motion was primarily used for desktop applications and games. But since early 2016, Leap Motion is also used a lot for virtual reality applications in combination with VR-headsets like Oculus Rift [leaa].

### 1.3.2 Nintendo Wii Remote

The Wii Remote (Figure 1.5b) is probably the best known of all these devices. It was revealed in 2005 to be the new controller for the next generation video game console of Nintendo, the Wii. It was also used for the Wii U, the console that succeeded the Wii in 2012. The Wii Remote can be connected to a computer through a bluetooth connection and mimics a remote control, featuring a similar infrared light setup. The user points with the Wii Remote at the 'sensor bar', which is a box with an infrared light on each side of it. The Wii Remote itself has an infrared light too, which together with the sensor bar is used to compute data like the

orientation of the Wii Remote. The Wii Remote also contains accelerometers, which can determine the three-dimensional acceleration of the remote. Combined, these functionalities can recognise crude gestures, like shaking and pointing. The Wii Remote Plus was released in 2009, which introduced improvements on the tracking data by adding gyroscopic sensors in order to allow for rotational tracking of a user's hand [ea12].

Due to the enormous success of the Nintendo Wii, combined with the fact that the Wii Remote was bundled with the console, a staggering amount of people already own a Wii Remote. In 2010, press releases stated that in the United States alone, 63 million Wii Remotes had been sold since its release in 2006 [McW10]. This fact makes it a very interesting option when designing a system using gesture-based input. However, its relatively limited tracking data does make it hard to design high-end interactions with the Wii Remote.

### 1.3.3 Microsoft Kinect

Kinect (Figure 1.5c) is a device developed originally for the Xbox 360 game console. It was later further developed to work on Windows PC's and is also compatible with the Xbox One game console. The Kinect hardware consists of components like a depth sensor and a color camera, which track the full-body movements of a user. In order to correctly perform the tracking, Kinect represents its data in several ways: as a depth map, through infrared light imaging and through skeletal tracking. Kinect also has a set of four microphones, which allow the user to give voice commands [Zen12]. Multiple people can be tracked simultaneously, the maximum number of people is claimed to be only constrained by how many people can fit within the field of view [Sch10].

### 1.3.4 Myo armband

Myo (Figure 1.5d) is a wearable armband, which can be expanded and contracted in order to fit around one's arm. The Myo contains a number of sensors which can sense the small electrical signals that go through a person's muscles when they flex or relax them [myo14] [Kli13]. These signals are called surface electromyography signals, or SEMG signals [ZLZ14]. By listening for these pulses in your arm and interpreting them, a program can track the movements of a user's arm and recognise gestures.

## 1.4 Research question

A regular mouse and keyboard setup is not as useful to BigEye as one might think. We have also seen that gesture-based input should be investigated, because it could improve the usefulness of BigEye in meaningful ways, and we wrote about a number of devices which can be used to arrive at such a gesture-based interface. We want to investigate if and how such a gesture-based system can improve the user experience provided by


(a) Leap Motion (b) Wii Remote

(c) Kinect (d) Myo armband

Figure 1.5: The four mentioned gesture-based input devices

a video wall. Therefore, our research question is: "How can we make BigEye more interactive through the use of gesture-based interaction devices?" One way to arrive at this would be to completely omit mouse and keyboard input, but there are clear use cases in which these devices still have merit. Because of this, combined with the complexity of designing a system to fully control a computer with only gesture commands, we chose to focus on the High Resolution Microscopy Images application. We also chose to focus on the Leap Motion to realize the gesture-based input. Thus, we arrive at a number of sub research questions:

- How can we use the Leap Motion to realize a gesture-based input system?
- How can we enrich applications featuring high resolution imagery using Leap Motion?
- How can we arrive at an intuitive and useful system?

## 1.5 Thesis overview

The remainder of this thesis is organized as follows. In chapter 2, we will write about the libraries and programs that we have worked with. In chapter 3, we will describe the gestures we developed for the different interactions. We will also cover a visualization program of the user's hands which we have adapted, and make a small detour to some work we have done with the Wii Remote. In chapter 4, we will describe the experiments we have done in order to test the usefulness of our gestures, with chapter 5 covering the results of said experiment. Finally, in chapter 6, we will present our conclusion and discuss some future work.

# Chapter 2

# Material and methods

In this chapter, we will discuss the libraries and API's we have used to develop our gesture-based interaction system.

## 2.1 Leap Motion Software Development Kit

The Leap Motion comes with its own Software Development Kit (SDK). Multiple programming languages are supported, like C++, Python and JavaScript. Since we wanted to use Leap Motion to control a JavaScript-based application, we decided to work with the JavaScript library, called LeapJS.

In this library, a function can be registered to be fired every time the Leap finished constructing a new frame. All of the frame information can then be passed to this function in a frame object in order to parse this data into instructions. Some examples of data which one could use are:

- The position of the hand palm, given as a three-dimensional vector covering the x, y and z-axes (frame.hand.palmPosition), as well as the direction (frame.hand.palmDirection) and the normal vector of a hand (hand.palmNormal). These are pictured in Figure 2.1. The coordinate system used by the Leap Motion is visualized in Figure 2.3.
- The velocity of the hand, again given as a three-dimensional vector: frame.hand.palmVelocity. The velocity of separate fingers can also be reached.
- Whether or not a single finger is extended or not (frame.hand.finger.extended). The position and direction of each finger are also tracked (frame.pointable.direction, frame.pointable.tipPosition), visualized in Figure 2.2.
- The amount of rotation of a hand over the z-axis, given in radians: frame.hand.roll(). The pitch and

yaw functions return similar rotation values over the other axes, but we did not use these. All three of these values are pictured in Figure 2.4.

- For every frame, an evaluation is made regarding how sure the Leap Motion is about what it sees: frame.hand.confidence. Because a three-dimensional image is constructed, but the hardware can only see the bottom of your hands, it sometimes has to guess about certain aspects of the data. The confidence factor is a measure of how much guesswork had to be done to construct the current frame and can be used to exclude frames the Leap Motion is unsure about.

For example, when someone would want to compute how much space is between his two hands, he will start off by counting the number of hands in the Leap Motion field of view. If two hands are in view, it might be checked whether the hands are in a certain orientation to signify the start of a gesture. For example, if the palms of the hands have to face each other, the palmNormal vector can be used to check whether this is the case. When the correct starting gesture is recognized, the palmPosition vector of both hands can be used to compute the distance between the two hands.

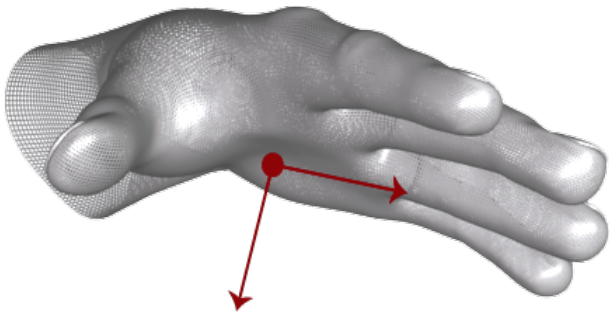


Figure 2.1: The PalmPosition, PalmDirection and PalmNormal vectors. Source: `https://developer.leapmotion.com/documentation/javascript/devguide/Leap_Overview.html`

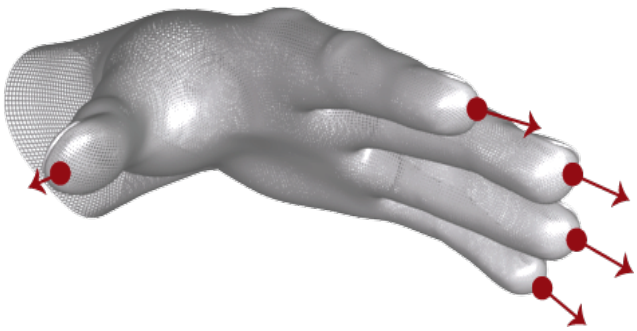


Figure 2.2: FingerDirection and position vectors. Source: `https://developer.leapmotion.com/documentation/javascript/devguide/Leap_Overview.html`

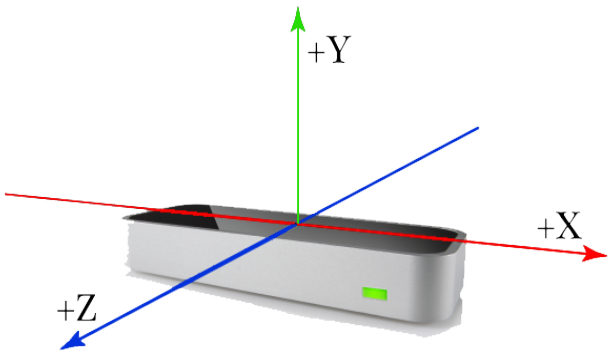



Figure 2.3: Coordinate system used by the Leap SDK. Source: `https://developer.leapmotion.com/documentation/javascript/devguide/Leap_Overview.html`

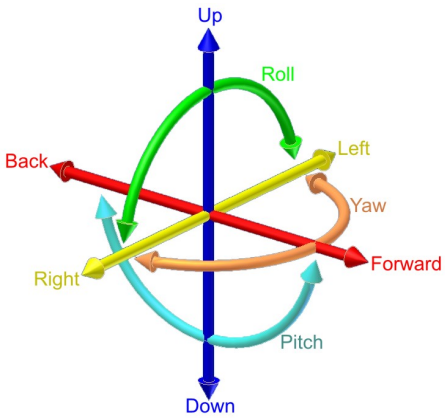



Figure 2.4: Roll, pitch and yaw. Source: `http://johnclarkeonline.com/wp-content/uploads/2011/12/6DOF_en.jpg`

## 2.2 OpenSeadragon

OpenSeadragon is an open source project that can be used to view high resolution images in JavaScript, with the main feature being that a user can zoom in on these images without loss of resolution [ope16]. It is based on the AJAX-based Seadragon framework, which was later acquired by Microsoft and integrated into Silverlight and Photosynth [yA07], after which it became proprietary software. A number of different image file formats are supported, like LIP (Legacy Image Pyramids), DZI (Deep Zoom Images) and TMS (Tiled Map Service). These file formats construct the image in a dynamic way based on the zoom leve. Within our application, we used DZI-images. In this image file format, the image is constructed as a pyramid-like structure of 'tiles', which are segments of the complete image. These tiles start small and are then combined as needed in order to create larger tiles and eventually the image that is shown on the screen. An example of how DZI images are constructed can be viewed in Figure 2.5. The important part is that no tiles of the image are constructed that are not directly in view, which allows for little to no computational overhead caused by all the tiles that are not in view. Moreover, the amount of detail of the tiles becomes smaller when the user zooms out, which means that when the user is zoomed out all the way, a version of the image is shown that is detailed, but with a resolution that is low enough to prevent stutter. This implementation allows for a large level of detail for each level of zoom, while limiting the amount of data that the user has to download in order to do so [Mic].

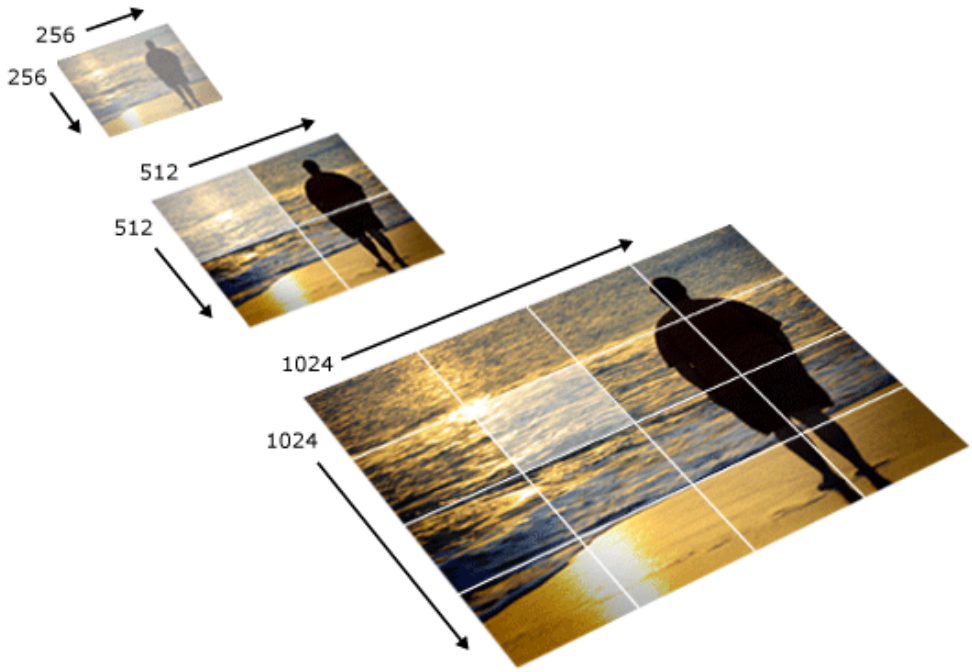



Figure 2.5: An example of how a DZI image is constructed. Source: `https://msdn.microsoft.com/fr-fr/library/cc645077(v=vs.95).aspx`

## 2.3 Programmable Input Emulation

As a side project during the experiment, we also researched Programmable Input Emulation as an interaction scheme for BigEye. A programmable input emulator is a program which can mimic the behavior of the mouse and keyboard through another device, or the other way around. Possible uses for such a program are for example to control the mouse with a joystick, or to use a mouse and keyboard to emulate a Wii Remote. We used a programmable input emulator to map Wii Remote movement to the mouse, and Wii Remote buttons to keyboard keys in order to allow the user to interact with the Paintings application using a Wii Remote.

As stated in Chapter 1, the Paintings application used a programmable input emulator called GlovePIE to translate Wii Remote input into mouse and keyboard input in order to control the application. However, we quickly found out that the old script was incomplete. Several of the Wii Remote buttons that were supposed to carry out certain functions did not do anything anymore. Furthermore, since the development of the Paintings application, GlovePIE has been discontinued. However, the project has been picked up by the open source community and has been further developed under the name FreePIE.

FreePIE features a GUI in which scripts can be loaded, edited and executed. The scripting language of FreePIE is largely based on the Python language, with some important differences [Jor16]:

- FreePIE programs are automatically executed continuously, so there is no need for never ending while loops. The program will loop until the user explicitly tells it to stop.
- Most devices (the keyboard and mouse are important exceptions) do not update every frame. Therefore, you can link a function to the update event of a certain device, which makes sure that the function is only executed whenever the device has new information available. This ensures that your program does not perform unnecessary computations.
- A global variable is defined which is true only during the first iteration of the program and always false afterwards. This can be used to check whether the program is in the first iteration and if so, to initiate other global variables and define event listeners.

# Chapter 3

# Implementation of the Gesture-Based Interaction System

This chapter will discuss the work that was done to develop two gesture sets with which the High Resolution Microscopy Images viewer can be controlled. We will also cover a Leap Motion visualizer program which we modified to suit our needs, and a FreePIE program that was written through which users can interact with the Paintings application using a WiiMote.

## 3.1 Gesture alphabets

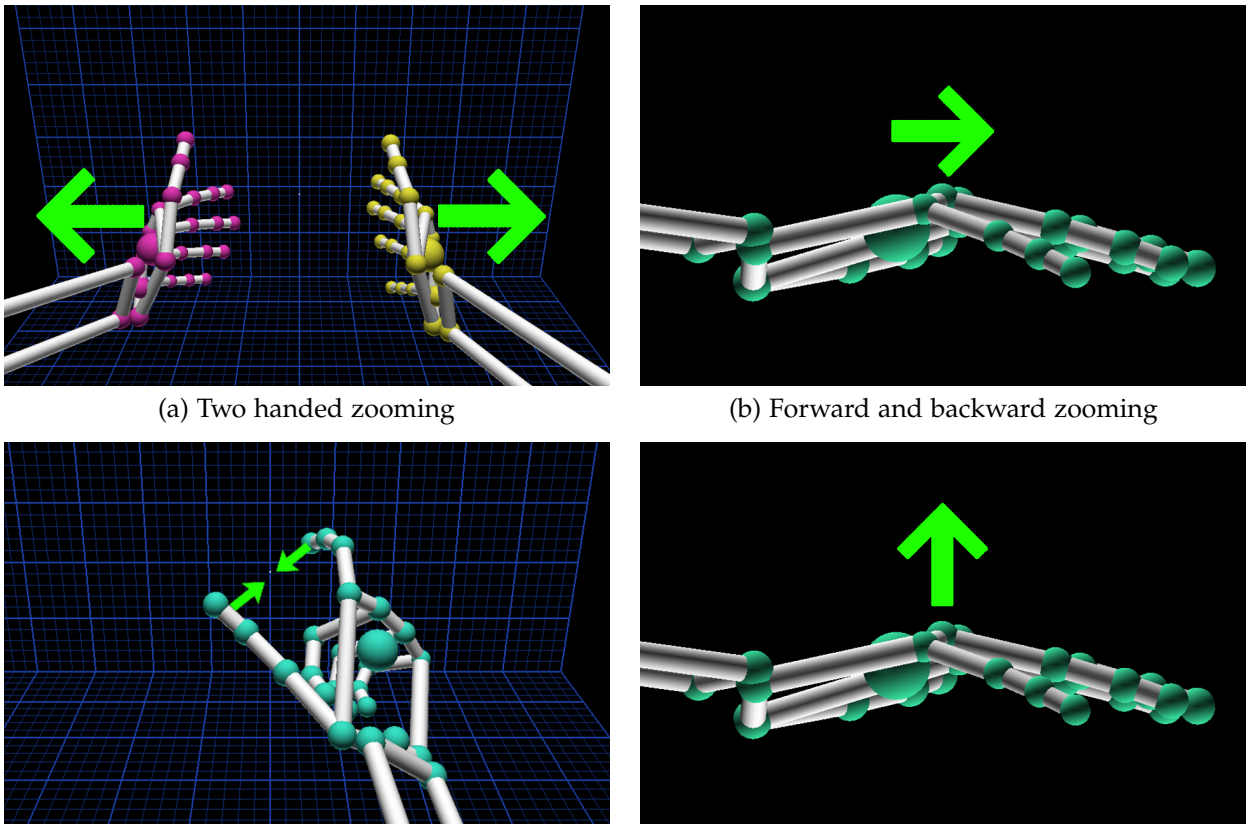
As written in Chapter 1, we have chosen to focus on developing a system to control the High Resolution Microscopy Images application using Leap Motion. Leap Motion seemed like it would provide a useful and desirable means of interaction for this particular application. The three functionalities easily translated into a number of possible interactions, making it an interesting device for trying different kinds of interaction schemes. In order to end with a system that is as useful and intuitive as possible, we designed several ways in which the three basic interactions could be performed. In this section, we will talk about the different gestures we developed, and we will conclude with two gesture sets, or gesture alphabets, to control the application. Note that every single gesture required a lot of tuning in order to improve its usefulness, but this will not be covered extensively in this thesis.

For all of these gestures, if the user makes a fist with any of his hands, the system will not perform any actions at all. This is useful for switching between hands, putting your hands in a certain orientation or retracting your hands altogether without the system recognising this as a gesture and performing unwanted interactions.

### 3.1.1 Zooming in and out

To allow the user to zoom, we developed four gestures, which can be seen in Figure 3.1:

1. Placing two hands above the Leap Motion and moving them apart to zoom in, and toward each other to zoom out. When both hands are put very close together, the system will zoom out as far as possible.
2. Placing a single hand with all fingers extended above the Leap Motion. The user zooms in by moving his hand further forward, and zooms out by moving it backward.
3. Placing one hand above the Leap Motion and extending two fingers. The zoom level is then determined by the pinching strength between these two fingers. A higher pinch strength means a greater zoom level.
4. Placing a single hand with all fingers extended above the Leap Motion and moving up to zoom in, and down to zoom out.



(a) Two handed zooming (b) Forward and backward zooming

(c) Pinch zooming (d) Up and down zooming

Figure 3.1: The different zoom gestures

Zooming is a relatively straightforward interaction, and there were only three issues while designing its gestures. Firstly, after a user zooms in, moves his hands away from the Leap Motion and puts them back above it in the active zoom-posture, the system might recognize this as an instance of zooming as well. To correct this, a system had to be introduced in which during the first frame the zoom-posture is detected,

a variable is instantiated which holds some information about the current state of the user's hand. In the following frames, the zoom level is corrected accordingly. Whenever a different posture, or a different number of hands is detected, the system has to take note and instantiate the variable again afterwards.

Secondly, allowing the user infinite precision in the zoom level that they want is desirable, but results in a terribly inconsistent user experience. Because of the fact that a person's hands are never completely still, the zoom level would continuously keep zooming in and out just a little. This kind of stutter quickly proved to be unacceptable for our system. In order to improve the stability of zooming, it is necessary to round down the values used for zoom level computation so that this natural amount of stutter is ignored.

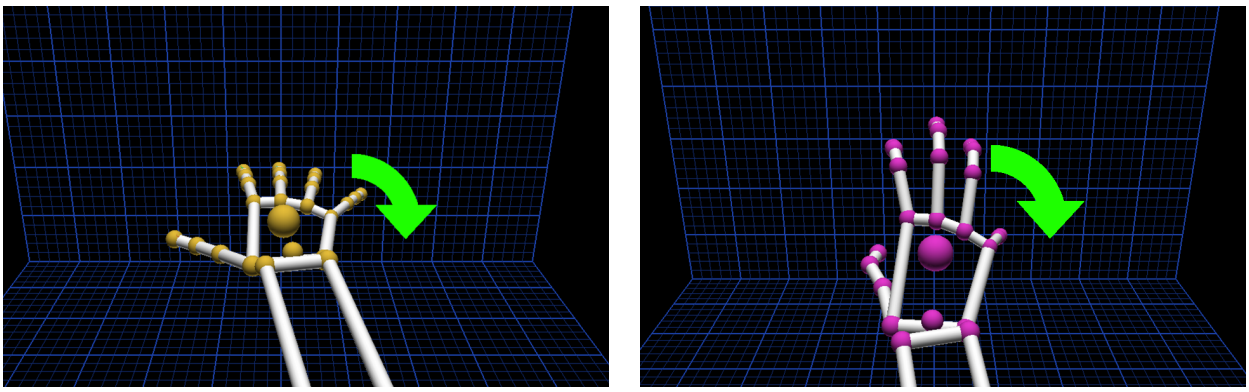
Finally, when looking at pinch zooming, the Leap Motion returns the pinching strength of a hand as a floating point number between 0 and 1. This means that small differences in pinch strength result in large changes to the zoom level, which calls for large thresholds between zoom levels. However, we also wanted to allow the user to zoom in as far as the system allows. This was a problem that was not solvable without having the user zoom in, retract their hand, adjusting pinch strength of their hand, putting it back above the Leap Motion, zooming in again and repeating this procedure until the desired zoom level was reached. Because of this problem, we decided that pinch zooming was not useful enough and we therefore did not include it in our gesture sets, even though it did seem very intuitive. Similar issues are present in the other three gestures, but since the range of possible values is much larger for the those gestures, it is less of a problem there.

### 3.1.2 Rotation

For rotation, we came up with two gestures, which are very similar. The first one, which we called "flat-hand rotation", is done by placing a single hand above the Leap Motion with all fingers extended and simply rotating your hand the way you want to rotate the image. Secondly, "doorknob-rotation" works by putting a single, fully extended hand above the Leap Motion while contracting the hand a little bit. In this way, your hand takes on a posture as if grabbing a doorknob, and rotating your hand then rotates the image. Both gestures are pictured in Figure 3.2. It is important to note that the only actual difference between these two gestures is the starting posture of the hand. As soon as the correct posture is recognized, the rest of the code is exactly the same for both gestures.

Rotation uses a system in which during the first frame of the rotational gesture, both the current degree of rotation as well as the orientation of the user's hand (in degrees) is saved. This is useful because of the fact that when a user starts the rotational gesture, either the system or his hand might not have a rotational angle of zero degrees. Using these values, the system can then correct the computed rotation angle of all following frames to account for these values.

The fact that a person's hands are never completely stationary was also an issue for rotation. However,



(a) Doorknob rotation (b) Flat hand rotation

Figure 3.2: The different rotation gestures

simply rounding the rotation angle down only had the desired effect when the value was rounded off to multiples of about 45°. This would only allow for eight distinct rotation angles, which was not enough. As a comparison, the mouse and keyboard setup allowed angles of multiples of 5, resulting in 72 distinct angles. We tried to find a solution such that the amount of different rotation angles would be as high as possible, while maintaining a stable system in which it had to be reasonably easy for users to hold a single rotation angle without unwanted turning. We ended up rounding the degrees down to multiples of 15. This value is what we call the rotation threshold.

Several ideas were tested to improve the stability of rotation further, like only parsing once every few frames and having to pass an extra rotation threshold to turn on rotation mode. However, most of them did not have a big enough effect on the stability of the system. One thing that did significantly improve the stability of the system was dynamically determining angle centers. Whenever the image is rotated, the current rotation degrees of the user's hands is saved. From that moment on, the system will only perform another rotation when it registers a hand with a rotation degree that is at least 15 degrees more or less than this value. This directly counters the natural stutter in a user's hands by ensuring that the user cannot go back and forth between a rotation angle of 14.5 and 15.5 degrees with a rotation triggering each time 15.0 degrees is passed.

### 3.1.3 Navigation

Navigation proved to have some remarkable problems:

1. Because the High Resolution Microscopy Images application uses a two-dimensional plane, there had to be a translation from the three-dimensional positional data that the Leap Motion returns, to the two dimensions of the canvas. In order to do this, we simply ignore the z-coordinates to end with only two dimensions.

2. The coordinate system of the canvas is different than one might expect, with its origin in the upper left corner. This means that in neutral rotation, an increase in the y-coordinate actually moves the user

down, instead of up.

3. The Leap can only register things in a conical shape above it, with any points outside of this cone not registering to the Leap Motion.

4. When the user places its hand too close above the Leap Motion, the tracking data will become unreliable, which can result in unpredictable results.

As a result of these problems, we envisioned a two-dimensional square above the Leap motion, which is pictured in Figure 3.3a and 3.3b. This square is positioned high enough that both of the last two problems will not occur within this square. Only movement from inside this box would be registered, and the corners of the square would correspond with the corners of the canvas. However, since the image is rotatable, the canvas is not a perfect square most of the time, but this discrepancy is hardly noticeable.

In order to move the canvas, we developed three different gestures, pictured in Figure 3.3, two of which involved around pointing with one hand toward this square:

- Directional navigation: The direction to which the user points becomes the relative coordinates of the new point of view. So for example an upward pointing direction of 45 degrees will result in moving to the upper boundary of the image. Note that this solution does not utilize the square mentioned above.

- Point navigation: The relative point to which the user points in the square is translated to coordinates on the canvas.

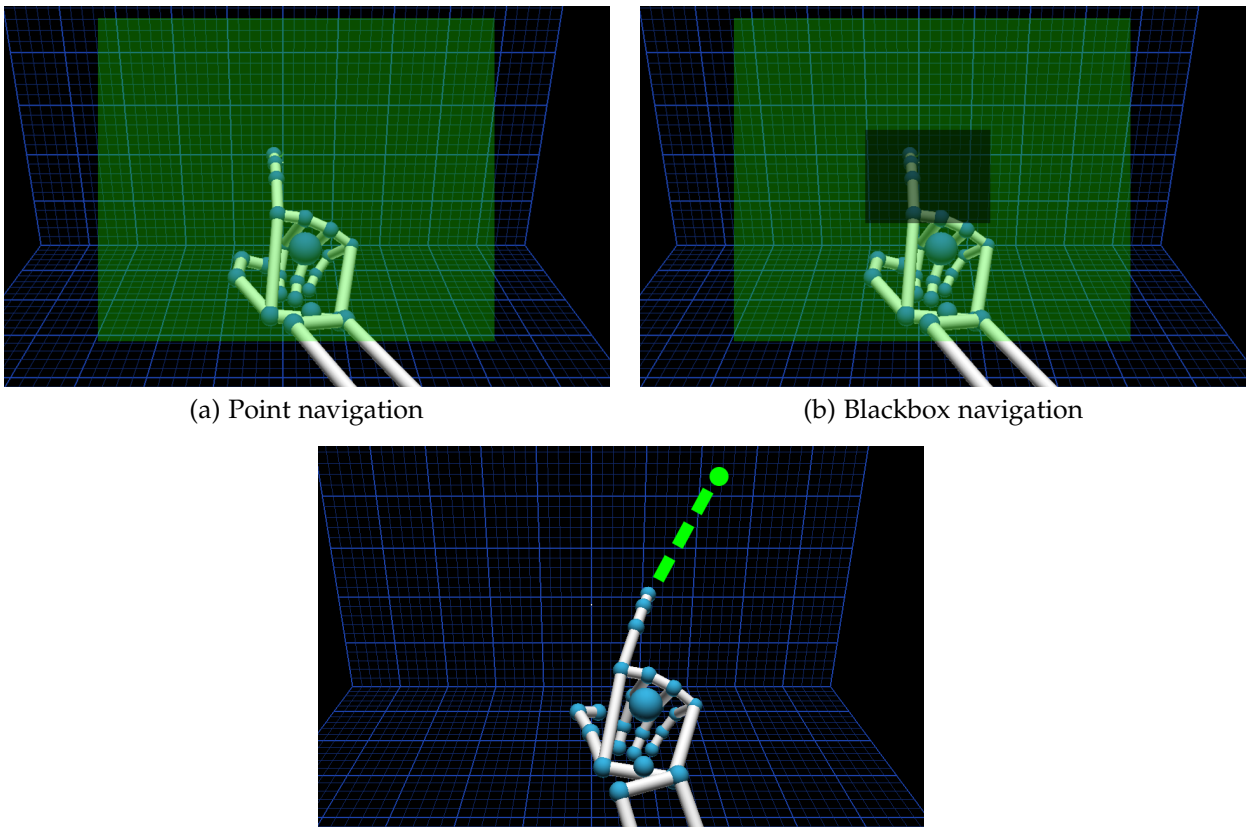
- Blackbox navigation: A smaller square within the pointable square is reserved as a kind of blackbox. While the user points inside this box, nothing happens. When the user goes out of this blackbox, the coordinates are gradually shifted in that direction until the user returns to within the blackbox, or the edge of the canvas is reached.

These forms of navigation worked correctly while the user had not rotated the canvas. However, during rotation, the origin rotates as well. This means that, for example, when the user has rotated the image by 90 degrees, decreasing the value of the x-coordinate in the program actually moves the system in an upward direction as viewed by the user. In order to account for this, both the x and y coordinates had to be corrected by multiplying the coordinates by the sine and cosine of the current rotation angle.

### 3.1.4 Final gesture sets

To be able to test the effectiveness of the gestures, we chose to group the following gestures together in two gesture sets:

- Set 1: Two-handed zooming, flat-hand rotation and point navigation



(a) Point navigation

(b) Blackbox navigation

(c) Direction navigation

Figure 3.3: The different navigation gestures

- Set 2: Forward/backward zooming, doorknob rotation and blackbox navigation

As for the remaining gestures, pinch zooming was omitted because of the issue described in Section 3.1.1. Up/down zooming felt comparable to, but less intuitive than forward/backward zooming, since the latter confers a suitable feeling of motion, going "into" the picture. Finally, direction navigation was something that the Leap Motion was not able to track reliably, which led us to choose the other two forms of navigation over it.

## 3.2 Hands visualizer

Most Leap Motion applications have some sort of visualization of the data that the Leap Motion returns, for example like the way it was done in Figures 3.1–3.3. Whether or not these applications use an abstract or a natural representation of the user's hands, the purpose is the same. Interacting with the Leap Motion is different to what most users are used to. Without any form of haptic feedback, the user has to rely on visual and audio cues to help one correctly interact with the system. The most direct way to to do this, is through graphical feedback, for example in the form of an actual representation of the user's hands. This is what we decided to use for the High Resolution Microscopy Images application. Audio cues were

not researched, because a direct visualization of a user's hands was hypothesized to be more directly and intuitively understandable than audio cues.

We found a program online that could perform the visualization of hands for us using three.js in combination with LeapJS [Arm15]. This program contains a number of features which were not needed, like a three dimensional representation of the coordinate system and an FPS-counter. This visualizer directly influenced the framerate of the whole application, so we stripped it to the bare minimum: only the hands themselves and no other information. The result of this can be seen in Figure 3.4. The visualizer was meant to run in a separate window, but we wanted to plug it into the microscopy images viewer, working as an overlay for the canvas. To accomplish this, we had to add an overlay element to the viewer and link the data generated by the visualizer to this overlay through JavaScript DOM-manipulations.

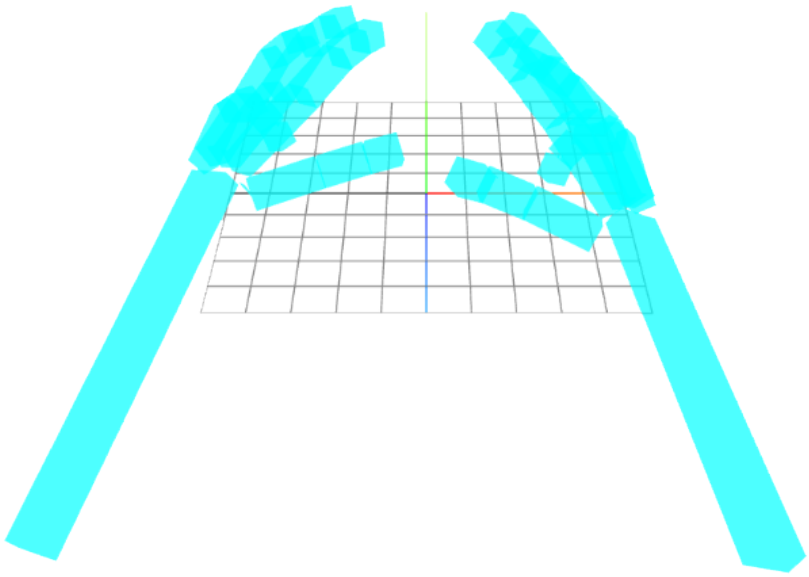


Figure 3.4: The visualizer of a user's hands

## 3.3 WiiMote as interaction device for the videowall

Because of the reasons covered in Section 2.3, we decided to rewrite Wii Remote script in the GlovePIE API. The script itself is fairly straightforward. We simply map the buttons of the Wii Remote to the correct keyboard keys and mouse buttons. Then, when the Wii Remote buttons are pressed, FreePIE will act as if the corresponding keyboard keys are pressed and the Paintings application will carry out the correct actions. Furthermore, the position of the mouse is changed according to the direction the Wii Remote is pointing to.

The system uses the same principles as blackbox navigation in the microscopy images application, with a directional vector being translated to a continuous movement in a certain direction instead of a one-to-one translation to screen coordinates. This is visualized in Figure 3.5. Whenever FreePIE detects a change in any of the parameters of the WiiMote, particularly movement and pressing a button, the current state of the Wii Remote is translated into the correct button presses and mouse movements in order to control the Paintings application.

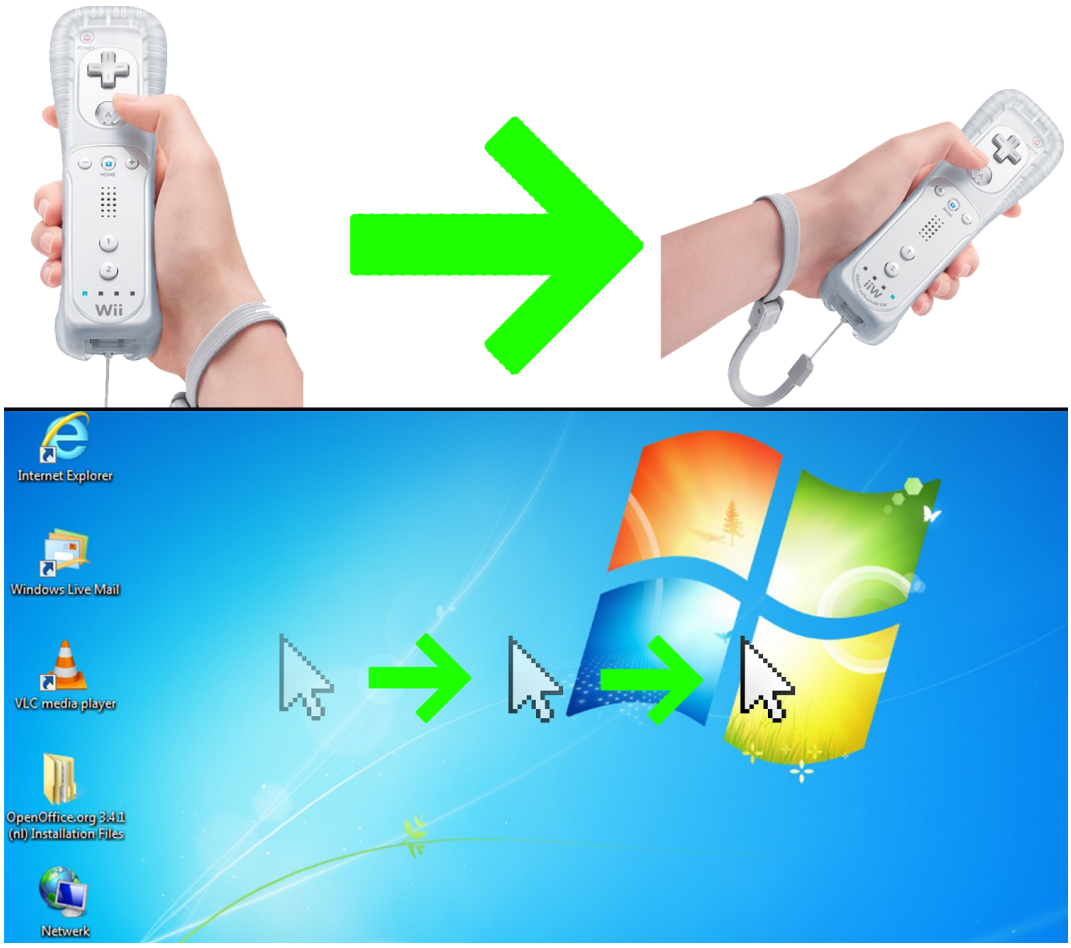



Figure 3.5: Visualization of WiiMote to mouse navigation

# Chapter 4

# Testing the gesture sets

In order to get a qualitative measure of the effectiveness of both of the devised gesture sets, we conducted an experiment. Participants were asked to complete a number of tasks in the High Resolution Microscopy Images application using only the Leap Motion. The experiment was carried out in room 410 of the Snellius building, on the BigEye computer. Since we wanted to find out which gesture set was perceived as more preferable, we divided the users in two groups. Each group was assigned a gesture set, without being told that a second gesture set was being tested as well. Through not informing the users about this second gesture set, we eliminated the possibility of a bias towards the assigned gesture set.

## 4.1 Procedure of the experiment

When a participant entered the room, we first told them a little bit about BigEye. He or she was then asked to read a document introducing the Leap Motion and the gesture sets. A number of important pointers were given for how to interact with the Leap Motion which we got from personal experience while testing the system. These were suggestions such as "don't move your hands too close to the Leap Motion" and "sometimes the system might not seem to react correctly to your actions. In this case, making a fist and returning to what you were doing will usually fix the problem". The hands viewer and mini-map were also noted and explained. The physical setup of the experiment can be seen in Figure 4.1.

Afterwards, the user would read an explanation about the assigned gesture set. The different interactions (zooming, rotation and navigation) were each explained individually, together with an illustration to help the user understand what they had to do. These illustrations were the same images we used in Section 3.1.

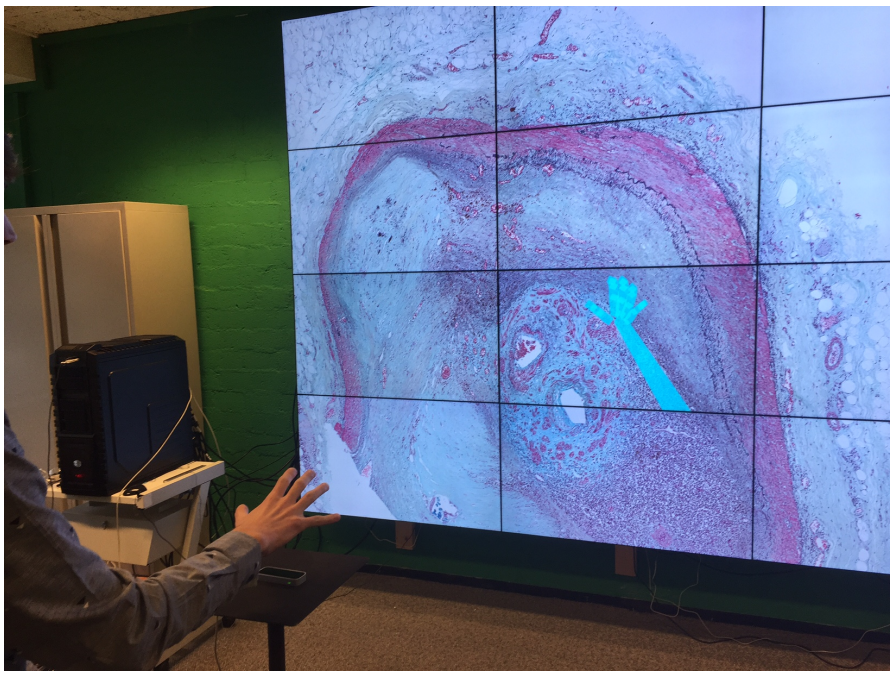


Figure 4.1: The setup of the experiment.

## 4.2 Tasks in the experiment

The introductory document ended with a short description of the microscopy image application. On the joined screen, an image was shown of a microscopy image with two red blood vessels. The slides were shown in full screen mode, with the hands viewer visible. On the document, six simple tasks were listed which the users were then asked to perform. These actions had to be carried out in order, and we would tell them when they could continue to the next task if any doubt would arise to the correctness of a user's actions. The document stated that the participant would be timed, but that the goal of the experiment was not to complete the tasks in as little time as possible. Instead, the users were asked to try and perform the tasks as correctly as possible. The different tasks were stated as follows:

- Zoom in (use your own judgment to determine how far).
- Navigate to the upper blood vessel.
- Rotate the image 90 degrees.
- Navigate to the other blood vessel.
- Zoom out as far as possible.
- Rotate the image until it is back in its original orientation.

During the experiment, we wrote down the number of mistakes participants made. Also, we kept track of how many interventions we had to give. We defined an intervention as us having to help a user, either through giving a hint or showing how a specific gesture had to be carried out. Afterwards, we asked the

users to complete the same tasks again, but this time we turned off the hands viewer. We hoped to see an improvement in execution time, number of mistakes and number of interventions, even though the hands viewer was disabled the second time.

## 4.3 Questionnaire

After the users had completed the tasks, we concluded the experiment by having them fill in a questionnaire. This survey started with some personal details like age, gender and scientific background. Afterwards, we asked the user the ten standardised System Usability Scale (SUS) questions, which can be viewed in Appendix A. The questions had to be answered on a one-to-five Likert scale, with 1 translating to "strongly disagree", and 5 translating to "strongly agree". Afterwards, we asked for each separate gesture how intuitive, useful and enjoyable the participants thought it was, all on a five-point Likert scale as well. We also asked the users for suggestions for improvements. Finally, we asked for their opinion on the hands visualizer. We wanted to know whether they thought the visualizer improved their understanding of the system, whether it helped them perform the tasks correctly and whether they felt they performed similarly well without the visualizer, again using a five-point Likert scale.

# Chapter 5

# Evaluation

A total number of 24 people participated in our experiment. Of this user group, 20 people were men and 4 people were women. With one exception, all participants were between the ages of 19 and 30, with everyone being a bachelor student in computer science, physics or math, or a master student in a computer science oriented area. Finally, one person was working in a computer science oriented job. About half of the participants had some sort of experience with gesture based input.

## 5.1 Results of the experiment

In the three tables below, the results of the experiment are shown. In Table 5.1, the combined results of both the testing round with the visualizer and without the visualizer are shown. In Table 5.2, the results are only shown for when the visualizer was on and in Table 5.3 are the results for when the visualizer was off.

| | Time | Mistakes | Interventions |
|---|---|---|---|
| Gesture set 1 | 2:43 ($\sigma$ = 2:12) | 3,42 ($\sigma$ = 1,81) | 0,96 ($\sigma$ = 1,51) |
| Gesture set 2 | 2:40 ($\sigma$ = 1:05) | 3,71 ($\sigma$ = 1,60) | 0,83 ($\sigma$ = 0,82) |

Table 5.1: Average results for both conditions combined

| | Time | Mistakes | Interventions |
|---|---|---|---|
| Gesture set 1 | 3:45 ($\sigma$ = 2:39) | 4,42 ($\sigma$ = 1,88) | 1,67 ($\sigma$ = 1,77) |
| Gesture set 2 | 3:11 ($\sigma$ = 1:12) | 4,0 ($\sigma$ = 2,00) | 1,25 ($\sigma$ = 0,87) |

Table 5.2: Average results with the visualizer on

| | Time | Mistakes | Interventions |
|---|---|---|---|
| Gesture set 1 | 1:35 ($\sigma$ = 0:26) | 2,64 ($\sigma$ = 1,36) | 0,27 ($\sigma$ = 0,65) |
| Gesture set 2 | 2:09 ($\sigma$ = 0:38) | 3,42 ($\sigma$ = 1,08) | 0,42 ($\sigma$ = 0,52) |

Table 5.3: Average results with the visualizer off

When looking at Table 5.1, the two gesture sets seem to be pretty similar in all fields. However, when comparing the results with and without the visualizer, this is not the case at all. When the visualizer was on (Table 5.2), users who were working with gesture set 2 performed better than the users who worked with gesture set 1. This group performed better in both the time needed to complete the tasks, the number of mistakes they made and the number of interventions we had to make. On the other hand, when looking at the results for when the visualizer was off (Table 5.3), this comparison is exactly the other way around, with gesture set 1 outperforming gesture set 2 in all areas.

With both gesture sets, users have performed significantly better without the visualizer. It should be kept in mind that this part of the experiment was done after the round with the visualizer on, so participants had had some practice with Leap Motion and the system at this point. One thing this data seems to suggest is therefore that both gesture sets have a relatively fast learning curve, since after practicing for only a few minutes, users had gotten considerably better at working with this system.

Using an unpaired student's t-test, we found a statistically significant difference for the time spent to complete the tasks for both the condition where the hands visualizer was on ($p = 0.0139$) as well as when the visualizer was off ($p = 0.0144$). However, both the difference in the number of mistakes made and the number of interventions needed during the test were not found to be significantly different.

## 5.2 Results of the questionnaire

In this section we will discuss the results we got from the survey that was filled in after the tasks had been completed. We will first cover the SUS scores. These scores are taken for the whole gesture set, but we will also compare the two implementations of each interaction side by side based on three properties of the gesture: intuitiveness, usefulness and enjoyment. In this way, we hope that we can come to a conclusion which combines the two gesture sets so that for each interaction, the best gesture can be selected. For each gesture, the results for both sets are visualized in Graphs 5.2–5.4 (for zooming), 5.5–5.7 (for rotation) and 5.8–5.10 (for navigation). These graphs show the participants on the x-axis, and the answers they gave to these questions on a one-to-five scale on the y-axis. The resulting lines are only meant as trend lines, the participants have been sorted according to their answers and the resulting graphs are therefore not supposed to be interpreted as a continuous line.

### 5.2.1 SUS scores

As mentioned in Chapter 4, we implemented a standardized SUS-questionnaire of ten questions in our survey. We processed the answers of this questionnaire by subtracting one from the scores of the positive questions and multiplying them by 2.5. The negative questions were subtracted from 5 and also multiplied by 2.5. In this way, every question becomes an integer between 0 and 10, and the final sum of these values is the SUS score of a single user, on a scale of 0 to 100. The average of these individual SUS scores is shown in Figure 5.1. From these values, it becomes apparent that gesture set 1 is considered to be more user friendly than gesture set 2, with a higher median and higher outliers.

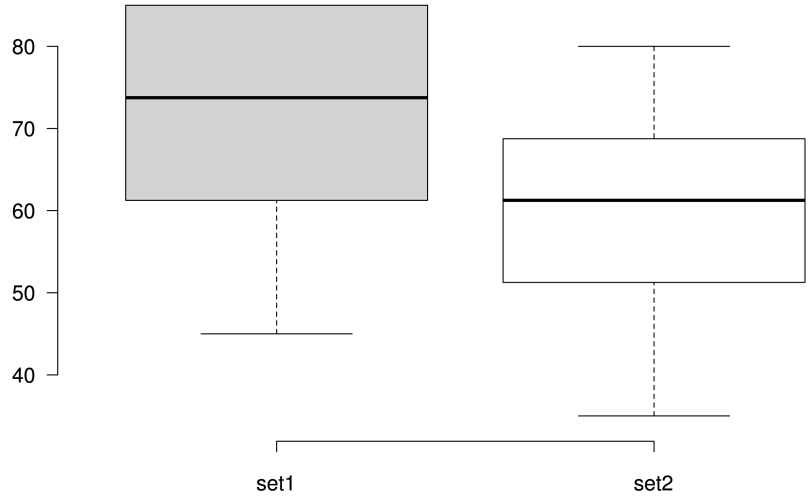



Figure 5.1: Boxplot of the SUS scores

### 5.2.2 Zooming

For zooming (Figures 5.2–5.4), one handed zooming has been evaluated slightly better than two handed zooming in both intuitiveness and enjoyment, with a small advantage for two handed zooming when considering usefulness. From this, we can conclude that one handed zooming is preferable over two handed zooming. With both gestures, someone suggested that zooming in through pinching your fingers could improve the system, exactly like the implementation of pinch zooming we discussed in Chapter 3 but which we ultimately rejected. Other suggestions were to implement a gesture to completely zoom out by clapping your hands, which was something we did implement, but only for two handed zooming. Some users would also

have preferred a smoother zooming speed.

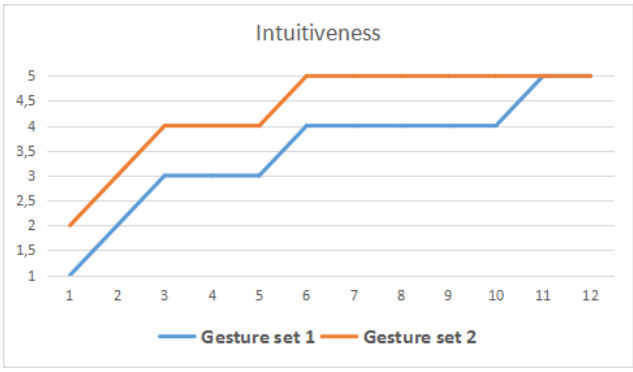



Figure 5.2: Intuitiveness of zooming

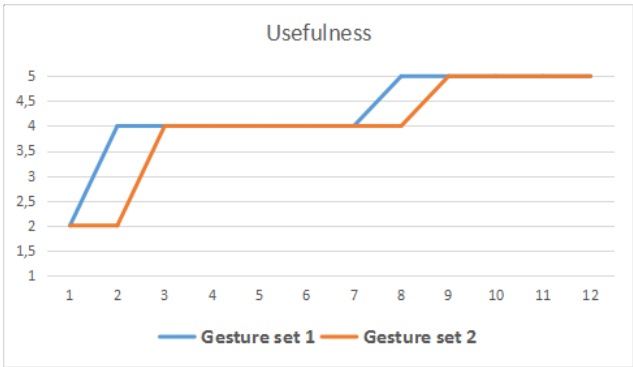



Figure 5.3: Usefulness of zooming


Figure 5.4: Enjoyment of zooming

### 5.2.3 Rotation

In the case of rotation (Figures 5.5–5.7), we see a nearly identical trend line for intuitiveness, with flat hand rotation scoring a lot better in the context of usefulness and slightly better for enjoyment. We conclude that flat hand rotation is preferable to doorknob rotation. Users did not have much suggestions to improve flat-hand rotation, but doorknob rotation did get a lot of suggestions, with multiple people writing that the gesture did not seem to work correctly. More on this in section 5.3.

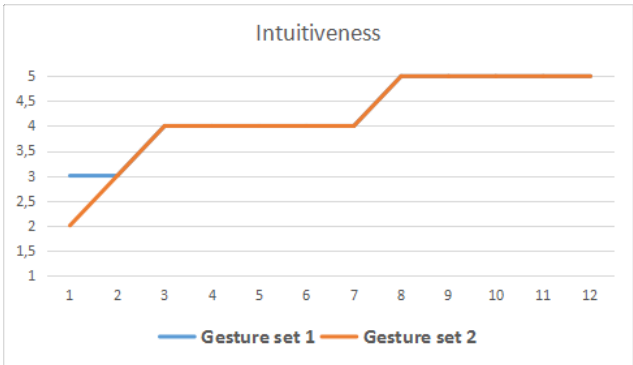



Figure 5.5: Intuitiveness of rotation

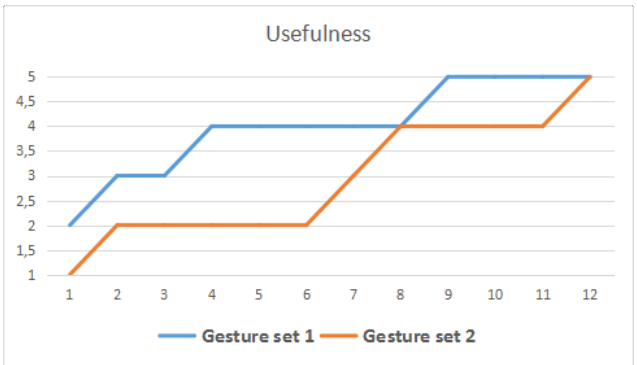



Figure 5.6: Usefulness of rotation


Figure 5.7: Enjoyment of rotation

### 5.2.4 Navigation

Through Figures 5.8–5.10, it is clear to see that Blackbox navigation preferable for this interaction. In all three aspects, blackbox navigation is evaluated better than point navigation. The suggestions mostly covered the fact that people had a hard time finding the imaginary boxes in which they could move their finger for nagivation. Some people suggested making some physical aid, others suggested to initialize the imaginary boxes dynamically when the pointing gesture was recognised. For point moving, there was also the suggestion to make navigation occur more slowly when zoomed in, something which was implemented for blackbox navigation, but which did not really translate well to point navigation.

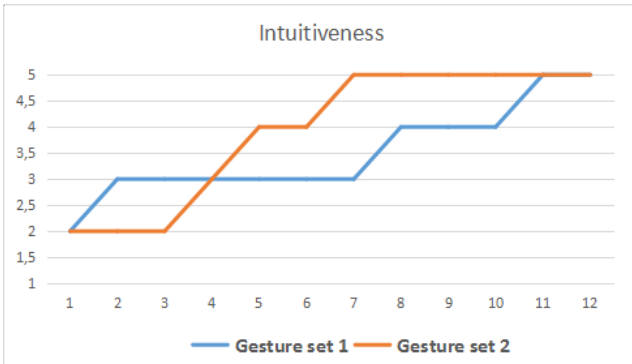



Figure 5.8: Intuitiveness of navigation

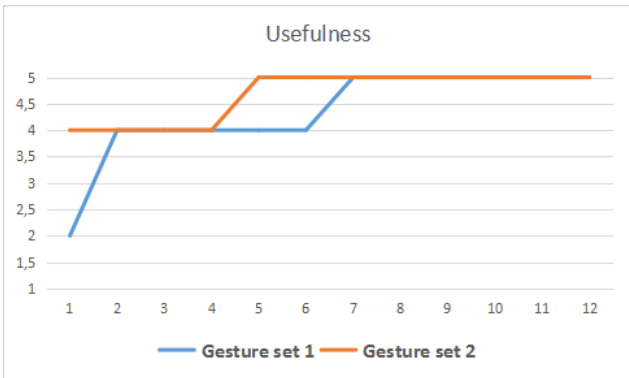



Figure 5.9: Usefulness of navigation

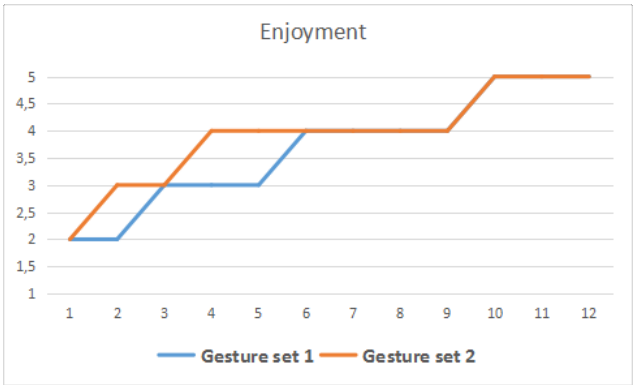



Figure 5.10: Enjoyment of navigation

## 5.3 Personal observations

One thing that immediately became apparent, was the fact that doorknob rotation did not work as intended. The shape that participants had to make with their hand was probably too specific, having to be clenched up enough to be recognized as a grab, but without being recognized as a fist. This resulted in the system often not recognizing doorknob rotation, which was frustrating for users. Since we developed the system, we tested this gesture so much that for us, there was not as much of a problem, but even we were not proficient with the gesture. Most of the people who had never used the system before were not able to learn this gesture very well in the few minutes they interacted with the system, meaning that this gesture ultimately failed to be as useful as the other gestures. One participant did discover that the gesture worked better when a user's fingers were pointing at the Leap Motion instead of forwards, but it still seemed more unstable than flat hand rotation.

Another observation we made during the experiment, was that the Leap Motion had trouble correctly tracking a user's hands when someone was wearing rings, a watch or even long sleeves, the Leap Motion website confirmed this suspicion. We discovered this trend halfway through the experiment and have since told participants to remove these items before testing. This might have negatively affected some of the data and should be kept in mind.

## 5.4 Discussion

Based on the data that we gathered from the user test, we can conclude that as a whole, gesture set 1 can be considered to be better than gesture set 2. However, this data was most likely heavily influenced by the broken rotation gesture of gesture set 2 and should therefore not be used to completely write off the other gestures in this set. When looking at the interactions separately, rotation obviously favors flat hand rotation.

For navigation, blackbox navigation is evaluated best. The problem here is that although blackbox navigation offers a lot more precision, it takes a lot longer to move around at higher zoom levels, so when moving long distances through the image, users might want to first zoom out. One possible solution for this is to combine both navigation modes, to allow for big and imprecise movements through point navigation and smaller, more precise movements through blackbox navigation.

Finally, for zooming there is a preference for forward and backwards zooming. However, when combining flat hand rotation and this form of zooming, the two interactions can interfere with each other since they utilize the same starting gesture. It is important to make clear distinctions between gestures, and to realize this, the two gestures should not be used together in this way. Since flat hand rotation has a big preference over door-knob rotation, while two handed zooming is only slightly less preferable compared to forward/backward zooming, we conclude that it is best to keep flat hand rotation and exchange forward/backward zooming for two handed zooming. In this way, the gestures are sufficiently different again, while keeping a gesture set that is as useful and intuitive as possible.

# Chapter 6

# Conclusions

In this thesis, we have discussed interaction with BigEye, a system featuring a multiple screen display. These screens, which alone are already of a size most people will consider to be big, were set up to work together to create the illusion of a single, massive screen. The size of this system introduced interesting challenges with regard to interaction, which made the classic mouse and keyboard setup less functional and desirable than it would usually be. In order to solve these difficulties, we looked at gesture based input devices to try and replace keyboard and mouse interaction with something better. Our main focus became designing a gesture set for the Leap Motion to control an image viewer for deep zoom images. The system had three basic interactions: zooming, rotation and navigation. Our goal was to create a gesture set which would be as intuitive and useful as possible.

In order to end up with the best system possible, we started implementing multiple different gestures for each interaction. When we had played around with the different gestures enough, the two gestures that we thought were the most useful for each interaction were chosen and these were developed as far as we could. When we were confident that each gesture had arrived at a stage where it would be useful for the system, we concluded an experiment in which participants were asked to use the system using one of the two gesture sets. We collected data during this experiment and the participants were asked to fill in a questionnaire afterwards. We found a statistically significant difference in the results of the two gesture sets. After taking into consideration the personal evaluations of the participants of each individual gesture, we ended up with the gesture which was best for each interaction. Using all this data we have proposed a final gesture set which, according to our research, should be best suited for interaction with this system. This final gesture set consists of two-handed zooming, flat-hand rotation and blackbox navigation.

However, it is impossible to deny the difficulties that still exist within the system. Although Leap Motion is a wonderful device with a lot of potential, the tracking data is not as robust as it should be. Examples of little mistakes that the Leap Motion makes are easily found, and this can be immensely confusing when

interacting with the device for the first time. The fact that we felt compelled to add a note in the document for our testers explaining what to do when the Leap Motion seemed to work incorrectly, should prove how often this happens. When designing a system with Leap Motion, we were continuously walking a fine line between recognizing a gesture too easily, and not recognizing the gesture early enough, trying to come as close as possible to the golden ratio of exactly recognizing a gesture when the gesture is actually performed.

We conclude that gesture based input does a lot to help a system like BigEye become more useful, and we certainly think that devices like Leap Motion could someday become a replacement for the classic Mouse and Keyboard setup. However, before such a day is reached, improvements should still be made to the tracking system. However, for applications that offer a small set of interactions and require big mouse movements, Leap Motion seems to be an upgrade over mouse and keyboard interaction.

As to the capabilities of the gesture set that we developed, we found some improvements which could still be made to further strengthen the usefulness of the current system. Most of these improvements can be made in the area of navigation. The proposed form of navigation in which point navigation can be used for big and unrefined movements, while blackbox navigation can be used for the smaller and more precise movements seems to be a big improvement to the system. The biggest problem remains the imaginary boxes. For a big boost in intuitiveness, these boxes should be visualized or a system with another kind of feedback could be developed. As for the other interactions, it seems it will always be possible to further improve the system. However, in its current state, the system already seems useful and intuitive enough that we have come to expect that most people will be able to interact with the system reasonably well after a quick, but absolutely necessary tutorial.

Future research should be done to test other interaction devices like Kinect and Myo, as well as to broaden the range of applications that can be controlled with gesture-based input devices.

# Appendix A

# SUS questions

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

# Bibliography


[Arm15] Theo Armour. https://github.com/jaanga/jaanga.github.io/tree/master/gestification-r2/template-leap-threejs/r10, 2015.

[BH14] Wouter Bouman and Thomas Helling. Wii-nterface! 2014.

[CV14] Joanna Camargo Coelho and Fons J. Verbeek. Pointing task evaluation of gestural interface interaction in 3d virtual environment. *IADIS International Journal on WWW/Internet*, 12(1):113—130, 2014.

[ea12] Kazumoto Tanaka et al. A comparison of exergaming interfaces for use in rehabilitation programs and research. 2012.

[EVV08] Otmar Klaas Luuk Langenhoff Diederik van der Steen Edit Varga, Jouke Verlinden and Jasper Verhagen. A study on intuitive gestures to control multimedia applications. *Proceedings of IADIS International Conference Interfaces and Human Computer Interaction*, pages 25—27, 2008.

[FAB10] Michelle Annett Fraser Anderson and Walter F. Bischof. Lean on wii: Physical rehabilitation with virtual reality and wii peripherals. *Annual Review of CyberTherapy and Telemedicine*, 8:181–184, 2010.

[GF17] Zengguo Ge and Li Fan. Social development for children with autism using kinect gesture games: A case study in suzhou industrial park renai school. *Simulation and Serious Games for Education*, pages 113—123, 2017.

[Jor16] Hans Jorgensen. Scripting. https://github.com/AndersMalmgren/FreePIE/wiki/Scripting, 2016.

[Kli13] Andreas Klintberg. How does myo wearable gesture control work? https://www.quora.com, 2013.

[leaa] https://leapmotion.com.

[leab] Javascript sdk documentation. https://developer.leapmotion.com.

[leac] Leaptouch. https://leaptouch.com.

[McW10] Michael McWhertor. It's official: Nintendo has sold a ridiculous amount of wii remotes. `http://kotaku.com`, 2010.

[Mic] Microsoft. Deep zoom file format overview. `https://msdn.microsoft.com/en-us/en-en/library/cc645077(v=vs.95).aspx`.

[myo14] Getting to know the myo armband. `https://developer.thalmic.com`, 2014.

[ope16] Openseadragon. `https://openseadragon.github.io`, 2016.

[PGS09] Naveen Aggarwal Pragati Garg and Sanjeev Sofat. Vision based hand gesture recognition. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 3(1):186—191, 2009.

[RAL17] Fons Verbeek Remi Alkemade and Stephan G. Lukosch. On the efficiency of a vr hand gesture based interface for 3d object manipulations in conceptual design. *International Journal of Human-Computer Interaction*, 2017.

[Sch10] Mike Schramm. Kinect: The company behind the tech explains how it works. `https://www.engadget.com`, 2010.

[SH97] J. A. Spierenburg and D. P. Huijsmans. Voici: Video overview for image cluster indexing–a swift browsing tool for a large digital image database using similarities. *BMVC*, 1997.

[yA07] Blaise Agera y Arcas. How photosynth can connect the world's images. `http://www.ted.com/talks/blaise_aguera_y_arcas_demos_photosynth`, 2007.

[Zen12] Wenjun Zeng. Microsoft kinect sensor and its effect. 2012.

[ZLZ14] Qiang Li Xu Zhang Zhiyuan Lu, Xiang Chen and Ping Zhou. A hand gesture recognition framework and wearable gesture-based interaction prototype for mobile devices. 2014.