Natural User Interface for Web Browsing on a PC

SW905E11

Software Engineering Aalborg University

Fall 2011

Aalborg University

Department of Computer Science

Title:

Natural User Interface for Web Browsing on a PC

Topic:

Natural User Interface

Project Period:

September 1st, 2011 - January 13th, 2012

Project Group: sw905e11

Lasse Andreasen

Rasmus Hummersgaard

Supervisor:

Jan Stage

Abstract:

The goal of this project is to study to what extent a natural user interface can replace standard input devices when web browsing on a PC. A natural user interface has been designed and developed that uses of a motion sensor to track users's body parts and movements. The development of the prototypes has been conducted using a iterative development process. The first prototype has been designed, developed and tested and the results from the test has lead to the design of the second prototype. The interface supports the possibility to control the cursor, clicking elements, scrolling up/down and navigating forward and backward in the web history using gestures.

Number of appendices: 0

Total number of pages: 102

Number of reports printed: 4

PREFACE

This report is the result of Software Engineering group sw905e11's 9th semester project at the Department of Computer Science, Aalborg University. The report is documentation for the project, which proceeded in the period from 1st of September until 13th of January 2012.

The research area for this project is human-computer interaction focusing on the following research question:

> *To what extent can a natural user interface replace the standard input devices used for basic web browser interaction on a PC?*

The reader is expected to have understanding of programming corresponding to a student that has completed the 8th semester of Software Engineering.

Unless otherwise noted, this report uses the following conventions:

- Cites and references to sources will be denoted by square brackets containing the authors' surname, and the year of publishing, if possible. The references each corresponds to an entry in the bibliography on page 99.
- Abbreviations will be represented in their extended form the first time they appear.
- When a person is mentioned as *he* in the report, it refers to *he/she*.

Throughout the report, the following typographical conventions will be used:

- References to classes, variables and functions in code listings are made in `monospace` font.
- Omitted unrelated code is shown as "..." in the code examples.
- Lines broken down in two are denoted by a ↵.

The code examples in the report are not expected to compile out of context.

We would like to use this opportunity to thank our supervisor Jan Stage for his help and support and all the people that participated in our tests.

Signatures

Lasse Andreasen

Rasmus Hummersgaard

CONTENTS

CHAPTER 1

INTRODUCTION

The recent years shows that the research in the area of more natural interaction forms, also referred to as natural user interface, between humans and computers has gained an increasing interest. Natural user interface refers to the interaction that is centralized around the user's body movements rather than standard input devices, such as mouse and keyboard. The body movements that are used in a natural user interface, are also referred to as gestures.

The new developments have already been exploited by the gaming industry. The gaming console Wii has been developed for the purpose of supporting natural user interface, whereas the PlayStation 3 and the Xbox has moved on from only supporting the original gaming controller to also supporting a natural user interface in form of the PlayStation 3 Move and the Microsoft Kinect for Xbox 360. For the Wii and PlayStation 3 Move the natural user interface consist of the movements of a wireless controller. The wireless controller can act as a real object like a tennis racket. The user's movement of the controller is then translated into movement of a virtual tennis racket. The Microsoft Kinect has implemented a natural user interface that works without any controllers. The Microsoft Kinect is capable of tracking the body movements of a user using a motion sensor. As an example the user's right arm can function as a virtual tennis racket without the use of any controllers.

A similar exploitation of natural user interfaces can be found in the mobile marked. Through the development of tablets and mobile phones with touch screen the interaction form is changing from physical buttons to touch screens, where the user can perform gestures using the fingers. The natural user interface for mobile devices concerns finger gestures on a touch screen that replace the buttons on the graphical user interface. As an example, navigating through a gallery on a mobile device can be done by swiping the finger to the left or to the right on the touch screen instead of clicking navigation buttons.

Mapping is a key concept, when considering natural user interface. The user's movement, the user's gestures, must be mapped directly to appropriate actions. If the mapping is not intuitive, this will increase the difficulty of using the natural user interface.

For a standard PC the use of natural user interfaces is not as common as with gaming consoles, and the most used input devices are still the keyboard and the mouse. A common usage for a PC is web browsing[1], people use social medias as Facebook, check their emails, find informations online etc. This requires basic web interaction, meaning that the users must be able move the cursor to different locations and clicking on buttons or links. It would be interesting to explore whether a natural user interface could be developed for a web browser on a PC.

The focus of this study is defined by the following research question:

> *To what extent can a natural user interface replace the standard input devices used for basic web browser interaction on a PC?*

This study is conducted as a proof-of-concept, where a system capable of acting as a natural user interface for a web browser is developed. No human touch or the need to wear any devices are required, when interacting with the interface. A study of related work is conducted, and a list of requirements are developed. The requirements are used in the design phase, where different ideas are tested and discussed. The most suited ideas are implemented, and the prototype is tested and evaluated through a usability test. The results from the usability test are discussed and used to create a design of a new prototype that is focused on the issues found in the usability test. To test the new prototype a usability test is carried out. Finally a conclusion is drawn in the final section along with plans for future work.

CHAPTER 2

RELATED WORK

This chapter describes related work, whether it is commercial products or articles. It must be noted that the products and articles mentioned in this chapter are only examples and do not cover all research and products in this area. The related work is divided into tables according to their topic of concern in order to present an overview.

The main element of the research question from Chapter 1 is natural user interface for web browsing.
From the term, natural user interface, three element have been chosen. The first element is using gestures, and this is necessary, because a gesture is needed in order to perform a click on a button or a link. To find inspiration on how gestures can be applied, articles and products that use gestures, are studied.

The second element is recognizing gestures, and this is related to the first element. In order to use gestures the gestures must also be possible to be recognized, and this element concerns the technical aspect of recognizing gestures.

The third element is cursor control since it is necessary to control the cursor to be able to select links and buttons on a web page. To make the mapping of the users movement to the control of the cursor as natural as possible, several techniques of controlling the cursor must be studied.
To sum up, the three elements are:

- Using gestures.
- Recognizing gestures.
- Cursor control.

Through the remaining of this report the gesture for moving the cursor is referred to as cursor control and not included in the term gestures, this is done to make a clear separation.

The related work is based on these three main elements and is placed in three tables that each matches one of the three main elements. For each cell commercial products are placed above the line, and articles are placed below the line. It must be noted that the title of the articles has been shorten for simplicity, and the elements are separated by **;**.

2.1 Using gestures

This section concerns commercial products or studies that make use of gestures. Table 2.1 shows the selected commercial products and articles. The horizontal axis of Table 2.1 is divided into three categories: *Mobile devices*, *Gaming consoles* and *PC*, referring to the device, on which the gestures are used. The vertical axis in Table 2.1 is divided into two categories: *Touch* and *Free hand*, referring to the interaction. Free hand means that the user is not touching a surface, but interacting with the system in 3D space, and touch is interaction with a touch screen.

		Mobile device	**Gaming console**	**PC**
Touch	Product	iPhone; iPad	Nintendo DS	HP TouchSmart
	Article	Gesture and Touch Controlled		Gestures in The Wild; Gesture Select
Free hand	Product	Crunchfish	Nintendo Wii; PlayStation 3 Move; Microsoft Kinect	
	Article	Development of an Everyday Hand Gesture Interface		WaveWindow; Mid-air Pan-and-Zoom

Table 2.1: Table for using gestures

The products and articles are described in detail below.

Touch on Mobile devices

Two well-known mobile devices that have touch interfaces are the iPhone[2] and the iPad[3]. The iPhone is a smart phone from Apple with a touch screen, and the iPad is a tablet from Apple that has basically the same touch interface as the iPhone. Both devices support a similar set of gestures to interact with the user interface. The devices have single touch and multi-touch gestures, where a single touch gesture can be tapping an element or holding the finger down and moving it in a direction, and the multi-touch is using multiple fingers to perform more advanced gestures such as zooming and rotating.

The article "Gesture and Touch Controlled Video Player Interface for Mobile Devices"[4] describes an interface for interacting with a video player on a mobile phone using gestures.

The gestures are either performed by moving and holding the mobile phone in different direction or by drawing lines on the screen of the mobile phone with a finger. Examples of this are that holding the phone upside down pauses the video, and drawing a line from left to right increases the volume. The interface is tested through a usability experiment, and the results show that the touch gestures drawn with diagonal lines and curves are less preferred than gesture with horizontal lines.

Touch on Gaming console

For gaming consoles with touch interface there is the Nintendo DS[5]. The Nintendo DS has a secondary screen that is a touch screen which can be used to draw images or write text. The touch screen is also used in games, where the user can be required to press elements on the screen during the game.

No relevant articles have been found for gaming consoles with touch screen.

Touch on PC

HP has developed a series of PCs that has a touch screen integrated, the HP TouchSmart PCs[6]. The HP TouchSmart PC is basically a standard PC that has touch screen functionality that the users can use to interact with Windows. The HP TouchSmart uses similar gestures as the iPad and iPhone, where e.g. swiping a finger to left or right is used for navigating to the next or previous item.

In the article "Gestures in The Wild:Studying Multi-Touch Gesture Sequences on Interactive Tabletop Exhibits"[7] the use of multi-touch gestures on a large horizontal interactive tabletop has been studied. The use of gestures is studied in an environment, where adults and children can walk up to the interactive tabletop and interact with it. Their use is recorded and analysed, and the results show that there is a difference in, what gestures adults use compared to what gestures children use. Children are more likely to use both hands and even arms and sleeves to interact, whereas the adults more tend to use single-handed gestures involving one or two fingers. Their results also show that one gesture is not necessarily enough for one command, since people use different gestures to perform one command. For example for the sweeping command used to push an image away from the person, five variations of the gesture were recorded during the test. They conclude that it is necessary to look at the context, of which the gesture must be applied, and not on the gesture in isolation. The gesture also depends on the person, as people tries alternative gestures, depending on the social context, their age and their overall intention.

Another system that uses touch interaction, is described in the article "Gesture Select: Acquiring Remote Targets on Large Displays without Pointing"[8]. The system is developed for selecting targets on a large display, and the idea is that the icons each have a gesture assigned, and when the user makes the gesture either with a pen or simple touching the screen, the icon is selected. During the development of the system they discover that it is easier for users to make simple lines, horizontal or vertical, whereas diagonal lines are more difficult. This is applied by making the most frequently used gestures consisting of only simple lines.

Free hand with Mobile devices

Crunchfish is a company that develops software for tracking a user's hand movements using the camera on the mobile phone. The user can either hold his hand or finger in front of the camera, move his hand or fingers over the camera in different directions or make a push movement towards the device.

The article "Design and Development of an Everyday Hand Gesture Interface"[9] describes the development of a customized gesture interface on a mobile phone. The customized gesture interface is based on input from the accelerometer, and for the development of such a gesture interface four design principles are developed, which are:

- Ubiquity, which means that the system should be available anywhere and not restricted by time or place.
- Unobtrusiveness, meaning that users should barely notice that they are using an interface. No glove or special controller should be used.
- Adaptivity, meaning the system should be capable of adapting to a specific user, either be letting the users define their own gestures or adapting the general gestures to the users.
- Simplicity, meaning that the system must be simple to use. It should not be expected that the users have a great understanding of the system, and the system should therefore be simple and provide feedback.

These four design principles are then used to create a prototype that is not described further in this article.

Free hand with Gaming console

There are three gaming consoles that use free hand interaction. The Nintendo Wii[10] and the PlayStation 3 with PlayStation 3 Move[11] use hand held controllers and the users's movements of these to create and interact with natural user interfaces. The controller is held like a real object as a sword, and the users move the controller, as if they were using a real sword. The user's movements of the controller are recorded by the gaming consoles and are translated to movements of a virtual sword.

The Xbox 360 with Microsoft Kinect[12] does not use any controllers, but tracks the human body and translates the body movements into movements of the virtual object using a motion sensor.

Free hand with PC

The article "WaveWindow: public, perfomative gestural interaction"[13] describes the system WaveWindow that is a system for navigating through video trailers. WaveWindow works with the use of wave gestures and by knocking on the display or the case, in which the system is placed, to navigate through film trailers.

The article "Mid-air Pan-and-Zoom on Wall-sized Displays"[14] describes a system that can be used for zooming and panning geographical data on large displays. For this system two types of gestures are tested, a linear gesture and a circular gesture, used for zooming. Several devices are tested as input devices, and two types of gestures, linear and circular, are tested with each device. The devices and gestures can be seen in Figure 2.1.

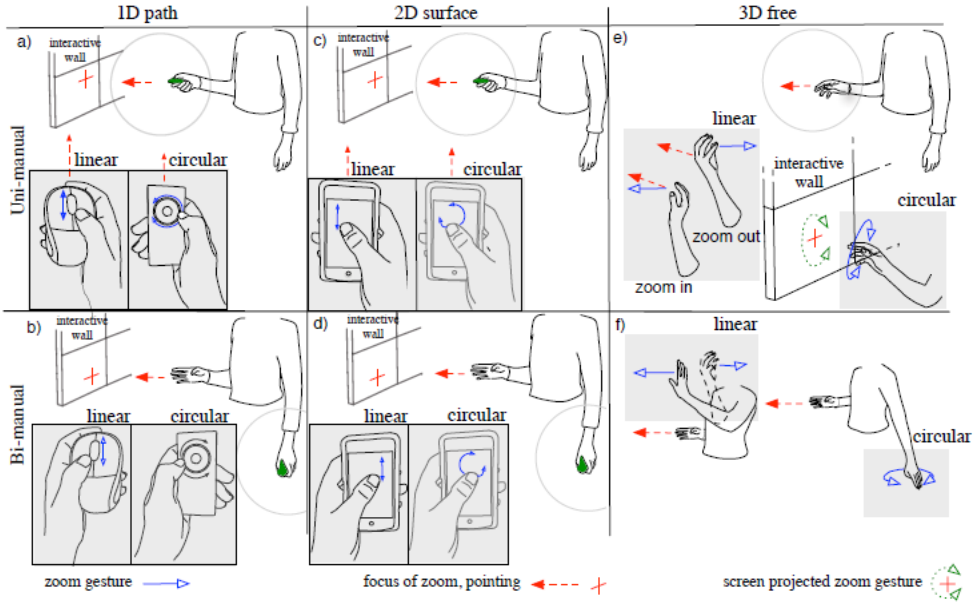

Figure 2.1: The input devices for Mid-air Pan-and-Zoom[14]

As seen the different techniques are either 1D path, 2D surface or 3D free. The different techniques are used uni-manual, meaning that only the dominant hand is used for zooming and pointing, and bi-manual, meaning that the dominant hand is used only for pointing, whereas the non-dominant hand is used for zooming.

The different techniques and gestures are tested through an experiment with 12 participants, and the result showed that :

- For each technique bi-manual is faster to use than uni-manual. The participants commented that using the same hand to make the zoom-gestures interfered with pointing.
- For each technique linear gestures are faster to use than circular gestures. The participants commented that linear gestures are easier to perform than circular, where there is no guidance of, how the gesture should be performed.
- For both uni-manual and bi-manual 1D path is faster to use than 2D surface, and 2D surface is faster to use then 3D free.

2.2 Recognizing gestures

This section describes the technical aspect of recognizing gestures, and the articles are shown in Table 2.2. No commercial products that describe in detail, how they recognize gestures,

have been found.
The table is divided into two categories on the horizontal axis, which are *Static* and *Motion*, referring to the type of gesture that is recognized. Static means that it is hand poses that are detected, and Motion means that it is a movement that is detected.

	Static	**Motion**
Recognizing gestures	Real-time Tracking of Hand Gestures	A Framework for 3D Spatial Gesture Design; Dynamic Hand Gesture Recognition; Gestures without Libraries, Toolkits or Training

Table 2.2: Table for recognizing gestures

The articles are described in detail below.

Static gestures

The article "Real-time Tracking of Hand Gestures for Interactive Game Design"[15] describes the development of a system called Germane. The system uses a webcam to retrieve a image feed in order to recognize gestures performed with the hand. The system is able to recognize a set of static gestures containing: Rock, paper, scissors and pointing with a finger. The system uses a hull point analysis algorithm to extract the hand from the image feed. An analysis of the convex hull is used to determine how many fingers that are visible to the camera which decides the gesture performed.

Motion gestures

The article "A Framework for 3D Spatial Gesture Design and Modeling Using a Wearable Input Device"[16] describes a framework which is able to recognize gestures performed in 3D space using a wearable input device. They implement and test two different algorithms used to recognize gestures, a Dynamic Time Warping (DTW) algorithm and a Hidden Markov Model (HMM) algorithm. The result shows that the HMM-algorithm performs 2.9% better than the DTW-algorithm, but one should note that HMM-algorithm requires a large set of training data, in this case the HMM-algorithm uses a set of 20, whereas the DTW-algorithm only uses 3 templates to compare against. Furthermore a test with only the HMM-algorithm is conducted to compare the success rates for user-dependent gestures and user-independent gestures. The user-independent gestures are recognized with a success rate of 90% and user-independent gestures are recognized with a successrate of 50%.

The article "Dynamic Hand Gesture Recognition Based on SURF Tracking"[17] describes a system able to recognize hand gesture performed using the DTW-algorithm. The system uses an ordinary webcam to capture the image feed where the hand is extracted using a SURF-algorithm. The system is able to recognize 26 hand gestures which is based on the alphabet. The system has a success rate of 87.1% on a training set and a successrate of 84.6% on a testing set.

The article "Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes"[18] describes the development of a gesture recognizer called the $1 Recognizer. The $1 recognizer is developed and compared to a DTW recognizer and a Rubine recognizer using a Pocket PC with a touch screen. The success rate of the $1 recognizer and the DTW were tested to be 99%. The Rubine recognizer recognizes gestures with a success rate of 95%, when providing it with six more sets of data. The $1 is the fastest algorithm of the three tested but it is not able to handle gestures performed at different speeds.

2.3 Cursor control

This section describes several techniques used to control the cursor when web browsing on different devices. Related work on this subject is inserted into a table based on the three categories: *Mobile devices*, *Gaming consoles* and *PC*.

		Mobile device	**Gaming console**	**PC**
Cursor control	Product	BlackBerry; Sony Ericsson	PlayStation 3; Nintendo Wii	Mouse; Trackball; Trackpoin; Touchpad
	Article			Face as Mouse; Does dynamic cursor control gain improve

Table 2.3: Table for cursor control

The different products and articles are described in detail below.

Cursor control on Mobile devices

Older models of mobile phones uses a cursor to highlight items and links while browsing the Internet. Some models of mobile phones from BlackBerry use a Trackball to control the cursor on the screen while browsing. The cursor is moved by rolling the trackball around using a finger. The Trackball is contained in a socket with sensors to detect a movement of the ball on both x- and y-axis. If a user rolls the Trackball upwards, the cursor moves up on the screen. Newer models like[19] uses a Trackpad which works in a similar way. If a finger is moved across the Trackpad, the cursor moves accordingly. The Sony Ericsson W995[20] has a 4-way button which is used to move the cursor while browsing. If e.g. the left side of the button is pressed the cursor moves to the left.

No relevant articles concerning cursor control on mobile devices have been found.

Cursor control on Gaming consoles

Two of the major gaming consoles, PlayStation 3 and Nintendo Wii, support the ability to browse the Internet. Originally the PlayStation 3 used a simple controller[21] with two analogue stick, where one is used to control the cursor around the screen. When the analogue stick is pushed in a direction, the cursor moves accordingly. The further the analogue stick

is pushed, the faster the cursor moves on the screen. Recently a new wireless controller was released for the PlayStation 3 called PlayStation 3 Move[11]. The PlayStation 3 Move controller works like a pointing device while browsing, enabling the user to point at a link. Similar the Nintendo Wii uses a wireless controller[10] which also works as a pointing device to control the cursor while browsing.

No relevant articles concerning cursor control with gaming consoles have been found.

Cursor control on PC

The most common way of interacting with a browser on a PC is using a mouse. A mouse works by moving the cursor on the screen according to the movement of the mouse on a surface. Another device for PC is a Trackball, which works like the Trackball on a BlackBerry as mentioned before. Older laptops from IBM used a TrackPoint[22] to control the cursor. When the TrackPoint is pushed in a direction, the cursor moves in the given direction. The more pressure that is applied to the TrackPoint, the faster the cursor moves. Newer laptops uses a Touchpad to control the cursor. The Touchpad translates the movement and position of a user's finger on the Touchpad to a relative position on the screen.

The article "Face as Mouse Through Visual Face Tracking"[23] describes a system which tracks the face of a user and and translate it into movement of the cursor. In the article three cursor control modes are investigated and compared.

- **Direct mode** - The face orientation is mapped directly to the cursor's X- and Y-coordinates.
- **Joystick mode** - The face orientation is used to determine the direction of the movement of the cursor. The angle of, which the user turns his head, determines the speed of the cursor.
- **Differential mode** - A combination of the two methods.

The optimal solution is the differential mode which means that direct mode is used to move the cursor to an area of interest. The joystick mode is then applied in order to fine tune the position of the cursor. In order to switch between the two modes they suggest that blinking of an eye can be used.

The article "Does dynamic cursor control gain improve the performance of selection task in wearable computing?"[24] describes whether it is beneficial to use dynamic cursor control or not. Dynamic cursor control means when the cursor approaches the area of interest, the cursor speed is decreased. The result of the experiment conducted by the authors shows that dynamic cursor control decreases the time it takes navigating from point A to point B.

2.4 Summary

This chapter has presented examples of related work for three different elements: using gestures, recognizing gestures and cursor control. The related work for the three elements con-

sists of both articles and commercial products. It was noted that no commercial products for free hand interaction with a PC were found. We find it therefore relevant to study this area further, and the focus for this project concerns therefore free hand interaction. Based on the knowledge acquired during the process of analysing relating work a set of requirements is constructed in the following chapter.

CHAPTER 3

REQUIREMENT SPECIFICATION

This chapter describes the requirements for the natural user interface, and the requirements are divided into two parts, one part for user requirements and another part for the technical functionality that the user interface must offer. Each requirement is explained and defined.

The different requirements and their explanations are listed in the following.

3.1 User requirements

The requirements from a user's point of view are described in this section. In order to create the user requirements a definition of natural user is used as a starting point:

> *A natural user interface is a user interface designed to reuse existing skills for interacting directly with content[25].*

The two main concepts in the definition are reuse of existing skills and interacting directly with the content. To give a better understanding of these concepts we present an explanation of both that are based on the source[25].
The term of reusing existing skills refers to the use of already known skills from human-human communication and human-environmental interaction for interacting with computers. People should be able to interact with an interface through intuitive actions, and the interface should be understandable with the use of metaphors drawn from real-life experience.

The term interacting directly with content is referring to the fact that the primary interaction should be the directly manipulation of the content. It must be noted that this does not mean that a natural user interface can not contain controls as buttons, but this is the secondary interaction form.

From this definition we use the term intuitiveness which is described and defined below.

Intuitiveness

Intuitiveness is derived from the explanation of reusing existing skills from the definition of a natural user interface. An effort to develop an intuitive interface is to make sure that the interface is easy to learn and remember, to ensure the users are able to use the interface from time to time without going through a training session each time. This is also supported by the book "Designing Interactive Systems" [26] which states that easy to learn and easy remember is two characteristics of a system with a high degree of usability. The way, the user is controlling the cursor, must be intuitive as well as the possible gestures to perform. A sub point to intuitiveness is the use of direct mapping, and direct mapping is derived from the concept of interacting directly with the content from the definition. The gestures must be directly mapped to actions in an intuitive way
To summarize, the term intuitiveness is divided into the following requirements:

- Easy to learn - The gesture should be easy to learn.
- Easy to remember - The gesture should be easy to remember.
- Direct mapping - The gestures should be directly mapped to actions.

3.2 Technical requirements

This section describes the technical functionality that the user interface must offer.

Web browser functionality

The interface must be able to communicate with a web browser, and the reason for this is the fact by having the ability to interact with a web browser, the system is able to function online. If the interface can function online, this will provide access to various online Web services.

As mentioned before when considering basic web browser functionality required to navigate web pages there are two functionalities: the ability to move the cursor and to click on buttons and links. In order to optimize the interface for web browsing we consider the functionality that is offered by the standard input device and the web browser. As a standard input device the mouse is considered, and besides the functionality of moving the cursor and clicking, the standard mouse also offers the functionality of scrolling vertical on a web page.

Besides scrolling newer mice also support the functionality to go backward and forward in the web history. Furthermore a study from Mozilla Firefox showed that the most used button for web browser navigation in Mozilla Firefox is the "back" button[27].
Based on this the functionality of the "back" button is chosen to be integrated into the interface. To keep the interface as consistent as possible the functionality of the "forward" button is added as well although it is not used as much as the "back" button.
This results in six functions, which are listed below:

- Moving the cursor.
- Clicking on a link or a button.

- Scrolling up on a web page.
- Scrolling down on a web page.
- Going backward in the web history.
- Going forward in the web history.

It must be noted that the web pages considered do not require input from a keyboard.

3.3 Summary

In this chapter the requirements for the natural user interface have been found, and the requirements have been divided into two parts: the user requirements and the technical requirements. In the user requirement a definition for a natural user interface is given, and from this definition the term intuitiveness is used. The term intuitiveness is furthermore divided into three requirements: Easy to learn, Easy to remember and Direct mapping. For the technical requirements the reason for implementing six functions required for interacting with a web browser is given.

CHAPTER 4

CHOSEN TECHNOLOGIES

This chapter describes the different technologies used in this project. First is a brief description of the hardware used. The chosen software development kit and its architecture are described. Lastly, the programming language used to create the software is introduced.

4.1 Motion sensor

In order to track people and their movements a motion sensor is chosen. By using a motion sensor people are not required to wear or use any devices.

Two different motion sensors are considered. The Kinect developed by Microsoft and the XTION PRO from Asus are both capable of tracking a user. Both products have been developed in collaboration with PrimeSense[28], which means they are based on the same technology.

The Kinect is primarily developed for use with the Xbox 360 whereas the XTION PRO is developed for use with a PC. It is possible to connect the Kinect to a PC without any modifications, and the Kinect is less expensive compared to the XTION PRO. Since both products are capable of tracking a person and its movements, and due to the price difference, the Kinect has been chosen for this project.

4.2 Software development kits

In order to be able to recognize a person and its movements the Software development kit (SDK) needs to support tracking of the user and the user movements. There are two well known SDK available when developing software supporting the Kinect. PrimeSense is offering a SDK, called OpenNI SDK, and Microsoft has released a beta SDK, called Kinect SDK. Both the Kinect SDK and OpenNI SDK support tracking of the whole body, and in order to find the most suitable SDK for this project a test has been conducted.

The test of the Kinect SDK and OpenNI SDK showed that the Kinect SDK had difficulties when the Kinect was not able to see the whole body of a person. If a user's legs were beneath the view of the Kinect, it returned misleading data. Figure 4.1 shows the position of the legs from the Kinect SDK, when a user's legs were positioned beneath the view of the Kinect. The misleading data for the legs could interfere with data for other body parts. This error can be avoided with OpenNI SDK, because OpenNI SDK makes it possible to define which specific parts of the body to track and in this case the upper body and both hands. OpenNI SDK is therefore chosen.

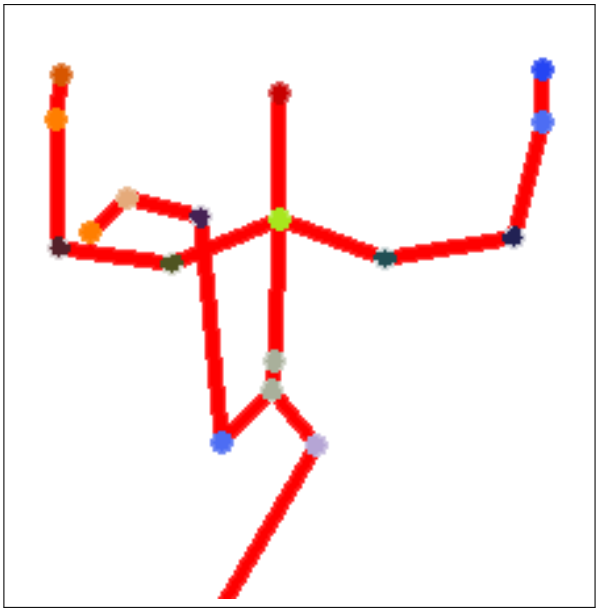

Figure 4.1: Skeleton view with Kinect SDK

4.3 OpenNI SDK architecture

After choosing OpenNI as the SDK, the architecture of OpenNI SDK is studied in order to get an overview of the functionalites that OpenNI SDK offers.
The architecture of the OpenNI SDK is shown in Figure 4.2.

1. PrimeSensor - The physical device, the motion sensor, which generates the raw sensory data.
2. Sensor Data Acquisition - An API which acquires the data from the motion sensor.
3. OpenNI - Provides communication interface that interacts with both the motion sensor's driver and the middleware components (NITE Controls and NITE Algorithms).
4. NITE algorithms - Processes the depth image produced by the motion sensor.
5. NITE Controls - Provides application framework for gesture identification and gesture-based UI controls.

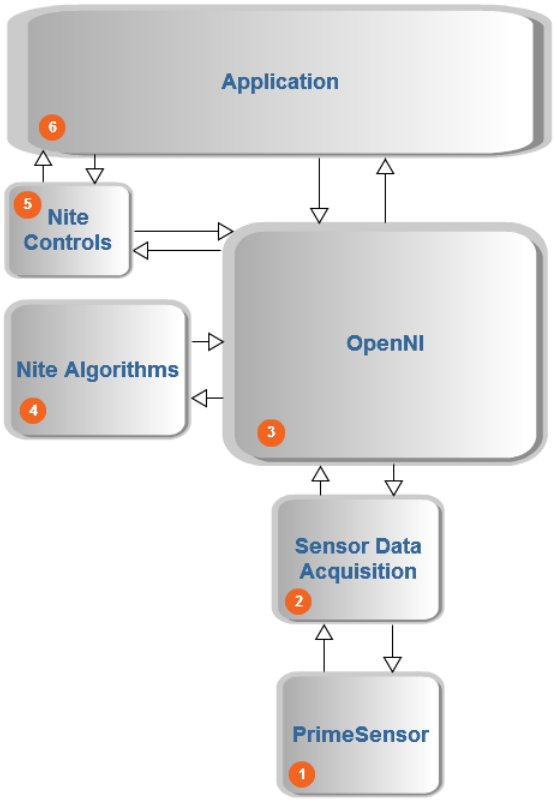

Figure 4.2: Architecture of the OpenNI SDK

6. Application - The layer is capable of using NITE Controls and also approach the OpenNI directly in order to access data generated by NITE Algorithms or the raw data produced by the motion sensor.

As mentioned above the NITE Controls offers methods for recognizing simple gestures. A test of the gesture recognition system was conducted and showed that it was not sufficient due to the fact that gestures was often mixed together.

The overview of the OpenNI SDK has been presented, and the functionality, which needs to be developed in order to developed a natural user interface for web browsing on a PC, is listed below:

- The mapping from a user's hand movement to the cursor on the screen.
- The ability to recognize gestures.
- A suitable user interface.

This functionality will be contained in the application layer described above.

4.4 Programing language

OpenNI SDK supports a range of programming languages, including C#, where there is no significant difference in the amount of features the different programming languages support. Because the authors have a great deal of experience with C# and the development of object oriented software, C# has been chosen.

4.5 Summary

This chapter describes the choice of a motion sensor used to track a user. Two different software development kits are examined and tested in order to determine which one is the most suited. The architecture of the OpenNI SDK is presented and based on this the functionality that must be implemented is defined and listed. Finally the choice of program language is desribed.

CHAPTER 5

DESIGN OF PROTOTYPE 1

This chapter presents different ideas for the design of cursor control, gestures and how the gestures are recognized. The different ideas are discussed, and an idea for the cursor control , the gestures and the recognizing of gestures is chosen.

It must be noted that the interface is developed with the right hand as the users's dominant hand. The reason for omitting the choice of the left hand as the dominant hand is, because it is a trivial implementation and not relevant in this study.

5.1 Cursor

In order to interact with a browser a similar interaction form as a cursor is required. The basic idea is to use the most common used input device, which is a mouse, as inspiration, where the movement of users's hand is mapped directly to the position of the cursor. If the user moves his hand to the left, the cursor should move to the left side of the screen. When tracking a hand using the Kinect and the OpenNI SDK, OpenNI SDK presents the data of the hand as a point in a coordinate system with the size of 640 x 480. The coordinate system has a X-axis going from -320 to 320 pixels and a Y-axis from -240 to 240 pixels as shown in Figure 5.1.
To use the data from OpenNI SDK to move the cursor on the screen the data must be converted. The need for converting the OpenNI SDK data is, because C# is only capable of working with a coordinate system, where the center of the coordinate system is located at the upper left corner of the screen. An illustration of the coordinate system used by C# for a 1920 x 1080 screen is shown in Figure 5.2.
When using a 1920 x 1080 screen the coordinate system used in C# has a x-axis going from 0 to 1920 and a y-axis from 0 to 1080. The data must therefore be converted from a 640 x 480 screen with center in the middle of the screen to a 1920 x 1080 screen that has center in the upper left corner of the screen. This can be archived with Formula 5.1 and 5.2.

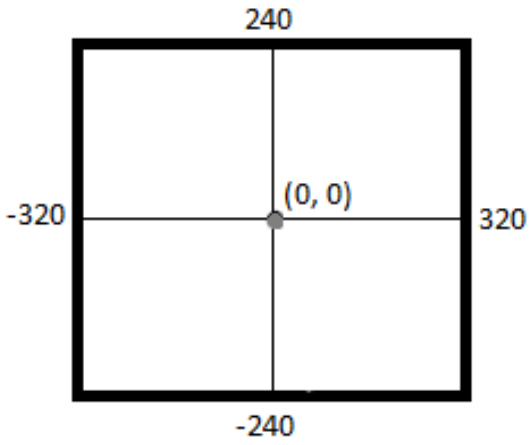

Figure 5.1: Coordinate system used by OpenNI SDK

Figure 5.2: Coordinate system used by C#

$$X_{cursor} = X_{OpenNI} + 960 \tag{5.1}$$

$$Y_{cursor} = -Y_{OpenNI} + 540 \tag{5.2}$$

The addition of 960 and 540 to respectively X and Y in Formula 5.1 and 5.1.1 adjusts the center of the 1920 x 1080 screen to the center of the 640 x 480 screen. If the user moves his hand to the left side, the OpenNI SDK returns a negative X-value for the tracked hand. When applying this X-value in Formula 5.1 a value below 960 pixels is returned, and this results in a movement of the cursor to the left of the screen.

The converting of the Y-value of the Kinect data is similar, however the Y-value must be inverted. The reason for this is because of the coordinate system that is used by the OpenNI SDK, as the Y-value for the tracked hand decreases, when the user lowers his hand. This gives the opposite effect when using this Y-value in the coordinate system used in C#, because a

smaller Y-value moves the cursor to the top. Therefore the data from the OpenNI SDK needs to be inverted.

The mapping of the users hand can be done in two different ways: absolute and relative.

5.1.1 Absolute Mapping

Mapping the hand absolute means that the movement of the users hand is mapped directly to movement of the cursor using Formula 5.1 and . To point at an object in the top right corner, the user needs to place his hand in the top right corner of the view of the Kinect.

The advantage of this is the ability of being more precise with the cursor, because the user has to make larger movement with his hand to make the cursor move on the screen. This is also the disadvantage of absolute mapping, since in order to move the cursor from one corner to another corner of the screen the user needs to move his hand significantly.

5.1.2 Relative Mapping

When using a relative mapping the user's hand is not directly mapped to a point on the screen. Instead the X- and Y-values from the OpenNI SDK is multiplied with a constant, before they are converted into coordinates for the cursor. If the user moves his hand 100 pixels to the left, this is multiplying by a constant as for example three. This results in a movement of the cursor 300 pixels to the left side of the screen.

The advantage of this is that it reduces the required movement of the users hand to reach the corners of the computer screen. The disadvantage of this mapping is that it also increases the uncertainty of the users hand. If the user shakes slightly on his hand, this is also multiplied with a constant and is displayed as movement of the cursor.

The advantages and disadvantages of mapping absolute and relative have been considered, and the relative mapping has been chosen. The reason for choosing the relative mapping is based on an experiment with both mappings. An experiment showed that the larger movement required by absolute mapping faster wear out the user than with relative mapping.

The jitter caused by multiplying the movement of the users hand can also be reduced, if needed.

5.2 Gestures

The functions that are listed in the requirement specification in Section 3 should be able to be performed with the use of gestures. The gestures required to perform these functions can be done in two ways, using the fingers or the arm. The different ideas to make these gestures have therefore been divided into two categories: finger gestures and arm gestures.

5.2.1 Finger gestures

The idea of finger gestures is to use the fingers to make the gestures. An example is by showing one finger the functionality of a mouse click is triggered, and by showing two fingers the

functionality of a double click on a mouse is triggered. In order to perform finger gestures the functionality of counting the number of fingers that are shown is required.

To be able to count the fingers, the available output from the Kinect and the OpenNI SDK has been studied. The available output from the OpenNI SDK is the depth image and the RGB-image, and based on this two possible ideas have been developed and tested. One idea is based on the depth image, and the other idea is based on the use of skin-detection on th RGB-image from the Kinect.

Depth image

The depth image from the Kinect provides the data needed to count the number of fingers that are showed. The depth image from the Kinect is shown in Figure 5.3, and as seen the fingers can be seen as outgoing tops from the palm.

Figure 5.3: Depth image from OpenNI SDK

The number of fingers can then be counted using the Convex hull algorithm. The Convex hull algorithm can be illustrated by having a board with nails in, where an elastic band is stretched around the nails to form an outer border. The Convex hull algorithm is capable of calculating the number of edges that are needed to form the border. This number can then be converted to the number of fingers shown, because each shown finger requires a new edge as seen in Figure 5.4.
The advantage of this idea is that the hand can be separated from background because of functionality provided by the Kinect and the OpenNI SDK. The disadvantage of this approach is the quality of the depth image that the OpenNI SDK delivers. The quality of the depth image is only sufficient in our implementation, when the user holds his hand within 80 centimeters from the Kinect. After the limit of 80 centimeters the depth image becomes blurry, and this increases the difficulty of detecting and separating the fingers from each other. Figure 5.5 shows a screenshot of, when holding the hand at the limit of 80 centimeters.
The OpenNI SDK can also only create a sufficient depth image, when the user is further than 50 centimeters away from the Kinect. If the user is closer than 50 centimeters, the quality of the depth image is reduced significantly and can therefore not be used. To sum up, this is only functional within the limit of 50 - 80 centimeters away from the Kinect, meaning a 30 centimeters spread.

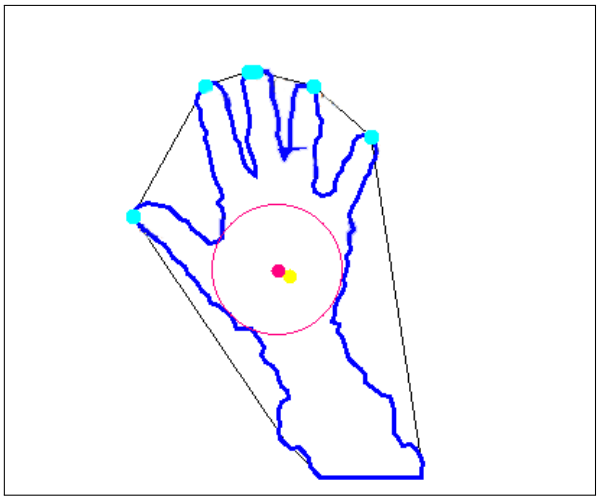

Figure 5.4: Depth image used with Convex hull algorithm

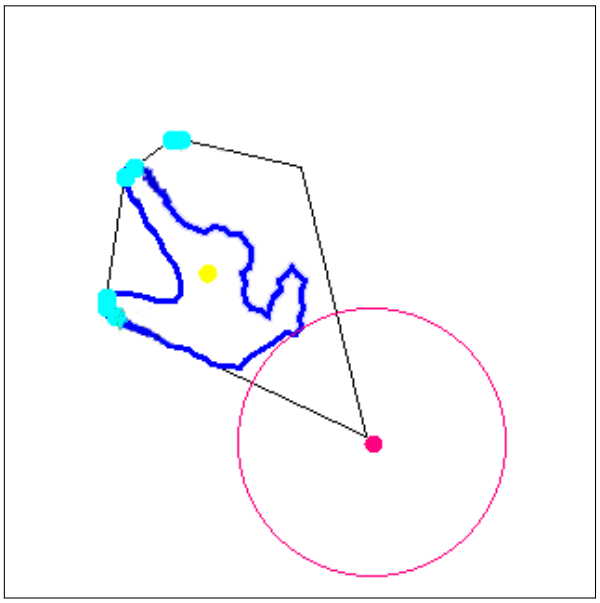

Figure 5.5: When holding the hand at the limit at 80 centimeters

Skin Detection

The other idea is to use the RGB-image from the Kinect, because the quality of the RGB-image is not reduced as significant as the depth image after 80 centimeters. The problem for this approach is to separate the hand from the background, and to handle this separation skin detection is applied. Skin detection used the fact that the human skin has a particular color spectrum that can be used to separate the skin and thereby the hand from the background. Using skin detection the shape of the hand can be detected, and this shape can be applied as data in the Convex hull algorithm. This will as with the depth image return the number of fingers that are shown.

As mentioned the advantage of this approach is that the quality of the RGB-image from Kinect

is sufficient at a larger distance than the depth image. The disadvantage with this approach is the fact that Kinect's perception of the color of the skin changes according to the light settings. The Kinect tries to adjust to the light settings in the room. By switching off the light, the color of objects can appear differently to the Kinect. Other objects than the actual skin can therefore be detected to be within the predefined color spectrum that the skin detection algorithm is searching for.

After considering the advantages and disadvantages none of the ideas for hand gestures are chosen. The reason for not choosing the depth image is the limit of working within 50 to 80 centimeter. This is not sufficient as it is a narrow virtual area, and the users must guess the borders of this area.

The approach with skin detection is discarded because of the sensitivity concerning the light settings. If the system is calibrated, and the light is switched on, this can break the system.
The idea of hand gestures is therefore discarded, as none of the ideas are appropriate.

5.2.2 Arm gestures

The other idea for performing gestures is to use a larger body part than the fingers, and the arms are suitable. The advantage of using the entire arm is that it is possible to perform gestures that are bigger than the finger gestures and therefore easier to detect. Since it is difficult to hold the cursor still and perform a gesture, two arms are needed for interaction in this approach. Otherwise the system would have to predict, when the user wants to perform a gesture, lock the cursor-position, recognize the gesture, perform the required command and unlock the cursor again. Since this is very difficult to do, the left arm is used to perform the gestures.

The idea is based on using OpenNI SDK to track two points of the left arm and compare the data to previous recorded gestures. The two points that should be tracked, are the left hand and the left elbow, and the OpenNI SDK provides functions to output the X-, Y- and Z-coordinates of these two points.

The advantage of arm gestures is that the data that are needed to detect gestures is available in different settings, at different lights and at larger distances. Changing the light settings will not have the same negative effect as with skin detection. The requirements for this approach is that the arm is within view of the Kinect. The disadvantage is that two hands are needed, one hand to control the movement of the cursor and one hand to perform the gestures, when needed. This can have a fatiguing effect on the user.

The disadvantage of using two hands is not considered as a significant problem compared to the advantages of this approach. The user is not required to perform gestures all the time and can therefore rest his other arm between the gestures. The use of arm gestures has been found sufficient.

5.2.3 Design of gestures

The functions that should be mapped to gestures, have been defined in the requirement section under Web browser functionality, and the functionalities are:

- Clicking a link or a button
- Scrolling up on a web page
- Scrolling down on a web page
- Going backward in the web history
- Going forward in the web history

Based on the requirements in Section 3 the gestures should be easy to learn, easy to remember and directly mapped to actions. In order to make the interface more usable the gestures should also not be physically unpleasant to perform.

All of the gestures are designed to start in an initial position where the left lower arm is held in a 90 degree angle according to the left upper arm, pointing the left hand forward. It has been considered, whether the use of an initial position makes the interface more difficult to use. If the initial position is discarded, the advantage would be that users could start their gestures anywhere, they wanted, as long as they performed the remaining of the gesture. However with the initial position it is more simple for users to indicate, when they want to make a gesture, and thereby reducing the possibility of a user making unintended gestures.

Click gesture

In order to make the "Click"-gesture easy to learn and remember, devices from the related work are considered to find a usable mapping. When the Xbox 360 and Kinect sensor are used in Xbox Live, people can click elements by hovering the cursor over the element. For the devices with touch screens, the smart phones and tablets, people press or tab the elements, they want to click. A similar technique is used by Crunchfish, where people make a push movement towards the device, when they want to click an element.

It is difficult to apply the technique used on Xbox Live with web pages, because the click-able elements are not static as in Xbox Live and change from page to page. It is impossible to use the same technique as touch screens, however the movement of moving a finger towards the element, when clicking on a touch screen can be mapped to a gesture. Based on the movement used to click elements on touch screens and the "Click"-gesture from Crunchfish the gesture is defined to be a push movement towards the screen as illustrated in Figure 5.6.

Scroll up and down

Based on the related work two possible ways of designing the "Scroll Up"- and "Scroll Down"-gesture have been considered, the scroll wheel of a mouse and the use of a scrollbar on a web page. The scroll wheel on a mouse works by a circular movement, rolling the wheel upwards makes the browser scroll up. When scrolling up and down on a web page one either clicks the up or down arrow on the scrollbar or drags the scrollbar upward or downwards.

When looking at the scroll wheel the "Scroll Up"- and "Scroll Down"-gesture can be designed with the use of a circular movement. However the article "Mid-air Pan-and-Zoom on Wall-

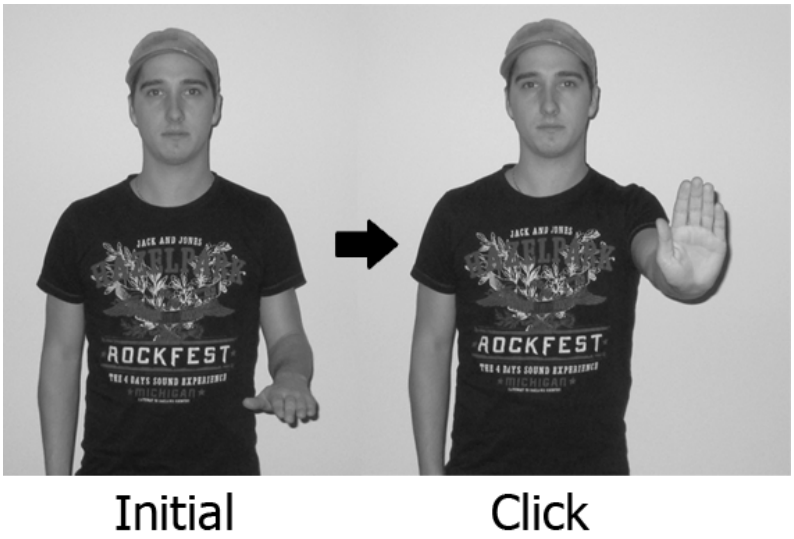

Figure 5.6: The "Click"-gesture

sized Displays" [14] states that linear gestures are faster to perform than circular gesture, therefore the use of a circular movements is discarded.

The dragging of a scrollbar using the mouse can be translated into two different gestures. The "Scroll Up"-gesture which is a linear gesture where the user moves the left arm from the initial position up to a position where the hand is located next the face of the user. The "Scroll Down"-gesture which is a linear gesture where the user moves the left arm from the initial position down to a position where the arm is stretched out pointing at the floor.
Both gestures are illustrated of Figure 5.7.

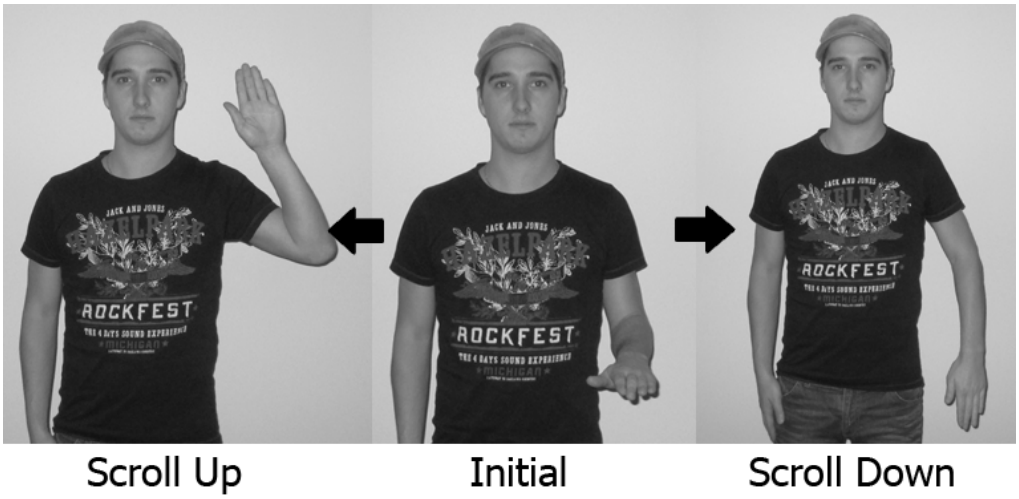

Figure 5.7: The "Scroll Up"- and "Scroll Down"-gesture

Go backward and forward

The definitions of the "Go Backward"- and "Go Forward"-gesture are inspired by the arrows on the buttons used in most browsers to go backward or forward in the web history. The "back"-button in browsers consists of an arrow pointing to the left, and this has led to a "Go Backward"-gesture where the user holds the arm in the initial position and then moving the arm to the left, until it is stretched to a vertical position, as if the user is pointing in that direction.
The "forward"-button in browsers consists of an arrow pointing to the right, which has led to a "Go Forward"-gesture, where the user is required to hold the left arm in the initial position and then moving the hand to a position near the right shoulder.

To keep the interface consistent the "Go Forward"-gesture should be a reflection of the "Go Backward"-gesture, but stretching the left arm to vertical position, while pointing to the right, was found to be physically unpleasant by the authors. The "Go Forward"-gesture has therefore been adjusted and is designed to be pointing to the right, but with the left hand positioned near the right shoulder.

Both gestures are illustrated of Figure 5.8.

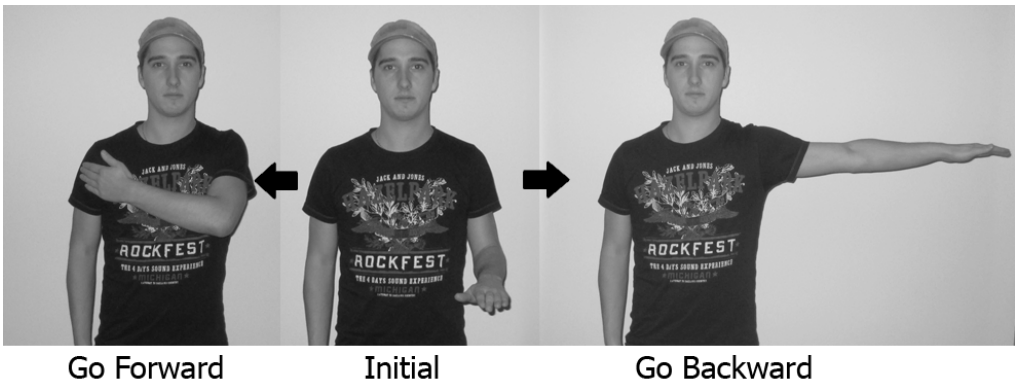

Figure 5.8: The "Go Forward"- and "Go Backward"-gesture

The user requirement from Section 3 specified that the gestures should be directly mapped to action which have been fulfilled through the design of the gestures.

5.2.4 Recognizing gestures

In order to have the functionality of performing gestures, an algorithm is needed for recognizing the users movement. The idea for the choice of algorithm has been inspired by the article [29] and the article in related work. To ensure that gestures can be performed with different speed, the $1-Recognizer is discarded. The DTW-algorithm performs better than the Rubine recognizer, which is the main reason for not choosing the Rubine recognizer.

The other choice is to use HMM-algorithm, which is compared in the article "A Framework for 3D Spatial Gesture Design and Modeling Using a Wearable Input Device"[16]. In the article, the HMM-algorithm and DTW-algorithm perform nearly identical, HMM-algorithm with a

success rate of 91.6% and DTW-algorithm with a success rate of 88.7%. The HMM-algorithm requires a large set of training data, in this case HMM-algorithm uses a set of 20, where the DTW-algorithm only uses 3 templates to compare against. Based on this, the DTW-algorithm has been chosen, because the gestures can then easier be defined and changed.

The DTW-algorithm is described in the paragraph below and is based on the book: "Information Retrieval for Music and Motion"[30].

Dynamic Time Warping-algorithm

DTW is an algorithm that is used to find an optimal alignment between two sequences under certain restrictions, where the two sequences may vary in time and speed. This is important, because DTW is original developed for recognizing for automatic speech recognition, where people pronounce words differently.

The DTW-algorithm requires the input data to be points in time from which a local cost measure or a local distance measure can be calculated. The local distance measure means the calculation of a distance between a point in the stored sequence and a point in the actual sequence. The local distance measure is also referred to as $c(x, y)$. If the cost is low, it means that the two point are similar to each other. This is illustrated in Figure 5.9, where two sequences, X and Y, are displayed.

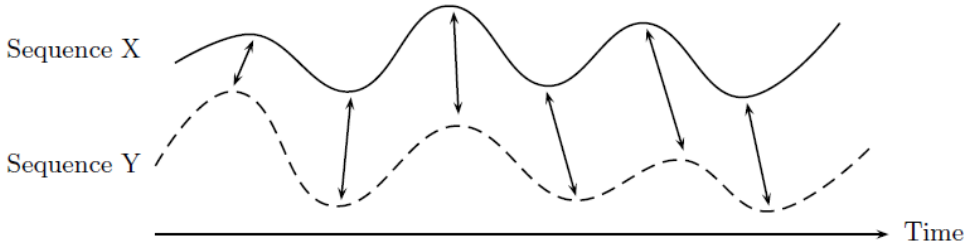

Figure 5.9: Time alignment of two sequences X and Y[30]

Sequence X shows the actual sequence with size `N`, meaning $X := x_1, x_2, ..., x_N$. Sequence Y shows the stored sequence with size `M`, meaning $Y := y_1, y_2, ..., y_M$. The arrows between the two sequences shows the aligned points from X and Y, meaning that these two points are used to calculate the local distance measure. For each pair of elements from sequence X and Y the local distance measure is calculated and inserted into a cost matrix with size `N*M`. The goal is then to find an optimal path that has minimal cost, but there are certain conditions that an optimal path must obey.

1. The boundary condition, meaning that the path must begin at (1,1) and end at (N,M).
2. The monotonicity condition, meaning the order of the point must be maintained.
3. The step size condition, meaning that the step size can not exceed 1 and therefore only go to (x+1, y), (x, y+1) or (x+1, y+1).

An example of a correct path and three paths that each violates one of the conditions, are illustrated in Figure 5.10.

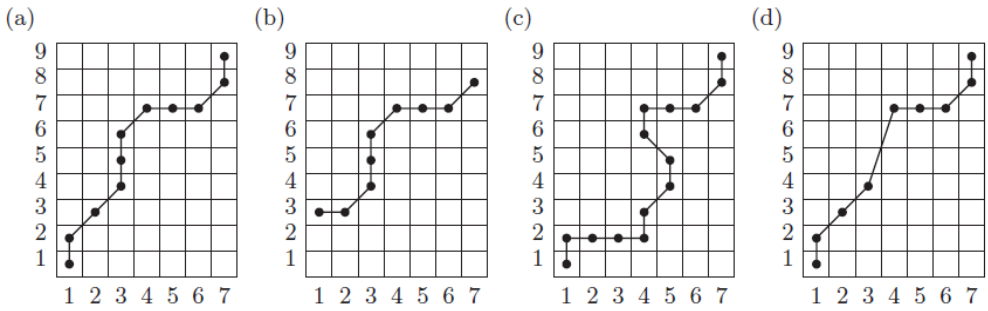

Figure 5.10: Examples of paths[30]

In Figure 5.10 **(a)** displays a correct path. **(b)** shows the path violating the boundary condition. **(c)** displays a path that violates the monotonicity condition, and **(d)** shows a path with a step size more than one and thereby violating the step size condition.

The optimal path between sequence X and Y is a path having the minimal total cost amongst all possible paths, and the length of this path is defined as the `DTW-distance` $DTW(X,Y)$. The `DTW-distance` can be calculated by going through every possible path and comparing them, but a quicker way is to use dynamic programming. The reason for using dynamic programming is the fact that an optimal path from 1 to 9 must consist of an optimal path from 1 to 2 and an optimal path from 2 to 9. By dividing the path into smaller paths that are easier to calculate, the computational complexity is decreased from exponential to the length of N and M to N*M.

To calculate an optimal path the accumulated cost matrix is introduced. The accumulated cost matrix is a N x M matrix, which consists of the values $D(n,m)$, where $D(n,m) := DTW(X(1:n), Y(1:m))$. This means that $D(n,m)$ is the length of an optimal path for $X := x_1, x_2, ..., x_n$ and $Y := y_1, y_2, ..., y_m$, and Formula 5.3 shows the calculation of D(n,m).

$$D(n,m) = min\{D(n-1,m-1), D(n-1,m), D(n,m-1)\} + c(x_n, y_m) \tag{5.3}$$

Formula 5.3 shows the use of dynamic programming, where $D(n,m)$ is calculated based on the previous paths. $D(N,M)$ is the length of an optimal path for $X(1:N)$ and $Y(1:M)$ and thereby also equal to $DTW(X,Y)$. $D(N,M)$ denotes the difference between the actual sequence X and the stored sequence Y, and this tells, how similar the two sequences are.

To sum the DTW-algorithm takes two sequences as input, the actual sequence X and the stored sequences Y. The distances between the aligned points in X and Y are calculated and stored in a matrix, the cost matrix. This matrix is used along with dynamic programming to find the lengths of optimal sub paths and final the length of a full optimal path. The length of a full optimal path shows, how similar sequence X is to sequence Y.

5.3 User Interface

A web browser is required and must therefore be integrated into the user interface. The user should also be able to see, when the system is recognizing a gesture. If there is no indication of, whether the system has recognized a gesture, this can increase the difficulty of using the system. If for example the user tries to click a link and performs the correct gesture, but misses, the user is unable to see, whether the user missed the link or made a incorrect gesture. Therefore to inform the user that he has performed the gesture correctly, the system writes the recognized gesture for example: "Going back" in a text box.

In order to lighten the learning of the gestures the RGB-image from the Kinect is shown. By having the RGB-image at their disposal the users can see the limits of the Kinect and is able to see them selves performing gestures. This can assist the learning of the gestures, since the users can see their movements, when performing gestures.
The user interface is therefore designed as shown in Figure 5.11.

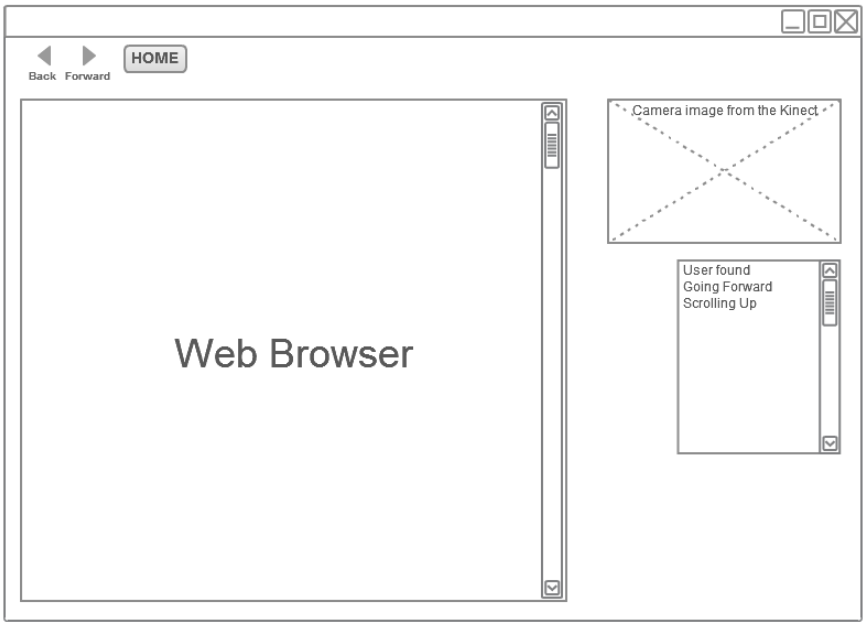

Figure 5.11: Mockup of user interface

5.4 Summary

This section has described and discussed ideas for tracking a hand and how to map the hand movement to the cursor control. Advantages and disadvantages with different ideas of how to make gestures with either the hand or with the arm have been described, and choice of using arm gestures is made. Finally as an algorithm for recognizing gestures the DTW-algorithm is chosen, and a description of the DTW-algorithm is given.

CHAPTER 6

IMPLEMENTATION OF PROTOTYPE 1

This chapter describes the implementation of the system capable of acting as a natural user interface. The most important parts of the system are introduced and described in detail.
The architecture of the application including OpenNI SDK is shown on Figure 6.1. Compared to the architecture of OpenNI SDK shown in Figure 4.2, the application layer has been divided into several parts. The parts that are described in detail in this chapter are listed below. The names marked with this `font` is equal to the names shown in the architecture in Figure 6.1.

1. **User tracking** - Tracking of a user to be able to retrieve data of the user located in the view of the Kinect. This is done through the `UserGenerator` object.
2. **Hand tracking** - Tracking of the right hand of the user in order to retrieve coordinates of the hands position. This is needed to be able to move the cursor on the screen according to the current position of the right hand. This is done using the `HandGenerator` object.
3. **Filtering of the data for the right hand** - The filtering is done to remove jitter in the data used to control the cursor. This is done using the `Filter` element.
4. **Recognizing gestures** - Describes the implementation which makes it possible to recognize gestures performed with the left arm. This is done using the `DTW`-algorithm

6.1 User tracking (UserGenerator)

The OpenNI SDK provides several objects which makes it possible to track a user. This is needed in order to recognize gestures performed with the left arm and control the cursor based on the location of the right hand. The `UserGenerator` is an object provided by OpenNI, which recognizes the number of users located in the view of the sensor. Furthermore it provides events which are triggered when a new user is detected, and when an existing user disappears. When a new users enters the view of the Kinect, the system has to be calibrated. In order for

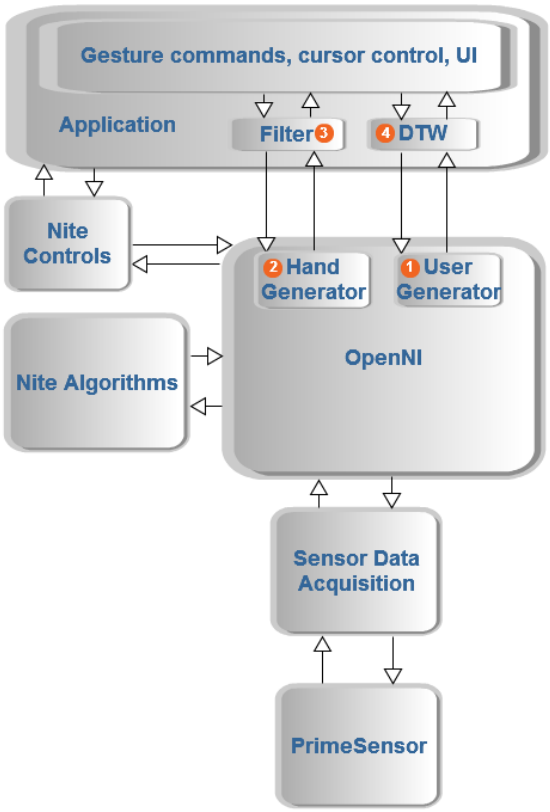

Figure 6.1: Architecture of the application

the system to recognize the different joints on a user's body, it has to be calibrated according to the user's build. This is done by requesting the user to stand in a specific pose as shown in Figure 6.2.
Source Code 6.1 shows the instantiation of the needed objects for detecting and calibrating a user.
Line No. 2 instantiates a `UserGenerator` object which is used to detect users located within the view of the sensor.

Line No. 5 instantiates a `PoseDetectionCapability` object which is used to detect when a user is standing the predefined pose.

Line No. 8 instantiates a `SkeletonCapability` object that is able to calibrate the data according to the users build.

```
1 //Creates a Usergenerator object
2 UserGenerator _userGenerator = new UserGenerator(this._context);
3
4 //Creates a PoseDetectionCapability object
5 PoseDetectionCapability _poseDetectionCapability = new ↵
      _userGenerator.PoseDetectionCapability;
6
7 //Creates a Skeletoncapability object
```

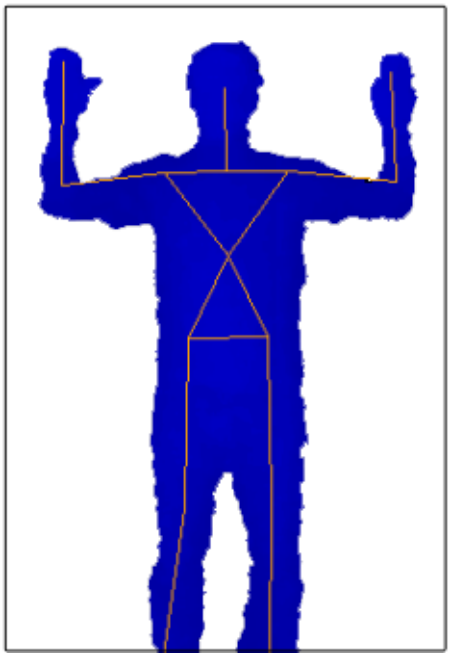

Figure 6.2: Pose used when calibrating a user

```
8  SkeletonCapability _skeletonCapability = new ↵
       _userGenerator.SkeletonCapability;
```

Source Code 6.1: Objects used to detect and calibrate a user

Source Code 6.2 shows the event that is triggered, when a new user enters the view of the Kinect. When a new user is detected the event handler begins searching for the predefined pose shown in Figure 6.2.

```
1  void UserGenerator_NewUser(object sender, NewUserEventArgs eventsArgs)
2  {
3       //Start pose detection
4      this._poseDetectionCapability.StartPoseDetection(this._calibrationPose, ↵
           eventsArgs.ID);
5  }
```

Source Code 6.2: Event handler for new user

If the right pose is detected the event handler stops the pose detection and requests calibration of the user as shown in Source Code 6.3.

```
1  void PoseDetectionCapability_PoseDetected(object sender, ↵
       PoseDetectedEventArgs eventsArgs)
2  {
```

```
3       //When the right pose is detected, stop further detection
4       this._poseDetectionCapability.StopPoseDetection(eventsArgs.ID);
5       //Request calibration of the user
6       this._skeletonCapability.RequestCalibration(eventsArgs.ID, true);
7   }
```

Source Code 6.3: Event handler for pose detection

If the system is able to successfully calibrate based on the performed pose the system continues or else the user is requested to assume the pose from Figure 6.2 again. The implementation is shown in Source Code 6.4

```
1   void SkeletonCapability_CalibrationComplete(object sender, ↵
        CalibrationProgressEventArgs eventsArgs)
2   {
3       //If calibration is successful
4       if (eventsArgs.Status == CalibrationStatus.OK)
5       {
6                   //Start tracking the user
7                   ...
8       }
9       else
10      {
11          //Do the pose detection again
12          this._poseDetectionCapability.StartPoseDetection(_calibrationPose, ↵
                eventsArgs.ID);
13      }
14  }
```

Source Code 6.4: Event handler for user calibration

When the calibration of a user is complete it is possible to retrieve the X-,Y- and Z-coordinates of the joints, represented by circles in Figure 6.3, on the user standing in the view of the Kinect by using the `_skeletonCapability`.

6.2 Hand tracking (HandGenerator)

To be able to control the cursor it is necessary to track the position of a user's hand. This can be archived in two different ways. One is to use `_skeletonCapability` to track the position of the right hand. Another way is to use the `HandGenerator` object which by default only tracks the left or right hand. In order to choose the most suited technique, the precision of both the skeleton tracker and the `HandGenerator` have been tested. The test showed that less jitter was experienced when using the `HandGenerator` compared to the texttt_skeletonCapability therefore the `HandGenerator` was chosen to track the right hand.

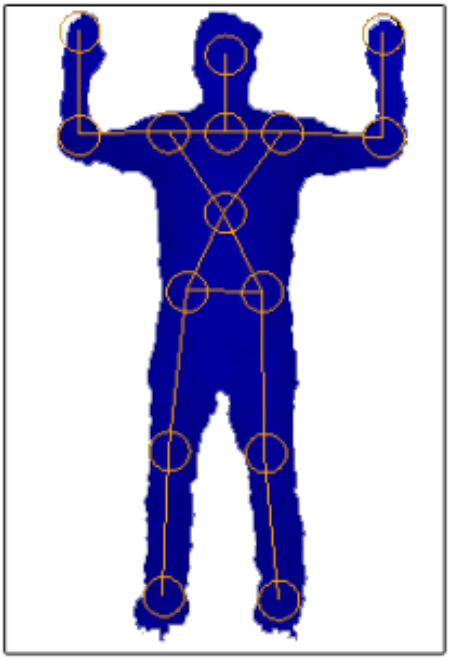

Figure 6.3: Joints to retrieve coordinates from

Source Code 6.5 shows the implementation of the `calibrationComplete` event handler which starts the tracking of the user's right hand using the `HandGenerator`. Line No. 7 instantiates a skeleton joint on the right hand, and Line No. 15 retrieves the position of the right hand. Line No. 17 starts the tracking of the right hand using the `HandGenerator`.

```
1  void SkeletonCapability_CalibrationComplete(object sender, ↵
       CalibrationProgressEventArgs eventsArgs)
2  {
3      //If calibration is successful
4      if (eventsArgs.Status == CalibrationStatus.OK)
5      {
6          //Instantiates a skeleton joint on the right hand
7          SkeletonJoint rightHand = SkeletonJoint.RightHand;
8          //Starts tracking the predefined joints
9          _skeletonCapability.StartTracking(eventsArgs.ID);
10
11         //Checks, whether a user is been tracking
12         if (this._skeletonCapability.IsTracking(1))
13         {
14             //Gets the position of the right hand using skeleton tracker
15             SkeletonJointPosition rightHandPosition = ↵
                   this._skeletonCapability.GetSkeletonJointPosition(1, ↵
                   rightHand);
16             //Starts tracking the right hand using the handgenerator ↵
                   given the position of the hand
17             _handsGenerator.StartTracking(rightHandPosition.Position);
```

```
18          }
19          ...
20      }
21      else
22      {
23          this._poseDetectionCapability.StartPoseDetection(_calibrationPose, ↵
                eventsArgs.ID);
24      }
```

Source Code 6.5: Start tracking of hand

This enables the system to retrieve the X- and Y-coordinates of the right hand, when a user is being tracked.

6.2.1 Moving the cursor based on data

Moving the cursor is done by setting the cursor position to the X- and Y-coordinates of the position of the right hand retrieved from the OpenNI SDK. Every time the `HandGenerator` retrieves data from the Kinect, the event handler `HandGenerator_Handupdate` triggers. The implementation of the event handler is shown in Source Code 6.6. The event handler sets the global variable `_rightHand`, that is a 3D-point containing the current coordinates of the hand, to the coordinates retrieved from OpenNI SDK.

```
1  void HandGenerator_HandUpdate(object sender, HandUpdateEventArgs ↵
       eventArgs)
2  {
3  _rightHand.X = eventArgs.Position.X;
4  _rightHand.Y = eventArgs.Position.Y;
5  _rightHand.Z = eventArgs.Position.Z;
6  }
```

Source Code 6.6: HandUpdate event handler

By importing the `user32.dll` that contains a method called `SetCursorPos` it is possible to specify the position of the cursor on the screen based on X- and Y-coordinates. Source Code 6.7 shows the method `setCursor` which calls the `SetCursorPos` method. An explanation of the computation made to change the coordinates from the `HandGenerator` to appropriate input parameters to the `SetCursorPos` can be found in Section 5.1.
In order to position the cursor on the screen relative mapping has been used, which means that a constant is multiplied with the X- and Y-coordinates of the hand, and the constant is the value of 3. This means that when the hand is moved 10 pixels, the cursor moves 30 pixels on the screen. The constant of 3 has been found to be sufficient through tests performed by the authors. The constant enables the user to position the cursor in any of the four corners of the screen without having to move around.

```
1 public void setCursor(float xCoordinate, float yCoordinate)
2 {
3    //Sets the cursor position to the coordinates retrieved from OpenNI ↵
         SDK
4    SetCursorPos((int)((xCoordinate * 3) + 960, (int)((-(yCoordinate * ↵
         3) + 540));
5 }
```

Source Code 6.7: The SetCursor method

6.3 Filtering of data for the right hand (Filter)

It was discovered after the implementation of tracking the right hand with relative mapping that the jitter of the users hand was enough to increase the difficulty of clicking links. The jitter made it difficult to position the cursor over a link and hold it still while performing the "Click"-gesture. A technique must be used to compensate for this, and one technique that is applicable is filtering the data retrieved by the `Handgenerator`. Two different filters were implemented and tested. The reason for choosing these two filters was based on experience from a previous project, where the filters were applied to similar data in form of GPS-coordinates and found sufficient.

Simple moving average filter

The first filter is based on the average of the last n number of coordinates. This should level out the "spikes" contained in the data received from the sensor. Spikes are coordinates that are either much higher or lower than the previous coordinates and often considered as incorrect data from sensors. The filters convert noisy data into smoothened data by calculating the average value based on the previous values. The formula for calculating the average value is shown in Equation 6.1.

$$average = \frac{p_0 + p_1 + p_2 ... p_n}{n} \tag{6.1}$$

The implementation of the simple moving average filter is shown in Source Code 6.8. The source code only shows the filtering of the data for the X-coordinate since it is identical to the filtering of the Y-coordinates. Line No. 4 checks how many coordinates the `_mouseXCoordinates` list contains. If the list contains `numberOfCoordinates` amount of coordinates, the oldest coordinate is removed. Line No. 6 add the newest coordinate received to the list. Line No. 8 and 9 add all of the coordinates in `_mouseXcoordinates` to the `_averageX` variable. Line No. 12 calculates the average by dividing `_averageX` with the total amount of coordinates. Finally Line No. 15 sets the position of the cursor by calling the method `setCursor` with the calculated averages as input parameters.

```
1 private void SetCursorWithSimpleAverageFilter(int numberOfCoordinates)
```

```
2  {
3      //Filtering of the X-coordinates
4      if (_mouseXCoordinates.Count == numberOfCoordinates)
5          _mouseXCoordinates.RemoveAt(0);
6      _mouseXCoordinates.Add(_rightHand.X);
7
8      foreach (float x in _mouseXCoordinates)
9          _averageX += x;
10           ...
11     _averageX = _averageX / _mouseXCoordinates.Count;
12     ...
13     setCursor(_averageX, _averageY);
14 }
```

Source Code 6.8: Simple moving average filter

This filter can be adjusted by lowering or increasing the amount of coordinates. With a low number of coordinates the cursor becomes more responsive, but suffers more from spikes. With a high number of coordinates the cursor is less sensitive to spikes, but the response time is decreased. Throughout experiments 10 showed to an adequate number of coordinates to have a sufficient response time and not to be too sensitive to spikes. If the amount of coordinates was increased to more than 10, the response time of the cursor became to high and thereby making it hard to control the cursor.

Exponential moving average filter

Exponential moving average filtering works in a similar way to the simple moving average filter. Exponential moving average filter is a moving average filter with exponential weights, where the more recent values are given more weight. Equation 6.2 shows the formula for exponential moving average filtering.

$$s_t = a \times Y_t + (1 - a) \times s_{t-1} \tag{6.2}$$

s_t is the smoothed value.

a is the smoothing factor which must be between 0 and 1. A low a value results in high level of smoothing and thereby a lower response time, whereas a high a value makes it more sensitive to spikes.
Through experiments 0.12 was found as a reasonable value for a.

Y_t is the output from the `Handgenerator` at time t. s_{t-1} is the previous smoothed value. The implementation of the exponential moving average filter for the X-coordinate is shown in Source Code 6.9. The implementation of the exponential moving average filter for the Y-coordinate has been omitted, because it is identical to the implementation for X-coordinate.

```
1  private void SetCursorWithExponentialAverageFilter(float a)
2  {
```

```
3         //Equation of valueX
4         float valueX = a * _rightHand.X + (1 - a) * _oldValueX;
5         _oldValueX = valueX;
6         ...
7         setCursor(valueX, valueY);
8 }
```

Source Code 6.9: Exponential moving average filter

Choice of filter

In order to choose a filter, experiments with each filter were conducted, and a comparison was made. The simple moving average filter was found to reduce the jitter, but the difficulty of clicking links was still high. The use of exponential moving average filter resulted in a fairly steady cursor. Depending on the smoothing value, the filter removed a large part of the jitter, while still maintaining a reasonable response time. Therefore the exponential moving average filter has been chosen.

6.4 Recognizing gestures (DTW)

To recognize gestures, data is needed, and as mentioned is Section 6.1, the OpenNI SDK provides an object called `SkeletonCapability` which can be used to retrieve the X-, Y- and Z-coordinates of the joints of the body. As mentioned in the design of arm gestures tracking of the two joints, left hand and left elbow, is required to recognize gestures, since the coordinates of these joints change, when a person moves the left arm. The set of joints chosen for recognizing gestures is extended with the right and left shoulder. The data from the left and right shoulder is used to centralize the data, which is explained in detail in Section 6.4.1. Finally in order to compare a movement with previous recorded gestures the DTW-algorithm is used and the implementation is described in Section 6.4.2

6.4.1 Centralizing data

When a user moves around within the view of the Kinect the coordinates of the different joints on the user's body change as well.

If a user takes one step to the side holding the left hand in the same position, the coordinates of the left hand changes according to the distance moved. This also means that if a gesture is recorded standing in the middle of the view of the Kinect , the gesture will only be recognized when the gesture is performed in the exact same place.

In order to enable users to perform gestures at any place within the view of the Kinect, the coordinates of the joints used to perform and recognize gestures must be centralized. The centralization of the coordinates is divided into two steps. First step is to ensure that it does not depend on where the user is located on the X- and Y-axis.

The idea is to remove the dependency of the actual coordinates of the joints and replace it with relative coordinates. The relative coordinates of a joint is based on the distance from the given joint to a point located between the shoulders. This means that if the user moves around in the view of the Kinect the relative coordinates of the joints remain the same. The coordinates of the point located between the shoulders are calculated by adding the coordinates of the left shoulder to the coordinates of the right shoulder and dividing it with two.
The relative coordinates for a joint are first calculated in XY-plane and are calculated by subtracting the actual coordinates of a joint that need to be centralized from the coordinates of the point located between the shoulders of the user. The formula is shown in Equation 6.3.

$$relativeCoordinatesForJointInXAndY = joint - \frac{leftShoulder + rightShoulder}{2} \tag{6.3}$$

The last step is to ensure that the user is able to perform gestures close to and far away from the Kinect, meaning that the user moves around on the Z-axis. When the user moves closer to the Kinect, the distance between the left and right shoulder increases. This means that the `relativeCoordinatesForJointInXAndY` changes. In order to ensure that the user can move around on the Z-axis, the `relativeCoordinatesForJointInXAndY` is divided with the distance between left and right shoulder on XY-plane. The result of this is the final relative coordinates of the given joint, and the formula for this can be seen in Equation 6.4.

$$finalRelativeCoordinatesForJoint = \frac{relativeCoordinatesForJointInXAndY}{\sqrt{(shoulderDistX)^2 + (shoulderDistY)^2}} \tag{6.4}$$

`shoulderDistX` and `shoulderDistY` is the distance between the left and right shoulder on the X- and Y-axis.

The joint is now centralized around a point between the shoulders which ensures that it does not matter where the user is located in the view of the Kinect. Figure 6.4 shows the relative coordinates and the actual coordinates of the left hand of two users located in the view of the Kinect at the same time. The difference in the relative coordinates of the left hand for each user is little to none whereas the actual coordinates differ a lot.

6.4.2 Recognizing gestures using DTW-algorithm

To be able to recognize gesture the performed gestures must be compared to previous recorded gestures. The recording of a gesture is done by storing the centralized coordinates of the chosen joints. A gesture is a sequence of centralized X-, Y- and Z-coordinates of the joints based on the last 30 observations.

The DTW-algorithm is used to compare a performed gesture to a recorded gesture. As mentioned in Section 5.2.4 the DTW-algorithm dynamically creates a matrix and computes the distance between the points in Sequence X and Sequence Y.

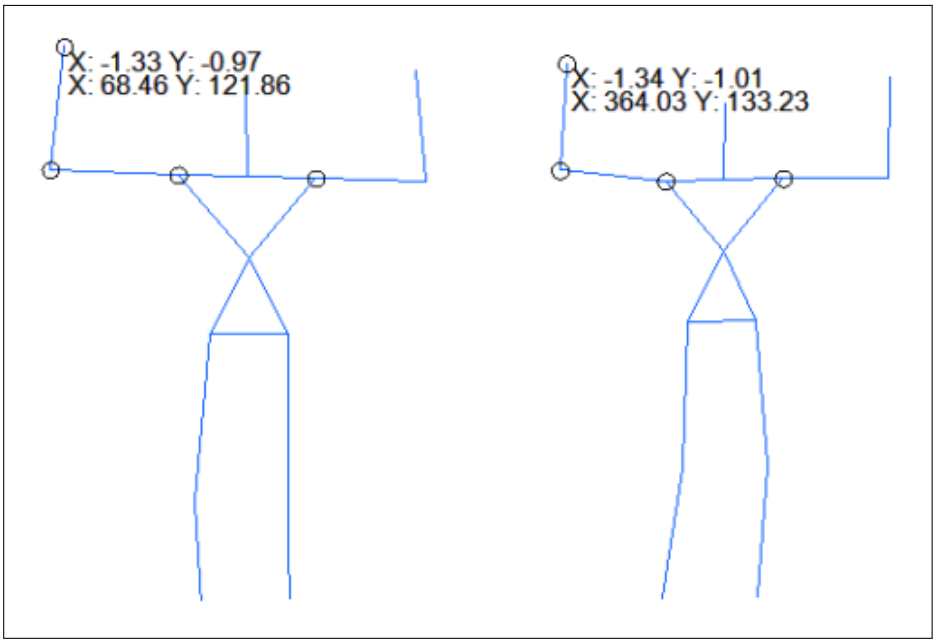

Figure 6.4: Coordinates of the left hand of two users

The distance between two points is calculated using the Euclidean distance formula shown in Equation 6.5. The Euclidean distance between point p and q is the length of the vector |pq|

$$|pq| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2} \tag{6.5}$$

The algorithm goes through the first step every time data is retrieved, and if the check returns true, the following steps are executed:

1. Checks that the end of the gesture performed is within a range of a recorded gesture.
2. Creates a cost matrix of size 30x30 and sets all cells to positive infinity.
3. Inverts the data contained in Sequence X and Y.
4. Calculates the cost of the optimal path to each cell.
5. Finds the path with the lowest cost that goes through the matrix.
6. Checks whether the cost of the path is within the specified boundary.
7. Returns either the name of the gesture recognized or `UNKNOWN`.

The steps are described in further detail below.

1) To ensure the algorithm is not executed every time the Kinect receives data a simple check is performed. The distance from the last point in the sequence of a performed gesture must be within a certain range called `firstThreshold` to the last point in one of the recorded gestures. This ensures that movements, which do not have similarity at the end points, are not compared to any of the recorded gestures.

2) The created matrix is of size 30x30 since both the performed gesture and the recorded gestures consist of a sequence of point over the last 30 observations. All cells are set to positive infinity in order to be able to find a smaller cost when finding a path through the matrix.

3) All gestures require the user to start in the initial position which means that it is not the beginning of a movement that defines the gesture, but more the end of the movement.

It is possible that the algorithm creates a path through the matrix which does not use all the points in one of the sequences. The reason is that a point in one of the sequences can be mapped to several points in the other sequence due to the dynamic time warping ability of the algorithm. The effect of this is that a part of the performed gesture is not taken into consideration when comparing it with a recorded gesture. To ensure that the end of a movement has a higher priority than the beginning of a movement the sequences are inverted.

4) To calculate the cost of the optimal path to each cell the Formula 6.6 from the description of the DTW-algorithm in Section 5.2.4 is used:

$$D(n,m) = min\{D(n-1,m-1), D(n-1,m), D(n,m-1)\} + c(x_n, y_m) \tag{6.6}$$

The algorithm calculates the cost of the path from (0,0) to all cells in the matrix using the Euclidean distance formula described above. The algorithm starts in cell `(0,0)` and ends in `(30, 30)` as shown in Figure 6.5. Figure 6.5 shows two possible paths through the matrix (p1 and p2).

5) The path with the lowest cost is found by locating the lowest value in the row marked with a square in Figure 6.5. The path itself is not interesting, but the cost is used to decide if the performed gesture is similar to the recorded gesture according to the boundaries. The original algorithm has been modified to suit the fact that the end of a movement has a higher priority than the beginning of a movement. This is done by not requiring the path to end in `(30, 30)`, but in the last row, which is marked with a square.

6) The boundary consists of a variable called `threshold`. `threshold` defines the maximum distance between any points in a gesture.

7) If the algorithm returns a minimum cost lower than the `threshold`, the algorithm returns the name of the stored gesture which matched the performed gesture.

6.5 Evaluation of the requirements

To evaluate the implementation, the technical requirements is compared to the functionality implemented. The technical requirements described in Section 3 are the following:

- Moving the cursor.
- Clicking on a link or a button.
- Scrolling up on a web page.

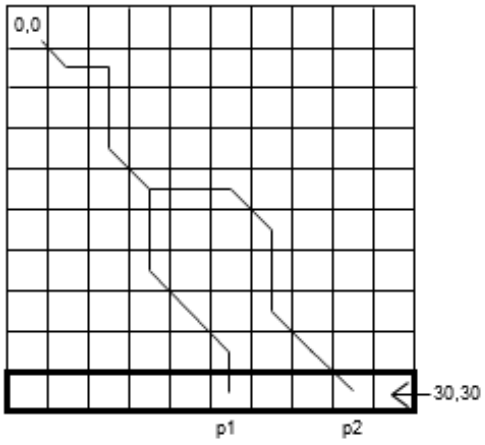

Figure 6.5: Matrix showing the path with a minimal cost

- Scrolling down on a web page.
- Going backward in the web history.
- Going forward in the web history.

Using the prototype the users are able to control the cursor with the right hand and at the same time perform gestures with the left arm. The prototype supports the ability to perform gestures for each of the functions listed above.

6.6 Summary

This chapter has described the implementation of the first prototype developed called Prototype 1. The most important parts of the prototype has been described in detail, including: User tracking, Hand tracking, Filtering and Recognizing gestures. Lastly the technical requirements have been evaluated to ensure that the prototype fulfills the technical requirements.

CHAPTER 7

TEST OF PROTOTYPE 1

This chapter describes the test of Prototype 1. It is divided into two parts, and the first part is the method part that describes the basics about the test, the setup, the procedure, the task etc. The second part is the result part that contains the results from the test, and a discussion of the results is given.

7.1 Method

The purpose of this experiment was to test, how the users navigated a website with this prototype using mouse-based and gesture-based interaction. Mouse-based interaction is with a standard computer mouse, and gesture-based interaction is using the Kinect.

7.1.1 System

The system consisted of a computer connected with a 42" screen, a mouse, a keyboard and a Kinect. The system was set up as shown in Figure 7.1, where the screen was installed on a shelf with a table placed in front of it, and the Kinect was mounted on top of the screen. A test monitor was present throughout the test to present the tasks that the test subject should solve.

The system is further described in the implementation Chapter 6. The system was tested with the two types of interactions, mouse-based and gesture-based, with a web page called dkkogebogen.dk. This web page is an online cook book, where different recipes can be found, and the reason for choosing this web page is the fact that the main part of the interaction with this web page can be done with the use of a mouse, so a keyboard is not required. For the two types of interactions the same computer and screen were used without making any adjustments, such as changing resolution. Otherwise the differences between the two interactions are described in the paragraphs below.

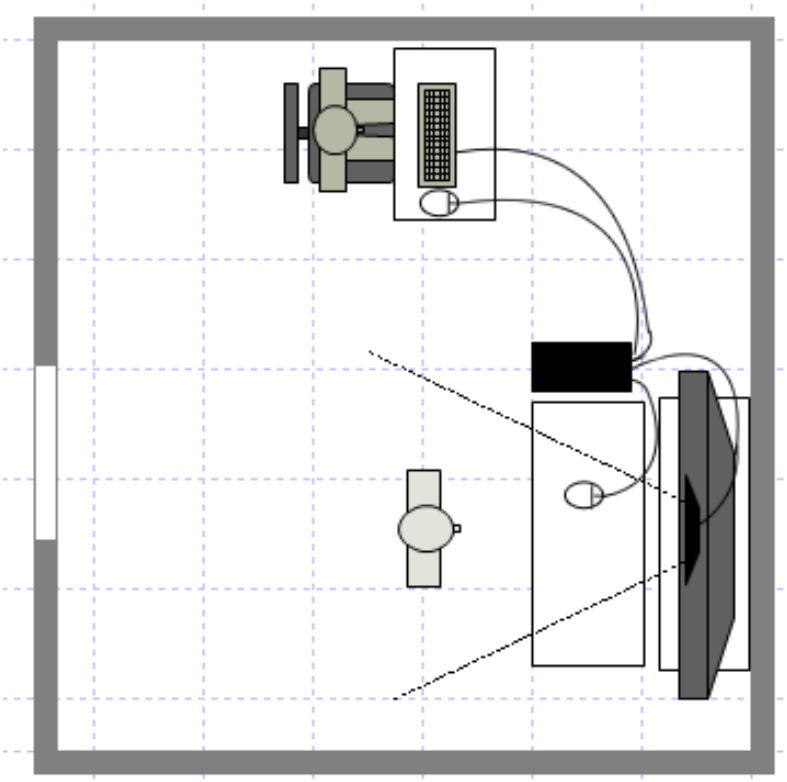

Figure 7.1: Setup of the system

Mouse-based interaction

For the mouse-based interaction only the mouse was positioned on the table and thereby available to the test subjects. The keyboard was placed on another table that only the test monitor had access to.

Gesture-based interaction

During the test session with the gesture-based interaction the test subjects were told to stand in front of the screen at a distance of their choice. The mouse and keyboard were placed at the test monitor's table, so the test subject could not interact with these.

7.1.2 Participants

A total of eight test subjects participated. The demographical data about participants is listed in Table 10.1.
As shown in Table 10.1 four males and four females with variating experience with IT participated. The categories in Table 10.1 are self-explanatory, apart from the Test order. The Test order refers to the order of, how the test subjects tried the different interactions. T-G-M means that the test subject had training in how to use the gesture-based interaction, then the gesture-based test and finally the mouse-based test. M-T-G means, that the test subject first completed the mouse-based test, then had training and finished with the gesture-based interaction.

Group:	Test person:	Sex:	Age:	Education:	Test order:
1	A	Male	27	Technical education	T-G-M
1	B	Male	24	Technical education	T-G-M
1	C	Female	20	Non-technical education	T-G-M
1	D	Female	23	Non-technical education	T-G-M
2	E	Male	26	Technical education	M-T-G
2	F	Male	23	Technical education	M-T-G
2	G	Female	20	Non-technical education	M-T-G
2	H	Female	23	Non-technical education	M-T-G

Table 7.1: Demographical data of participants

We tried to choose the test subjects based on sex, age and experience with IT to get a larger variance of test subjects. However there was a limited number of test subjects available, which meant that there were some limitations with balancing educations. As seen in Table 10.1 it is only the males that were studying or had finished a technical education. We are aware that this can have an influence on the test results, but as mentioned these were the test subjects that were willing to participate at the time. None of the test subjects had prior experience with a Kinect. The test subjects received no payment, but after completion they were given bag of candy for their effort.

7.1.3 Setting

The test was conducted in the usability laboratory at the university. A test monitor was present throughout the test to present the tasks. In the usability lab two cameras were setup to record the test subject. One camera was placed to face the front of the test subject. The other camera recorded the participant from behind, and the screen was also recorded by that camera.

7.1.4 Procedure

The test subjects were introduced to the test and instructed to announce when they had completed a task. A test monitor was presented during the tests and was in charge of dictating the tasks to the test subject. The test monitor was the same person throughout all the tests in order to ensure that the test subjects were given as equal training as possible. Each test subject was given approximately 30 minutes to carry out the test which includes the introduction and the training session. The test monitor tried not to interfere in the tests, however if the test monitor observed that the test subject needed help in order to complete the tasks, the tests subjects received help from the test monitor. Half of the test subjects started with using gesture-based interaction, and the other half started with using mouse-based interaction.

As mentioned the test subjects would receive a training session, where the test subject would be introduced to the gesture-based interaction before the actual test. The training session did always take place right before the test with gesture-based interaction. Therefore the test subjects that started with the mouse-based interaction, would have the mouse-based interaction, then the training session and finish with the gesture-based test. The other half of the test

subjects started with the training session, then the gesture-based interaction test and had the mouse-based interaction test as the last test.

There were two reasons for having the training session right before the test of the gesture-based interaction. One reason was that it would be easier for the test subjects to remember the different gestures, since there was a minimal time between learning the gestures and using them. The other reason was to consider the fact that the gesture-based interaction uses the human body movement, and this could make the test subjects feel tired. If the test subjects had the training session before the mouse-based interaction test, they would have time to rest before the gesture-based interaction test oppose to the test subjects that had the training session and then the gesture-based interaction test. After completing the two tests each test subject was interviewed by the test monitor to find advantages and disadvantages with the two different interactions. The test subjects were asked to describe their thoughts about controlling the cursor with the gesture-based interaction and their opinions of the different gestures.

7.1.5 Training

The test subjects were introduced to the gesture-based interaction, and the test monitor told the test subjects how to control the cursor and the possible gestures. After the explanation the test subjects were allowed to play around with the gesture-based interaction for a couple of minutes in order to learn how to control the cursor and how to perform the gestures. The test started, when the test subjects expressed that they were comfortable with the gestures and the control of the cursor .

7.1.6 Tasks

All test subjects were asked to perform each of the following tasks in presented order:

Assignment 1

- Select "Weekly recipes" from week 26
- How many eggs should be used in the dessert for 4 people
- What is the grilled dish from the same week (week 26)
- Click on the recipes
- Find the list of low-fat soups
- Choose Brussels sprouts soup (low-fat)
- How many potatoes should be used?
- Which soup is placed just beneath the Brussels sprouts soup in the list?
- How many leeks should be used in the soup for 4 people
- What is the first recipe in the list of low-fat soups

Assignment 2

- Select "Weekly recipes" from week 24
- How many leek should be used for "Chops in foil" for 4 people?
- What is the dessert from the same week (week 24)
- Click on the recipes
- Find the list of diabetic diet
- Find the list of main courses
- Select Mango Chicken
- How many apples should be used for Mango Chicken?
- What is the main course that is located just below the Mango Chicken in the list?
- How many grams of onions should be used for mango chicken?
- What is the last recipe in the list of diabetic main courses?

The tasks were developed in such a way that no input from the keyboard was needed.

7.1.7 Data Collection

The collection of data was done using two video cameras and a screen capture application. Two camera were used to record the test subjects movements of the upper body. One camera was used to record the test subjects from behind, and the other camera was placed in front of the test subjects. To record the movement of the cursor, the recognized gestures and the video feed from the Kinect a screen capture application was used. The interviews were recorded in the usability lab with the two cameras.

7.1.8 Data Analysis

In order to analyze the collected data the parameters for which to measure needed to be defined. The parameters are:

- **Gesture success rate**, meaning how often a gesture is recognized according to how many attempts the user makes. The idea for measuring this parameter is inspired by the articles [31], [32] and [8] that measure this in the test of their systems.
- **Task completion time**, how long it takes for a test subject to finish a task. This is inspired by the articles [33] and [34], where the completion time for tasks completed with a system called iCon and an ARC-Pad is measured.
- **Relation between link size and missed clicks,** meaning the relation between the size of the links and the number of missed mouse clicks. This is inspired by the article [8] that consider, whether it is easier for the users to click larger objects.

All data was collected before the analysis, and it consisted of two videos for each test subject, a total of 16 videos. The video material for each test subjects was carefully analyzed by the authors. The process for the data analysis consisted of going through the video and creating a data set for each test subject. The data set was created by studying the video and noting:

- The time and type of gesture, when a gesture was correctly recognized.
- The time that was required for a test subject to perform a correct gesture.
- The time and type of gesture, when system incorrectly recognized a movement as a gesture, referred to as a false hit.
- The time, type of gestures and number of tries when the test subject tried to perform a gesture that was not recognized by the system.

Addition to correctly recognized gestures it was also noted for the click gesture, whether the test subject misses the link and the font size of the link.

Two of the eight test subjects were randomly chosen, and the data from the two test subjects were analyzed by both authors. The two data sets were compared, and Cohen's Kappa[35] was calculated to 0.932. The six remaining videos were divided between the two authors and analyzed. After completing the analysis of a video the other author watched the same video and verified the data set. In case of a disagreement or misunderstanding the video was analyzed again by both authors in co-operation.

The interviews were transcribed in order to note both suggestions from test subjects and potential problems experienced during the test sessions.

7.2 Results

This section presents the findings from the usability test of Prototype 1.

7.2.1 Gesture success rate

The success rate for a given gesture is defined by the number of correctly recognized attempts to perform gestures divided by the total number of attempts. An attempt is defined as a movement that has been recognized by the system or as a movement that the authors have considered as being similar in some degree to the expected movement. If the test subject is solving a task requiring going back in the web history, the "Go Backward"-gesture is expected. This ensures that the test subjects movements are actually attempts to perform the given gesture. In order to illustrate this Figure 7.2 shows a correct attempt to perform the "Go Backward"-gesture. Contrary Figure 7.3 and Figure 7.4 show an incorrect attempt to perform the "Go Backward"-gesture.
For the test the success rates of the gestures have been calculated based on the collected data and are displayed in Figure 7.5.

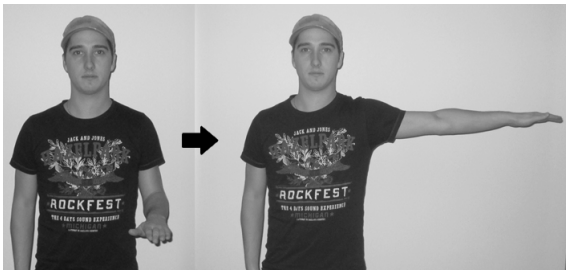

Figure 7.2: Correct attempt to perform "Go Backward"-gesture

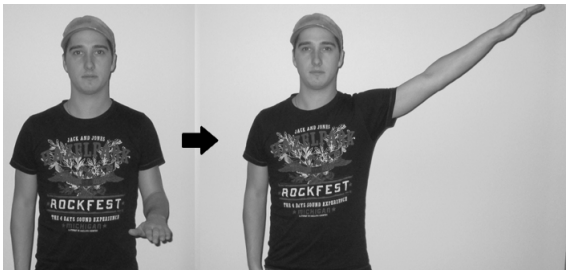

Figure 7.3: Incorrect attempt to perform "Go Backward"-gesture

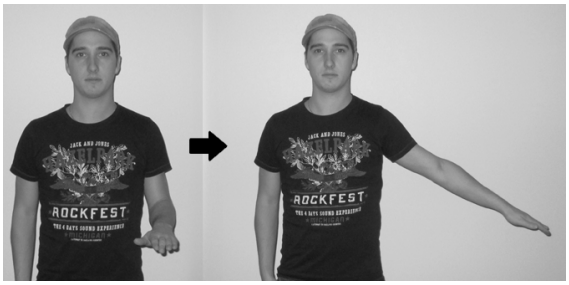

Figure 7.4: Incorrect attempt to perform "Go Backward"-gesture

As seen in Figure the "Scroll Up"-gesture has a success rate of 100% and is therefore considered as the easiest gesture to perform. The "Go Backward"-gesture has a success rate of 35% which means that it is the most difficult gesture to perform correctly for the test subjects.
The overall success rate for all gestures is 58%, meaning that 223 of the 383 attempts are recognized as the expected gestures.

In addition to the 223 correctly recognized attempts there were 16 false hit, which was when the system recognized a movement from a test subject that was not intended as a gesture. This is illustrated in Figure 7.6. An example of a false hit was when one of the test subjects scratched his nose, it was recognized as a "Scroll Up"-gesture.

7.2.2 Task completion time

In order to pinpoint the actions that are time-consuming the data for task completion time is divided into parts according to the Keystroke-level model[36]. The Keystroke-level model

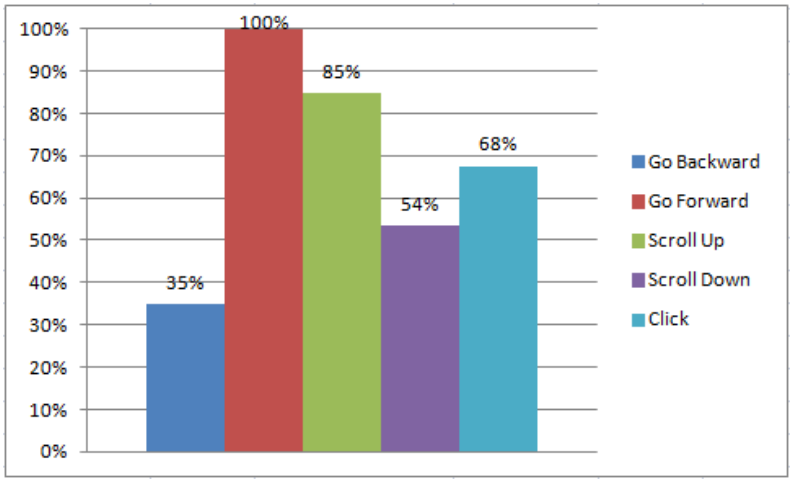

Figure 7.5: Success rates for the gestures

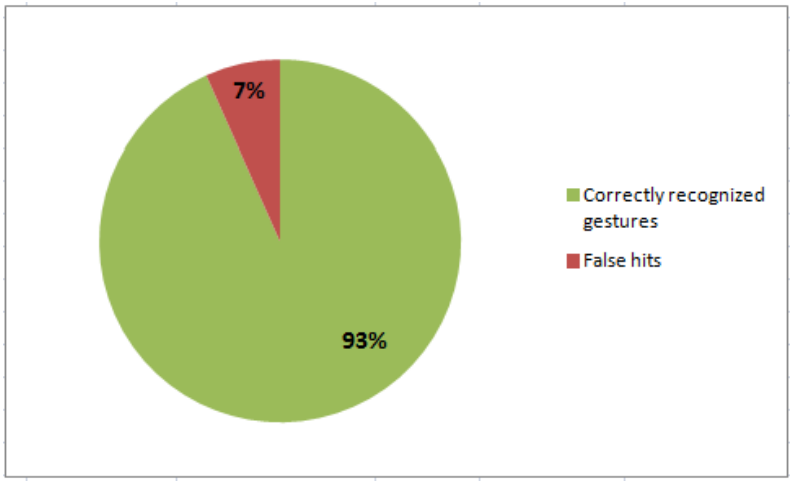

Figure 7.6: False hits

is used to make a comparison of the data from the mouse-based interaction and the gesture-based interaction. The elements that are used with the Keystroke-level model, are:

- Moving the cursor to a target.
- Clicking a target.
- Going forward in the web history.
- Going backward in the web history.

For the mouse-based interaction going forward and backward in the web history are considered as moving the cursor to the respectively "forward"- and "back"-button and clicking. For the gesture-based interaction going forward and backward in the web history are the use of the "Go Backward"- and the "Go Forward"-gesture. Scrolling up and down on the web pages have been omitted, since this is not supported by the Keystroke-level model.

As mentioned the required data for the Keystroke-level model has been gathered by analyzing the task completion times and divided them into elements from the Keystroke-level model. An average time for each element has been calculated and is displayed in Table 7.2 and Table 7.3 with the theoretical time from the Keystroke-level model. It must be noted that with the Keystroke-level model a click is considered as pressing the mouse button down, taking 0.100 second, and releasing the mouse button again, taking 0.100 second, resulting in a total of 0.200 second. The time that a user holds down the mouse button, is not considered. This can result in deviations in the Keystroke-level model time and the recorded time for the mouse-based interaction.
For Table 7.2 and Table 7.3 Keystroke-level is the theoretical time, when using a mouse, Mouse-based is the recorded time, when using a mouse, Gesture-based is the ideal time, meaning the recorded time, where errors have been omitted, and Gesture-based with errors is the total time with the errors that the test subjects made.

It can be seen in Table 7.2 and Table 7.3 that it is the movement of the cursor that is the time-consuming part. Moving cursor in gesture-based interaction is on average four times slower than with mouse-based interaction. However the values for the Keystroke-level model and the actual data for the mouse-based interaction are adjacent, when taking the possible deviation into consideration.

When looking at the data for the gestures, it is faster to use gesture-based interaction to go back and forward in the web history than with mouse-based interaction under the consumption that the test subjects are able to perform the gestures correct

Interaction	**Move Cursor**	**Click**	**Total Click Sequence**
Keystroke-level	01.100	0.200	01.300
Mouse-based	01.825	00.325	02.150
Gesture-based	05.620	00.850	06.470
Gesture-based with errors	-	-	08.770

Table 7.2: Time for Keystroke-level model and the average time for mouse- and gesture-based interaction

Interaction	**Going Back**	**Going Forward:**
Keystroke-level	01.300	01.300
Mouse-based	01.665	01.135
Gesture-based	00.800	00.867
Gesture-based with errors	11.150	00.867

Table 7.3: Time for Keystroke-level model and the average time for mouse- and gesture-based interaction (continued)

7.2.3 Relation between link size and missed clicks

This is the result for when a test subject tries to click a link, but misses the link when performing the "Click"-gesture. Throughout the test there were three sizes of links, and the results have been divided according to those sizes. The three sizes of links are Link 1, Link 2 and Link 3, where Link 1 has the smallest font size. Table 7.4 shows the results from the tests.

Link:	Font size:	Hit:	Miss:	Percentages:
Link 1	12	16	7	69.6%
Link 2	18	32	13	71.1%
Link 3	24	23	7	76.7%
Total	-	71	27	72%

Table 7.4: Table of hit and miss on link sizes

In this table it can be seen that the overall accuracy of mouse click is 72%, and Link 1 is the most difficult link to click. However the table also shows that there is only a difference of 7% between the smallest and the largest size of link, even though the font size is doubled.

7.2.4 Interviews

Important quotes from the interview are presented in this part. The interviews were held in danish and have therefore been translated. Words that have been added by the authors are shown in ( ) and ... means that words have been left out of the quotes, as they were empty words. The interviews focused on the control of the cursor with the gesture-based interaction, and how the test subject adopted and thought about the different gestures.

Cursor control

The overall opinion was that the cursor was easy to control. The test subjects said that the cursor followed the hand, and the test subjects did not express any significant problems controlling the cursor. However all test subjects pointed out the situation, where they missed a link, because they moved the cursor, while performing the "Click"-gesture, as a major source of irritation. Examples of these statements from the interviews are shown below:

> *You should hold it (the hand) still, and when you then move (to make the "Click"-gesture), the whole body moves as well.*

> *It was very easy, but it was difficult to be precise, when you should click at the same time.*

Gestures

The test subjects were asked questions about difficulty of which the gestures were to learn and remember. The overall opinion was that the gestures were easy to remember, and the mappings from the gestures to the functions were logical. The test subject were able to remember gestures by the function that they wanted to perform. For example the "Scroll Down"-gesture

was mentioned to be easy to remember, since the test subject wanted to move down on the web page. Examples of these statements from the test subjects are shown below:

> *It was very logical that you should do in this way. (Move the hand down) I think that it was very easy.*
>
> *It made good sense that it worked in this way. I think that it would be fast to learn.*
>
> *I think that they were easy to remember, because it made good sense that the backward gesture reminded of the back arrow.*

7.2.5 Discussion

From the results it is possible to see that the time consuming part of the interaction is the movement of the cursor. The movement of the cursor is on average three times slower than with an actual mouse, whereas the time used for going back and forward in the web history gesture-based interaction is less than the time used with mouse-based interaction. The subjects expressed their satisfaction with the overall control of the cursor, but complained about situations, where they were required to be precise. This indicates that the control of the cursor is satisfactory, but the system requires improvement to enable the test subjects to be precise in certain situations.

An interesting part of this is that even though all of the test subjects complained about the precision of the cursor while performing the "Click"-gesture, they were on average able to hit the links 72% of the times. As a comparison 58% of the performed gestures was recognized. Even though the data showed that the test subjects had more difficulties with performing gestures than clicking links, missing the links was experienced as the major source of irritation.

As seen from the results half the attempts are recognized as gestures, which may seem relatively low, however only seven percent of the recognized gestures is a false hit. It must also be considered that this is the first time the test subjects are using the system and have on average used the system for less than 20 minutes. In order to heighten the overall success rate the gestures them selves can be adjusted. The gestures can be too difficult or unnatural for the test subjects to perform. However during the interviews all of the test subjects expressed their satisfaction with the gestures and their mapping to the given functions. This indicates that it is not the gestures them selves that are difficult to remember or illogical, but rather the precision needed to perform a gesture. In order to clarify it is not the movement of moving the left arm to the left to perform "Go Backward"-gesture as shown in Figure 7.2. It is about recognizing less precise movements as those shown in Figure 7.3 and Figure 7.4. This indicates that the DTW-algorithm is implemented too strict for first-time users and not allowing enough variation of the gestures.

The user requirements described in Section 3 specifies that the interface should be: Easy to learn, Easy to remember and use Direct mapping between gestures and functions. The quotes concerning the gestures support the fact that the gestures are easy to learn and easy to remember.

7.3 Summary

In this chapter the test of Prototype 1 has been described. Success rates for each of the five gestures have been calculated along with an overall success rate. Problems of Prototype 1 have been identified from the results and the statements from the test subjects. Time-consuming parts of the interaction have also been identified. Based on this ideas for a new prototype are presented.

CHAPTER 8

DESIGN OF PROTOTYPE 2

This chapter describes the design of Prototype 2 based on the results from the test of Prototype 1. New ideas for controlling the cursor are described and discussed to decrease the navigation time and to increase cursor precision. Furthermore new ideas are found and described for recognizing the gestures in order to heighten the overall success rate.

8.1 Cursor

Based on the result of the test of Prototype 1 the most time-consuming part is the movement of the cursor. The test subjects also claimed in the interviews that a major source of irritation was missing a link, when clicking, because they could not keep the hand still, while performing the "Click"-gesture.

To change the technique of which the cursor can be controlled, the focus is turned to the articles and products from related work for cursor control in Section 2.3. Another technique for cursor control is used by TrackPoint[22] and the PlayStation 3 controller[21]. The cursor is moved in the direction, of which the TrackPoint or the left analogue stick on the PlayStation 3 controller is pushed, and the more the TrackPoint or the left analogue stick on the PlayStation 3 controller is pushed, the faster the cursor moves. The idea of variable cursor speed is also supported by the article "Does dynamic cursor control gain improve the performance of selection task in wearable computing?"[24] from related work that points out that it can improve the time required for selection tasks.
If the technique is compared to the technique from Prototype 1, there are two parameters that are different:

- Cursor speed.
- Movement of cursor.

For the cursor control technique used in Prototype 1 the cursor speed is fixed, and the movement of the cursor is directly mapped to the user's hand. For TrackPoint the cursor speed is variable, and the movement of the cursor is directional according to the initial position of the TrackPoint. By combining the parameters, three new techniques are developed, and this leads to a total of four techniques:

- **Technique 1** that is the technique used in Prototype 1.
- **Technique 2** that uses direct-mapping and variable cursor speed.
- **Technique 3** that functions very similar to the TrackPoint and uses directional control and variable cursor speed
- **Technique 4** that is based on the TrackPoint, but without variable cursor speed.

To present an overview of the four techniques, they have been plotted in Figure 8.1 along with the parameters.

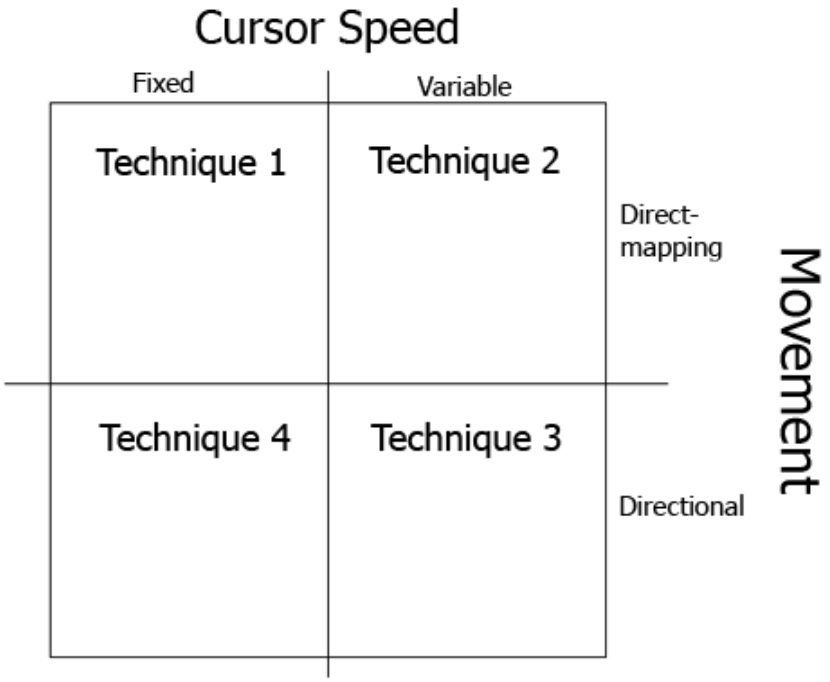

Figure 8.1: The parameters for the ideas for controlling the cursor

A further description of the three new techniques are given below:

8.1.1 Technique 2

This technique uses direct-mapping of the user's hand, but it has the ability of variable cursor speed. The idea is that when the user wants to make more precise movements, the user holds

his hand still, and the cursor speed is reduced. This can be done by comparing the coordinates of the users hand for the past period of time, and if the difference between them are within a predefined limit, the cursor speed is reduced. With the reduced cursor speed the user can make the same movements as before, but the cursor will not be a sensitive as before and is therefore more precise. In order to disable the reduced cursor speed the user makes a larger movement and thereby increases the difference between the previous and current coordinates for the hand. When this difference exceeds a limit, the cursor speed is returned to normal cursor speed again. The reason for activating Technique 2 based on, whether the users hold their hand still, is, because of the observation that when users want to click an element, they position the cursor over the element and tries to hold it still, while performing "Click"-gesture.

8.1.2 Technique 3

This technique is based on the TrackPoint, and the idea is that there is an area, where the user can place his hand within and hold the cursor still. When the user moves his hand outside of the area, the cursor moves according to the direction of which the user moved his hand. The cursor speed changes based on, how far the user moves his hand according to the initial point. An illustration of this is seen in Figure 8.2.

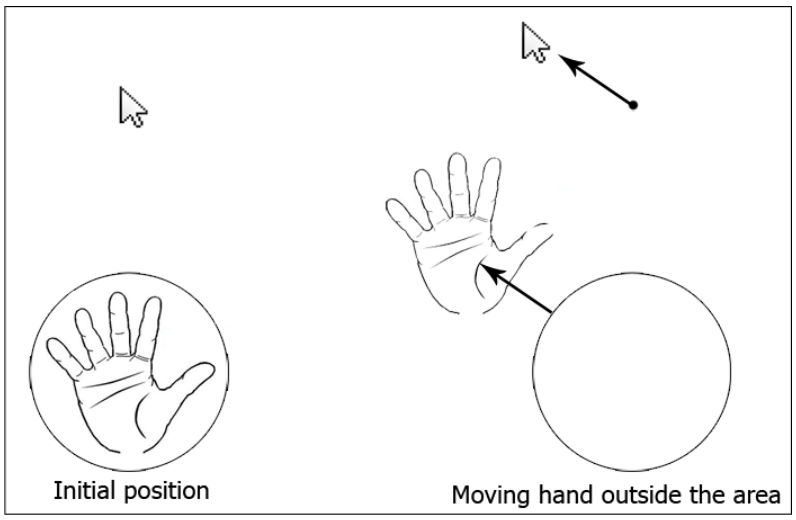

Figure 8.2: Illustration of Technique 3

The design idea for this technique is to compare the coordinates of the user's hand to the coordinates of the user's elbow, and when the distance between the X- and Y-coordinates for two points exceed a predefined limit, the cursor moves. The direction of the cursor movement can then be defined from a vector that is calculated based on the X- and Y-coordinates from the hand and elbow, and the vector will determine the speed, of which the cursor moves.

8.1.3 Technique 4

Technique 4 is basically identical to Technique 3, but without variable cursor speed. The difference is that the cursor speed is fixed and does therefore not depend on, how far the user

moves his hand from the initial point. Only the direction of the user's hand according the initial position is used to control the cursor movement.

8.2 Gestures

The overall success rate of gesture recognition from the test of Prototype 1 was 58%, which means that 3 out of 5 gestures are recognized. In order to increase the success rate new ideas are introduced. One idea is to loosen the DTW-algorithm and thereby allowing the DTW-algorithm to recognize more imprecise gestures. Based on observations made in the test of Prototype 1 of, how the test subjects variated the gestures, another idea is to have several variations of the same gestures. This also complies with the idea in the article "Gestures in The Wild:Studying Multi-Touch Gesture Sequences on Interactive Tabletop Exhibits"[7] from related work, where the system has been implemented to have several gestures for one function. The parameters for the two ideas are:

- DTW-algorithm settings.
- Number of gestures.

By combining the parameters in three new techniques for recognizing gestures are developed, which leads to a total of four techniques. The four techniques are the following:

- **Technique 1** that is the technique used in Prototype 1 with one gesture per function and strict setting of the DTW-algorithm.
- **Technique 2** that has a loose setting of the DTW-algorithm and one gesture per function.
- **Technique 3** that has the same strict setting for the DTW-algorithm as Technique 1, but has multiple gestures for each function.
- **Technique 4** that has a loose setting for DTW-algorithm and multiple gestures.

To present an overview of the four techniques and the two parameters they have been plotted in Figure 8.3.

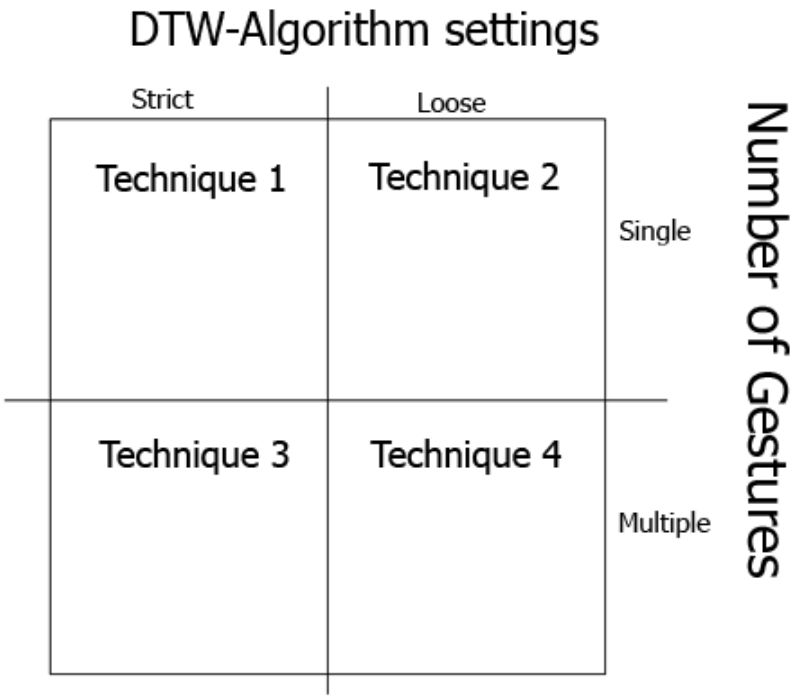

Figure 8.3: The parameters for the ideas of recognizing gestures

8.3 Summary

This concludes the design chapter for Prototype 2. In this chapter the products and articles from related work have been used as inspiration to developed three new techniques for cursor control. Three new techniques for recognizing gestures have been developed with inspiration from articles and observations made in the test of Prototype 1.

CHAPTER 9

IMPLEMENTATION OF PROTOTYPE 2

This chapter describes the implementation of the techniques designed in Chapter 8 and is divided into two parts: Cursor techniques and Gestures. Cursor techniques describes the implementation of the three new techniques used to control the cursor. Gestures describes the changes made to the settings of the DTW-algorithm and the use of multiple gestures for each function.

9.1 Cursor techniques

This section describes the implementation of the three additional cursor control techniques described in Section 8.1.

9.1.1 Technique 2

This technique works identically to Technique 1, described in Section 6.2, with the exception that Technique 2 has the functionality of detecting, if a user holds his hand nearly still for one second. When it is detected that the user has held his hand nearly still for one second, Technique 2 reduces the cursor speed. The activation time of one second has been defined based on tests conducted by the authors while implementing Technique 2. If the activation time was reduced to less than one second, it was difficult to use the cursor and not unintentionally activating the reduced cursor speed. In order not to increase the navigation time more than necessarily, one second has been found sufficient.

The implementation of Technique 2 is divided into two parts, where the first parts describes the functionality for detecting, and the second part describes the cursor control.

Detection in Technique 2

The detection part of Technique 2 is shown in Source Code 9.1. In order to detect, if the user has held his hand nearly still for one second, it is possible to use of the fact, the Kinect out-

puts data 30 times per second. This means that if it is possible to create a list with 30 subsequent points for the hand that fulfills two requirements, it can be registered, as the user has held his hand still for one second. The list, containing the 30 points, is referred to as `_rightHandCoordinates` in Source Code 9.1.

The first requirement, shown in Line No. 11, is that the difference between a point in the list and the previous point in the list must not exceed a limit of 5 pixels based on the coordinate system used by OpenNI SDK, and 5 was found to be a suitable value for this purpose. Otherwise the hand is not considered as been held still. It would not be possible to require the difference to be zero, because it is nearly impossible for users to hold their hand completely still.

The second requirement, shown in Line No. 23, is that the difference between the first point in the list and the last point in the list should not exceed the limit of 5, otherwise a slowly movement of the hand in a direction could be detected as holding the hand still as shown in Figure 9.1.

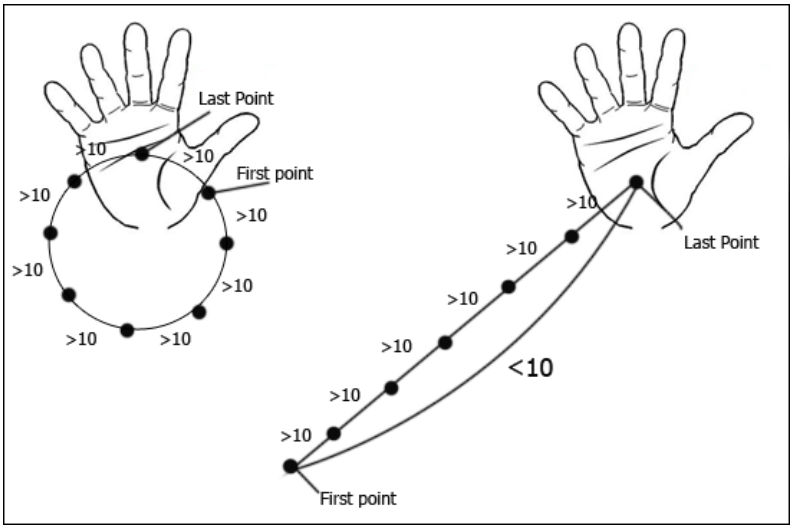

Figure 9.1: Illustration of the second requirement

If both requirements are fulfilled, this is detected, as the user has held his hand nearly still for one second. When detecting that the user has held his hand still, the method `ChangeCoordinateSystemPlacement` shown in Line 27 in Source Code 9.1 is called. `ChangeCoordinateSystemPlacement` method moves the center of the coordinate system used to positioning the cursor on the screen based on input from OpenNI SDK from (960, 540), which is the center of the screen, to the current position of the cursor.

```
1  //Technique 2
2  if (_cursorTechnique == 2)
3  {
4          if (_firstTime)
5          {
```

```
6             //Ensure that the required data is available in the list, ↵
                  _rightX and _rightY are the current X- and ↵
                  Y-coordinates of the hand
7             _rightHandCoordinates.Add(new ↵
                  System.Windows.Point(_rightHand.X, _rightHand.Y));
8             _firstTime = false;
9         }
10          //Requirement 1 - Check if the current point of the hand is ↵
                within the limits of the previous point.
11        if (Math.Abs(_rightHandCoordinates[_rightHandCoordinates.Count ↵
              - 1].X - _rightHand.X) < 5 && ↵
              Math.Abs(_rightHandCoordinates[_rightHandCoordinates.Count ↵
              - 1].Y - _rightHand.Y) < 5 && !_firstTime)
12            _rightHandCoordinates.Add(new ↵
                  System.Windows.Point(_rightHand.X, _rightHand.Y));
13        else
14        {
15            //The current point of the hand is not within the limits of ↵
                  the previous - Reset
16            _rightHandCoordinates.Clear();
17            _firstTime = true;
18        }
19
20        if (_rightHandCoordinates.Count == 30)
21        {
22            //Requirement 2 - Check if the difference between the first ↵
                  point in the list and the current point of the hand is ↵
                  within the limit.
23            if (Math.Abs(_rightHandCoordinates[0].X - _rightHand.X) < 5 ↵
                  && Math.Abs(_rightHandCoordinates[0].Y - _rightHand.Y) ↵
                  < 5)
24            {
25                //The hand is kept within the limits for one second
26                //Activate the variable cursor speed.
27                ChangeCoordinateSystemPlacement(_rightHand.X, ↵
                      _rightHand.Y);
28                return;
29            }
30            _rightHandCoordinates.RemoveAt(0);
31        }
32    }
```

Source Code 9.1: Implementation of Technique 2 part one

Cursor Control in Technique 2

Source Code 9.2 shows the implementation of the cursor control used in Technique 2. The cursor is positioned on the screen using the same functionality as in Technique 1 that calls the `SetCursorPos` method, which is able to position a cursor based on a X- and Y-coordinate. The `SetCursorPos` call in Technique 2 is shown in Line No. 4 where the X- and Y-coordinate of the hand is divided by two and added to the `_coordinateSystemCentrum`. This means that when the hand is moved e.g. 20 pixels, the cursor is only moved 10 pixels on the screen.

The cursor speed remains reduced, until the user makes a larger movement, and the larger movement is detected by comparing the current coordinates of the hand with the previous coordinates of the hand, which is shown in Line No. 2. If this exceed the predefined limit of 10, the cursor speed is returned to normal, and the center of coordinate system is set to (960, 540), which is the center of the screen. 10 was found to be sufficient value through tests, and a limit below 10 was found to be difficult to use without accidentally returning to normal cursor speed, and a limit above 10 required the user to make a movement that was considered to be too large.

```
1  //Check, if the user has made a larger movement
2  if (!(Math.Abs(_comparingPoint.X - x) > 10 || ↵
       Math.Abs(_comparingPoint.Y - y) > 10))
3   {
4      SetCursorPos((int)((x/2) + _coordinateSystemCentrum.X), ↵
           (int)((-y/2) + _coordinateSystemCentrum.Y));
5      _comparingPoint = new System.Windows.Point(x, y));
6   }
7  else
8  {
9      //Return to normal cursor speed - Reset.
10     _coordinateSystemCentrum = new System.Windows.Point(960, 540);
11 }
```

Source Code 9.2: Implementation of Technique 2 part two

9.1.2 Technique 3

This technique is based on the TrackPoint, where the cursor moves in the direction, of which the hand is moved. The cursor speed increases the further the hand moves. The implementation is shown in Source Code 9.3.

Because Technique 3 and 4 is very similar and share code, they have been implemented together. Technique 3 and 4 use a point called `_cursorPoint` to position the cursor on the screen, and when the user moves his hand, the X- and Y-coordinates of this point change. `_cursorPoint` is first set to (960, 540) in Line No. 2, to place it at the center of the screen.

The changes of the X- and Y-coordinates are based on, how far the user moves his hand from his elbow. To calculate the direction of which the hand is moved according to the elbow, a

vector is computed, which is done in Line 11-13. The vector is calculated by subtracting the coordinates for the hand from the coordinates for the elbow. The cursor speed therefore depends on this vector.

Line No. 5 checks whether it is Technique 3 or Technique 4 that is used at the moment, to ensure that the following code is not run, when using Technique 1 or 2. The distance that the hand can be moved away from the elbow, without effecting the cursor, is defined to be 10, and this is shown in Line No. 8. This distance makes it possible to move the hand around in a circle that has the elbow as center and a radius of 10. This is also referred to as the safe area. Tests showed that a smaller distance than 10 made it difficult to keep the hand inside the safe area, and when returning to the safe area, it was easy to accidentally move the hand outside the opposite border of the safe area and thereby moving the cursor in a wrong direction. To reduce the distance that the user is required to move his hand to leave the safe area, the smallest possible value has been chosen.

To check, whether the hand is outside the safe area, the length of the vector between the elbow and the hand is calculated, and this is done in Line No. 16. The check of, whether the hand is moved outside the safe area, is performed in Line No. 20.

The variable `factor` is used to reduce the vector and thereby reduce the cursor speed. It was found necessary to reduce the cursor speed to be able to hit targets. `factor` is set, when the length of vector between the hand and elbow reaches certain limits, as shown in Line No. 29-39. The lower the factor is, the faster the cursor moves.

The changes of the X- and Y-coordinates are then based on the values of the reduced vector, shown in Line No. 42-43.

Line No. 55-56 sets `_cursorPoint` to its own value added with the values of the reduced vector.

Finally the position of the cursor is set on the screen in Line No. 61 using the `SetCursorPos` method and the X- and Y-coordinates of `_cursorPoint`.

```
1  //The initial cursor point.
2  Point _cursorPoint = new System.Window.Point(960, 540);
3
4  //Technique 3 or Technique 4
5  if (_cursorTechnique == 3 || _cursorTechnique == 4)
6  {
7          //Gives a circle with a diameter of 10 pixels where the hand ↵
               movements do not effect the cursor (safe area)
8           int distanceFromElbowLimit = 10;
9
10          //Vector used to determine the direction
11         System.Windows.Point vector = new System.Windows.Point();
12         vector.X = rightElbowPoint.X - rightHandPoint.X;
13         vector.Y = rightElbowPoint.Y - rightHandPoint.Y;
14
15         //Calculates the length of the vector from the hand to the elbow
16         double distanceFromElbow = Math.Sqrt(Math.Pow(vector.X, 2) + ↵
               Math.Pow(vector.Y, 2));
```

```
17
18
19          //Checks whether the hand is moved out of the safe area.
20          if (distanceFromElbow > distanceFromElbowLimit)
21          {
22                  double changesOfXCoordinates= 0;
23                  double changesOfYCoordinates= 0;
24
25                  //Technique 3
26                  if (_cursorTechnique == 3)
27                  {
28                          //The cursor speed is set to its maximum
29                          int factor = 3;
30
31                          //Cursor speed is set to low
32                          if (distanceFromElbow < 15)
33                                  factor = 15;
34                          //Cursor speed is set to medium
35                          else if (distanceFromElbow < 25)
36                                  factor = 10;
37                          //Cursor speed is set to high
38                          else if (distanceFromElbow < 30)
39                                  factor = 5;
40
41                          //The interval of which the cursor should move ↵
                                using variable cursor speed (factor)
42                          changesOfXCoordinates = -(vector.X) / factor;
43                          changeOfYCoordinates= -(vector.Y) / factor;
44                  }
45
46                  //Technique 4
47                  if (_cursorTechnique == 4)
48                  {
49                          //The interval of which the cursor should move ↵
                                using a fixed cursor speed (5)
50                          changesOfXCoordinates = (-(vector.X) / ↵
                                distanceFromElbow) * 5;
51                          changeOfYCoordinates = (-(vector.Y) / ↵
                                distanceFromElbow) * 5;
52                  }
53
54          //Cursor point is set to its own value + the interval the ↵
                cursor is moved on both axis.
55          _cursorPoint.X += changesOfXCoordinates;
56          _cursorPoint.Y += changeOfYCoordinates;
57
58          //Ensure that the cursor is within the screen
59          ...
60  }
```

```
61 SetCursorPos((int)_cursorPoint.X, (int)_cursorPoint.Y);
```

Source Code 9.3: Implementation of Technique 3 and 4

9.1.3 Technique 4

Technique 4 is similar to Technique 3, but with a fixed cursor speed. The implementation is shown above in Source Code 9.3. Note that the implementation is the same as described above but with minor changes.

The difference in the implementation of Technique 3 and 4 is that the unit vector is used instead of the vector to calculate the cursor speed.
The unit vector is calculated by dividing the vector with the length of the vector. The unit vector is then multiplied with 5 to increase the cursor speed. The value 5 was found during tests done by the authors to be the maximum value that the cursor speed can be multiplied with, while still being able to position the cursor on elements.
The changes of the X- and Y-coordinates for Technique 4 is calculated in Line No. 47-52.

9.2 Gestures

This section describes the implementation of multiple gesture for each function. Furthermore the different settings of the DTW-algorithm are described.

9.2.1 Multiple gestures

As mentioned the idea is to have more gestures for one function, because users tend to variate the gestures. The variations and amount of variations of the gestures are based on observations made during the test of Prototype 1.

The amount of variations of the gestures are listed below:

- The "Go Backward"-gesture is defined by 5 variations of the gesture shown in Figure 9.2.
- The "Go Forward"-gesture is defined by the 3 variations of the gesture shown in Figure 9.2.
- The "Scroll Up"-gesture is defined by the 3 variations of the gesture shown in Figure 9.3.
- The "Scroll Down"-gesture is defined by the 3 variations of the gesture shown in Figure 9.3.
- The "Click"-gesture is defined by the 5 variations of the gesture shown in Figure 9.4.

To enable recognition of a greater amount of gestures, the variation of the gestures have been performed and recorded.

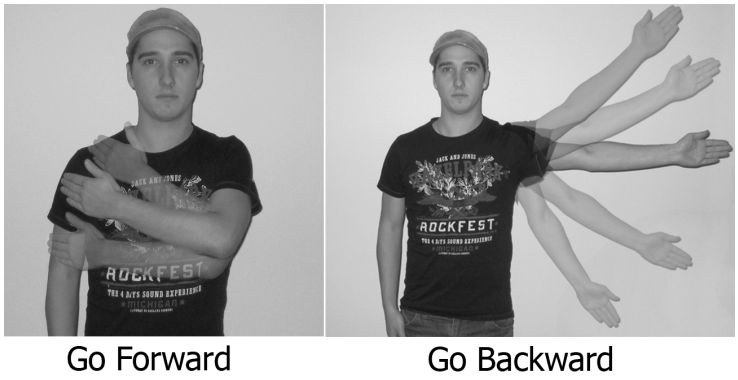

Figure 9.2: The "Go Forward"- and "Go Backward"-gestures

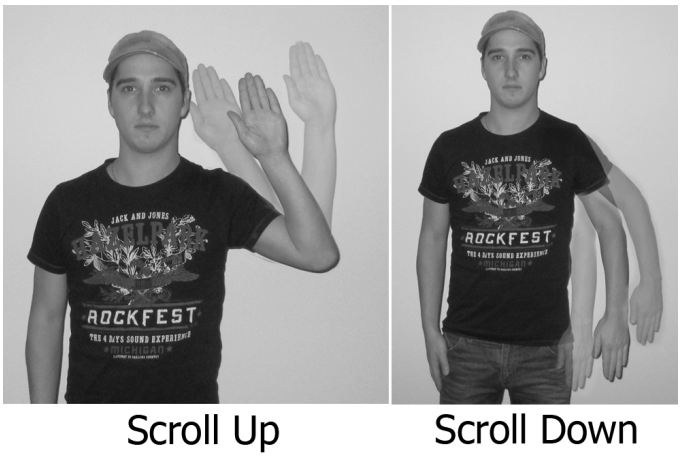

Figure 9.3: The "Scroll Up"- and "Scroll Down"-gestures

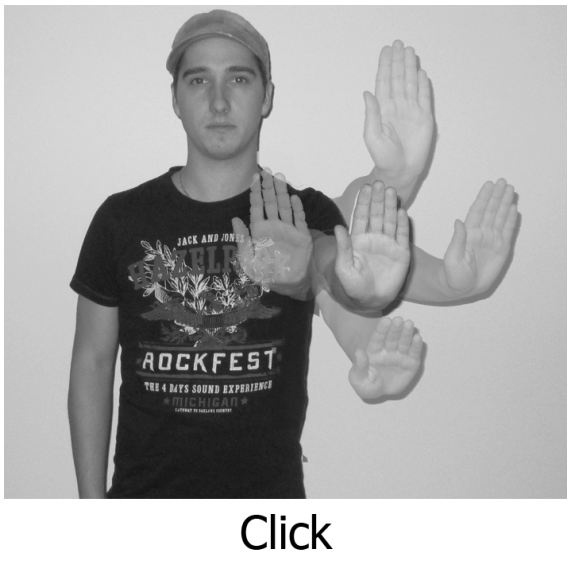

Figure 9.4: The "Click"-gestures

9.2.2 DTW-algorithm settings

A way to achieve a higher success rate can be by allowing the algorithm to accept gesture that are performed less precise. There are two variables, `firstThreshold` and `threshold`, that is used to define the required precision of the gestures performed. `firstThreshold` defines the maximum distance between the last point in a gesture sequence and the last point of a sequence of a stored gesture. `threshold` is used to define the maximum distance between all the points in a gesture sequence and all the points in a sequence of a stored gesture. Both variables are described in detail in Section 6.4.2.

Two different settings are constructed for the DTW-algorithm, loose and strict. The strict setting is the setting used in Prototype 1. The loose setting has been found through a test of a range of settings conducted by the authors. The chosen setting was found to be close to the point, where the DTW-algorithm started recognizing unexpected gestures. Using the loose setting, the distance between the points in a performed gesture sequence and a recorded gesture sequence is allowed to be nearly twice as much as the strict setting.

9.3 Summary

In this chapter the implementation of the three new cursor techniques has been described. Furthermore the variations of the gestures are shown in figures, and the two settings of the DTW-algorithm has been described.

CHAPTER 10

TEST OF PROTOTYPE 2

This chapter describes the test of Prototype 2, and it is divided into two parts: Method and esults. Method describes the setup for the test, the participants etc, and results presents the results found in this test. A discussion of the found results is then given.

10.1 Method

There were two purposes for this test:

1. To test the four techniques for recognizing gestures, which differ on the number of gestures per function and the settings of the DTW-algorithm.

2. To test four techniques for controlling the cursor, which differ on the possibilty for variable cursor speed and how the hand is mapped to movement of the cursor.

10.1.1 System

The system consisted of the same elements as the test of Prototype 1 which were a PC, a 42" screen, a mouse, a keyboard and a Kinect. The setup was identical to the setup used in the gesture-based interaction part in the test of Prototype 1, where the screen was installed on a shelf with a table placed in front of it, the Kinect was mounted on top of the screen, and both the keyboard and mouse were placed on the test monitors table. The system was tested with four techniques for recognizing gestures and four techniques for controlling the cursor, and the setup was the same for both recognizing gestures and controlling the cursor.

10.1.2 Participants

A total of eight test subjects participated. We tried to choose the subjects based on sex, age and experience with IT. The demographical data about the participants is listed in Table 10.1.

Group:	Test person:	Sex:	Age:	Education:	Test order:
1	A	Male	24	Technical education	1-2-4-3
1	B	Male	22	Non-technical education	2-3-1-4
1	C	Female	21	Technical education	4-1-3-2
1	D	Female	23	Non-technical education	3-4-1-2
2	E	Male	24	Technical education	1-2-4-3
2	F	Male	24	Non-technical education	2-3-1-4
2	G	Female	23	Technical education	4-1-3-2
2	H	Female	25	Non-technical education	3-4-1-2

Table 10.1: Table of participants

The Test order refers to the order of which the test subjects were given both the four techniques of recognizing gestures and the four techniques for controlling the cursor. In order to select the order of which the test subjects should be presented with the different techniques, Latin Square[37] was applied. Latin Square was used in order to balance the learning effect from the four techniques.

The age of the test subjects varies from 21 to 25 and there is therefore a small variation of age. We are aware that this can have an influence on the results, but because of the limited number of test subjects these test subjects have been used. The test subjects were given a bag of candy after completion as appreciation of their help, but did not receive any further payment.

10.1.3 Setting

The setting for this test is identical to the setting for the test of Prototype 1 in Section 7.1.3, meaning that the test was conducted at the usability lab at the university. Two cameras were used to record the test subject from behind and from the front.

10.1.4 Tasks

All test subjects were asked to perform each of the following two tests:

Gesture recognizing-test

A task for one of the four techniques for recognizing gestures consisted of the test monitor asking the test subject to perform a gesture. When the test monitor said: "Scroll Up", the test subject should try to perform the "Scroll Up"-gesture, and the test subject should try to perform the "Scroll Up "-gesture, until it was recognized by the system. If the gesture was recognized, the test subject should stop and wait for the next gesture to perform. Each gesture should be recognized by five times to end the test.

The order of which the test subject was asked to perform the gestures, was kept the same. The reason for this was that according to observations made in the test of Prototype 1 people had less difficulty to perform gestures that they had recently used. If a test subject had performed a "Scroll Up "-gesture that was recognized, it was easier for the test subject to remember the

proper movement. In order to make this factor as equal as possible for the five gestures, the order was kept the same and thereby also the number of other gestures between the same gestures.

Controlling cursor-test

A task for one of the four techniques of controlling the cursor the test subject consisted of five steps, which the test subject had to complete ten times for each technique.

1. Move the cursor within Circle 1 on the screen.
2. Move the cursor to the new, smaller Circle 2 that appers after completing step 1
3. Hold the cursor within Circle 2 for one second.
4. Click the link that appears after completing step 3.
5. Repeat with new location of Circle 1.

For each repetition the locations of Circle 1 and 2 and the link changed, however the distances between the elements were kept the same. The locations did not change between the techniques, meaning that the locations of Circle 1, Circle 2 and the link were the same for the second repetition in the test of Technique 1 as for the second repetition in the test of Technique 2.

The reason for step 3 was to ensure that the test subject had to be precise with the cursor and not accidentally hit Circle 2, when moving the cursor around.

10.1.5 Procedure

The test subjects were introduced to the test and instructed to only perform an action, when the test monitor asked them to. A test monitor was presented during the tests and was in charge of reading the tasks to the test subject. The test monitor was the same person throughout all the tests in order to ensure that the test subjects were given the same training.

10.1.5.1 Recognizing gestures-test

All of the test subjects started with the test of the four techniques of recognizing gestures. The reason for starting with the recognizing gestures-test was that the "Click"-gesture is required to complete the controlling cursor-test. If the tests subjects were introduced to the "Click"-gestures in the controlling cursor-test, they would have had more practice with this gesture compared to the others.

Training

Before the recognizing gestures-test the test subject was given a training session. For the training session the test subject was taught the five gestures by the test monitor. The test monitor demonstrated a gesture, and when the test subject was able of performing the gesture correctly

one time, a new gesture was shown to the test subject. The gestures that the test subjects were taught, were the gestures that could be recognized by all of the four techniques of recognizing gestures. The test subjects were also instructed that if their gestures were not recognized, they should try to be more precise with their movements and recall the gestures they had been shown.
It was considered to time the training session to try to have the same training for each test subject. However by first continuing the training, when the test subject made a gesture that was recognized, it prevented that the test subject failed to make a correct gesture, before the time ran out, and thereby not learning the gesture.

10.1.5.2 Controlling cursor-test

For the controlling cursor-test the test subject was presented with a training session of approximately 30 seconds before the test of the technique. The training consisted of an explanation from the test monitor of how the technique worked, and then the test subject was given time to play around with it. After finishing the test with one technique the test subject was given a new training session with another technique, and it was then repeated, until the test subject had tried all of the four techniques. After completing the controlling cursor-test the test subjects were asked questions about the four techniques for controlling the cursor The questions mainly concerned how the test subject felt, when using each of the four techniques, what technique they preferred, and what technique they disliked the most.

10.1.6 Data Collection

The data collection of this test was identical to the data collection in the test of Prototype 1 in Section 7.1.7 with two cameras recording the test subject from two angles. As an addition both of the authors noted during the test for recognizing gestures, when the test subject performed a gesture, whether it was recognized or not, and when a false hit was recognized. For the test of controlling the cursor a log system had been implemented that recorded the time, when the test subject left Circle 1, entered Circle 2 and the number of missed clicks.

10.1.7 Data Analysis

For the test of controlling cursor the parameters for which to measure were:

- **Time to move the cursor.**
- **Missed clicks.**

For the test of recognizing gestures the parameters for which to measure were:

- **Success rate for the gestures.**
- **False hits rate.**

After the tests the notes for the four techniques of recognizing gestures were compared, and whenever a disagreement was found, the video material was analyzed by the authors in collaboration.

The test subjects's number of false hits and attempts to make a gesture that was not recognized, was counted and used to calculate an individual success rate and false hits rate for each technique. A total success rate and false hit rate for each technique have been computed by calculating an average of the test subjects's success rates and false hits rates.

For each test subject the average time used with each technique to move the cursor from Circle 1 to Circle 2 was calculated, and the number of missed clicks with each technique was counted. An overall average time for each technique was computed based on the average times for the test subjects. An average of missed clicks for each technique was computed based on the number of missed clicks for the test subjects. The interviews were transcribed in order to note the test subjects opinions on the four techniques of controlling the cursor.

10.2 Results

This section presents the findings from the test of Prototype 2.

10.2.1 Time to move the cursor and Missed clicks

The overall average time used to move the cursor from Circle 1 to Circle 2 and the percentages of missed clicks according to all clicks for each technique are inserted in Figure 10.1.

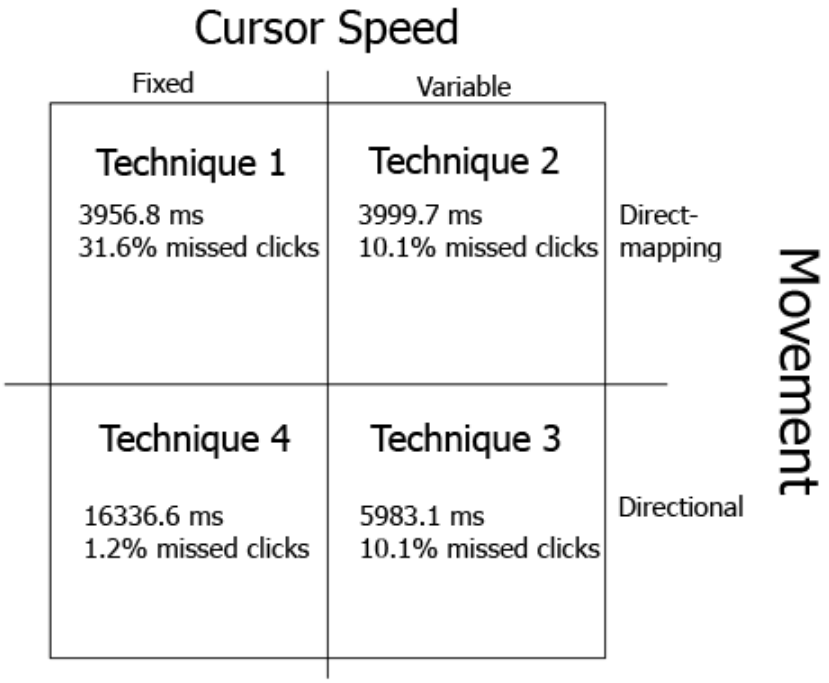

Figure 10.1: Results of the four techniques for controlling the cursor

When considering the lowest time, Technique 1 is the optimal choice. If the smallest number of missed clicks is preferred, Technique 4 is the optimal choice, and if it must be a combina-

tion, then Technique 2 is the best solution.

Based on Figure 10.1 it is also possible to see a pattern that when changing the movement of the cursor from direct-mapping to directional the time for moving the cursor increases. Another pattern is, when going from direct-mapping to directional the number of missed clicks is maintained or at best decreased.

Our results do not show any patterns when considering fixed and variable cursor speed.

10.2.2 Success rate for the gestures and False hits rate

For each of the four techniques of recognizing gestures the total success rate, called SR, and false hits rate, called FHR, have been calculated and are shown in Figure 10.2. The relation between success rate and false hits rate for each technique has been calculated and is also inserted into Figure 10.2

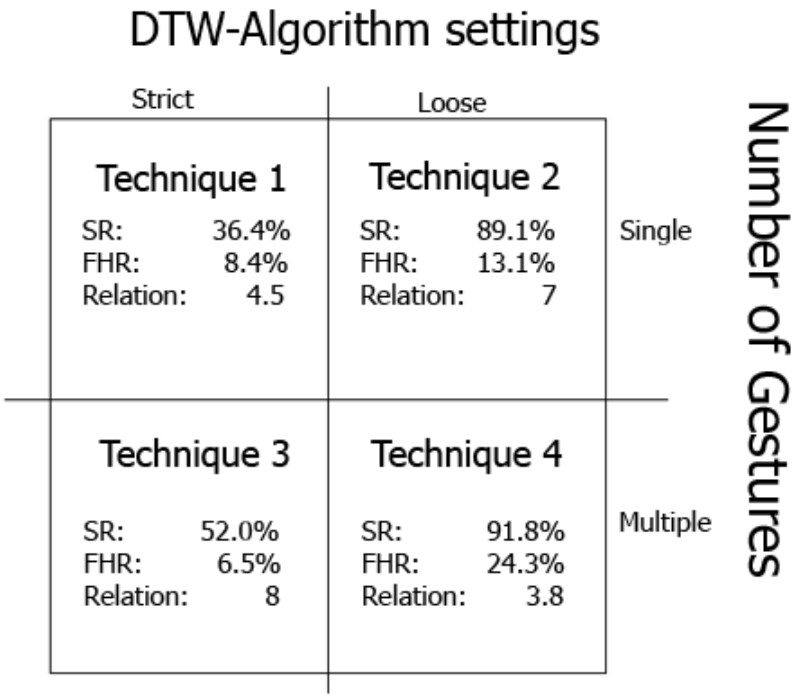

Figure 10.2: Results for the four techniques of recognizing gestures

It must be noted that the success rate of Technique 1 can not be compared with the gesture success rate from the test of Prototype 1, even though it is the same technique. The test subjects received a longer training in how to perform the gestures in the test of Prototype 1. Secondly the test of Prototype 1 was a realistic use of the system, whereas the test of Prototype 2 was a performance test designed for comparison of the different techniques.

Based on Figure 10.2 it is possible to see that by increasing the number of gestures and by using a loose DTW-algorithm the success rates increase. However the false hit rate only increases

when moving from a strict DTW-algorithm to a loose DTW-algorithm. When moving from single gestures to multiple gestures the false hit rate decreases with a strict DTW-algorithm.

10.2.3 Interviews

The interviews were held in danish and have therefore been translated. Words that have been added by the authors are shown in ( ), and ... means that words have been left out of the quotes, as they were empty words.

The focus of the interviews were on the different techniques for controlling the cursor, what the test subject preferred and what they disliked. The choice of quotes is based on the test subjects general opinions of the different cursor techniques. Quotes from the interviews concerning the different techniques are shown below:

Technique 1

> *I think it was hard to keep it still (the cursor).*
>
> *Just so it felt natural to use it, but it was like when you held the hand still, the mouse still moved a little.*

Technique 2

> *I think it was the easiest. It is because you had more control over the adjustment.*
>
> *That was certainly the easiest. The thing about it was that it became slower when the hand was kept still.*

Technique 3

> *I think it was easy to control*
>
> *You got a better feeling for it since you could adjust it a little bit.*

Technique 4

> *It was hard because you did not have control over when it (the cursor) should stop moving.*
>
> *Way too slow.*

10.2.4 Discussion

When looking on the results from the test of controlling the cursor, Technique 1 is the fastest, however the modifications that led to Technique 2, reduced the number of missed clicks, while being only 34 ms slower on average. The interviews indicates that the test subjects prefer Technique 2 due to the fact that it is easy to control and easy to adjust the cursor precisely. The optimal solution for controlling the cursor is found to be Technique 2 due to the fact that it

is nearly as fast as Technique 1, it produces 20% less missed clicks and it is the most prefered technique by the test subjects.

For the test of recognizing gestures the most suited technique depends on the requirements. If the highest success rate is desired, Technique 4 is the best choice, however this technique also has the highest false hit rate. If the lowest false hit rate is required, Technique 1 is the optimal technique, but this also means that on average only a third of the gestures are recognized.

For a combination of success rate and false hit rate the solution is either Technique 2 or 3. Technique 2 has been chosen as the optimal solution for this system because of the gain of a 37% higher success rate, when accepting an increase of 7% in false hits rate.

There is a trade-off between the success rate and the false hit rate, however Technique 3 displays an interesting result. With the same strict DTW-algorithm and multiple gestures Technique 3 should in theory have a higher chance for a false hit than with Technique 1, but the results shows a decrease in false hit rates when going from Technique 1 to Technique 3. No sufficient reason for the disagreement of the results has been found.

10.3 Summary

This chapter has described the test of Prototype 2, which focused on comparing four techniques for controlling the cursor and four techniques for recognizing gestures. Based on the results an optimal solution for controlling the cursor and recognizing the gestures has been found.

CHAPTER 11

DISCUSSION

This chapter contains the discussion of this project, and through this discussion observations from the two tests are considered, results are compared with related work, and the interface is compared with one system that offers a similar interface for interacting with a PC. Finally the development of the natural user interface is discussed.

11.1 Observations

After the test of Prototype 1 observations of not-expected results were made, and these observations were used in the design of Prototype 2. The observations that were not used to design Prototype 2, are described below. The three observations that were made during the test of Prototype 1 and 2 are categorized as:

- Consistency of gestures.
- Ignoring the possibility of moving.
- Interpretation of own body movement.

Consistency of gestures

It was observed during the test of Prototype 1 and 2 that some test subjects were able to guess gestures based on what they were told previously and had tried. The test subjects that were able to guess the gesture, were all classified as test subjects with a technical education. When some of the test subjects were showed and told how to perform a "Scroll Up"-gesture they performed the "Scroll Up"-gesture and immediately performed the reflected gesture "Scroll Down" without being introduced to this. The rest of the test subjects did not try any gestures and waited, until the test monitor told them of a new gesture. This indicates that the gestures are intuitive and that the choice of having consistency between the gestures increased the

learnability. This is also supported by the interviews from the test of Prototype 1 in Section 7.2, where statements from the test subjects refer to the gestures as intuitive and easy to remember.

Ignoring the possibility of moving

Another interesting observation that was made during the two tests, was that the participants tended to stay in the same spot, when using the interface, even though they were specifically told that they could move while using the interface. It was observed that the test subject would move a few meters, when they were notified of the opportunity. However during the tests the participants would rather stay in uncomfortable positions to hold the cursor at a certain position, while clicking than taking a few steps to the side. Only one of the test subjects commented on the possibility of moving around, while using the interface, and stated in the interview:

When you are focused on the screen, it seems strange that you should move around.

This indicates that it is not natural for people to move around, when their focus is on the screen. The observation therefore suggests, when developing systems with natural user interfaces it should either not be necessary to move around, or the users should clearly be notified of this possibility during their use of the interface.

Interpretation of own body movement

It was also observed during the two tests that the participants had difficulties to translate the gestures that they were shown and told, into their own body movements. The participants were showed a "Go Backward"-gesture by raising the arm to be in a horizontal position, but when asked to perform the gesture, the participants made several variations of the gestures, and the figures in Section 9.2 shows the most common variations. The movements used to perform the gestures were intuitive according to the participants, but the issue was rather, how precise the movement should be performed to be recognized. This suggests that it should be considered, how the gestures should be taught to the users.

11.2 Comparison with related work

This section compares results from the two tests with results found in related work.

The results from the test of Prototype 2 in Section 10.2 showed that variable cursor speed decreased the time required to navigate from point A to point B between Technique 4 and Technique 3. This complies with the result from the article "Does dynamic cursor control gain improve the performance of selection task in wearable computing?"[24] that states the use of dynamic cursor control, or in this case the use of variable cursor speed, would decrease the navigation time.

However this was not the case between Technique 1 and Technique 2, because Technique 2 had an increase of 40 ms in navigation time compared to Technique 1. In order to find a reason for the disagreement the implementation of Technique 2 has been analyzed. Technique

2 requires the user to hold the cursor nearly still for one second to decrease the cursor speed, and the one second used for activation is included in the navigation time. If the system was able to automatically decrease the cursor speed, when the cursor approaches a link or button the one second delay in the navigation time could potentially be decreased or completely removed. This can imply that the navigation time for Technique 2 can be reduced to a lower navigation time than Technique 1, as there is only a difference of 40 ms and thereby comply with the results from the article.

11.3 Comparison with similar systems

At the final stage of this project two systems that provide a natural user interface for PC's, were discovered, KinEmote[38] and Win&I[39]. Both systems use the Kinect to track users.

11.3.1 KinEmote

There are two versions of KinEmote, KinEmote v0.4 Beta and KinEmote PCD v0.1 Beta, and both were released April 29th 2011. KinEmote started as an open source project, but changed to Free License project. This means that there is no access to the source code, and we can therefore not go into details of the technical aspects.

The two versions are described in the following and a comparison of each version with our system is conducted.

KinEmote v0.4 Beta

KinEmote v0.4 Beta is primarily designed for interacting with Boxee and XMBC, which are two media center systems used to watch movies. KinEmote v0.4 Beta supports six gestures for interaction with the two media centers: up, down, left, right, push forward and push backward, and all gestures are performed with the same hand. Moving the hand to the left activates the left-gesture and similar when moving the hand up, down and right. The push forward- and push backward-gestures are made by making a push movement respectively forward or backward. The up, down, left and right gestures are by default mapped to the arrow keys on the keyboard. The push forward and backward-gestures are used to select and deselect items.

KinEmote v0.4 Beta also supports cursor control and the functionality to perform a left click. When using the cursor control, the movement of the hand is mapped to movement of the cursor instead of the up-, down-, left- and right-gestures, and the click function is activated by making the forward push gesture with the same hand.

To make a comparison with our own system we have installed KinEmote v0.4 Beta and tested it as a user. A main difference is that the cursor control and click function is controlled by the same hand. The cursor control itself worked well and was fairly easy to control both over great distances and when trying to hit small targets. However it was discovered during the test that the cursor was likely to move, when trying to perform a click. This also refers to the problem mentioned in the design section 5.2. To summarize the discovered pros and cons are listed below.

Pros

- Easy to control the cursor.
- Uses only one arm.

Cons

- Difficult to keep the cursor still, while performing a click.
- No gestures to perform basic operations when browsing.

To conclude, the system can be used to navigate web pages, but it was found to be very difficult to hold the cursor still, while performing the click-gesture.

KinEmote PCD v0.1 Beta

KinEmote PCD v0.1 Beta is designed with normal PC use in mind. The system supports cursor control and a click function. The cursor is controlled by moving a hand around in front of the sensor. To be able to control the cursor without performing a click, the fingers on the hand must be separated at all times. If the gab between the fingers are closed, the system performs the functionality of holding the left mouse button down, until the fingers are separated, which triggers the functionality of releasing the left mouse button. This makes it possible to perform a click and to drag and drop items.

We have tested the system in order to compared it with the system we have developed. The cursor is controlled the same way as with KinEmote v0.4 Beta. The click gesture seemed easy to perform, but keeping the fingers separated during cursor control stressed the muscles in the fingers and arm. At a distance of approximately 1.5 meter away from the sensor, we were also having difficulties with spreading the fingers sufficiently to not activate a click. KinEmote PCD does not support gestures to scroll or moving forward/backward when navigating web pages in a browser. KinEmote PCD supports the functionality of adjusting the sensitivity of the cursor, but trying the use this functionality resulted in a system crash every time.
To summarize the discovered pros and cons are listed below.

Pros

- Easy to control the cursor.
- Uses only one arm.

Cons

- Hard to keep fingers separated.
- No gestures to perform basic operations when browsing.

To conclude, the system enables users to navigate web pages, but based on experience the click gesture stressed on the fingers and arm muscles.

11.3.2 Win&I

Win&I is a commercial product that claims to offer a natural user interface for interacting with a PC. We contacted the company behind Win&I to ask them, whether we could have the opportunity for comparing their interface with our interface, but the company answered that this unfortunately was of no interest. We have considered the opportunity to use the promotion videos to make a comparison, but the promotions videos are created to sell the product and therefore only shows the product at its best. Win&I has therefore not been compared with our own interface.

11.4 Development

The development of a natural user interface was a new area that none of the authors have had any previous experience with. A lot of effort has been put in analysing, implementing and testing different ways of tracking a users hands and fingers. Simple prototypes have been implemented in order to test the possibility of using the techniques described in Section 5.2. Prototype 1 and 2 consist of approximately 1800 lines of code which have been written by the authors, and the lines of code are distributed as such:

- Tracking user movements - 34%
- UI and Test - 32%
- DTW-algorithm - 17%
- Cursor techniques - 16%
- Recording gestures - 0.5%
- Executing actions mapped to gestures - 0.5%

Besides that there are approximately 200 lines of code which are based on work from others that have been modified to fit the purpose of this system. The 200 lines of code are divided between Tracking user movement and DTW-algorithm. It must be noted, that the distribution of lines of code does not reflect which areas that have required the most time and focus.

It was discovered that debugging an application that receives data in real time based on a user's movement is very difficult, because it was nearly impossible to reproduce errors discovered at runtime.

During the development of the system we have gained experience on how a natural user interface can be implemented, how users's movements can be recognized using a DTW-algorithm, and how users interact with such an interface.

CHAPTER 12

CONCLUSION

The focus of this study was defined by the following research question:

> *To what extent can a natural user interface replace the standard input devices used for basic web browser interaction on a PC?*

This study has been conducted as a proof of concept and has led to the development of Prototype 1 and 2.
Prototype 1 and 2 makes it possible for users to navigate web pages using only their hands without the need of human touch and wearing any devices. The users are able to control the cursor with their right hand and clicking elements by performing a gesture with their left hand. Beside the functionality of clicking elements, the users can also navigate backward and forward in the web history and scrolling up and down by through gestures with their left hand.

What we find as the optimal technique when recognizing gesture in the final prototype is on average capable of recognizing 89.1% of the performed gestures.

A limitation when using the final prototype is the fact that the time it takes to move the cursor from A to B is approximately three times greater than using a standard mouse. Another limitation when using the optimal technique is the fact that 13.1% of all recognized gestures were unintended by the test subjects. The tests also showed a limitation in the difficulty of holding the cursor still, when performing the "Click"-gesture, also referred to as missed clicks. When using, what we found to be the optimal technique for controlling the cursor, the users had on average 10.11% missed clicks.

Based on the test of Prototype 1 and 2 it is possible to replace the standard input devices with a natural user interface when interacting with a web browser on a PC with above mentioned limitations.

A limitation of the study is the fact that the tests of Prototype 1 and Prototype 2 were both conducted using eight test subjects. It can not be garantied that the findings from the two tests can be applied on the general population due to the fact that only eight test subjects were used and all of them were students. In order to make a general statement a larger number and a larger variation of test subjects are required.

Future work

For future work there is a desire to improve the time required to move the cursor, increase the success rate and reduce the false hit rate, which is the number of movements that should have been recognized as gestures. However the test of Prototype suggests that there is a trade-off between the success rate and the false hit rate, meaning that an increase of the success rate produce a higher number of false hits. It must be noted that there was one deviation in the test, where the false hit rate decreases, while the success rate was increased. No explanation has been found for this.

If the system was to be released as a commercial product, it would be necessary to implement the possibility for the users to choose their dominant hand. The interface could be extended with the possibility to enter data using an on-screen keyboard. This would enable users to enter urls, logging in on web pages etc. This could be implemented using the Windows on-screen keyboard that could be activated and deactivated using a predefined gesture.

BIBLIOGRAPHY

[1] T. Beauvisage, "Computer Usage in Daily Life." http://laborange.academia.edu/ThomasBeauvisage/Papers/417510/Computer_usage_in_daily_life, 2009. [Online; accessed 19-December-2011].

[2] Apple, "iPhone." http://www.apple.com/dk/iphone/. [Online; accessed 28-December-2011].

[3] Apple, "iPad." http://www.apple.com/dk/ipad/. [Online; accessed 28-December-2011].

[4] Shelley Buchinger, Ewald Hotop, Helmut Hlavacs, Francesca De Simone Ecole, Touradj Ebrahimi, "Gesture and Touch Controlled Video Player Interface for Mobile Devices." http://dl.acm.org/citation.cfm?id=1874055, 2010. [Online; accessed 03-January-2011].

[5] Nintendo, "Nintendo DS Lite." http://www.nintendo.dk/ds/nintendo_ds. [Online; accessed 30-December-2011].

[6] HP, "HP TouchSmart." http://www.hp.com/united-states/campaigns/touchsmart/#/TouchSmart-610/Features. [Online; accessed 30-December-2011].

[7] U. Hinrichs and S. Carpendale, "Gestures in The Wild: Studying Multi-Touch Gesture Sequences on Interactive Tabletop Exhibits." http://dl.acm.org.zorac.aub.aau.dk/citation.cfm?id=1978942.1979391&coll=DL&dl=ACM&CFID=60086728&CFTOKEN=41733495, 2011. [Online; accessed 30-December-2011].

[8] A. Bragdon and H.-S. Ko, "Gesture Select: Acquiring Remote Targets on Large Displays without Pointing." http://dl.acm.org/citation.cfm?id=1978942.1978970, May 2011. [Online; accessed 09-November-2011].

[9] Z. Prekopcsák, P. Halácsy, and C. Gáspár-Papanek, "Design and Development of an Everyday Hand Gesture Interface." http://dl.acm.org/citation.cfm?id=1409318, 2008. [Online; accessed 30-December-2011].

[10] Nintendo, "Controls for Wii." http://www.nintendo.com/wii/what-is-wii/#/controls. [Online; accessed 28-December-2011].

[11] Sony, "This Is How I Move." http://us.playstation.com/ps3/playstation-move. [Online; accessed 30-December-2011].

[12] M. Kinect, "Introduction to Kinect." http://www.xbox.com/da-DK/Kinect/GetStarted. [Online; accessed 28-December-2011].

[13] M. Perry, S. Beckett, and K. O'Hara, "WaveWindow: public, performative gestural interaction." http://dl.acm.org/citation.cfm?id=1936672, 2010. [Online; accessed 30-December-2011].

[14] M. Nancel, J. Wagner, Emmanuel, O. Chapuis, and W. Mackay, "Mid-air Pan-and-Zoom on Wall-sized Displays." http://dl.acm.org/citation.cfm?id=1978969, 2010. [Online; accessed 30-December-2011].

[15] H. V. L. S. H. K. Chan, A.T.S., "Real-time tracking of hand gestures for interactive game design." http://ieeexplore.ieee.org/arnumber=5219910, 2009. [Online; accessed 3-January-2011].

[16] M. G. Doo Young Kwon, "A Framework for 3D Spatial Gesture Design and Modeling Using a Wearable Input Device." http://ieeexplore.ieee.org/arnumber=4373771, 2007. [Online; accessed 3-January-2011].

[17] Y. G. H. T. Jiatong Bao, Aiguo Song, "Dynamic Hand Gesture Recognition Based on SURF Tracking." http://ieeexplore.ieee.org/arnumber=5777598, 2011. [Online; accessed 04-January-2011].

[18] Y. L. Jacob O. Wobbrock, Andrew D. Wilson, "Dynamic Hand Gesture Recognition Based on SURF Tracking." http://dl.acm.org/citation.cfm?id=1294238, 2007. [Online; accessed 04-January-2011].

[19] BlackBerry, "BlackBerry." http://us.blackberry.com/smartphones/blackberry-curve-9350-9360-9370. [Online; accessed 30-December-2011].

[20] S. Ericsson, "Sony Ericsson W995." http://www.sonyericsson.com/cws/products/mobilephones/overview/w995?cc=dk&lc=da#view=overview. [Online; accessed 30-December-2011].

[21] S. C. E. Inc, "Using shortcut keys / mouse / keyboard." http://manuals.playstation.net/document/en/ps3/current/browser/shortcut.html. [Online; accessed 13-December-2011].

[22] IBM, "TrackPoint." http://www.pc.ibm.com/ww/healthycomputing/trkpnt.html. [Online; accessed 13-December-2011].

[23] T. Jilin Tu, Huang and H. Tao, "Face as mouse through visual face tracking." http://ieeexplore.ieee.org.zorac.aub.aau.dk/search/srchabstract.jsp?tp=&arnumber=1443150. [Online; accessed 30-December-2011].

[24] S. Y. M.-J. K. K.-H. H. Ji-Young Hong, Haeng-Suk Chae, "Does dynamic cursor control gain improve the performance of selection task in wearable computing?." http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1550822&isnumber=33046, 2005. [Online; accessed 31-December-2011].

[25] J. Blake, "Natural User Interfaces in .NET." http://manning.com/blake/MEAP_Blake_ch01.pdf, 2010. [Online; accessed 04-January-2011].

[26] D. Benyon, *Designing Interactive System.* Pearson, 2nd ed., 2010.

[27] M. Firefox, "What are the most popular Firefox menu items?." http://mozillalabs.com/testpilot/2010/03/17/popular-menu-buttons/. [Online; accessed 04-January-2011].

[28] PrimeSense, "PrimeSense Natural Interaction." http://www.primesense.com. [Online; accessed 18-December-2011].

[29] J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes." http://dl.acm.org/citation.cfm?id=1294238, October 2007. [Online; accessed 14-November-2011].

[30] M. Müller, "Information Retrieval for Music and Motion." http://www.springer.com/978-3-540-74047-6, October 2007. [Online; accessed 15-November-2011].

[31] M. Tang, "Recognizing Hand Gestures with Microsoft's Kinect." http://www.stanford.edu/class/ee368/Project_11/Reports/Tang_Hand_Gesture_Recognition.pdf, June 2011. [Online; accessed 09-November-2011].

[32] T. R. Trigo and S. R. M. Pellegrino, "An Analysis of Features for Hand-Gesture Classification." http://www.ic.uff.br/iwssip2010/Proceedings/nav/papers/paper_128.pdf, June 2011. [Online; accessed 09-November-2011].

[33] K.-Y. Cheng, R.-H. Liang, B.-Y. Chen, R.-H. Liang, and S.-Y. Kuo, "iCon: utilizing everyday objects as additional, auxiliary and instant tabletop controllers." http://dl.acm.org/citation.cfm?id=1753326.1753499, April 2010. [Online; accessed 09-November-2011].

[34] D. C. McCallum and P. Irani, "ARC-Pad: absolute+relative cursor positioning for large displays with a mobile touchscreen." http://dl.acm.org/citation.cfm?id=1622176.1622205, October 2009. [Online; accessed 09-November-2011].

[35] J. Lazar, J. H. Feng, and H. Hochheiser, *Research Methods in Human.computer interaction.* Wiley, 2010.

[36] T. P. M. Stuart K. Card and A. Newell, "The keystroke-level model for user performance time with interactive systems." http://dl.acm.org/citation.cfm?id=358895, July 1980. [Online; accessed 28-November-2011].

[37] E. W.Weisstein, "Latin Square From MathWorld–A Wolfram Web Resource." http://mathworld.wolfram.com/LatinSquare.html. [Online; accessed 10-December-2011].

[38] KinEmote, "KinEmote Gesture Driven Control." http://www.kinemote.net/. [Online; accessed 05-January-2011].

[39] Evoluce, "Gesture Control Software for Kinect & Windows 7." http://www.evoluce.com/_win-and-i/en/software/overview/index.php. [Online; accessed 05-January-2011].