Implementing Eye Movements in Business Applications

Master's Thesis in Computer Science

Olav Hermansen

February 2, 2015
Halden, Norway

Abstract

The work presented in this thesis show how eye movements are implemented in business applications. The work provides two prototypes, one gaze based and one gaze added. Prototype 1 is gaze based and is designed to handle navigation between top-level applications in a Windows operating system by only using the position of the gaze as input. Prototype 2 is gaze added and is designed to modify the tab stop order of a form were multiple input fields are present. The position of the gaze is used to determine the direction of the tab stop, still using the tab key to trigger the tab event. The prototypes are evaluated by testers and provide a discussion on how to implement natural eye movements in business applications. In addition a discussion is provided on how eye movements in business applications are perceived in relation to the use of keyboard and mouse. The findings provide two prototypes that implements natural eye movement and discuss the importance of implementing natural eye movement in business applications.

Keywords: gaze interaction, eye tracking, gaze added, gaze based

Acknowledgments

I would like to thank my supervisor Harald Holone for challenging me and inspiring me throughout the process of this master thesis. It has been a long process and I am grateful for the guidance provided.

In addition I would like to thank PostNord and some of its employees who were willing to be observed throughout their workday. You were all welcoming and quite eager to explain what you were doing and how you were doing it.

Finally I would like to thank my family whom have supported me all these years and throughout the work of this master thesis, and I would especially like to thank my mom, June whom have inspired me to be whom I am today. Without your key decisions throughout the last years, this would not be possible. Thank you.

Contents

List of Figures

List of Tables

Listings

Biometrics on the other hand is essentially adding more communication bandwidth between the game and the person playing it, especially in ways the player isn't necessarily conscious of. Biometrics gives us more visibility. Also, gaze tracking. We think gaze tracking is going to turn out to be super important.

Gabe Newell, CEO and founder of Valve

Chapter 1

Introduction

From the moment we are born we start to move our body, either if it's our arms, legs or even the mouth screaming. For each time a baby is born, an exciting adventure is started. While not going in depth of such a journey, it often involves its first words spoken and the first steps accomplished by its own. It is within the first years that one develops general motor functionality to walk, talk, move and play.

Skipping ahead in time we learn to use computers by enabling a keyboard and mouse. At first one might use a lot of time and concentration in order to type a word with the keyboard. During time and experience as one advance in writing on a keyboard, it becomes natural. One can say you are typing fluid when typing and no longer looking at the keys to do so. This is a motor skill we develop over time, pretty much like cycling.

Even though our hands start to move faster and more efficiently while typing and the mouse cursors moves directly to the appointed location, another human factor will still remain faster, our eyes. Our eyes have been measured to be the fastest moving human organ. Eye tracking software have been around for quite some time already, utilized within research, usability testing and acute communication for severe disabled people. Yet it remains a niche cause of the steep pricing. At least up until this year where hardware producers announced affordable eye trackers for the consumer market [6, 10].

1.1 Background and Motivation

Ever since the dawn of computers, the computer was made to solve a problem and to do it faster than a human. Computers are well employed in the typical business model as they make businesses and employees more efficient. Looking at the common usages of computers across businesses, we find the recurrences of mouse and keyboard as the standard methods for computer interaction.

The standardization of mouse and keyboard has proven over time to be the best suited interaction technique. However what if we could improve ourselves? What if the computer was given an extra incentive or understanding of what task we would like to perform, before we even initiate an action? To use an extra input modality that is already present. Our eyes can provide context to what we are doing and what we want to achieve. By introducing a third input that we already use without thinking. Could we achieve a more emerged experience work hours. Could eye tracking engage us into improving our self-performance and self-efficiency while still being at the workplace?

Until recently, eye tracking technologies have been restricted to studies of psychology, usability studies or in acute communication for critical severe disabled patients. This is due to the high prices of eye tracking equipment. However the recent changes in the market targeting the consumers, allow for cheaper and affordable eye tracking equipment. The eye tracking company the Eye Tribe started shipments of affordable eye trackers for the consumer market in early 2014 for the price of $99 US Dollars [6]. Approximately the same time a couple of days before CES 2014 the global leading eye tracking company Tobii Technology announces in a press release a partnership with SteelSeries to launch the world's first mass-market consumer eye tracking devices for gamers [1].

It is intriguing to play with the idea of using eye movements as an input for interaction and a sensor that knows where we are looking. Giving applications the possibility to determine what we want to achieve based on where we look, long before any interaction is physically initiated from moving our fingers, hands or body.

1.2 Research Question and Method

Having embellished the possibility of using eye tracking on a daily basis, raises questions of how the technology can be beneficial for businesses. This section will further specify the objectives for the thesis and provide a brief overview of the research method used to accompany the objectives.

1.2.1 Research Question

Based on previous work of eye movement in human-computer interaction techniques [21, 32, 23, 30, 28, 24, 16] the thesis aims to investigate in how eye movement can be accommodated in business applications. The thesis creates two prototypes tailored to be used at the workplace in an attempt draw close similarities to real business applications. The motivations for the prototypes are gathered by observing employees from different departments within a transportation company.

The thesis evaluates the implementation of eye movement in the prototypes by enabling natural eye movement. Further taking into consideration how the users respond to the different prototypes we look at how the eye movement is perceived in relation to the use of keyboard and mouse.

Through research, prototyping and evaluation the following research question will be focused on:

RQ 1 *How can eye movement be accommodated in business applications?*
Secondary relevant research questions are:

RQ 1.1 *How can natural eye movement be implemented?*

[1]http://www.intomobile.com/2014/01/07/ces-2014-tobii-brings-eyetracking-pc-gaming/

RQ 1.2 *How is the use of eye movement perceived in relation to the use of keyboard and mouse?*

1.2.2 Research Method

In the adventurous quest to create business application that implements eye movement in day to day work routines, I inserted myself into a fully working company that utilizes computers on a daily basis. By observations and interviews of various employees in the company I gained knowledge about how the employees would use the computer and insight in different application structures. By further analyzing the different uses of the computer to complete their work, I establish some common grounds for the different usages across the employees. Based on the observation of the employees, I created two high fidelity prototypes that implements eye movement to accommodate some of the reoccurring tasks in an attempt to simplify the workflow. The prototypes are thereafter tested by students in order to validate the validity.

1.3 Report Outline

- Chapter 1 holds the motivation of this master thesis and its research goals.
- Chapter 2 provides background information for how eye tracking is achieved before presenting the Windows design guidelines and at last present the related work.
- Chapter 3 provides the toolbox of methods used to carry out the work of this thesis.
- Chapter 4 covers technical details on how to implement eye tracking before producing the design and implementation on natural eye movements in the two business applications.
- Chapter 5 starts by establishing domain knowledge on business applications and observations from visiting PostNord. Further providing a description of how the eye tracker was configured and the lab environment were the tests were conducted. Details on the execution of tasks and test design is described individually for each prototype and results from the tests are presented.
- Chapter 6 discusses the results gathered from testing both of the prototypes. Separating the discussion on natural eye movement and the perception of eye movement in relation to keyboard and mouse for each of the prototypes.
- Chapter 7 Finally the last chapter summarizes the discussion before suggesting further work.

Chapter 2

Background

2.1 Eye Tracking

Eye tracking is achieved by analyzing the position and movement of the users gaze and translate where they are looking into a format. Applied with computers its common to translate the users gaze into screen coordinates making it possible to determine down to pixels where they are looking. Depending on the required precision it can also be used to indicate if the user is present or get directional data. In the literature this procedure is often described as gaze tracking as it is a more correct description of the procedure. While eye tracking could indicate that the eye organ itself is tracked, we are interested in the translation of the gaze. However the use of eye tracking is well established as the tracking procedure of the users gaze [23]. Eye tracking is therefore used within this thesis as the translation of the user's gaze establishing an absolute point of where the user is looking.

Eye tracking have been around for quite some time and its area of use is quite broad. Duchowski [16] an expert within the field created a hierarchy of eye tracking application. Duchowski separates the two main categories for eye tracking applications, namely for interactive or diagnostic usages. Duchowski emphasizes that this hierarchy is not deterministic as eye-tracking applications is emerging and that is should be used to spur further interest towards the field. Moreover the community have embraced the hierarchy and uses it to clarify their usage and placement of eye tracking applications [23].

This thesis points towards the branch of interactive applications. Thus it should be mentioned that eye tracking application in a diagnostic matter may concern usability studies for testing the efficiency of interfaces. In a usability study the eye tracker is used to gather objective and quantitative results of the user's visual and attentional process. Schiessl et al. [26] discusses the importance of eye tracking in usability studies. In the results Schiessl et al. presented, eye tracking gave insight to the user's behavior before the actual mouse click and important information before the user even starts interacting with the mouse. The overall criteria for diagnostic eye tracking applications are that the eye tracker doesn't need to interact with the application. The eye tracker is used to gather analytical information and it is usually represented as unobtrusive device [15].

Eye tracking as a tool for interaction has and still is playing a major role to the accommodation of people with disabilities. Duchowski [16] writes about universal accessibility and eye tracking as a tool for severely handicapped. Eye tracking is also used for eye typing and gaze pointing in acute communication systems where the user is unable to speak or move and still able to see. Such systems are usually implemented on a computer screen with a virtual keyboard. Writing with eye typing can be achieved by looking at one of the virtual keys and by hovering over the key for a short period of time, that will lead to a selection or typing of the key [16]. Eye tracking as a tool for interaction requires real-time processing of the users gaze as opposed to an analytical point where eye tracking can be processed at a later time.

2.1.1 The Eye

To fully understand how eye tracking is accomplished it's important to have some basic knowledge of the eye, how it works and what makes it possible for us to see the environment as we know it. The first concept of understanding how we see color, shapes and object is a source of light that hits a surface and is reflected on towards our eyes. See figure 2.1 for illustration and orientation to further understand the structure of the human eye as we break it down.

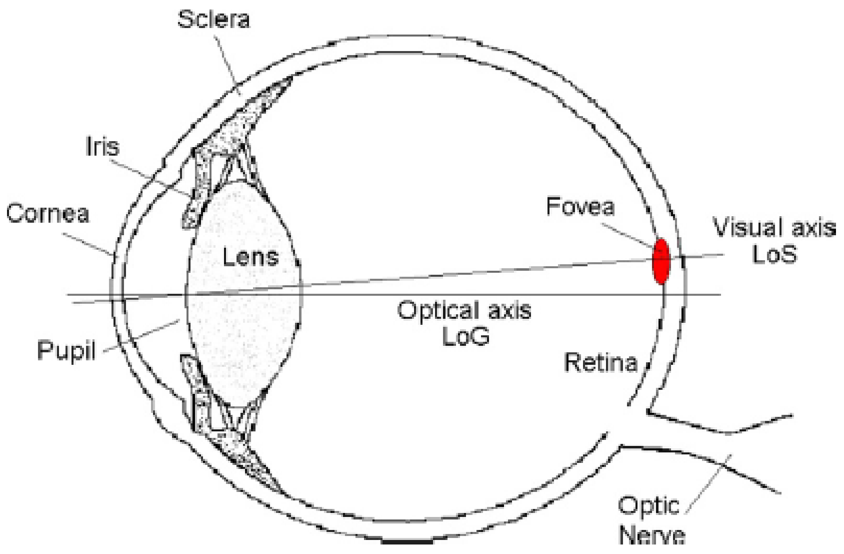


Figure 2.1: Structure of the human eye [24].

The light is first perceived through the cornea that is shaped as a dome refracting the light into the pupil. The iris is situated around the pupil, also known as the colored part of the eye, adjusting the opening size of the pupil to regulate the amount of light. Then the light goes through the lens that refracts the light a second time. The second refraction through the lens turns the light upside down, suggesting that we should perceive the world upside down. Finally the light hits that back of the eye and the retina. The retina consists of approximately 127 million receptors that can be divided into rods and cones [16, 24]. The distinction between the rods and cones is that the rods cannot sense colors of the light, where the cones can sense color. The neural signals obtained by these receptors are sent further onto the optic nerve. The optic nerve sends the impulses to the occipital lobe of the brain for interpretation and turns the image right side up again.

As we move the eyes around we steer the opening of the eye around changing the refraction from the cornea allowing other light reflections to be perceived. The reflection that meets the fovea perfectly within one-degree is considered to be where objects are perceived with optimal clarity[21]. This gives the basics for our visual attention indicating directions of where we are looking.

2.1.2 The spire to the field of study

The first eye tracker developed was in 1898 by Edmund B. Huey [18]. While he performed a experimental study of the physiology and psychology of reading, Huey noticed that the eyes where moving along the text by little jerks and not in a continuous steady flow. At first Huey tried to monitor the jerks by direct observation, however he failed as his capacity to observe was over swamped when the test subjects where reading at a normal speed.

In order to make a record of the eye movements during reading, Huey arranged for a apparatus to track the eye-ball. This consisted of a fixed frame where the head could be mounted in order to keep it from moving. To record the eye movement a cup was placed on the left eye. This technique is intrusive as the test subjects can't move freely and there is a cup attached to the eye-ball. This cup was molded of plaster-of-Paris after a steal ball shape. The cup was sand-papered until it was light and thin and a hole was made in the middle for the pupil to look through. A short lever was attached to the cup and a kymograph for recording. Figure 2.2 is a simplified illustration from the paper [18] of how the apparatus is constructed. Huey managed to complete 18 successful tracings with the apparatus.

Huey's eye tracking apparatus is the first eye gaze tracking (EGT) system developed that is documented [18]. There has been a development of several systems during the years since 1898. The major differences in the development of eye tracking techniques can be intrusive or non-intrusive eye tracking.

2.1.3 Different methods of eye tracking

The development of eye tracking methods have spired into multiple techniques since Huey's first apparatus. Solutions to day can still be considered intrusive, such as placing electrodes [21] on the skin around the eyes or a contact lens [21] that is perfectly fitted over the cornea with a slight suction. The use cases for these techniques are far apart when it comes to precision as electrodes makes it hard to find an absolute location, while a contact lens can provide a very good precision. The downside, apart from its intrusive usage is the required time and expertise needed to setup the eye trackers. These eye tracking methods are often limited to a controlled laboratory study due to its large scope of necessities.

Other solutions that are considered non-intrusive are remote eye gaze tracking (REGT) systems. Such systems utilize a remote camera analyzing the eyes in real-time. REGT systems are comfortable to use as it requires no physical contact with the user. This

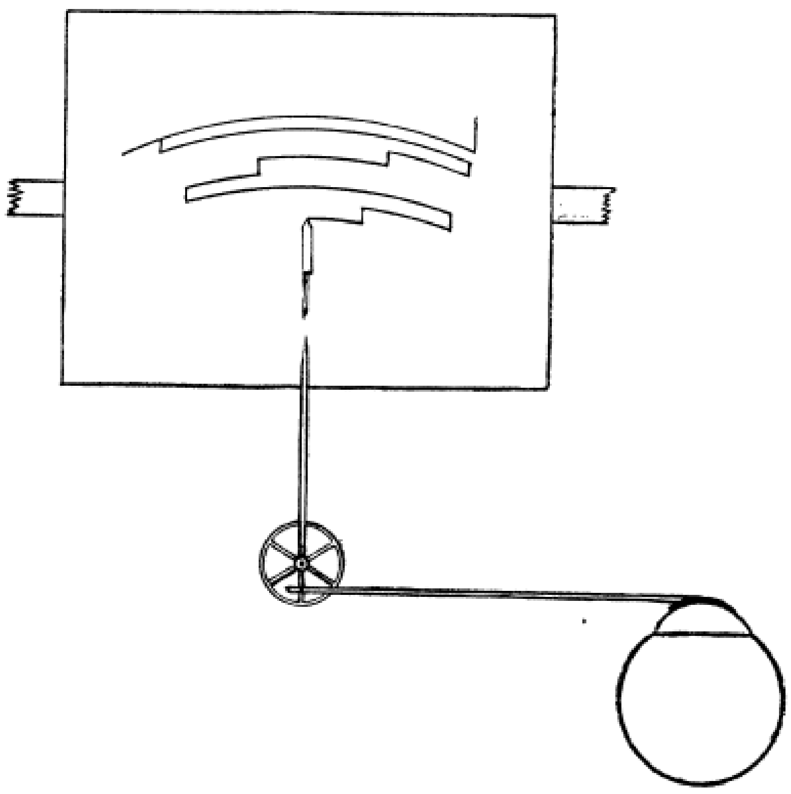


Figure 2.2: First eye tracking apparatus [18].

made it possible for users to utilize the system for a longer period of time without causing discomfort [24]. However the accuracy may vary due to head motion and calibration. REGT systems are known to be easy to setup and easy to calibrate as it usually only require a camera.

To briefly summarize some of the different methods of eye tracking its important to keep in mind that each method serves it purpose with different situations at hand. Even though some of the intrusive solutions gives great accuracy it comes with a trade-off such as discomfort. Vice versa is it for the REGT systems that can vary in accuracy and more comfortable to use. Further improvement of a REGT camera system is the usage of infrared light (IR) and is known as corneal reflection.

2.1.4 Corneal Reflection

Corneal reflection is the most common eye tracking method up to current date. It's a REGT system that incorporates an Infrared (IR) light source and IR camera for image processing. This is known as the pupil-corneal reflection technique[24]. When IR light hits the eye, the corneal gives a reflection, see figure 2.3. While the eye is shifting it's view of gaze the position of the cornea reflection stays at the same position. The cornea reflection can therefore be used as a reference to create a relationship between the reflection and the pupil[24]. By performing calibration on the relationship it is possible to get point of regard and almost know exactly where the user is looking. Keeping in mind that it's required to perform a calibration before any use, the quality of results may vary. Some of the challenges with calibration and potential flaws are discussed in further detail under challenges with eye tracking.

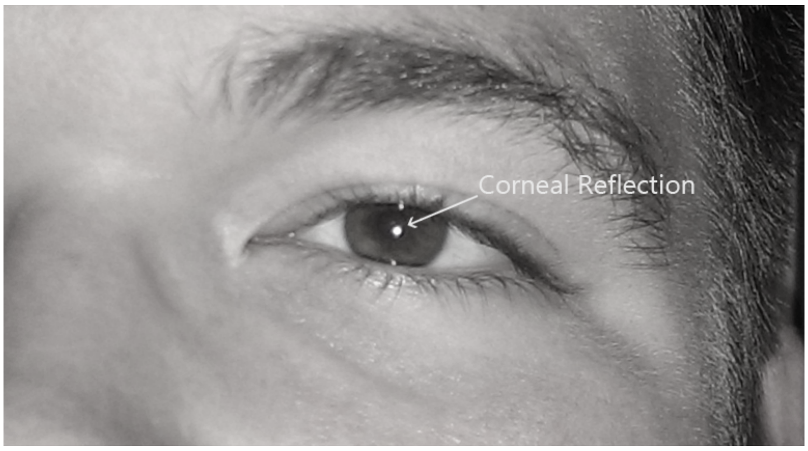


Figure 2.3: Illustration of the corneal reflection.

Infrared lightning is best suited to create the corneal reflection as it is not visible to the human eye. It's possible to use a regular light source as well to see the reflection; however it will easily cause discomfort and forces the pupil to contract, making it harder to distinct the relationship between the reflection and the pupil.

It is important to highlight some of the health concerns related to IR light exposure as it can damage the eyes. As mentioned the eyes defense towards too much light exposure is to contract the pupil, narrowing the amount of light that can pass through. IR light is not visible to the eye and therefore has no way of protecting itself. Luckily for us all of the commercial eye tracking products must undergo a thorough inspection and make sure that all IR lights passes a safe exposure limit set by international standards [23].

There is no correct way of how to structure IR lights when it comes to configuration of eye trackers. Some eye tracker structures IR lights in arrays of up to 40 light sources [23]. The safety exposure limits of IR lights must therefore take the following properties into consideration; positioning, number of sources, brightness, size and distance. The test takes into consideration the worst case scenario for all the given properties to prevent any damage to the eyes.

2.1.5 Midas Touch

There are many challenges when trying to implement eye movement in application, for instance when eye movement is designed to replace the mouse. In order to make a selection, there is no way for the eyes to indicate that what they want to select what they are looking at. This problem arises hence the input from the eye tracker is continuous, leaving the state to always be active. When dealing with input from the keyboard we can act on different states like key down or key up choosing when to activate which key whenever desired. While dealing with an eye tracker we don't have the luxury of knowing its intention; should it type the letter I am looking at? Or am I just searching for the right letter? This problem is known as the Midas Touch effect and occurs when a input device is overloaded.

2.1.6 Eye tracking in HCI

Human-computer interaction (HCI) is a term used to identify study and design relationship between people and computers. Since the dawn of computers the keyboard and mouse have been the primary input methods from people to computers. The field of HCI is striving to create natural interaction between humans and computers.

The human eyes are daily used to aid in conversations between people. The eyes can stipulate the spoken language by direct eye contact or facilitate turn taking in conversations [29]. Using eyes as input to create a relationship between humans and computers is quite natural in relation to how we already use our eyes. Bolt [14] is one of the first to address our eyes as a system component and to consider our eyes as a way of output. By addressing the means of context the eyes give when combined with an action. Bolt makes the following example by asking a crowd of people the following question "What is your favorite sport?". By looking at a particular person, all around as well as the person of interest understands that the question is directed to this specific person.

The Midas Touch effect is one of the big barriers when it comes to development of eye tracking applications. Jacob [20] proposes two techniques to encounter the Midas Touch problem. The first technique is solved by using a keypad. The user can freely look around on the screen and press a key to perform an interaction. The second technique proposed is dwell time. The user can indicate interaction by looking at an object for a longer period of time. The key challenge is to use sufficient time so the Midas Touch problem is not encountered and not to long time causing the user to be agitated and frustrated. A third possible interaction technique can be presented by using eye winks. This can be achieved by blinking on one of the eyes or closing the eyes for a longer time. However Jacob rejects this as a possible interaction technique since it is not natural in an eye movement-base dialogue, making the user to think about when he or she should blink. With respect to the field of HCI eye winks should be discarded where possible. The keypad is often added to stipulate interaction from an extra input device. Applications combining eye tracking with other input devices is referred to as gaze added interface. Applications restricted to only using the eyes is referred to as gaze base interface.

Comparing our eyes to other movable parts of our body qualifies them to be the quickest of them all. Zhai et al. stresses the desirability within the field of HCI to increase the speed of user input between humans and computers. In cases where eye tracker is used as an auxiliary to cursor positioning, Ware and Mikaelian [30] found that Fitt's Law is still applicable for all eye tracking applications that functions as pointer. They also found that the constant time was slightly faster with eye tracking as opposed to the time it took to grab the mouse, making the difference in improved time a constant.

To accomplish best possible interaction between humans and eye tracker we must become aware of how our eyes are perceived and the jitter that follows. Even during fixations jittery motions are encountered rapidly with small eye movements [21]. In Ware and Mikaelian's evaluation of eye tracker as a device for computer input, they try to address the recommended size of items to be selected. They found that the selection speed increased dramatically from 0.45 degrees of visual angle to 0.75 degrees. The speed gain flattened out where the visual angles exceeded 1.5 degrees. With a viewing distance of 90 cm, the 1.5 degrees is translated into squares of 24 mm indicating optimal size for selection. Zhai et al. [32] also reported that targets contracting from 1.5 degrees proved drastically harder to select.

2.1.7 Challenges with Eye Tracking

Schnipke and Todd [27] describe the challenges associated with eye tracking applications. In their investigation only six of sixteen participants where successfully tracked. Therefore some cases of REGT where combined with chin rests or bite bar to restrict the head motion [24]. Further Schnipke et al. discuss the challenges tied to performing successful eye tracking. Challenges that occurred were problems with multiple light reflections in the face or that the pupil did not reflect enough light. This would confuse the eye tracker leaving inaccurate or no results. Dry eyes would also affect the reflection making it less bright and challenging to track. In some cases the eyelashes would disturb or cover the eye to that extent that it was not possible to accomplish successful eye tracking. Cases where the user squinted leaving the eyelids to occlude, this would also prove difficulties to perform tracking. They also pointed out that in their search for users to perform usability testing, they would need to pick users that dint wear glasses as they couldn't track eyes with glasses in front.

2.2 Windows Components and Guidelines

To simplify the process of creating applications for Windows, Microsoft have create numerous components such as menu bars, text fields, lists etc... Through the standardization and reuses of these components across applications we get a lot of "out of the box" functionality. While not going in depth of all the functionality the components bring with them, I would like to highlight some features that is reflected towards the user and the fulfilment of Windows as a consistent operating system. A closer look at the textfield component, we have certain expectations of how it should behave. For instance, if we click the text field a marker should appear, indicating it's ready to receive input. Further keyboard shortcuts such as *Ctrl+A* should select all text within the text field and *Ctrl+C* to copy the selected text to the clipboard. These shortcuts are well defined with the operating system and expected to work with all text fields present across all applications.

To create a best possible interaction between the operating system and all its applications, Microsoft created some fundamental design guidelines for developers [9]. While not going in depth of all the guidelines, its necessarily to embellish the guidelines that affects the prototypes developed within this work.

2.2.1 Keyboard navigation

From above we know that Windows have numerous components that can be enabled to build applications more faster, more fluid and keep consistency. This section will focus on the elements that are used to allow navigation of components with keyboard. The following reduced list highlights the key concepts of navigating components.

- **Input focus** - To indicate when a control is selected a marker appears within the text field. If the control is not a text field it will draw a dotted rectangle around the control.
- **Tab key and tab stops** - The Tab key is the primary mechanism for navigating between components within an application. When the tab key is pressed the input focus is moved to the next component that has enabled tab stop.
- **Tab order** - Tab order specifies the next tab stop when the tab key is pressed. *Shift + Tab* can be used to traverse the tab order in the opposite direction.
- **Control groups** - When then user is presented with too many components with tab stop; it can be more convenient to group certain components into control groups. Reducing the amount of tab stops to one tab stop for each control group.

Navigating components such as the text field by utilizing the tab is highly desirable for experienced users. The necessity for using the mouse to navigate between text fields when the primary action for a text field is input of text, forces the user to swap between the keyboard and mouse. Not only is this more time consuming and fatigue can it also cause potential repetitive stress injury.

2.2.2 Keyboard navigation of top-level applications

The keyboard can also be used to navigate between top-level applications. Using the key combination *Alt+Tab* will start to iterate through the z-order of top-level applications. During iteration of the top-level applications, thumbnails of the applications are centered on the screen with focus on the last active application as seen in figure 2.4. Holding *Alt* and continue pressing *Tab* will change the focus to the next application in the z-order. It is first when the *Alt* key is released a selection is made. When a application is selected, it will be brought to the top-most order and gain focus. The z-order will always place the most recent application first and the least used application in at the end. The key combination *Shit+Alt+Tab* allows reverse iteration of the top-level applications.

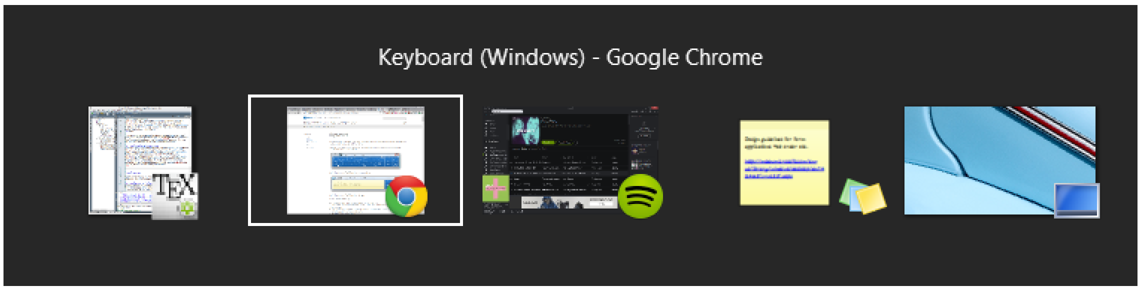


Figure 2.4: Navigating top-level applications with *Alt+Tab* in Windows 8.1

2.3 Related Work

Related work within the scope of this thesis is concerning similar projects that addresses eye movement as a point of regard real time in interactive systems and applications.

2.3.1 What You Look At is What You Get

Breaking the barrier to bring eye tracking into the grasp of human-computer interaction. Jacob [20] proposes six interaction techniques that fall within the categories of Object selection, Continuous Attribute Display, Moving an Object, Eye-Controlled Scrolling Text, Menu Commands and Listener Window.

Object selection the goal is to make a selection from multiple options presented on screen. As presented a selection can be obtained by looking at an object and by pressing a key to indicate a selection. The second alternative for selection is achieved by dwell time, simply by looking at an object for a sufficient amount of time. The two alternatives where implemented simultaneously to allow the user to perform an immediate selection instead of waiting for the dwell to kick off the selection. Jacob reports from observations that in practice this did not work very well arguing the dwell selection is more convenient. He did not describe why pressing a key is less convenient, as it could be possible to use other keys. It could be less convenient due to the task at hand as it is purely about making selection, not involving the necessity of using the keyboard at all. The observations done for dwell time are quite interesting. Dwell selections that are easy to correct by simply selection a new option should use less dwell time. This meaning that selections that are harder to correct should require a longer dwell time. This improves the correct selection reducing accidental selection for options that are harder to correct.

Continuous Attribute Display is a display that is divided into two parts vertically. The left side of the display consists of a list view and occupies 30 percent of the display. The right side occupying the remaining 70 percent can be considered as a map. With Continuous Attribute Display Jacob suggest that by placing points of interest around on the map and when a point is triggered by a fixation, properties for the selection is displayed conveniently in the list view. This fits well with the previous idea with object selection, where dwell time can be considerable reduced as it is easy to correct selection. Further Jacob emphasizes that the change in the list view is updated fast and that the change of properties is hardly seen unless the user looks straight at the properties while it changes. Since the changes of properties are updated while the user is looking at a selection, Jacob concludes that the user must be looking at a selection in the first place for a change to be made. Further concluding that based on the focused vision towards the selection, the users is not disturbed by the changes in properties. While I disagree, I think Weinschenk [31] can elaborate moreover the psychology of design. She refers to research on the peripheral vision and how blinking and movement attracts our attention. Meaning that even if Jacob found that changes to the properties should not distract the user, the peripheral vision is too good at detecting movement and changes suggesting that the distraction should be greater than described.

Moving an object on the display consists of selecting an object and perform a manipulation of the object. Jacob proposes two methods for moving an object on the screen. The first method is described to use the eye as a selector and the mouse to move the object. By looking at a object the user can click and hold the mouse button and drag the object around. While performing a drag operation the mouse is always moved relative to the selected object. The cursor is always hidden for the user. For the second method, the eye is used to select the object and a key is used for selection. By holding down the key when looking at the object, it is thereafter moved around from the eye movements. The object is only moved to where the user fixates for over 100 milliseconds, in order to counter the natural eye jitter creating a smooth transition. From the observations Jacob reports that their initial thoughts of the methods, the first select/mouse-to-drag method would be the best suited. However as the users got a hold of using their eyes to move the objects in the second method, it became more natural. Further Jacob reports that for the second method it was easier to move the object when there was a recognizable feature to look at. Because it is hard for the eye to fixate on a blank spot, it is recommended to have different features that object can be drawn towards.

Eye-Controlled Scrolling Text is applied to text that flows over the available space in a display. To indicate that there is more unveiled text to be read, directional arrows are placed above and or under the text. In order to scroll or move the text, the user must look at the arrow. While not looking at the arrow and reading the text, it remains still. Jacob makes the assumption that as soon as the text starts to move, the eye is attracted towards the moving text and away from the arrow forcing the text to stand still again. Reading until the end of the line and beneath where the arrow is situated will cause the text to scroll further down creating a natural reading order.

Menu Commands is a interaction technique for selecting a menu option from a pull-down menu. The steps acquired in order to make a selection consist of three different distinctive dwell operations. The first step is to open the pull-down menu. This is done by looking at the header of the pull-down menu for 400 milliseconds. This will cause the body of the pull-down menu to be shown. Each item is highlighted after a brief look of over 100 milliseconds indicating what item the user is looking at. However the item is not executed yet, as the user must be allowed to examine the different choices. If the user highlights one of the items and additionally looks at it for over one second, the item selection is executed. Furthermore Jacob allows for a keystroke to execute a item selection if it is already highlighted, allowing the user to interact faster. Additionally the pull-down menu allows the user to look outside of its bounding box for 600 milliseconds before it closes. Based on the initial experience Jacob made observations towards the beneficial use of a button to select the menu option after a highlight. The button is more favored as the dell-duration of one second seemed to agitate the user giving them a feeling of "staring" at the menu item. As Jacob elaborates that the user must be left with a reasonable amount of time allowing to comfortably reading the options and not causing unintended executions.

Listener Window attempts to create an interaction technique for setting the active window in a system. To give an example, let's say we have two documents opened for editing. The two documents are aligned next to each other making it possible to shift attention between them and read/write content without changing the position of the documents. However in order to write in both documents, the focus of the current window will need to be changed to the other document at some point. Back to Jacobs description, this interaction is usually obtained by positioning the mouse cursor over the document and clicking the window. It is also possible to hover the mouse over windows and just like a little dwell, the window is set active. Similar to the mouse Jacob proposes the use of eyes to determine the active window. A slight delay is implemented to the system allowing the user to briefly look at other windows without disrupting the current active window. Jacob also suggests for future work that the mouse position can be stored for each individual window saving the state it was left with.

Summary

From Jacobs overall observations from 1991 the eye tracker as an input device was far from perfect arguing the limitations of current equipment and the nature of human eye movements. Jacob draws similarity of eye tracking towards touch screen behavior while it is far from the accuracy we are familiarized with the mouse. Under optimal circumstances eye tracking can give a powerful impression where the system behaves on user intention. It is highly desirable to make interaction techniques based on natural eye movements where ever possible. Jacob found it also important to find fixations in the user's eye movement where precision matters the most. Rather than a jittery movement, fixations seemed smoother. Short dell time is desirable, however only if mistakes are easily to correct. If long dwell time is required where mistakes are harder to correct, one should exhaust the possibility of using other means of input first or at least make it optional with a button. I do however believe that the different distinctive dell-time options could be reduced as the user advances in the current system and become more familiar. However determining the dwell time based on user advancement, system familiarization and speed of reading can prove to be difficult.

2.3.2 Eye Movements for Video Game Control

Smith and Graham [28] experiments with eye movements for video game control. They target three different game genre with three different interaction approaches. While not trying to develop a game centered on eye movements, they implement eye movement in three different existing games. An overview of the three different approaches will be outlined as: Orientation in a First-Person Shooter, Communicating with an Avatar and at last Targeting Moving Objects.

Orientation in a First-Person Shooter modifies the game Jake2, this is an open-source Java port of Quake 2. The game is a typical first-person shooter (FPS) game where a weapon and a projectile path are centered on screen. The perspective of the game is through first-person perspective (FPP) where action takes part through the characters eyes. By default the orientation of the FPP is achieved by moving the mouse around. The implementation of eye movement is achieved by replacing the mouse behavior for FPP making the FPP center around the object the user is looking at. Character movement was accommodated with the keys WASD and the firing of weapon was assigned to the Ctrl key. Smith and Graham reports that when the user walked past an interesting object such as a box on the floor, the user would start moving towards it unintentionally because it fetched the users eyes. To correct the moving path the users had to fixate in the direction they wish to travel. Drawing the conclusion from Jacob on the observations on moving an object it's desirable for the eye to have a recognizable feature to look at. Features should therefore be present in some way to easily correct undesirable changes in order to continue the desired path.

Communicating with an Avatar is facilitated through the game Neverwinter Nights. Neverwinter Nights is a point and click game where an avatar is moved around in the environment in order to inspect interesting objects. The game uses a 3rd person perspective tinted from a bird view. By clicking on the ground, the avatar is moved to the clicked position. The perspective is always centered on the avatar. Interesting objects are inspected by left clicking the mouse as long as the avatar is within reach. Eye movement is implemented to replace the mouse as a pointer allowing the user to look at interesting objects and move towards them by clicking. During eye tracking the left mouse button remains as action facilitator. From Smith and Graham's discussion, this game was strongly preferred with eye tracking as it reduced the amount of effort put into moving the mouse. By simply clicking the mouse button the game would already know point of interest.

Targeting Moving Objects is situated within the game Lunar Command. In Lunar Command you are using a turret to defend the bases on ground from rockets in midair. The rockets are fired at increasingly velocity towards the ground trying to destroy the bases and all of its mankind. The ground turret can fire counter rockets, however these rockets must have a fixed time of detonation so they can blow of midair. The turret is controlled with the mouse, and sends a rocket to where the user clicks. This way the user is required to estimate where defending rockets blast area will overlap with the hostile rocket and take it out of play. Implemented with an eye tracker, the eyes take control of where the defending rocket should be fired. The rocket is always fired with left mouse button. Smith and Graham report that this was the most difficult game to succeed with eye tracking. The users would have to look ahead of the projectile path and determine where the two rockets would intersect. Not surprisingly the users would have a problem with looking ahead as there is no recognizable feature ahead of the rockets, as it is difficult to fixate on a blank spot and determine where and how far out it should be.

Summary

Smith and Graham present three comparisons with eye movements for video game control. The goal of the study was to test the effectiveness of eye tracking in existing commercial games. With this in mind, the games are mostly altered with a eye tracking device to substitute the behavior of the mouse. With knowledge from what Jacob did in his observations, they focus on measuring performance and subjective measures from participants without any effort on accommodating the previous work by Jacob. The three commercial games can easily be distinguished by the usage of player perspective. In the first game, Jake2 the user altered the FPP with eye movements continuously which can easily create error during game play. The second game Neverwinter Nights had the player perspective follow the avatar in a fixed bird view. From a fixed bird view, it seems easy to correct any error selections and never "loose" orientation. The last game Lunar Command also had a fixed view making it easy to keep orientation, however the difficulty was within targeting the moving objects by trying to look out in space with no recognizable features as well as estimating the projectile path in space. Also interesting they chose to keep left mouse button for executing actions in Neverwinter Nights. This might create expectations of the mouse or the illusion of using it. In Jake2 they chose to assign the Ctrl button to handle shooting eliminating the mouse completely.

2.3.3 Eye Tracker Input in First Person Shooter Games

Isokoski et al. [19] wrote a paper on gaze controlled games where they report on challenges with eye tracking in different game genre. The challenges are present in applications where the game requires constant or frequently interaction. In a basic first person shooter game you move around in the game where the objective usually is to kill the opponent or enemies. This involves player movement and strategic navigation. While moving around in the scenery the player must also be able to aim and shoot. The difficulty of managing these tasks by only using an eye tracker is extensively hard and impossible when you need to move agile and make multiple decisions at the same time.

Isokoski and Martin contacted a student group for creating a FPS game that could be used to measure the efficiency of an eye tracker and mouse. The game should be as simple as possible where terrain and trees where randomly generated. Targets of billboards with portrait of penguins would slightly move around giving the user somewhat of a challenge. Billboard within game development is typically a image that is always facing the player while the player moves around it. Isokoski and Martin found out early in their work that navigating the scene with eye movements requires too much effort. The basis of their research was to test for possible interaction techniques that would give a typical FPS gamer an advantage by using eye tracking software. The game was thereafter altered so that the mouse was used to orientate the FPP and within the perspective the crosshair was moved to POG based on data from the eye tracker. Instead of trying to replace the mouse, Isokoski and Martin tried to alter the game and find other usages for eye movement to increase player efficiency. The implementation of eye tracking is exclusively used to control the aim while movement and shooting is done by other input devices. The results gathered show that an eye tracker won't improve or accelerate the player performance leaving no advantages with eye tracking in FPS games. However they reported that eye tracking made the game more immersive to play.

Set aside the goal of performance increase, the results reported by Isokoski and Martin show that playing with the eye tracker would lead to least amount of wasted bullets. Given that if the player in the trails was given a limited amount of ammo, it could prove improved performance with more potential targets to be killed.

2.3.4 Summary of related work

The work done by Jacob establishes important factors in accommodating eye movement in HCI. When performing selection it is desirable with short dwell times, however it must be weighed against the cost of accidental selection and the effort required correcting. Where the eye is used to manipulate input from continuous eye movement, it is better to act on fixations rather than on the continuous jittery movement. Applications should try to utilize the naturalness of eye movement where possible.

Smith and Graham's work on eye movement in different game genres makes little effort in accommodating eye movement in games. In their approach they override the mouse functionality with POG from the eye tracker. It is however interesting to learn how eye tracking works on the different perspectives presented within the different game genres. Games where POG easily created errors and would make major changes to the perspective would make it less desirable to use. Relating to Jacob's work on dwell time and error correction gain experience on suitable eye movement in games.

Isokoski and Martin try to find uses where eye tracking can become an advantage in FPS games. Instead of trying to override the mouse behavior in existing games, they had a small and simple FPS game created. Addressing the need for games to be created around eye tracking support rather than trying to add eye tracking support to traditional games with established means of interaction.

The related work presented should highlight important factors towards the work that has been done on eye tracking interaction. The step between business application and game applications are quite apart, however the experience is applicable. The related work is also to be considered the foundation in further development of eye movement in business applications.

Chapter 3

Methodology

Eye tracking has been dramatically improved over that last two decades following the parallel development of better cameras and beneficially from cheaper production costs. The steep price reduction of eye tracking technology opens up the possibility of eye trackers in the consumer market. As discussed in related work, eye trackers have been utilized for generic computer interaction and tested for various interaction in computer games. This motivates the question of how business applications can utilize eye tracking and how applications can be design to accommodate users gaze.

Methodology determines the research design. Where the two major directions for how to conduct the research falls within quantitative and qualitative design. Quantitative research has its bearing ground in a large amount of data that allows mathematical models to be applied. It is highly desirable to convert data into numerical values applying a statistical view. Quantitative research is often applied to test theories on large populations and within geographical areas. Qualitative research focuses on a smaller group of people. The data collected in a qualitative research is often words and sentences as opposed to numerical values in a quantitative approach. Qualitative research can include focus groups, in-depth interviews and observations. The goal of qualitative research is to explore and find relativism. The truth can be found when qualitative research is applied through quantitative exploration.

The process of conducting this research is achieved by reading up on previous and related research. The foundation of the prototypes created lies within the accumulated knowledge from related research and experience gained through experimentations. The prototypes are tested and evaluated within a controlled laboratory following the experimental design approach with a qualitative take. Further throughout this chapter the research methods used will be outlined, how they were executed and how the data was gathered and analyzed.

3.1 Order of Execution

The fulfilment of the project consists of several phases. Where the list on page 22 gives an order of execution it does not show the various steps of iteration between specialization within the field and developing the problem statement. The phases have a natural overlap and to consider the execution order a sequential order would be an overstatement. Emphasizing that the different stages where affected from a continuous learning and discovering process.

Execution order

1. Specialization within the field of eye tracking
2. Code and interaction experiments with eye tracking
3. Develop first prototype
4. Developing problem statement
5. Shadowing various people in a business
6. Designing experiment
7. Develop second prototype
8. Design tasks for prototypes
9. Develop support applications for testing prototype one
10. Recruit testers and test prototype one
11. Analysis of data and results for prototype one
12. Recruit testers and test prototype two
13. Analysis of data and results for prototype two
14. Finalizing document

3.2 Within-Subject Design

Within-subject design is a type of experimental design often used within the field of psychology. The concept of the design is that all of the participants are exposed to more than one or all the treatments. The term treatment is used to describe the different levels of the independent variable. An example of within-subjects design could be where we are interested to know if listening to music before holding a presentation have any effect on the participant. The independent variables could be listening to music while preparing for a presentation or enduring a quiet room before preparing for a presentation. The participants can for instance be subjected to the first condition where they listen to music before the presentation. The same participants are also subjected to the second condition where they get to sit in a quiet room preparing for the presentation. This is the concept of within-subject design where all participants are exposed to all the independent variables. The participants are scored on their delivery of the presentations. The scores can at first hand be compared to each individual as each will have different experience with doing presentations. Further the overall scores for each condition can be compared to figure out which condition helped improve the participant the most.

There are of course strengths and weaknesses tied to within-subject design. By having all the participants participate in all of the conditions they might improve their skills at presenting due to quantity of training. Suggesting that for each time a participant presents, there is a learning curve regardless of the different conditions. This is known as the carryover effect or sometimes referred to as testing effects. The within-subject design requires fewer participants as all the participants participate in all of the conditions. This eases the recruiting process resulting in the half amount of participants required and should not be confused with the between-subject design where the participants are limited to participate in one condition. Looking at the example again, if the participants exhausted all their energy singing to music while getting ready for the presentation, they might be to worn out when going on to the second condition. If even more conditions are present, the last condition could suffer from the fact that the participants are tired and want to go home rather than giving their full effort. One attempt to reduce the carryover effect is to counterbalance by changing the order of execution for each participant.

To enlighten some of the strengths and benefits of within-subject design as already mentioned is the required amount of participants. Also by having all of the participants attending all of the conditions will reduce the errors of individual differences across conditions. Back to the example; if a particular participant is excellent at doing presentations. It would still be possible to validate the results as the excellent participant will effect both conditions, striving a reduced bias and applicable result.

Within-subject design is a well-established research design within the area combining HCI and eye tracking [33, 17, 22]. Especially since most of the research is experimental, the within-subject design allows multiple independent variables where the goal is to find the best suited condition. This does not imply that the research method is experimental. However while the research is experimental, the methodology of within-subject design aims for a controlled experiment with a qualitative approach. It is also implied by Hansen and Aoki [23] that the independent variables and the dependent variables set the tone for the entire experiment.

3.3 Prototyping

In order to find suitable interaction techniques for eye tracking in business application one must first create an idea of how this could be achieved. The idea may seem brilliant once first glanced upon, however the rule of thumb may be different once the idea is put to practice. This creates the idea of a prototype, a representation of the idea in form of a sketch, mock-up or even a sophisticated computer program. The main purpose of a prototype is its ability to be analyzed and evaluated.

When working with eye tracking technology it's hard to facilitate good prototypes on paper. Even software prototypes can prove hard to evaluate, as there is potentially so many things that can go wrong. The prototypes endured within this thesis aims to facilitate a mild form of sophistication while still building on dummy data and lack the overall structure.

3.3.1 Prototype 1: Navigating Between Top-Level Applications

This prototype was built to allow the eyes to navigate between the top-level applications in a Windows operating system. The prototype displays a menu icon at the bottom left corner of the screen upon startup. Once the gaze is registered within the trigger area of the icon, a new menu will instantaneously spawn. The appearing menu floats horizontally along the bottom of the screen listing all available top-level applications. The appearing list items are afflicted by the z-order of the top-level applications. Meaning that the first menu item displayed starting from the left to right will be the current top-most application. Each item is structured as a large square with appearing icons centered in the middle, see figure 4.4. A text field at the bottom of the square contains the title of the representative application. The background of the menu item gazed upon will change to indicate selection when the gaze hovers over. However no selection is made before the gaze exits the top boundary by looking back to the center of the screen. This action will close the menu and bring the selected application to the front and maximize it. The gaze pattern is illustrated in figure 4.2.

3.3.2 Prototype 2: Gaze Directional Tab Order

This prototype proposes a way to use the gaze to manipulate the tab-order in business applications. Especially in business applications that inhabits multiple input boxes that are used for data entry, update or delete. Figure 4.5 is an illustration of an application with 21 input boxes. Each of the boxes is numbered to indicate the direction of the tab order specified by the developer. If the cursor is set to the first input box and its desirable to update the fifth field, the user could click the fifth field directly or press the tab key four times to move the cursor ahead. Prototype 2 implements a directional tab-order allowing the cursor to move in the direction of the gaze, rather than following the predefined tab order as seen in figure 4.5. This makes it possible to look at the fifth field and press tab twice by having the cursor move diagonally from field one to field five. The gaze pattern is illustrated in figure 4.7.

3.4 Development platform

The prototypes are developed with the programming language cSharp and the .NET Framework 4.5 in Microsoft Visual Studio IDE. Visual Studio allows for easy creation of Windows Form Application and is a great tool for developing applications for Windows. The .NET Framework makes it possible to access core functionality and access functionality from the operating system.

3.5 Participants

The participants recruited for testing the prototypes are students from Østfold University College and derived from the department of Computer Science. Their background in Computer Science qualifies them as experienced users of computers. The target audience of the prototype inhabits employees using computers on a daily basis, making students from Computer Science excellent substitutes. Throughout the thesis the participants a referred to as testers. Establishing that it's the prototypes being tested and evaluated by experienced testers.

A total of eleven students took part in testing the prototypes. Six testers participated in testing prototype 1 and five testers participated in testing prototype 2. From a total pool of eleven students, two were female and the remaining where male. The mean age of all the students was 26 years where the youngest was 20 years and the oldest where 35 years old. None of the participants did wear glasses, though this is only a coincidence. Some of the participants did wear contact lenses.

3.6 Bias

Having testers from the same faculty as the candidate in writing can affect the testers in not being straight forward. From the pool of testers somewhere familiar with the research being conducted, others where only acquainted with the candidate without any knowledge of the work. Afraid of having testers not being honest, it was repeated for the testers in writing and through dialogue that it was the prototypes that where measured and not them as individuals.

Knowing that testers are easily affected by the behavior of the candidate during testing and interviews, the candidate tried to keep a reflected feedback, promoting positive and negative feedback from all of the testers. Having developed the prototypes at an earlier stage before conducting the tests, helped the candidate in distancing ownership and pride towards the prototype. Yet there will always exists a possibility that the testers where affected by the candidate. Ideally a third party should be used for testing the prototypes.

Introducing eye tracking technology can also have bias towards the testers. Due to the technology being unfamiliar and exiting the testers can be colored by the technology and not the interaction as being evaluated. Bias may also exist in the technology itself, where the eye tracker is inaccurate. The testers may blame the prototype, while it could be the technology used that failed.

3.7 Details about Data Collection

Data is gathered throughout the whole test phase of both prototypes. For each of the test conditions and for all of the testers, video recordings where made. A camera was placed at an angle behind the tester capturing the keyboard and mouse as well as the computer screen where the prototypes are tested. The recordings are stored so that they can be analyzed later, making it possible to discover other aspects that where not noted while testing.

Simultaneously as the camera would record the test conductor would take notes on irregularities and remarks that are not clearly visible through the recordings. For instance facial expressions and confusion are not easily spotted from the video recordings. It is always neat to have a sheet of paper to note ideas as they emerge during the tests.

The web application from the workflow for the first prototype implements a registration form that acquires name, age and test condition before starting. The web application would also silently record the time the tester spent solving the tasks starting from task three and until the test is completed. The two first tasks where not recorded as they were used to solve in collaboration with the test conductor. See appendix B for more details on the web application.

After completing the tests, the testers where interviewed. A phone was used to record the interview so the interview could be transcribed and analyzed at a later time. A phone provides better audio than the camera placed further back in the room. Having recordings of the interview also makes it easier to get correct quotes from the testers as well.

3.8 Tasks

Tasks were created when testing both of the prototypes. The tasks or workflow where created to resemble real work. In addition to that, the tasks would give incentive for using the prototype and attempting to reduce the focus on the technology, giving more valuable feedback on the prototypes and the interaction it self.

For prototype 1, a workflow where created that would require the testers to use three applications. Were each application serves a purpose in providing information for completing a task. The task is repeated several times creating a workflow. The data gathered where randomly generated each time. See Chapter 5.4.2 for more details on the workflow design for prototype 1.

For prototype 2, a workflow where created to make the testers update a new random text field in the prototype. When using the prototype, one of the text fields would always have a missing value. The value could be extracted from sheets of paper, updating a different text field each time. See Chapter 5.6.2 for more details on the workflow design for prototype 2.

3.9 Ethics and Privacy

From the interview with PostNord, June has agreed upon to be mentioned by name. All other employees interviewed are anonymized by refereeing their department, giving an idea of what they do for work. Insight into confidential information is not shared and observations of employees will never be shared with other employees or management.

Given the focus towards evaluating the prototype and not the testers, the data from the tests will be anonymized. The master thesis have no use of going into specific details about the testers and therefore this information is hidden behind references to testers #1, #2 etc... Additionally if material of the testers such as images where the tester is recognizable and its desirable use the image in the document, the testers must consent to each individual image before it can be used.

3.10 Eye Tracking

During development and testing, a REGT system from The Eye Tribe is used to track the gaze in real time. The REGT system needs to be calibrated for each individual user and preferably for each time the system is being used. The most basic data provided by the REGT system is the gaze translated into screen coordinates provided in the format of *(x,y)*.

The eye tracker form the Eye Tribe where chosen because of its cheap pricing and its status as a up-and-coming company. Alternately an eye tracker from Tobii could have been used without supposedly changing the result of the outcome of the research as both REGT systems should provide approximately the same data.

The implementation of the EyeTribe tracker is further described in Chapter 4.1 and the settings of the REGT system is outlined in Chapter 5.2.

Chapter 4

Design and Implementation

This chapter describes how the REGT system from the Eye Tribe is implemented for both of the prototypes. Further the chapter goes in depth of each of the prototypes developed, going through the gaze pattern developed and technical details about the prototype.

4.1 Implementation of the Eye Tribe Tracker

Before the eye tracker can be used, it must be connected to a computer with a USB3 cable. The computer needs to run the Eye Tribe server to ensure that the eye tracker is properly connected and ready to use. The Eye Tribe server receives the raw data from the eye tracker and handles the image processing. By default the server is idle when started and remains idle until eye tracking is requested by any of the clients. In terms of client, both of the prototypes are considered clients and may subscribing to the data feed.

In order to start receiving data the client must connect to the Eye Tribe server over the TCP protocol. The connection is kept alive by sending a heartbeat on intervals. The heartbeats therefore serves two purposes, where one is to tell the server to keep sending data and two, let the server rest if no clients requests any data.

The communication between the server and client is JSON formatted. The JSON object of interest is referred to as frames. Frame objects contain all the available information about the tracked eyes. The eye tracker can provide gaze data individually for both eyes and additional information of pupil size and normalized pupil coordinates. However the information that is utilized in the prototypes are the average smoothed coordinates from both of the eyes. A JSON snippet of the frame object is illustrated in listing 4.1 where line 5 reefers to the utilized gaze data.

Listing 4.1: Snippet of the frame object from the eye tribe documentation.

```
1 ...
2 "frame" : {
3   "time": int,    //timestamp in milliseconds
4   "fix": bool,    //is fixated?
5   "avg": {        //smoothed gaze coordinates in pix
```

```
    "x": int,
    "y": int
  },
...
```

4.1.1 Technical overview for C#

The Eye Tribe provides a great implementation guide [2] for implementing the eye tracker through the programming language C#. This section will therefore not try to create a new implementation guide, but rather try to give a rough overview of how the eye tracker is implemented.

To start receiving data from the eye tracker, a TcpClient is instantiated to handle the communication. The first method of interest is the Connect [2]. The Connect method sets up the TcpClient with the connection information to the eye tracking server. Thereafter spawning a ListenerLoop in its own thread for reading information from the server. Further the Connect method handles the heartbeat timer. The heartbeat timer is set to message the server once every 250 milliseconds to tell the server that the client is still alive. The threaded ListenerLoop reads the socket stream from the TcpClient continuously.

From the socket stream, the frame objects are parsed with the Newtonsoft.Json [3] library. Where the *(x,y)* coordinates are extracted to further be used in the prototype.

Alternative it is now possible to download a library [7] for C# where multiple interfaces can be used to simplify the implementation. Also making it possible to add a interface for calibration directly within the application implementing the library.

4.2 Prototype 1: Navigating Between Top-Level Applications

Windows 8 allows navigating between top-level applications with keyboard shortcuts and is further described in the background chapter on page 14. It is also possible to navigate the top-level applications with the use of a mouse as each top-level application will appear on the taskbar. However these shortcuts are not affected by the z-order, but the order of which application was first started. The order of the applications on the taskbar is also affected by applications pinned to the taskbar as they will not rearrange when other applications are brought to front.

Introducing eye tracking for navigating top-level application raises new challenges towards interaction styles and software designs to accommodate the interaction. While many of the challenges tied to eye tracking interaction is discussed in related work, the thesis aims to use the acquired information in improving situational software, also implying that related work is not definitive. This chapter will elaborate towards how the prototype is born and the different considerations taken during development. The end goal for the prototype is aimed to be a gaze base interface with easy error correction and limit the use of dwell time making more efficient to use during work.

4.2.1 Prototype Design

The prototype is inspired from how Windows already handles navigation of top-level applications with keyboard shortcuts. Figure 2.4 shows the appearing menu in Windows 8.1 when *Alt+Tab* is pressed. By holding down *Alt* after pressing *Alt+Tab* will keep the menu active, allowing the user to continue pressing *Tab* in order to cycle through the applications. When the *Alt* key is released, the selected application will be brought to front.

Continuing with the concept of a eye tracking implementation, the menu needs a initiator to tell when the menu should show and when to disappear. We are already familiarized with the Midas touch effect and the problem of distinguishing action from looking. While there is numerous proposed ways to work around the Midas touch effect [20, 19], they are generally not really fast. Taking into consideration the work by Ohno and Mukawa [25], providing criteria's for effective interaction. Saying that repeated use will improve the way we use eye tracking technology because the processes becomes automated over time. This means that dwell-time will create a bottleneck as the dwell-time is constant no matter how experienced the user gets.

This lead to the decision of reserving a area of the screen dedicated to trigger the menu selection. Figure 4.1 illustrates the trigger area with a green rectangle. Within the prototype the green rectangle is gone, as the trigger area is not visible within the prototype. Whenever the user was to look at the bottom left corner of the screen would trigger the menu to appear. The trigger area is reserved for the screen and it is independent, no matter which application that is currently top-most and active. Clarifying that the trigger area will work for any given application and seem integrated with the operating system.

To make the trigger area more accessible, a recognizable feature where added to the bottom left corner within the bounds of the trigger area. As suggested by Jacob [20], Smith and Graham [28] the eye should have a recognizable feature to look at when it is expected of it to trigger a action. The icon serves no other purpose than being an attention grabber for the eyes. The icon was designed to fit well within the trigger area so it would not contaminate the view of other applications or at least as little as possible.

When the menu is triggered, each of the top-level applications will be represented by boxes along the bottom of the screen. Where each box represents a application by placing an icon of the application in the middle of the box and the window title of the application is placed beneath the icon as illustrated in figure 4.4. The size of the boxes is four times larger than regular shortcuts in Windows 8, making it easier for the eyes to select the desired application. This is based on Ware and Mikaelian's [30] findings when evaluating the recommended size of items to be selected when a eye tracker is used for computer input. Since the pixels per centimeter may vary, the boxes size is set to 140x109 pixels, achieving boxes that are rendered to approximately 4.4x3.5 cm to assure that the sizes are greater than 2.4x2.4 cm as suggested for the minimum size for optimal performance. Placing the application icon in the middle of the box is also with regards to Jacob, Smith and Graham where the icon is used as a recognizable feature for the eyes.

To indicate when a application is selected, the background color of the box is changed to a higher density. However the status of the application remains unchanged until the gaze resumes to the middle of the screen. Similar to Jacob's interaction technique Continuous

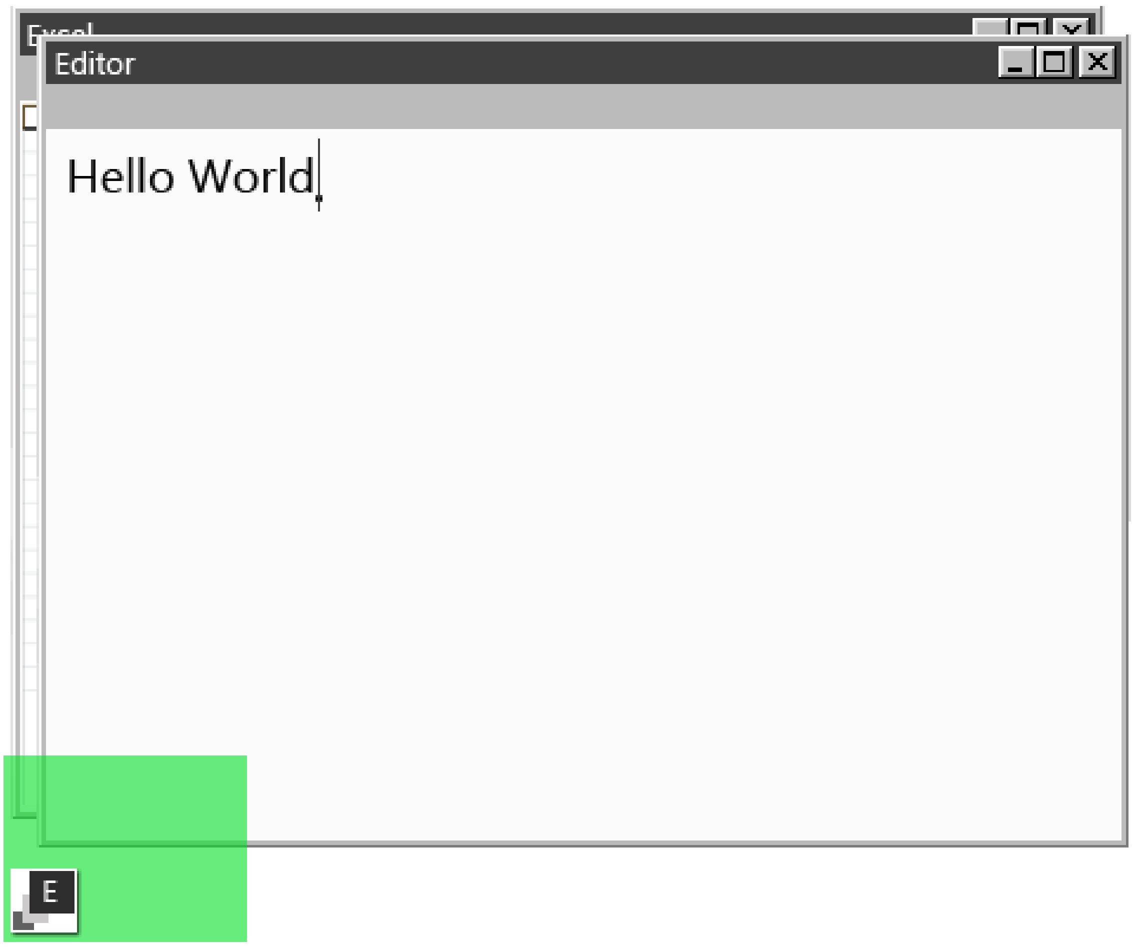


Figure 4.1: Illustration of navigating top-level applications with trigger area highlighted in green.

Attribute Display as highlighted in Chapter 2.3.1. The interaction technique by Jacob emphasizes easy error correction where a list is updated when coherent points of interest are triggered by dwell. Removing the use of dwell, the application is selected when the gaze exits upwards to the center of the screen. If the application where to be selected and brought to front each time the gaze looks at one of the menu boxes. The constant switching of application would cause noise and request the attention of the gaze too often. Causing disruption in the interaction achieving the same effect as blinking adds when browsing the internet [31].

4.2.2 Gaze Pattern and Desired Usage

To navigate to a new application, the user would look at the icon in the bottom left corner of the screen. Once the gaze is registered within the trigger area, the menu will appear with the available applications opened. The current top-most application will appear first in the list. If the gaze enters the trigger area by accident, the current application would be selected to prevent error selections. By searching through the available applications will keep the menu present and when the desired application is found, the user may look at the center of the screen, triggering the last selected application to appear in front of the eyes. Figure 4.2 illustrates the typical gaze pattern for navigating to a new application.

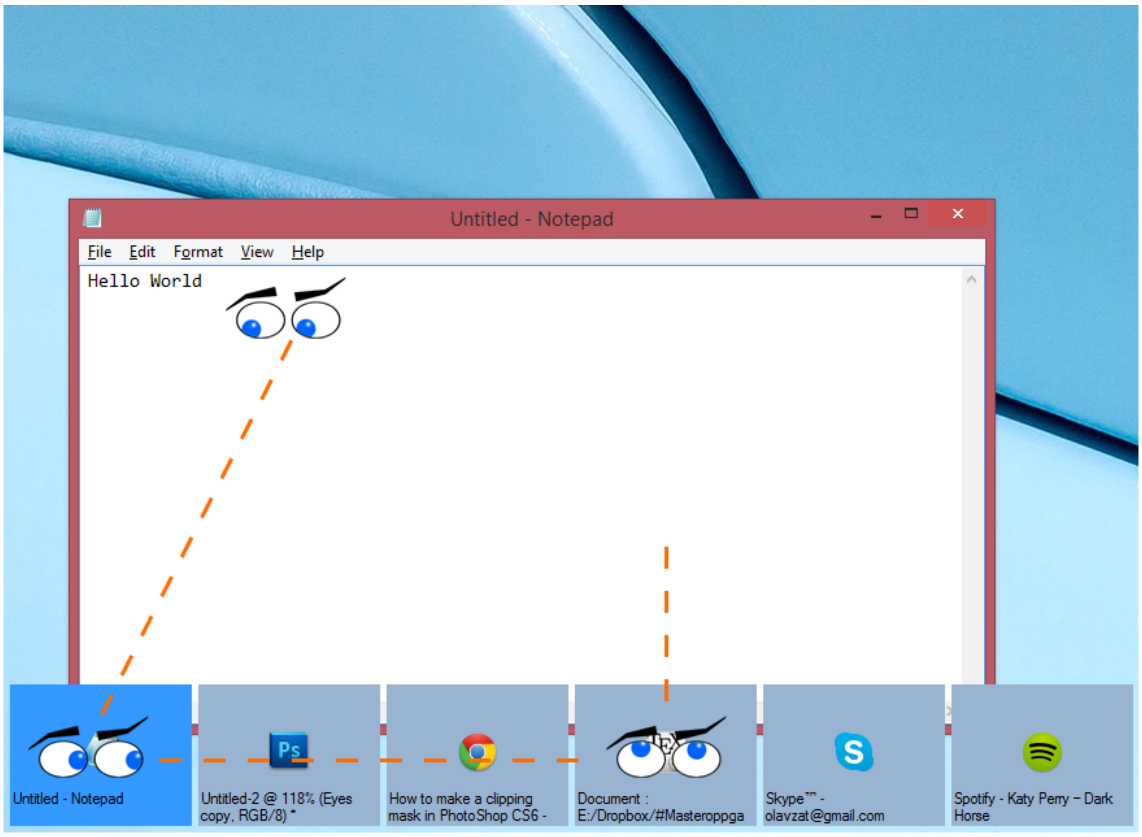

Untitled - Notepad
File Edit Format View Help
Hello World
Untitled - Notepad
Untitled-2 @ 118% (Eyes copy, RGB/8) *
How to make a clipping mask in PhotoShop CS6 -
Document : E:/Dropbox/#Masteroppga
Skype™ - olavzat@gmail.com
Spotify - Katy Perry – Dark Horse

Figure 4.2: Gaze pattern for typical navigation for Prototype 1.

4.2.3 Technical Implementation of the Prototype

The prototype is created by creating a new Windows Forms Application in Visual Studio. The prototype uses two distinct forms. One form dedicated to show the menu and one to only show the icon used as a recognizable feature for the eye as seen in figure 4.1. To ensure that both the trigger icon and the menu are display above any running applications, both of the forms have set the attribute of TopMost.

To avoid the necessity of distinguishing the coordinates from the eye tracker and the forms, the forms have set the FormBorderStyle to none and adjusted to fit the whole screen. This allows the coordinates from the eye tracker to be directly applied within the prototype without further adjustments. In earlier attempts where the form was resizable and have the FormBorderStyle set to sizeable. The offsets where calculated by requesting the forms top and left position. However the top and left position would return the position starting from the outer border. Meanwhile the internal coordinates form the form would begin from the inside of the border and beneath the form title and menu. This is illustrated in figure 4.3. Of course there is possible to add a constant to compensate for the border width and title height, but depending on how Windows 8 would render the text size in the title and border style, the offset will quickly become inaccurate. Thus the prototype is set to fit the whole screen reducing the concern of miscalculations of the gaze.

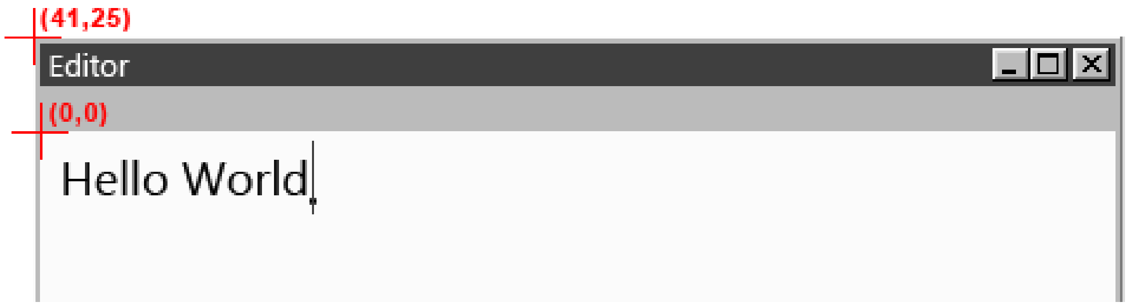


Figure 4.3: Illustration of the forms position and were the coordinate system for the form begins.

To ensure that the icon used as a recognizable feature and is always placed in the bottom left corner. The image is anchored to the bottom and left in the form element. The form border is removed along with the minimize, maximize and close button from setting the FormBorderStyle to none. Since the forms within the prototype are set to cover the whole screen, the forms property of TransparencyKey is used to make the whole form invisible. Both of the forms use a green color for the TransparencyKey. These results in the forms being placed over the whole screen and all elements that obtain the same green color are rendered invisible. By rendering the form invisible will also allow the form to be click-through and not interfere with ordinary mouse clicks.

The prototype imports the User32.dll library and the EnumWindows [1] function. The EnumWindows function allows the prototype to obtain all of the top-level applications within the Windows operating system. The prototype specifies a callback function that is passed as a parameter to the EnumWindows function. The callback function EnumTheWindows is set by passing the function as a parameter to a new EnumWindowsProc class as the first parameter in the EnumWindows function. To clarify see the code in listing 4.2. The second parameter is not used, however required, so its set to zero from the static class IntPtr.

Listing 4.2: EnumWindows function with specified callback.

```
EnumWindows(new EnumWindowsProc(EnumTheWindows), IntPtr.Zero);
```

The function EnumTheWindows that is specified within the prototype will be called from all of the running processes. Even if the documentations [1] specifies the following: "**Note** For Windows 8 and later, EnumWindows enumerates only top-level windows of desktop apps". Indicating that the callback function will only be passed to the top-level applications present at desktop. However the the callback is triggered by over 100 processes where many are services included. It could be that this is the correct behavior, thus it is clearly not easy to understand the documentation.

The workaround to filter out the applications that are open and present at desktop is done with the help of two additional Windows functions imported from the User32.dll library, namely the IsWindowVisible and the GetWindowTextLength functions as seen in listing 4.3. They both take hWnd as a argument that is obtained by the parameter

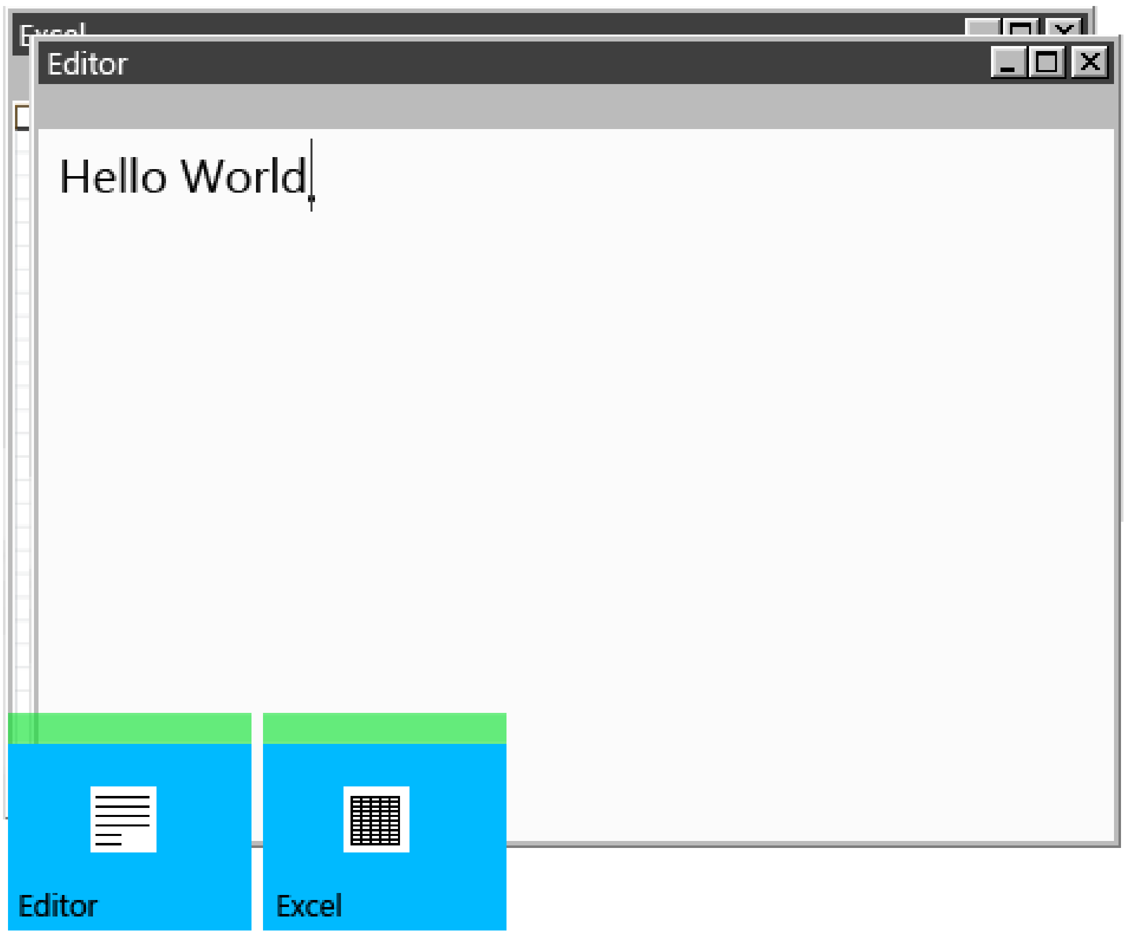


Figure 4.4: Illustration of navigating top-level applications showing two menu items.

of the EnumTheWindows function. The hWnd is a handle to the forms present and is required by all Windows applications in order for the Windows API to show, minimize, maximize or resize the application. The hWnd handle can be viewed as a reference to the process of the application and that all of the forms within the applications have its own hWnd. By passing the hWnd handle to the isWindowVisible function, the function will return true if the form is visible or false if not. The isWindowVisible function will filter a lot of the processes that are not visible. However some applications does still exist with multiple forms and therefore multiple hWnd handles to the same application. The function GetWindowTextLength is therefore used to filter out the last uninteresting forms, the forms without a title. The GetWindowTextLength function gets the text length of the title in the form. If the form does not have a title, indicated by a text length of zero. The form is probably not a top-level application either.

Further the prototype extracts key information about the remaining applications as seen in listing 4.3. The key information is later used to represent the application in the prototype, such as the title and icon of the application. To extract the title of the form, the function GetWindowText from the User32.dll library is used. Similar to the GetWindowsTextLength the GetWindowText will return the title of the form instead of the text

Listing 4.3: EnumTheWindows is the callback function passed onto EnumWindows.

```
private bool EnumTheWindows(IntPtr hWnd, IntPtr lParam)
{
    int size = GetWindowTextLength(hWnd);
    if (size++ > 0 && IsWindowVisible(hWnd))
    {
        WindowInfo wi = new WindowInfo();

        StringBuilder sb = new StringBuilder(size);
        GetWindowText(hWnd, sb, size);
        wi.name = sb.ToString();

        uint id = GetWindowProcessId(hWnd);
        Process p = Process.GetProcessById((int)id);
        wi.process = p;

        Icon ico = Icon.ExtractAssociatedIcon(p.MainModule.FileName);
        wi.image = ico.ToBitmap();

        /*
        * Dont want to show process for explorer
        * or DesktopEye, so we skip them.
        */
        switch (wi.process.ProcessName)
        {
            case "explorer":
                return true;
            case "DesktopEye.vshost":
                return true;
            case "DesktopEye":
                return true;
            case "EyeTribe":
                return true;
        }

        listWindowInfo.Add(wi);
    }
    return true;
}
```

length. In addition the GetWindowsText, the function GetWindowsTextLenght is used as a third parameter to the function GetWindowsText as a safeguard, setting the maximum allowed characters to read.

By having the hWnd handle, the prototype uses the Windows API function GetWindow-ProcessId to obtain the ID of the process tied to the form. Thereafter the ID is used with Process.GetProcessById, to obtain the process of the application. Further the process is used to access the file name, which thereafter is used with Icon.ExtractAssosiatedIcon to obtain the icon of the application.

A new WindowInfo object is instantiated to manage the details of each application. The class contains three public variables that holds the title as text, the process as process and the application icon as image. By representing each applications as a WindowInfo object makes it easier to manage the applications by only utilizing one array that holds all of the WindowInfo objects. It is interesting to point out that the order of the WindowInfo objects are added to the array, are stored chronologically after the z-order of top-level applications. To clarify the first WindowInfo object added, is the top-most application or the application with the highest z-order. This is the basis for reflecting the same behavior as the keyboard shortcut for tabbing between applications. This meaning that the first menu item displayed starting from the left to right will be the current top-most application. This promotes the desired behavior, that if the trigger area where ever to be accidentally selected, it would initially select the current top-most application. Promoting easy error correction from accidental selections.

To represent the applications within the prototype, a UserControl is created to design a generic representation of the applications. The UserControl has a rectangle shape with a picture box and a label to show the application icon and title as illustrated in figure 4.4. By mapping the WindowInfo object to the UserControl simplifies the structural overhead. Listing 4.4 of the buildControlls method show how the WindowInfo object is tied with the UserControl ItemView.

The UserControl uses a callback to update the prototype with the currently selected process each time a UserControl is highlighted. The prototype needs to know the last known selected process in order to show the application when the gaze exits the menu. This trick is used as the prototype have no idea of the selected process when the gaze exits the menu and since the prototype is always updated with the last selected application, the prototype can assume that this is the desired application to be shown.

Within the prototype it is the menu form that handles all of the gaze data and logic. The UserControl exposes therefore the method setAppFocus to change the background of the UserControls as seen in listing 4.5. When the public method setAppFocus is set true, will change the background of the UserControl and also trigger the callback to update the prototype with the current selected process.

The callback method updateSelectedProcess does only update the selected process as the prototype only needs the process to determine which of the applications that should be brought to front. The function setFormVisibility is the highest method in hierarchy in the prototype and is managed by the gaze logic as seen in listing 4.7. When the gaze enters the boundaries of the trigger area the setFormVisibility is set to true and when the gaze exits the horizontal boundary, the setFormVisibility is set to false. When the gaze triggers the

Listing 4.4: The function buildControlls that instantiates all of the ItemView controlles.

```
private void buildControlls(ArrayList listWindowInfo)
{
    for (int i = 0; i < listWindowInfo.Count; i++)
    {
        WindowInfo wi = (WindowInfo)listWindowInfo[i];

        ItemView iv = new ItemView();
        iv.setWindowInfo(wi);
        iv.Top = screenHeight - iv.Height - 5;
        iv.Left = i * (iv.Width + 5) + 5;
        iv.setCallback(this);
        this.Controls.Add(iv);
        listItemViews.Add(iv);
    }
}
```

Listing 4.5: The function setAppFocus is public and exposed in the ItemView object.

```
public void setCallback(frmMenu frmMenu)
{
    this.frmMenu = frmMenu;
}

public void setAppFocus(bool focus)
{
    if (focus) {
        this.BackColor = System.Drawing.SystemColors.Highlight;
        frmMenu.updateSelectedProcess(process);
    }
    else
    {
        this.BackColor = System.Drawing.SystemColors.ActiveCaption;
    }
}
```

setFormVisability to show the menu, the prototype will get all of the top-level applications through the EnumWindows call, assuring that the prototype is always updated with current available top-level applications. Moreover when the setFormVisability is set to false, closing the menu and show the selected application, the prototype will use the functions ShowWindow and the SetForegroundWindow. Both of the functions are imported from the User32.dll library.

The ShowWindow requires the hWnd handle as first parameter and is obtained by asking the process of its main window handle. The second parameter specifies how the application should be shown. Currently the prototype specifies the second parameter to be 3, which means the application will be maximized. From the documentation [5] the ShowWindows function can specify multiple different states such as, 6 to minimize, 10 to show default, 0 to hide and so on. This makes the prototype to enforce the application to maximize. While ideally it should be shown as the last know state that was not minimized. In addition if the application is maximized and the ShowWindow function is passed 10 (default) as the second parameter, the application would restore to its previous size before the state was changed to maximize. A key decision after fiddling with the challenge for a while, were to force the applications to always be maximized. This decision also afflicted how the workflow where designed as discussed in Chapter 5.4.2.

As also mentioned the prototype calls the SetForegroundWindow when the menu is closed. This is necessarily as the ShowWindow function specifies the show state of the window, but does not specify which z-order it should have. Result in potentially maximizing the selected application behind the last used application. The second function SetForegroundWindow is therefore used to bring the application to front. The documentation for the SetForegroundWindow [4] function gives the following description of the function; "Brings the thread that created the specified window into the foreground and activates the window. Keyboard input is directed to the window, and various visual cues are changed for the user". The function SetForegroundWindow will therefore bring the application to the foreground and set the window active.

The prototype is required to use both of the functions, ShowWindow and SetForegroundWindow to ensure that the application is to appear in the foreground. For example, if the SetForegroundWindow function where to be called on a window that is minimized, it would become active, however it would not appear on screen.

4.2.4 ListnerLoop Frame Cycle

The implementation of the Eye Tribe eye tracker is covered in Chapter 4.1. However the ListnerLoop from the EyeTribe implementation is used to trigger different actions in the prototype, based on the position of the gaze.

Within the ListenerLoop the gaze is parsed from the JSON structure into integer screen coordinates. Further the ListenerLoop checks if the gaze coordinates are within the specified boundaries as illustrated in figure 4.1 and figure 4.4. Listing 4.7 shows the three different states the gaze may trigger. The prototype takes into consideration that the ListenerLoop may be updated up to 60 times each second, stressing the importance of removing any unnecessarily logic in the ListenerLoop. Also since the ListnerLoop runs on

Listing 4.6: The setFormVisability is the heartbeat for building the menu structure managing which of the forms to be displayed as well as showing the desired application when the menu is closed.

```
private void setFormVisability(bool visible)
{
    if (visible)
    {
        // Need to find active windows and build menu before we show.
        getWindowsAndBuild();
        this.Opacity = 1;
        formVisible = true;
        fta.Hide();
    }
    else
    {
        // Move selected application to top on exit.
        if (selectedProcess != null)
        {
            // TODO: Should detect window state and not force window state
                to maximized.
            // http://msdn.microsoft.com/en-us/library/windows/desktop/
                ms633548(v=vs.85).aspx
            ShowWindow(selectedProcess.MainWindowHandle, 3);
            SetForegroundWindow(selectedProcess.MainWindowHandle);
        }
        this.Opacity = 0;
        formVisible = false;
        fta.Show();
    }

}
```

a separate thread, the method calls to the menu form are invoked in order to run on the main thread.

For the prototype to know which of the applications that are currently selected, the ListenerLoop will check all of the UserControls to check if the gaze has entered. If the gaze is caught within one of the UserControls, the method setAppFocusFromIndex is invoked with the position of the UserControl. The setAppFocusFromIndex as seen in listing 4.8, triggers the public method setAppFocus for the UserControl (ItemView).

Listing 4.7: Triggers implemented within the ListenerLoop().

```
if (formVisible)
{
    // Hide form if eyes exits on the y-axis.
    if (yOff < (screenHeight -200))
    {
        if (InvokeRequired)
        {
            this.Invoke(new Action(() => setFormVisability(false)));
        }
    }

    // Iterate applications and give focus as we look at them.
    for (int i = 0; i < listItemViews.Count; i++ )
    {
        ItemView iv = (ItemView)listItemViews[i];
        if (xOff > iv.Left && xOff < iv.Left + iv.Width)
        {
            if (yOff > iv.Top && yOff < iv.Top + iv.Height)
            {
                if (InvokeRequired)
                {
                    this.Invoke(new Action(() => setAppFocusFromIndex(i)));
                }
            }
        }
    }

}
else
{
    // Looking down in the left corner will trigger the menu to show.
    if (xOff < 200 && yOff > screenHeight -200)
    {
        if (InvokeRequired)
        {
            this.Invoke(new Action(() => setFormVisability(true)));
        }
    }
}
```

Listing 4.8: Method for setting the ItemView's focus.

```
private void setAppFocusFromIndex(int index)
{
    for (int i = 0; i < listItemViews.Count; i++)
    {
        ((ItemView)listItemViews[i]).setAppFocus(false);
    }
    ((ItemView)listItemViews[index]).setAppFocus(true);
}
```

4.3 Prototype 2: Gaze Directional Tab Order

Chapter 2.2 gives a glance at how Windows components are structured and how they can be design to accommodate the use of keyboard shortcuts. The use of tab stops is used to accommodate the use of keyboard shortcuts within applications for input fields. Figure 4.5 is an illustration of how an application with many input fields could construct the tab order where the first tab-stop starts in the upper left corner. Further the tab stops follows a horizontal line until the last field is reached, then it continues on the next line, following a horizontal pattern. This is the most common structure where sometimes little regard is given to the order the fields traversed. Sometimes the tab order is modified to accommodate the workflow of the application. For instance, it could be logical to traverse the columns in figure 4.5 along the vertical axis instead of along the horizontal axis. Figure 4.6 illustrates two possible tab orders by drawing a continuous line through all of the input fields.

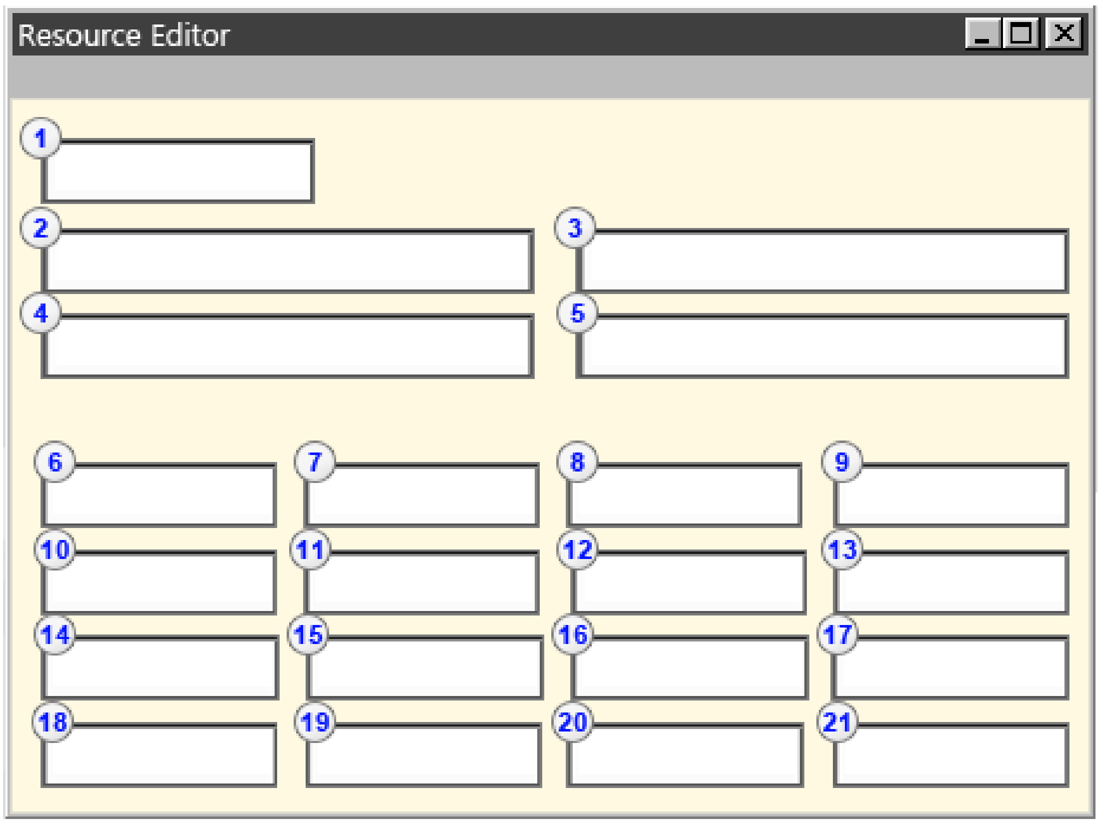
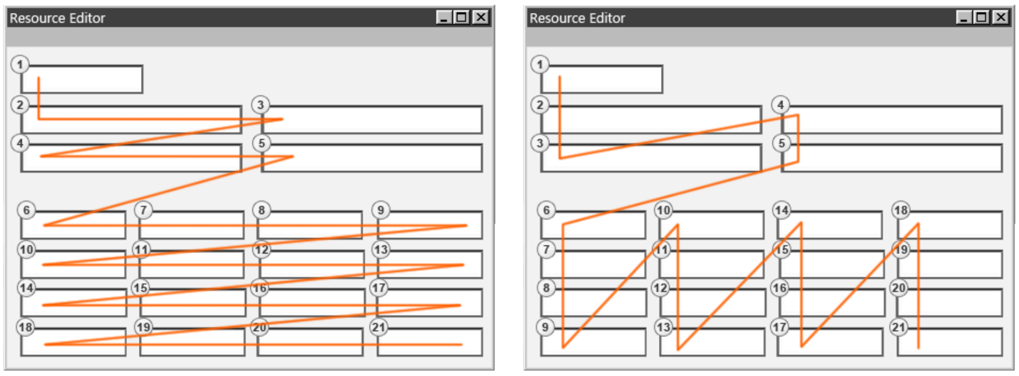


Figure 4.5: Illustration of typical tab order in a typical application. The tab order is specified by the numerical values.



(a) Horizontal pattern (b) Vertical pattern

Figure 4.6: The figures show two different concepts of implementing directional tab-order.

4.3.1 Prototype Design

Prototype 2 proposes the possibility of manipulating a fixed tab order by tabbing towards the point of gaze. For instance, if the cursor is located in the sixth input field in figure 4.5 and the user only wants to update the twentieth input field. The user would look at the twentieth input field, while starting to tab, the cursor could be traversing the input fields 14, 18, 19 before reaching the twentieth input field. If it were to follow the horizontal pattern as seen in figure 4.6(a), the cursor would traverse the input fields 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, and 19 before reaching the twentieth input field. Reducing the amount of step required to thirteen steps.

4.3.2 Gaze Pattern and Desired Usage

To navigate between input fields within the prototype, the user can look at the desired input field. While pressing tab multiple times, the cursor will traverse the shortest path of input fields towards the point of gaze. When pressing tab, the cursor will only traverse one input field at a time. Since the input fields are potentially too small for optimal selection [30], making it hard for the eye tracker to determine the actual point of interest. The prototype allows the gaze to compensate by looking slightly beneath or slightly to the left of the input field while pressing tab until the cursor reaches the desired field. A potential gaze pattern when navigating input fields in the prototype is illustrated in figure 4.7.

4.3.3 Technical Implementation of the Prototype

Prototype 2 is built by creating a new Windows Forms Application in Visual Studio. The application consists of one form element where multiple input boxes are placed around. Figure 4.8 shows the prototype and how the input boxes are structured. The input field in the top left corner holds the order number. Beneath the order number are some labels with

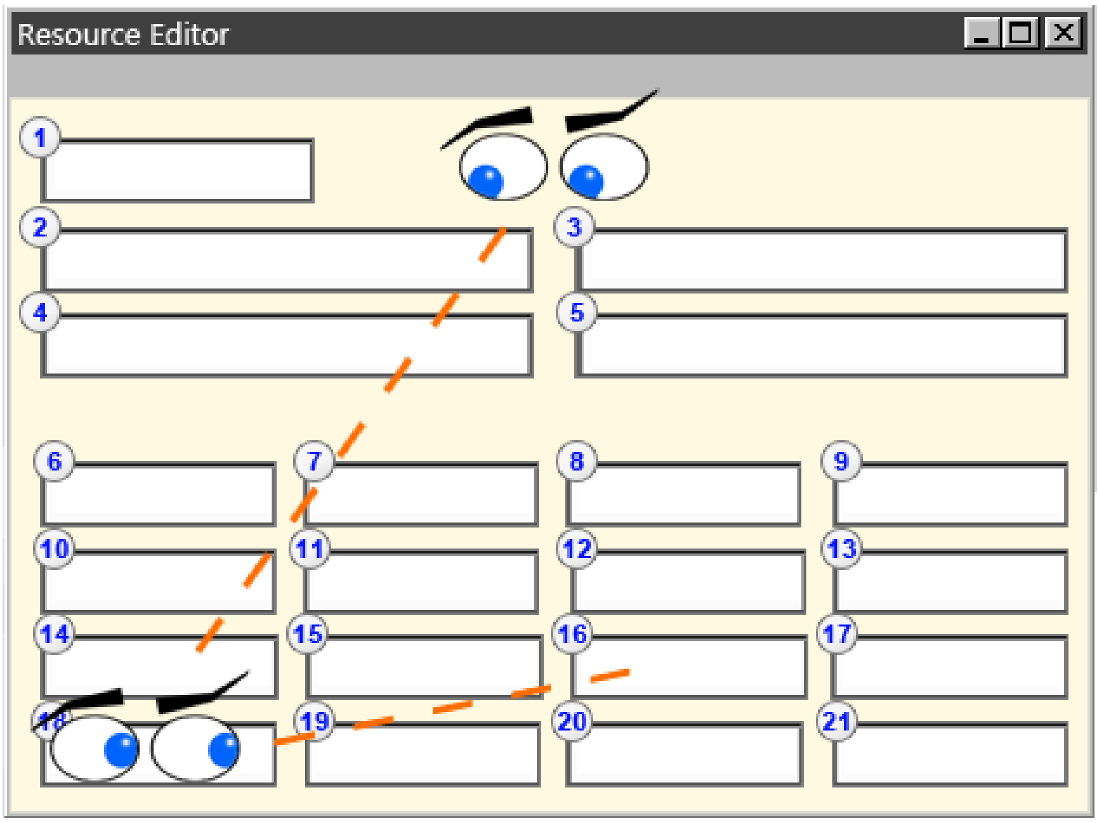
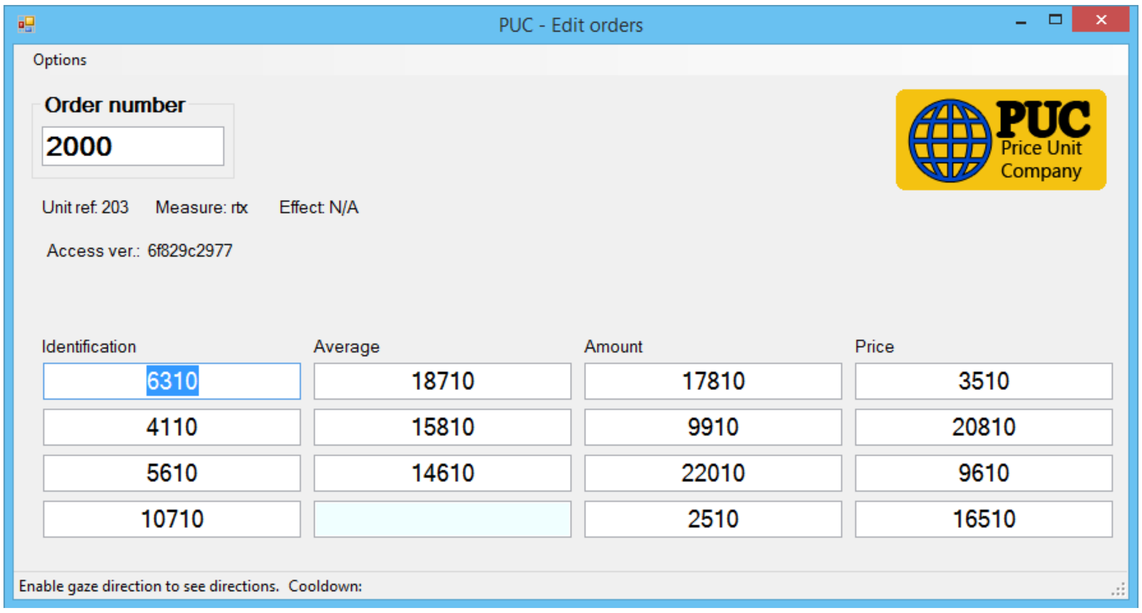


Figure 4.7: Gaze pattern for navigating to input field 18 and 16 in Prototype 2.

fixed information. The labels don't serve any purpose with the exception of application cosmetics. The input boxes beneath the labels are structured as a 4 x 4 grid. The columns are labeled as identification, average, amount and price where each row represents a product.

The design of the tab order in the prototype is seen in figure 4.9 where it follows the concept of horizontal tab order alignment. It is argued that the tab order should be horizontal as a product is represented for each row. If you were to update a product, it would seem reasonable to go through all of its details before moving onto the next product. Only the input boxes with a tab order between 0 and 15, both included have specified the tab stop. Implying that the fields within the orange rectangle in figure 4.9 will be traversed horizontally when pressing tab. All other elements does not obtain the property of TabStop and is therefore excluded from the tab cycle.

The text fields are structured by creating a dictionary of text boxes where the first text field is assigned the index of 11 (read one one). If 11 is divided in half in the sense of being a string and not a number, the first half would be 1 and the second half would be 1. Creating an illusion of a two-dimensional array, where the first number represents the row and the second representing the column in the grid. See listing 4.9 for the structure of the text fields. The text fields are also assigned event handlers for focus and key down. The function called when the text field obtains focus, updates the application with the



Order number
2000

Unit ref: 203 Measure: rtx Effect: N/A

Access ver.: 6f829c2977

Identification	Average	Amount	Price
6310	18710	17810	3510
4110	15810	9910	20810
5610	14610	22010	9610
10710		2510	16510

Enable gaze direction to see directions. Cooldown:

Figure 4.8: Illustration of Prototype 2.

selected text field to be the most recent selected text field. The function assigned for the event KeyDown is seen in listing 4.11 and is called when the text field is selected and keys are pressed. This is to detect when enter is pressed to confirm order and generate next order. The enter key is used instead of using a button that would require interaction with the mouse.

Listing 4.9: Initializing of the text fields.

```
txtGridElms = new Dictionary<int, TextBox>();
txtGridElms[11] = txtGrid11;
txtGridElms[12] = txtGrid12;
...
txtGridElms[43] = txtGrid43;
txtGridElms[44] = txtGrid44;

foreach (int key in txtGridElms.Keys)
{
  txtGridElms[key].GotFocus += tb_GotFocus;
  txtGridElms[key].KeyDown += GridTxt_KeyPress;
}
```

To determine the direction of the gaze, an algorithm was created, see appendix D. The algorithm uses the position of the currently selected input field and the gaze coordinates to determine if the direction of the gaze is to the left, top, right, bottom or none. None indicating that the gaze is looking at the current selected input field and the remaining directions indicate the direction of the gaze from the selected input field. The algorithm runs from a timer with a interval of 250 milliseconds to reduce the possibility of a bottleneck. Instead of running the algorithm from the ListenerLoop, that runs multiple times faster than required, the ListenerLoop is only concerned with updating the most recent coordinates of the gaze.

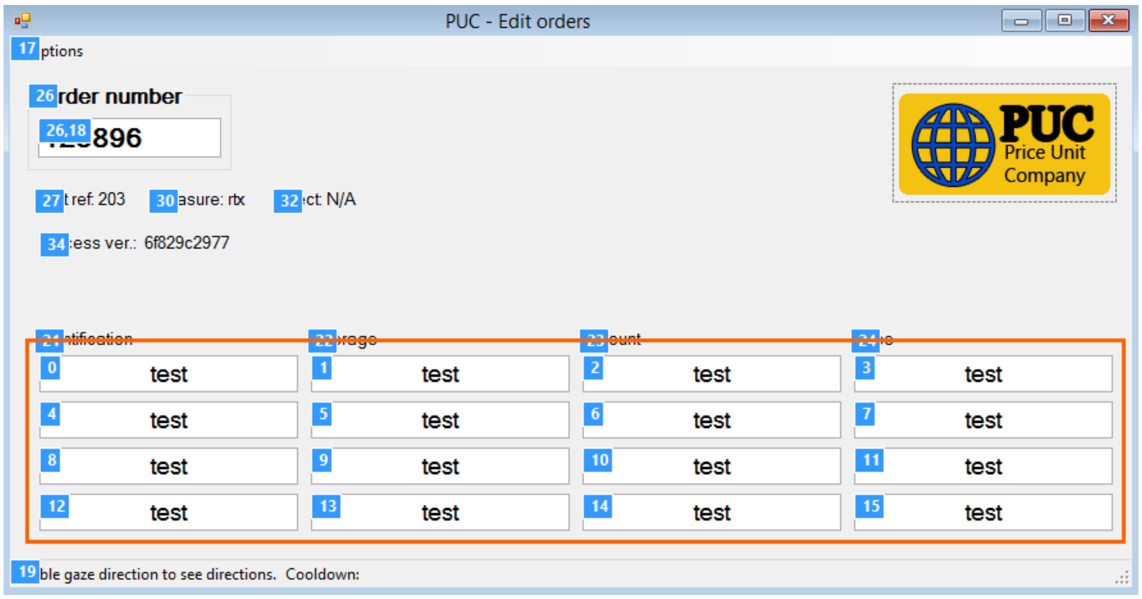


Figure 4.9: Prototype 2 in Visual Studio with the tab order editor enabled.

From the algorithm that is updated every quarter of a second, updates the application with the gaze direction so that the gaze direction is always available within the scope of the application at all time. So that when the tab key is pressed, the application will already be aware of the gaze direction. To further determine which text field that is next depending on the direction. The text fields where structured as a two dimensional array to resemble a grid system. The first product entry for identification is represented within the array with the index 1,1 and the next line for identification have the index of 2,1. From this structure the application is only concerned with keeping track of the index for the selected text field. Allowing the application to increment the first index to move downwards in the grid or increment the second index to move to the right in the grid. Decreasing the indexes will reverse the directions. Listing 4.10 shows the control flow for the increments and decreasing of the index for determining the next text field. In addition to setting the focus of the text field, the input is also selected.

The prototype overrides the applications OnKeyDown and ProcessTabKey functions to manipulate the tab order from the gaze direction. The function OnKeyDown is used to detect keyboard input for the application and when the tab key is pressed. The ProcessTabKey function is used to override the default tab behavior of the tab key, since it's not desirable to use the tab order when acting on the gaze direction. In addition to using the tab key, the plus key is also made available to use as a second smart tab key. Through the KeyPress event for the text fields as seen in listing 4.11, the function calls the SmartTabAction function similar as the tab key would be pressed. By setting the SuppressKeyPress to true for the event, will hinder the plus sign to be appended to the text field.

The application life cycle is created by drawing random numbers in the text fields with one exception where one of the text fields are empty. This is achieved by the function drawRandomFieldRed as seen in listing 4.12. The function is run by default upon startup to initialize the data and called each time enter is pressed as seen in listing 4.11. The

Listing 4.10: Method for handling the tab direction from gaze.

```
private void SmartTabAction()
{
    if (gazeDirection == GazeDirection.NONE || gazeDirection ==
        GazeDirection.TWODIRECTIONS || gazeDirection == GazeDirection.
        THRESHOLDMAX)
    {
        gazeDirection = lastKnowGazeDirection;
    }

    int numCol = 4;
    int numRow = 4;

    foreach (int key in txtGridElms.Keys)
    {
        if (txtGridElms[key] == focusedTextBox)
        {
            double selRow = Char.GetNumericValue((key.ToString()[0]));
            double selCol = Char.GetNumericValue((key.ToString()[1]));

            switch (gazeDirection)
            {
                case GazeDirection.UP:
                    selRow--;
                    break;
                case GazeDirection.DOWN:
                    selRow++;
                    break;
                case GazeDirection.LEFT:
                    selCol--;
                    break;
                case GazeDirection.RIGHT:
                    selCol++;
                    break;
                default:
                    break;
            }

            if ((selCol > 0 && selCol <= numCol) && (selRow > 0 && selRow
                <= numRow))
            {
                foreach (int keyn in txtGridElms.Keys)
                {
                    if (selRow + "" + selCol == keyn.ToString())
                    {
                        txtGridElms[keyn].Focus();
                        txtGridElms[keyn].SelectAll();
                    }
                }

            }
            break;
        }

    }
}
```

Listing 4.11: Function handles key press events for text fields.

```
private void GridTxt_KeyPress(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Return)
    {
        // Check and generate new order.
        drawRandomFieldRed();
        e.SuppressKeyPress = true;
    }

    if (e.KeyCode == Keys.Add)
    {
        SmartTabAction();
        e.SuppressKeyPress = true;
    }
}
```

function is concerned with checking that all of the fields are filled out, as it is expected by the user to fill in the blank spots. The numbers generated as fill have been restricted to a number between 1111 and 9999, to not supersede digits with a greater length than four, nor less. From the necessity of creating a relationship between the digital world and the paper flow, the order numbers are specified within a array of allowed order numbers. Thereafter one of the allowed order numbers are randomly picked each time.

To allow the prototype to be used with and without eye tracking technology, the application allows toggling of the option from the options menu. By enabling the SmartTab in the menu options as seen in figure 4.10, will allow the prototype to use gaze data. By not having SmartTab enabled promotes the default behavior of the horizontal tab order. Also added to the menu options is the possibility to show the updated gaze direction, the point of gaze and toggle the visibility of the status strip.

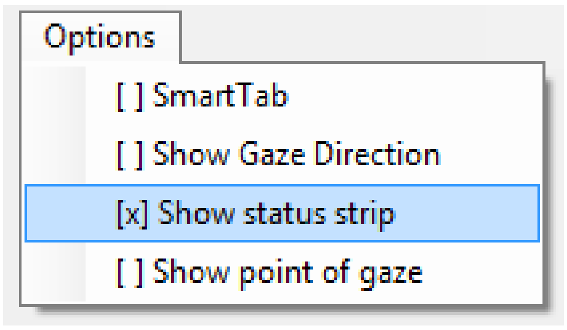


Figure 4.10: Tool strip and menu options for the prototype.

Listing 4.12: Function for checking and generating new orders.

```
private int[] allowedOrders = { 2000, 2023, 2213, 2491, 3266, 3270 };
private void drawRandomFieldRed()
{

    foreach (int key in txtGridElms.Keys)
    {
        if (txtGridElms[key].Text.Equals(""))
        {
            txtGridElms[key].BackColor = Color.Red;
            return;
        }
    }

    Random rand = new Random();
    foreach (int key in txtGridElms.Keys)
    {
        txtGridElms[key].BackColor = Color.White;
    }

    // fake order
    int getFakeOrder = rand.Next(allowedOrders.Length);
    txtOrderNumber.Text = "" + allowedOrders[getFakeOrder];


    string randomString = (rand.Next(4) + 1) + "" + (rand.Next(4) + 1);
    foreach (int key in txtGridElms.Keys)
    {
        txtGridElms[key].Text = "" + (rand.Next(8888) + 1111);
        if (key.ToString().Equals( randomString ))
        {
            txtGridElms[key].BackColor = Color.Azure;
            txtGridElms[key].Text = "";
        }
    }
}
```

Chapter 5

Results

5.1 Domain Knowledge and Justification

To further understand and establish a mild form of domain knowledge of business applications and how they are utilized by employees, I sought out a company in order to observe current trends of computer interaction. The means of understanding business applications must extend to the generalization of the different applications. To emphasis that to specific applications should be disregarded if it concerns a limited amount of employees across practices. One example of application that has a wide area of use and concerns a many employees is Microsoft Outlock. The email application is used widely among different business practices and is therefore highly suitable to be considered for eye tracking support. Some knowledge of employee's technical skills can also provide useful when developing interaction styles.

I approached the company PostNord where I first arranged for a meet with a family member June. I was allowed to observe record and ask questions during a half day. The material gathered is confidential as it may contain personal material and business insight. Due to this discretion and relevance I reflect on the overall usage of the computer and the interaction, which is also the most relevant for exploring the possibility for gaze supported interfaces. Therefore the material is processed confidential and video recordings where destroyed when the thesis was submitted.

The meet gave valuable insight and confirmation of current use of applications. The usual pattern of digital flow within businesses consists of data entry, data update and data finalization. Within the business this flow is known as the creation of a new order. Data entries to update provide documentation of the process and work being carried out. When the task is completed, the order is updated and marked as fulfilled. The last step is to get paid by sending billings for the fulfilled tasks. The different parts of the process are managed by different employee's where the consistency is the same application used throughout the process.

In my first visit with June I learned about her tasks and the different applications that she combined during her work day. One aspect was when she used lists in Excel to lookup values to use in a different application. Also during work a random but steady flow of emails where received and sent, often interrupting the workflow. Observing June I learned that she mastered the numpad on the keyboard. When I asked about it, she explained that numbers are used widely for references and representation, mentioning order number, amounts, weights, time and dates throughout the workday. Over many years of experience it occurs natural to use the numpad fluidly without even looking at the keys. Figure 5.1 shows June at her workspace during observations.

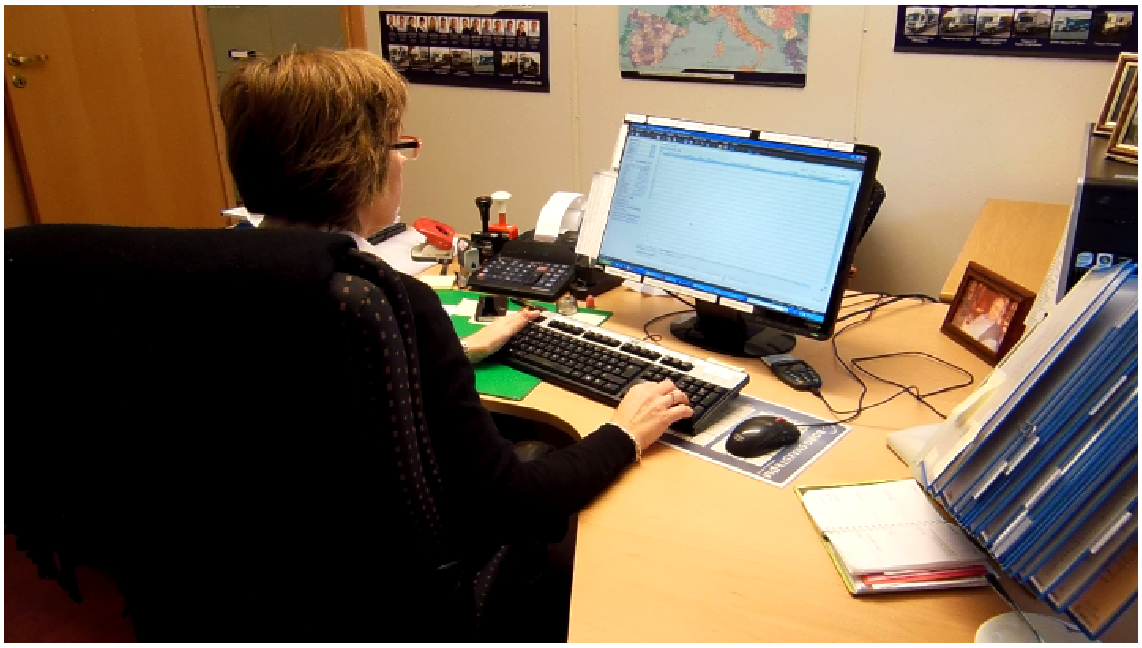


Figure 5.1: Photo of June during employee observations. Printed with permission.

After my meeting with June I reflected on the information gathered. Through the observation of June I learned a lot about how June uses the computer and what role the computer plays during June's workday. What I experienced during my observation of June also inspired to conduct more observations from different employees with different work routines. By interviewing additional employees in order to gather additional confirmation and establish a more rich understanding of employee's technical skills and how they interact with the different applications during a workday.

From my interview with June a verbal agreement was enough. In order to conduct wider observations and interviews from additional employees, a written agreement was stipulated to protect the firm and the employees. The written agreement did only concern the project holder and the employee, however the management required approving the agreement before any employees could agree to be interviewed. The agreement allowed me to observe the employees as they worked. During observation I was also allowed to ask questions about the work and what they did. The agreement did also emphasis that I was not there to evaluate the employee nor give feedback on the employee to the management.

Before proceeding to interview additional employees, a interview guide was created in a attempt to establish consistency. The guide is attached as appendix E. The guide was first intended as a questionnaire, however as I prepared for the interview, it could seem like the employees would evaluate them self. This was not my intention so I tried to use the form as a guide rather than a questionnaire. The first question in the guide looks at level of expertise when it comes to using the computer as a tool. This information is used to see if the employees have learned to fully utilize the shortcuts and efficiently use the applications and operating system. Question two and three was asked to look at the most time consuming and tedious tasks. This is to highlight challenges and motivate the possibility of substituting those tasks with eye tracking. Also trying to look at tasks where the employees considered themselves efficient and in a good workflow. With regards to don't fix anything that already works great. Question five also loops back towards question two and three and serves as a purpose to detect tasks that potentially could be improved. Question four and a sub question of four is based on the assumption that employees constantly have a paper flow to accommodate the digital flow of information. However some employees have greater paper flow than others and it could be other things than paper that requires the attention and work of the hands, forcing the arms to switch focus between the keyboard and other sources. Where eye tracking potentially can free up a hand and enable eye tracking to accommodate a workflow suited for multitasking.

5.1.1 Observations from PostNord

From observing the four employees I learned that the computer expertise did vary and very much affected by the work routines. It is safe to assume that most of the employees master the numpad due to heavy usage.. It surprised me that not all where familiar with neither tabbing between applications nor even familiar with the concept of tabbing between fields. When pointing it out in a particular observation I quickly learned that the application in question was lacking a reasonable tabular order. Suggesting that the tailored application does not fully comply with Microsoft's design guidelines as addressed in the background chapter on page 13.

When looking at the most time-consuming tasks across the employees, they shared the common factor of paper flow tied to their routines. Employees working with invoices are required to check the physical invoice in order to validate the electronic invoice. This task is considered to be very time-consuming as the company handles many invoices and each individual invoice needs to be marked by hand. Not only invoices require hands on attention. Employees with in the firm working on an hourly basis are required to submit weekly paper work on working hours. Besides the paper flow they barely shared the same opinion of tasks that where time-consuming. One employee considered the amount of time on phone to be the most time-consuming task. Using the information from the computer to negotiate over the phone in order to distribute workload. The use of phone in the particular matter was also considered by the employee where he was the most efficient.

During the observations I was also looking for when the employees where multitasking or combining other tools when using the computer. As already mentioned the phone was heavily used among one of the employees in order to distribute and assign work. Also the combination of paper and computer seems critical as numbers where gathered from paper and inserted into the computer and vice versa by writing down information on paper from the computer. Sometimes multiple applications where utilized to extract information and update other applications. One example given was information gathered from e-mails and inserted in a different application.

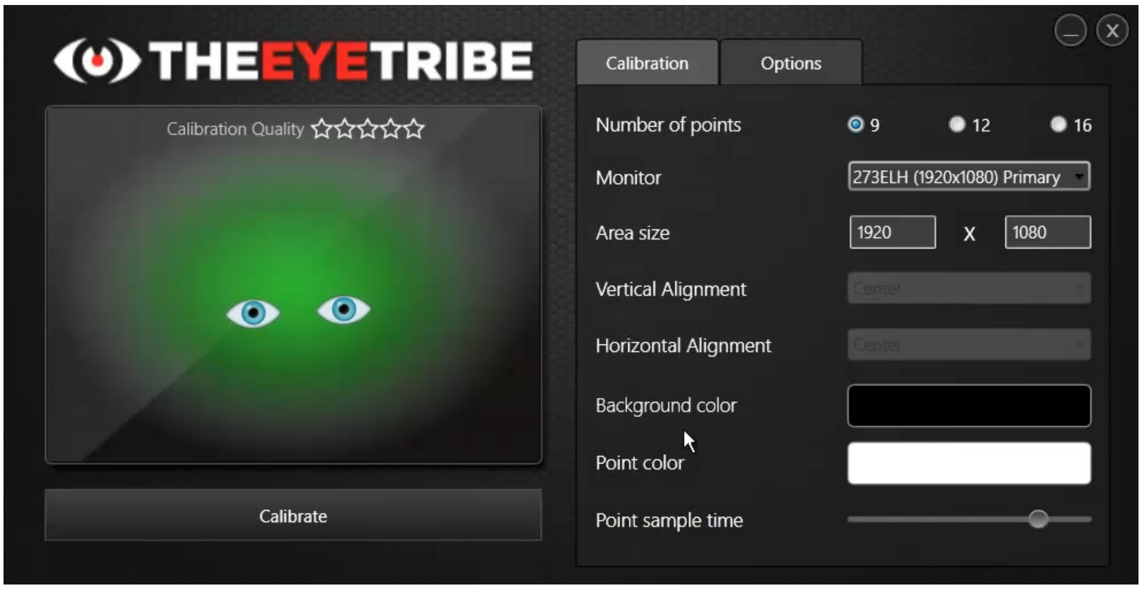
In addition to the observations I also challenged June to explain how she would interact with the computer, given that the computer knew where she was looking and somewhat knew her intention. She points towards screen and says, ”It would be nice if the screen saw what I was looking at outside of the monitor. Or if that’s not possible, maybe I could look at the field on the screen and say DK23451”. Were DK23451 where an example of a truck identification number. When trying to explain how it would work, she tried to use the existing interface and add logic onto that.

Considering the overall information gathered from the observations, it can be hard to highlight all the aspects of the information and how it affects the decision making when it comes to creating eye tracking prototypes. Some ideas that have emerged after and maybe during the observations is the idea of freeing up hands so that they can be used to accommodate other tasks. For instance one arm can freely handle paper while the other hand uses the numpad for data entry. The eyes would navigate the data entry point, making it unnecessarily to use the mouse.

5.2 Setup of the Eye Tracker

In order for the prototype to start receive gaze data, the EyeTribe server was started. The server runs in Command Prompt and does not require any further care once it is running. Further launching EyeTribe UI to calibrate the eye tracker and getting it ready for use. As seen in figure 5.2, the left side of the EyeTribe UI is used to locate the users eyes. It shows the users eyes in relation to the eye trackers position. If the eyes are not present, the eye tracker can be adjusted until the eyes appear. The background color for the eyes will change from red to green, indicating the most optimal position for the eyes to be tracked. A full green background indicates that the EyeTribe UI is ready to proceed with best possible grounds for acquiring best possible accuracy. A red background goes without saying and there is no point in proceeding with the calibration.

Proceeding with the calibration, a nine point calibration was used. This is also the minimum amount of points allowed in the EyeTribe UI. The benefit of only using nine points calibration is that the calibration process is faster. The reason for using additional calibration points is to further increase the accuracy of the eye tracking, however nine points have proven to be sufficient for proceeding to use the prototypes. When pressing calibrate in the EyeTribe UI, see figure 5.2, the whole screen turns black. Thereafter nine points will appear on different parts of the screen. The user is instructed to look at each point as they appear. Once the procedure completes, the user is presented with the quality of the results, indicated with stars ranging from one to five, where five stars is the best obtained accuracy. Once satisfied with the calibration, the EyeTribe UI is no longer necessarily and can be closed. The server is calibrated for the current user and is ready for use. When proceeding with a new user, the EyeTribe UI must be started and proceed with a calibration to update the old calibration. The server is updated with the latest calibration and will only work for the latest calibrated user.



Calibration Quality ☆☆☆☆☆

Number of points [✓ 9] 12 16

Monitor 273ELH (1920x1080) Primary

Area size 1920 X 1080

Vertical Alignment Center

Horizontal Alignment Center

Background color

Point color

Point sample time

Figure 5.2: Screen capture from the EyeTribe UI.

5.3 Lab Environment

In order to test the prototypes, a lab at Østfold University College was utilized. The room is located in the basement beneath ground level. The room does not have any windows in the room, excluding the possibility for sunlight as a disruptive factor for multiple reflections in the face as discussed in Chapter 2.1.7. Multiple ceiling light illuminated the entire room, stimulating a acceptable contrast for good readability on paper, following the guidelines on a digital workplace from the Norwegian Labour Inspection Authority [13]. Figure 5.3 is a picture of the laboratory ready to accommodate a tester. Figure 5.3 shows the lab where there is a table, a chair and on the table is a computer screen, beneath the screen is the eye tracker located, as well as the keyboard and mouse. Also on the table to the right of the mouse is the fictitious paper flow created.

A camera is mounted a few feet behind the table. The camera is angled to see the screen and the testers arms. This angle captures what happens on screen and how the tester interacts with the keyboard and mouse. The observer is placed to the far end right of the table. This position allows the observer to approach in order to instruct or assist the tester on the tasks at hand and thereafter contract to the observer position. Also the observers view angle can reassemble the camera view angle where it's possible to observe the user, as well as the screen. Figure 5.4 is a blueprint of the test environment to enlighten the overall structure of the lab.

5.4 Test Environment for Testing Prototype 1

The lab environment is previously described in Chapter 5.3 is concern with describing the physical environment and overall circumstances where the prototype where tested. This

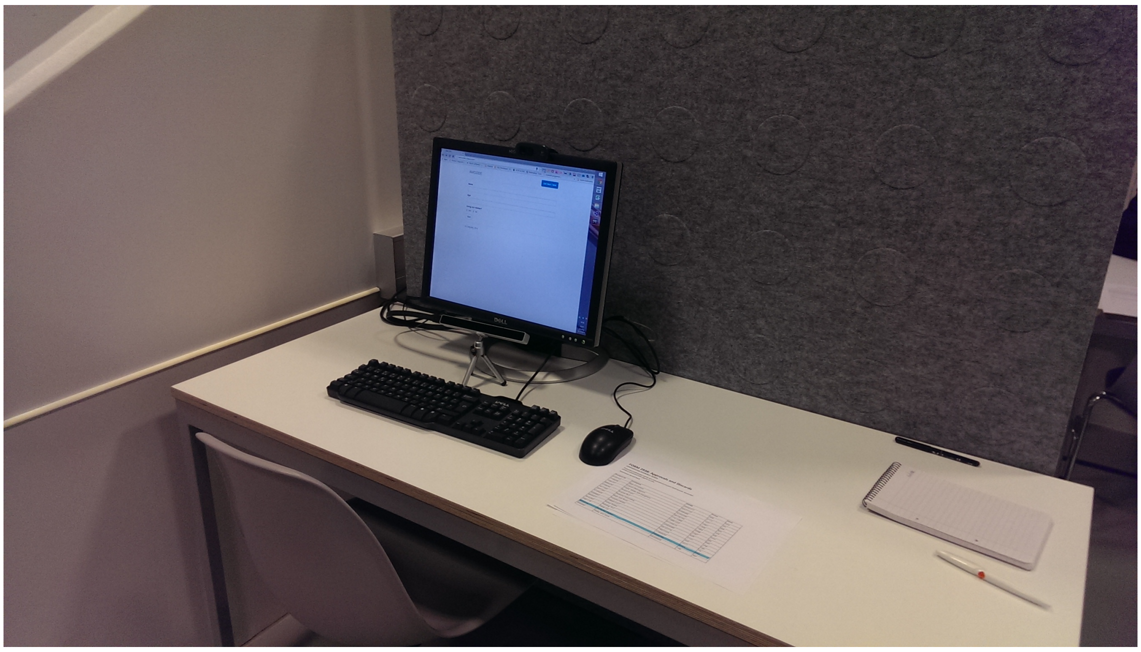


Figure 5.3: Photo of the lab.

section focuses on describing the procedure of how the prototype where tested and how the workflow where design to resemble a possible realistic workflow.

5.4.1 Test Design

The prototype is design to solve a problem of navigating between top-level applications. In order to test the prototype as intended, a basis of a potential workflow is needed. A workflow that forces the tester to switch between multiple applications or else there would not be any use for the prototype. To form a potential workflow, we want to create a scenario that reassembles a possible real life workflow. The workflow is fully described in Chapter 5.4.2. Further arguing the need to measure the prototype and not the testers, we still want to establish a baseline and understanding of how the user would perform under normal circumstances without eye tracking support. Following the methodology of within-subject design, each tester will be tested with and without the prototype in the same workflow. A Latin square is used to determine when the prototype is utilized first or last to reduce carryover effects.

A Latin square is a *n x n* array filled with different combination of testers and conditions. With six participating testers and two conditions, forms a 6 x 2 array as seen in table 5.1. Since there are only two different conditions, the array is only concerned with swapping the first and second condition for each tester.

Before starting the test, each participant is given a consent form explaining that the results obtained are used to evaluate the prototype and not the individual. Further consenting to the use of a camera recordings the tester and screen during the test. If photos from the recordings are to be used as illustration in the report, further approval is required

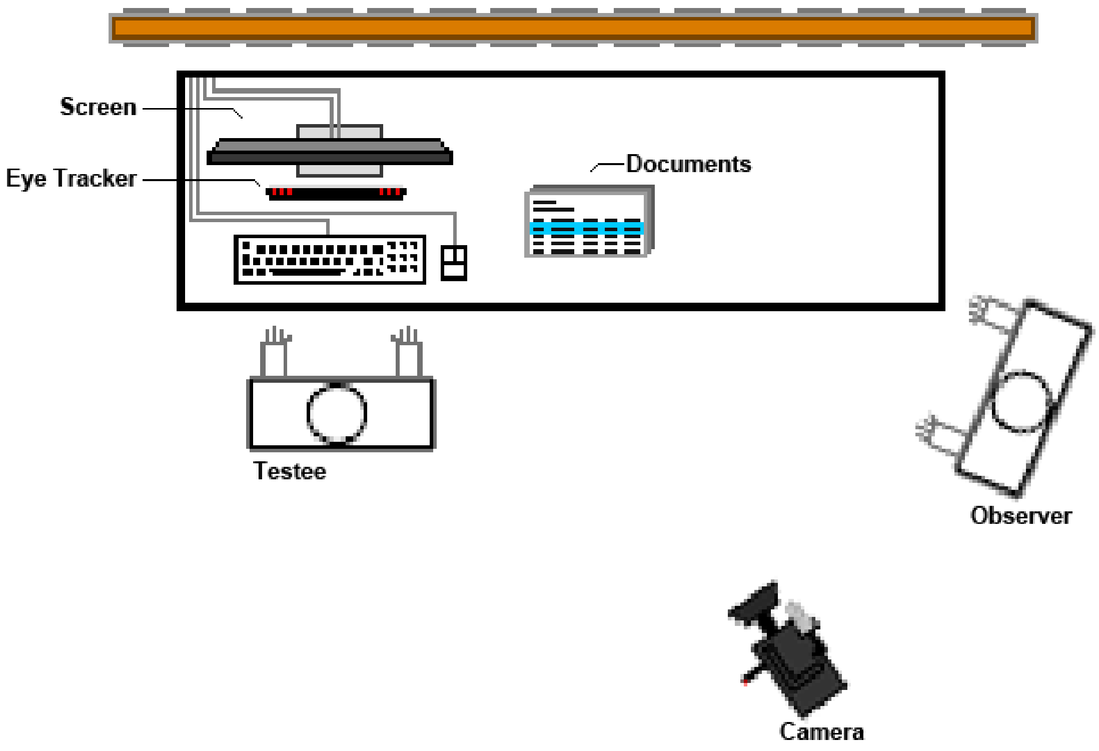


Figure 5.4: Illustration of lab environment.

from the tester. Thereafter the tester is given instructions on how to complete the work as described in Workflow design. For the free condition, no limitations are given towards how the tester chooses to solve the problem. By not instructing the tester how to navigate between top-level applications, the tester will choose the most natural way of navigating the top-level applications, where expected outcomes are either with keyboard navigation or mouse navigation. When the prototype is tested, the participant is instructed to use the prototype to navigate between top-level applications instead of personal preferences. Before the test can continue with the use of the prototype, a calibration of the eye tracker is required. The calibration procedure is described in Chapter 5.2. When the calibration is over, the tester is allowed to play around within the calibration software by looking at the different calibration points. This is to assure that the eye tracker was successfully calibrated and to reduce the overwhelming effect of using a eye tracker for the first time. Thereafter the tester is instructed on how to use the prototype and perform some exercises by cycling through the different top-level applications. Before proceeding with the second test condition, a short break of 1-2 minutes where given to reduce tension if any. When both conditions where completed, the tester was interviewed, mainly on reflections from the use of the prototype.

Tester	Start Condition	Second Condition
1	Keyboard or Mouse	Eye Tracking
2	Eye tracking	Keyboard or Mouse
3	Keyboard or Mouse	Eye Tracking
4	Eye Tracking	Keyboard or Mouse
5	Keyboard or Mouse	Eye Tracking
6	Eye Tracking	Keyboard or Mouse

Table 5.1: Testers with coherent test conditions ordered by a Latin Square.

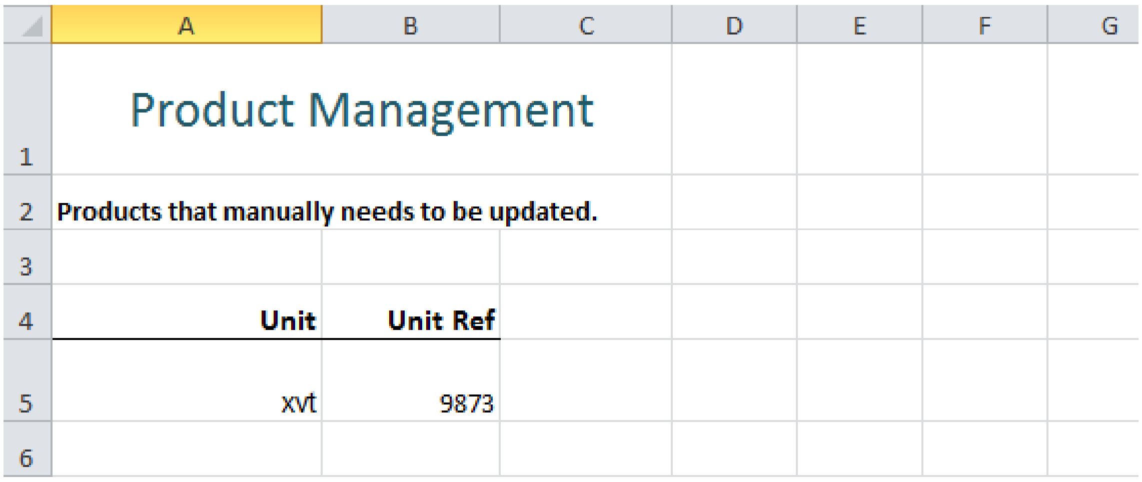
5.4.2 Workflow Design

When testing the prototype we want to look at how the tester handles switching between applications with gaze manipulation. However we don't want the user to only be tasked to switch between applications on their own initiative, as it can seem meaningless and demotivate the tester. Also a possible side effect is that the tester can become too much obsessed with eye navigation making it the primary focus. As we test top-level navigation with eye tracking, we want it to stimulate as close as possible to a real work environment where eye navigation is supplemented to solve a problem and not to be the main attraction.

The workflow created to test the prototype, forces the tester to switch between three top-level applications. All of the applications are opened beforehand and the tester may not be concerned with opening or closing the applications. The first application is located within the web browser where the tester starts by registration for a work session. The tester needs to enter name, age and marks which condition that will be tested. When proceeding the user is presented with some company information and a list of products as seen in figure B.1. One of the items in the list is highlighted with red to indicate that the item has a unregistered unit. This creates the foundation of the workflow of updating unregistered units of measure.

New unit values are registered in a Excel spreadsheet, which is the second application the user is required to interact with. The spreadsheet requires a field with a unit reference that was obtain from the web browser. Figure 5.5 shows one completed entry in the spreadsheet with a unit and updated unit reference. The second column in the spreadsheet holds the updated unit reference; however the unit reference is not available from the web browser, nor the spreadsheet. In order to obtain the unit reference for the entry update, the tester must navigate to the third application called Unit Ref, see figure A.1. In Unit ref the tester can enter the three letters representing the new unit of measure and Unit Ref will return a reference number. By updating the spreadsheet with the unit reference obtained from Unit Ref completes the work cycle. Thereafter the tester must return to the web browser and request the next client update. Thus the cycle starts all over again.

From my observations obtained at PostNord I learned about the constant necessity of changing between top-level applications throughout the day. By taking a closer look at one of June's many tasks, one of them concerns the payment routing from truck washes. Hence there is a second company providing cleaning facilities. Trucks from PostNord may use the facilities at any given time and they are logged in and out of the facilities by



	A	B	C	D	E	F	G
1	Product Management						
2	Products that manually needs to be updated.						
3							
4	Unit	Unit Ref					
5	xvt	9873					
6							

Figure 5.5: Screenshot from Excel spreadsheet.

writing down the license plates. Thereafter the wash company invoices PostNord based on the log entries. When June receives the log entries, she needs to assign the expenses to the specific truck in order to keep track of the trucks expenses. June's work procedure starts off by checking all of the log entries to ensure that all of the entries derive from PostNord. Since PostNord have multiple agreements when it comes to pricing, June is also required to check that the price is correct according to type of wash and the agreed upon prices. From the workflow we observe the necessity of multiple steps that are taken in consideration to ensure that invoices are in fashion of the agreed terms.

From observing June's workflow, we learn about the necessity to interact with different application when only focusing on this specific workflow. Which is also what I would consider reasonable, based on the observations. However this scenario places the user to only be tested in a condition where the computer is solely used. In order to create a more realistic approach, some of the pages from the web browser where replaced with paper. Creating a more varied workflow by telling the user to take a sheet of paper from the stack every second time the user requests a new task. The participant needs to solve a total of seven unit updates, where three updates are from paper and four from the web application. The first two updates are intended for learning, covering how to register entry from web and paper.

A chronological order of the tasks executed in the workflow is listed in table 5.2.

5.5 Testing Prototype 1

When evaluating the implementation of eye movement used in the prototype, the thesis takes into consideration the overall usage of the prototype, how the testers would acknowledge strengths and weaknesses of the interaction and by analysing recordings of the experiment to check for consistency. Also important to acknowledge is eye fatigue and overall wealth. Since it is hard to see if a person gets tired in theirs eyes or detect strain on musculature. The feedback from the testers is therefore crucial in detecting fatigue.

#	Description
1	Register web browser session.
2	Walkthrough of registering new unit update from web.
3	Walkthrough of registering new unit from paper.
4	Register new unit from web.
5	Register new unit from paper.
6	Register new unit from web.
7	Register new unit from paper.
8	Register new unit from web.
9	Completed

Table 5.2: A chronological order of the workflow.

Further the accuracy is evaluated, how the testers perceived the accuracy of using the prototype and if it affects the testers concentration.

From the calibration results, all of the testers scored five stars, suggesting that the accuracy of the eye tracking technology should be equally among the testers. Also suggesting that the eye tracker was being used under optimal conditions. Though the exception exists for one of the testers that where visually impaired with a reduced view angle, however still achieving a calibration of three stars.

The testers where all quick learners and had learned how to use the prototype within a couple of minutes. The gaze pattern for navigating between applications consists of three steps, similar to how one would press the start menu and select an application. Suggesting that the gaze pattern reassembles patterns for selecting applications that we are already familiar within earlier versions of Windows. Figure 4.2 shows the gaze pattern where the menu is triggered by looking at the icon in the bottom left corner, looking towards the desired application before looking towards the center of the screen.

5.5.1 Advantages and Disadvantages

Tester #2 thought the prototype made it easier to navigate between applications. It reduces the necessity to fiddle with the navigation. She further explained and I quote; "If you only need to see in order to explain, that would make it easier". She further emphasized that if the prototype would work 100% she would use the prototype rather than the mouse. When referring to a prototype that works 100%, the argument should be applied for the robustness of head motion in the equipment as the prototype does not modify the gaze data.

While tester #4 would favor the use of keyboard to navigate between top-level applications, he could see himself using the prototype just as fast when he gets a hold of it. He would further argue that it was easy to switch between applications when using the prototype and that it makes it more simpler to switch between top-level applications.

When tester #5 where asked about advantages and disadvantages concerning the prototype, he suggested that the future him would acquire more training in order to perform faster. He further described the idea of using the prototype to navigate between top-level applications at work and that it was highly desirable to use the eyes for navigating between applications. He continued expressing that the order of the menu items kept changing as he switched between the applications. This behavior where not desirable as progressing to learn the position of the top-level applications can never be achieved when it is required of the user to always look over the options when deciding which of the applications that should be brought to front.

Tester #6 thought the prototype was fast and that it reacted faster then what he were expecting. The eye tracking technology was also surprisingly more accurate than what he could imagine possible. When moving onto the question regarding advantages and disadvantages, he argued that the need to move the mouse around to navigate between applications is removed when using the prototype. He further simplified with this explanation and I quote; ”It was like looking at it and it would change the application and you could continue what you were doing”. He further proclaimed it was comfortable and it felt quite natural. He further continued reasoning and I quote; ”If you intend to switch between applications, you would likely look at the icon in the first place anyway”. Having the eyes already looking at the menu to indicate that a selection is desired, making it possible to act on the user’s intentions. Moreover arguing the disadvantages of the prototype, he highlighted that he would loose control of the menu if he moved out of the calibrated position.

5.5.2 Evaluating Fatigue

As mentioned in the introduction of Chapter 5.5, where it is crucial to evaluate the implementation of the gaze interaction. It is important to evaluate how the testers would experience fatigue. It is not uncommon to experience eye fatigue when the gaze interaction forces the tester to think about how they use their eyes and forget to blink [23, p. 261]. During testing, testers would discover that eye tracking would work optimally from the seated position during calibration. As a result the testers could stress the desirability to hold the original position putting strain on the back, neck and overall musculature of the body.

Tester #1 asserts it was comfortable to use the prototype. When she was asked about eye fatigue or exertion she replayed with no change or anything noticeable. After being asked twice that could sound like a cry for help, she replayed and I quote; ”I could perhaps be that if I were to use eye tracking over a longer period, I could have a tendency to notice fatigue”. This was also the case for neck, shoulders and arms, not any noticeable change.

When asking tester #2 about any discomfort with regards to using the prototype, she replied with the not so surprising answer of no. When further asked about what she liked and not liked about the prototype, she further expressed; ”The only thing I did not like was that it was unfamiliar. But it was actually quite fun! Moreover when I got used to using the prototype I enjoyed the idea of not using the mouse”. She would also express that the short period of usage did not affect here in any way when asked about fatigue in eyes, neck, shoulders and arms.

Tester #3 where more concerned about the discomfort of being observed rather than noticing any discomfort from the use of the prototype. When asked about any fatigue from the use of eye tracking and the prototype, he as others thought it was to little of work to actually notice anything. He would further argue that as long as the eye tracker can be adjusted to a relaxed seated position it shouldn't be any problem.

When talking with tester #4 about eye fatigue, he started to argue that the eyes are not used to operate software and therefore it would seem logical that they got a bit tired. Further expressing that the eyes where a bit tired and weary. When proceeding to talk about fatigue in neck, shoulders and arms he expressed an understanding for the eye tracker, that the eye tracker was calibrated to his current position and therefore chooses to be seated in a relaxed position from the beginning.

Tester #5 also indicated that he was a bit stressed in the beginning of the test. He did also report something that could seem like a stiff neck from using the prototype. However as he progressed with the test it went away. Tester #5 as tester #1 felt the necessity to defend or argue that the short period of testing the prototype was not sufficient to notice any fatigue. Tester #6 would also claim that the first ten seconds where strange because of the unfamiliar use of eye tracking technology. When it comes to eye fatigue, tester #6 could not claim that he really noticed anything.

5.5.3 Accuracy

When evaluating the eye tracking accuracy we need to take into consideration how the tester experienced the accuracy of the prototype. As we already know that eye tracking is not very precise, or at least precise down to a area. It is therefore natural to evaluate if the testers felt the prototype behaved accurately or if trigger areas optionally are not well adjusted for the prototype.

The first tester reports that she managed to select the desired choice each time and that as she progressed using the prototype, it became more natural. She did also mention that she felt the need to concentrate for a short period in the beginning of the test. Coherent with tester #2, she expressed that it seemed harder to use the prototype the more she concentrated on using it. Moreover when she stopped to think about how to use the prototype and just used it, it seemed to work surprisingly well. When asked about any miss selections, she reported that it would occur in the beginning and that the prototype refused to select the menu icon she was looking at.

Tester number #3 claimed that he needed to increase 30% of his brain activity in comparison to using the mouse when working the prototype. However he also added that the prototype behaved accurate enough for his use. He also felt to clarify that he would from time to time during the test that he would blank out, our can interpret as drift away. This lead to randomly selecting applications without any motivation or regards to actually solving the task. He explained that this could happen due to stress or memory loss, also expressing when the test was over that he felt some uncomfortable being observed.

The fourth tester did also experience operational issues when using the prototype. He described as follow; "I did not feel the calibration was so accurate. When I was looking a bit out on the side, the prototype thought I was looking at this, but I was actually looking at that". Where he intended to look at the first menu item, the prototype highlighted the second menu item. He also suggested that there should be more spacing between the menu items, so it would be easier to separate the menu items from each other. Tester #4 also reported that at one point when he looked at the keyboard, the prototype was unintentionally triggered leading to the menu being opened. When he was further asked about how the prototype was experienced with respect to accuracy he continued as follow; ”It was quite accurate. Because it was cool to only look down and then back up again”. He further elaborated that it’s only a case of *getting used to* as he also had problems adjusting to the idea of using the eye tracker instead of keyboard shortcuts.

Tester #5 also described the instant need to concentrate when starting to use the prototype. He described that once starting to concentrate; he felt the need to concentrate even more. He further gave the following reflection; ”When I started to concentrate, I began to concentrate even more. It would seem like it got easier when I learned to relax. I noticed it gradually as I where using the prototype”. When further asked about the precision and if he managed to select the intended application each time, he claimed that the prototype would select the menu item he intended to select and that it was pretty much okay to use. However this was not the case for all of the selections. He also reported that in the beginning of the test he was stressed and tried to concentrate on where he should look. As a result it became harder to use and less accurate when trying to concentrate on where to be looking. He got the impression that it was harder to use the prototype the more he was thinking of how to use it.

Not surprisingly, tester #6 also reported the need to concentrate in the beginning of the test and as he progressed in using the prototype, it would ease up and become more comfortable and natural. Tester #6 where scheduled to use the prototype in the first condition. A result of this is the following quote; ”It seemed strange to go back, without having prototype”. Indicating that he quickly adapted to the use of the prototype. He also noticed that the prototype or the eye tracker to be specific would become inaccurate when he leaned forward. At that point he was unable to navigate back to the Excel spreadsheet. By leaning backwards again to the original calibrated position, the prototype would start behaving more properly without any further issues. Tester #6 would draw the following conclusion; ”Yes, this is how I where seated when calibrated. So I thought okay, move back to the original position and it should be good and it worked!”.

5.5.4 Reflections

Throughout using the prototype, a workflow where created to stimulate real work being carried out and to redirect the focus of the testers away from the technology itself. In this attempt it became clear after a couple tests that the task described in appendix B required that the tester used the mouse in order to get the next client. This behavior contradicts the desired usage of the prototype of being free from using the mouse.

From observing the total of six testers, I learned that they could be divided into two significant groups. Were the determining factor was the fluid use of the keyboard to navigate between top-level applications or the extensive use of the mouse to navigate between top-level applications. It was to some extent expected that some testers would prefer to use the mouse, however the fact that four of six testers preferred to use the mouse to navigate is somewhat surprising. Especially since all of the testers derived from students specializing within the field of Computer Science. This is interesting as the testers preferring the keyboard have their preferred routines for navigating between top-level applications and indicated that they were less likely to adapt to the prototype. While the testers using the mouse seemed more interested in the idea of adapting to the prototype as it made it easier.

From the timestamps derived during testing, see appendix C, indicates that two of six testers managed to perform faster with the prototype when compared to using the mouse. Both of the testers preferred to use the mouse and are identified as #1 and #3 in figure C.1. Moreover it could be explained that the timestamps are afflicted by carryover effects, since the starting test condition for tester #1 and #3 where keyboard or mouse as found in table 5.1. Carryover effects can also be the case where keyboard or mouse was the start condition for tester #2, #4 and #6. Even though it is promising that the prototype could be more efficient than using the mouse, also indicating a low adoption threshold.

Throughout the testing phase few error selections were made. Promoting that the size of the menu items where sufficient in accommodating the testers gaze. Also all of the testers managed to complete the workflow without assistance.

5.6 Test Environment for Testing Prototype 2

The lab environment as previously described in Chapter 5.3, describes the physical environment where prototype 2 also where tested. This section will focus on describing the procedure of how the prototype where tested and how the workflow where designed to resemble a realistic workflow.

5.6.1 Test Design

The prototype challenges the possibility of manipulating the tab order with the use of gaze in a typical Windows application. To test the prototype as intended, a workflow is created to stimulate a realistic workflow. The workflow forces the tester to navigate between multiple input fields and is further described in Chapter 5.6.2. While the workflow design creates the play rules for how the work should be carried out, the test design describes how the different independent variables for testing the prototype.

When testing the first prototype, a Latin square where used to reduce the carryover effects. The test design for prototype two does not use a Latin square for the conditions. For testing the prototype, three conditions where used. The first condition uses the gaze and the tab key to manipulate the tab order. The second condition uses the gaze and the addition key within the numpad region. The last condition does not utilize the gaze and the tester may choose the desired choice for solving the workflow. Table 5.3 gives a overview of the order the different conditions were tested.

Before starting the test, each participant is given a consent form explaining that the results obtained are used to evaluate the prototype and not the individual. Consenting to the use of a camera recording the tester and screen during. If photos from the recordings are to be used as illustration in the report, further approval is required from the tester. Thereafter the tester is given instructions on how to complete the work as described in Chapter 5.6.2.

Tester	Start Condition	Second Condition	Third Condition
1	Eye Tracking + Tab	Eye Tracking + Addition key	Mouse or Tab
2	Eye Tracking + Tab	Eye Tracking + Addition key	Mouse or Tab
3	Eye Tracking + Tab	Eye Tracking + Addition key	Mouse or Tab
4	Eye Tracking + Tab	Eye Tracking + Addition key	Mouse or Tab
5	Eye Tracking + Tab	Eye Tracking + Addition key	Mouse or Tab

Table 5.3: Testers with coherent test conditions in a fixed order.

For the first condition the tester would start by calibrating the eye tracker and use the tab key to move the focus to the next text field in the direction of the gaze. For the second condition the tester where to use the gaze to indicate direction, however the addition key on the numpad would be used instead of the tab key. For the third and last condition, eye tracking was disabled and the tabular order would stay fixed to the horizontal pattern as seen in figure 4.6(a). The tester would choose the most natural way of navigating the text fields, where expected outcomes would be the use of tab or the mouse to navigate set the focus of the text field.

The prototype is used through all of the conditions, where eye tracking is only utilized in the two first conditions. Before proceeding with the first conditions where the eye tracker was used, a calibration of the eye tracker was required. For the second condition the tester would calibrate again if necessarily. The calibration procedure is described in detail in Chapter 5.2. When the calibration is over, the tester where allowed to play around within the calibration software by looking at the different calibration points. This is to assure that the eye tracker was successfully calibrated and to reduce the overwhelming effect of using a eye tracker for the first time. Thereafter the tester was instructed on how to use the prototype and performed some exercises by tabbing back and forth in the prototype. Instead of interviewing the testers after the conditions were completed. The testers were asked questions during all of the conditions.

5.6.2 Workflow Design

Prototype 2 is developed to test the concept of using the gaze to modify the tab order. Prototype 2 is therefore designed to resemble a realistic application used at work. The workflow can be described as having sheets of paper with order details. The orders exist in in the prototype as incomplete orders that need to be updated in order to be fulfilled. The sheets of paper contain the remaining information that needs to be updated. When the prototype is started, it shows the order number in the top left corner. Thereafter if shows the product lines where all of the fields have values with the exception of one field being empty. The empty field is random will be random for each order. For instance the first order may miss the product id for the first product, while the next order might be missing the price for the last order. The tester must identify the field missing a value. Thereafter find the coherent paper with matching order numbers. The paper will have a value for the missing field that is required by the prototype. When the field is filled, the tester may press enter to get the next order. Continuing by identifying the missing field and find the coherent sheet of paper and update the missing information. Pressing enter before updating will yield an error highlighting the field missing a value with a red background.

Instead of having testers complete the workflow from a fixed number of repetitions. The testers are set to repeat the tasks for eight minutes for each condition.

5.7 Testing Prototype 2

Feedback from testing the prototype 1, where the testers reported that the test was not sufficiently long enough to notice any fatigue when using the eye tracker. Therefore the second condition where introduced, increasing the time spend using the eye tracker from 2-3 minutes to 20 minutes in total. Similar as evaluating prototype 1, prototype 2 will share the same values. Maintaining focus on the overall usage of the prototype and how the testers acknowledges strengths and weaknesses with the interaction. Eye fatigue and overall wealth is also important to consider for prototype 2 and not to forget the perceived accuracy when using the prototype.

5.7.1 Advantages and Disadvantages

While tester #7 where managing papers from the workflow and entered numbers into the prototype, he would hold the paper above the keyboard in his left arm while typing on the numpad with his right arm. Sometimes the tester would hold the paper to high disrupting the view angle from the camera to his eyes, causing the prototype to stop responding.

The text fields in the prototype are much smaller than the menu items in prototype 2 and contradict the recommended trigger size when dealing with the gaze [30, 32]. Tester #9 noticed over time that the prototype became inaccurate and started to compensate to correct the selection. He achieved this by looking ahead to the next field in the desired direction he wanted the cursor to traverse. Moreover when the calibration would disintegrate due to head motion, the accuracy would also disintegrate, making it necessarily to compensate for head motion.

Tester #10 did also notice the disintegration in the accuracy and would from time to time look ahead of the destination text field. Otherwise he would click the tab key multiple times until the text field he was looking at until selected. He would further argue that the cursor should jump directly into the desired text field. Rather than having the cursor traverse multiple text field of no significant.

Pressing the tab key and having gaze enabled in the prototype makes the cursor traverse in the direction of the gaze. However if the algorithm thinks the gaze is looking at the current text field, pressing tab will not perform anything. Tester #11 encountered some inaccuracy with the eye tracker, where the prototype where tricked into believing that the tester where looking at another text field. While nothing would work at that point the tester where made to believe that something was wrong with the equipment or the prototype. Further the tester would learn how to compensate by shifting the gaze ahead of the text field, similar as tester #9 and #10. Moreover having the ability to compensate for the disintegration in accuracy by looking ahead of the target text field proved to be quite useful.

Enabling the addition key on the numpad to replace the tab key. Making the tab behavior to appear on the numpad, makes all the keys available from the right arm. This proved to be quite useful as it freed up the left arm completely. Were the left arm could handle the paper flow worked well and where embraced as the preferred location of the tab key. This is ideally when only dealing with numbers as input and not operations that requires the use of addition.

5.7.2 Evaluating Fatigue

Evaluating fatigue among the testers the feedback was close to identical with the feedback gathered for fatigue when testing prototype 1. Even though the test-duration using the eye tracker where increased to two session of 10 minutes each. Tester #10 emphasized that he dint notice any more eye fatigue than out of the ordinary. Arguing it was the use of contact lenses that affected the eyes in some extent.

Tester #10 would also notice fatigue in his arm as he where solving tasks in the workflow. While solving tasks in the workflow for the first condition where the tab key where used, he would constantly pick up the sheets of paper, before putting them down, continuing to use the numpad to enter the updated numbers.

5.7.3 Reflections

During the workflow for the first condition were the tab key was used to navigate between text fields Most of the testers would use the numpad to enter the numerical values. While there also was one case where one of the testers used the numbers at the top of the keyboard, the majority used the numpad. It is thereafter easier to notice the disruptencies the testers encounter. Further describing that the tab key is located on the far left side of the keyboard and the numpad is located on the far right side of the keyboard. Were it is natural to have the left arm handling the tab key and the right arm for handling the numpad. Both hands are therefore used on the keyboard to solve the tasks. In addition paper flow is added to the workflow where sheets of paper are used to identify information. Making it necessarily for one of the hands to leave the keyboard in order to sort or hold the papers during input.

Chapter 6

Discussion

6.1 Implementation of Natural Eye Movement

In the introduction is the research question formulated to ask how natural eye movement can be implemented in business applications. The most common challenge with implementing eye movement in general is the Midas touch problem, see Chapter 2.1.5. Here the state of the eyes can be closed or open. The eye tracker will provide a continuous flow of gaze data when the eyes are open, further challenging how or when the data should be interpreted. By means of natural eye movement, Jacob [21] emphasizes that the user should not be required to make specific eye movements or think about how eye movement will trigger actions. Further as emphasized in Chapter 2.1.6 that blinking the eyes is not a natural eye movement. Moreover when eye movement first is used to trigger action, Jacob adds that to long dwell time is not natural. Implementing eye movement in a business application should not expect the user to stare at a point for a longer period, as its not natural for the eyes to stare and wait for something to happen. The eyes will always move themselves rapidly around or skip little jerks at a time as observed by Huey [18].

To get something happen by just looking around is quite the opposite of natural. As the technology is new and unfamiliar for the testers, most of them would focus to much on trying to steer their eyes in the beginning of the test. Furthermore by starting to think about how and where the eyes should move would cause even more frustration as the eyes are not contemplated to be controlled on demand and will look exactly where they desire. One of the testers expressed that concentration in the beginning would cause an evil cycle and said: ”When I started to concentrate, I began to concentrate even more.”. Moreover this did mostly occur in the beginning of the test. The same tester added that: ”It would seem like it got easier when I learned to relax.”. All of the testers would describe similar experiences when starting to use eye tracking in the prototypes. Progressing into the test, the testers would learn to relax and get used to the idea that something would happen based on the idea that the computer knew where they were looking.

6.1.1 Prototype 1: Navigating Between Top-Level Applications

Prototype 1 is gaze based, meaning that the prototype only uses input from the eyes. Eye movements in the prototype is used to open a menu and select an application that should be brought to front as illustrated in figure 4.2. Jacob [21] have experimented with similar prototypes, such as the Continuous Attribute Display as discussed in Chapter 2.3.1 where fixations and dwell time is weighted. Having created a gaze based prototype that will respond to specific eye movement. The necessity of dwell time is eliminated by reserving a trigger area in the bottom corner of the screen. The selection of the menu is triggered instantaneously when looking in the bottom left corner of the screen, complying with Jacob's recommendation for reduced dwell time to create natural interaction. Moreover, by not introducing dwell time, Ohno and Mukawa [25] suggest that the users can become more efficient when the prototype is used over a longer period of time. Allowing users to adopt and become more efficient also suggests that the eye movement must be as natural as possible to avoid eye fatigue. In addition to Ohno and Mukawa's recommendation to achieve efficiency over time, one of the testers expressed that he could become faster by having more time for training, suggesting that he identified the prototype to become better when used over a longer period of time.

Having the eyes focused on a blank page is not natural as the eyes have nothing to look at or "hold onto". The eyes need elements that have contrasts and are recognizable. As suggested by Jacob [21], Smith and Graham [28] in related work, the eyes should have a recognizable feature to look at when specific eye movement is expected. The prototype has therefore implemented a small icon placed inside of the trigger area, making it easier for the eyes to recognize and look at. Looking at the icon will spawn the menu instantaneously, where different menu options are rendered horizontally along the bottom of the screen as seen both figure 4.1 and figure 4.4. Also the menu items in the prototype use coherent application icon as recognizable features. The application icons are placed in center of the menu items, allowing margins between the surrounding boundaries and the icon to serve as trigger area.

The size of the selection boundaries as used for the trigger area as well as the menu items are adjusted by the recommendation from Ware and Mikaelian [30]. Since eye movement is not particular precise as the eyes are rapidly moving around. Even during fixation the eyes produces small jittery movements [21], making it difficult to determine precisely where the eyes are looking. Ware and Mikaelian found that the selection speed is increased by having bigger selection fields, striving to achieve natural eye movement where the selection speed is optimal. Zhai [32] also confirms that square targets of 24 mm have the most speed gain and further increasing the size will barely improve the selection speed. None of the testers have made any remarks on the selection speed. Not surprisingly it can be argued that none of the testers have any experience with eye tracking and therefore expected that the prototype would behave accordingly. In future work it could be interesting to compare the current state of the prototype with a prototype that uses dwell time and recognizes fixations as Jacob used in his prototypes (see Chapter 2.3.1).

While by building on natural eye movement, the prototype may also encounter rapid eye movement or be afflicted by eye jitter to cause unintended selections. One attempt to encounter error selections caused by eye jitter is to show the top-most application as the first menu item. So that when a unintended selection occurs, it will select the current application, making it seem like nothing changed. Moreover this occurred once during testing, where one of the testers looked at the keyboard. In the progress of the gaze moving towards the keyboard, the gaze entered the trigger area resulting in spawning the menu before the gaze exited the screen. When the gaze returned back towards the center of the screen, the current application where selected as if nothing had changed. Whenever the prototype would avoid a error selection, the tester would never notice the selection and only be distracted by the menu popping up. The error selection could have been avoided in this scenario by introducing a short dwell time to trigger the menu selection. However deciding the reasonable amount of dwell time and taking into account the number of error selection can be discovered in future work.

The gaze pattern for the prototype as seen in figure 4.2, shows that the gaze is moving towards the bottom left corner and triggering the menu. The eyes have visual cues for triggering the menu and the different menu items. However the last step in the gaze pattern, where the actual selection is acquired does not have a recognizable feature. In earlier attempts of the application where selected for each time a menu item was gazed upon. Moreover changing the top-most application rapidly would seem to get the screen to blink for each selection. Having the prototype change the top-most application rapidly is easily detected by the peripheral vision, grabbing the eyes attention even if the central vision remains on the menu item [31]. Therefore introducing the last step of looking up towards the center of the screen. It can't be argued that the last step in the gaze pattern is natural, as it can be compared other research [21, 28, 19] where the gaze don't have any recognizable features. Moreover acknowledging that the last step in the gaze pattern is not a natural eye movement, can it be contradicted since it is less specific than controlling the detonation of a rocket in midair? (see Lunar Command, Chapter 2.3.2). None of the testers reported that it was strange to look up in order to have the application selected. However one of the tester said: "Because it was cool to only look down and then back up again.", where it's only a matter looking back at the screen. Finding that with no findings regarding the matter, it can be considered normal when the eye movement is not required to be specific.

6.1.2 Prototype 2: Gaze Directional Tab Order

Prototype 2 uses the location of the gaze to modify the tab order where multiple input fields are introduced. The prototype is gaze added, meaning that the prototype uses a button to instantiate a action based on information from the gaze. Eye movements in the prototype is used to determine which field the user wants to edit and by pressing tab will move the cursor towards the input field the user is looking at. The gaze pattern is illustrated in figure 4.7, where the gaze location is used navigate the shortest path of two input fields.

In prototype 1 the trigger areas where adjusted to comply with Ware and Mikaelian [30] recommendations to allow optimal selection. Moreover prototype 2 do not have the luxury of increasing the input fields to the recommended size as to many and to big input fields would clutter the interface. As a result of not having recommended selection sizes, testers would from time to time notice that the prototype became inaccurate. When the accuracy disintegrated some of the testers would try to look ahead to the next input field. Usually the disintegrated accuracy would mean that the gaze data provided where not centered for the testers gaze, still making it possible to trick the eye tracker by looking ahead to the next field. Having to trick the eye tracker allows the testers to compensate for the loss in accuracy. In addition some testers would try to move back and forth trying to return to the position they were seated during the calibration. Only three of the five testers experienced the necessity to compensate for the selections. Were it would only happen a couple of times during the test. While the need to compensate a couple of times during the workflow, it was accepted by the testers. Possibly because they were overwhelmed by the idea of using the eyes as a form of input or that they felt like they could not expect more of the technology. Further when one of the testers encountered a input field that was required to be updated in the far right and at the same time where required to compensate for the selection, he was not able to update the input field.

Moreover the other testers that did not encounter the disintegration of accuracy nor experienced the necessity to compensate with eye movement, these testers would indicate that the eye movement felt natural. However they would question why they needed to press the tab key multiple times. Suggesting that the cursor should go directly towards the input field they were looking at. As suggested by the testers, to have the cursor go directly to the gazed upon input field was considered during development. Moreover it was not implemented due to implementation difficulties and the time it would require. Having the input fields smaller than the recommendations from Ware and Mikaelian, the prototype would rather try to use the position of the gaze to decide the direction of the tab order. Further allowing testers to compensate for the lack of accuracy. From observations, it is not particular intuitive to be clicking the tab key multiple times when waiting for something to appear at the input field looking. Moreover the amount of tab key presses required can be compared to using dwell time. If compared to Jacob's observations [21] where the amount of tab keys equals a longer dwell time, it cannot be considered a natural eye movement. However if the amount of tab keys equals a minimal dwell time, it should be considered a natural eye movement.

6.2 Perception of Eye Movement in Relation to the Use of Keyboard and Mouse

Using gaze data to perform selection have been established as not being perfect, considering the jitter and rapid eye movements. Researchers [30, 32] have proposed recommended selection sizes for optimal selection. Jacob [21] compares the accuracy obtained by gaze data to have more similarities with a touch screen in the 90's. He implying that the difference in accuracy obtain with eye tracking technology is not equal to the accuracy obtained from the mouse or keyboard. Moreover in the introduction, the research question is formulated to ask how eye movement is perceived in relation to the use of keyboard and mouse. Smith and Graham [28] compare the use of mouse and eye tracking technology for video game control where they claim that the eye tracker would give a more immersive experience when playing.

6.2.1 Prototype 1: Navigating Between Top-Level Applications

When developing a prototype that is intended to replace how we switch between top-level applications, it is important to acknowledge that this is usually achieved by using the mouse or through keyboard shortcuts. From the observations the majority preferred to use the mouse when navigating between top-level applications. It is intuitive to grab onto the mouse and move the cursor around before proceeding to click on the icon of the application that should be brought to front. The two other testers that did not prefer to use the mouse, desired to use keyboard shortcuts to navigate between the top-level applications. Details on how to navigate through keyboard shortcuts is provided in Chapter 2.2.2. Learning how to use the shortcuts on the keyboard is desirable when trying to improve the efficiency of using the computer. Having learned how to navigate top-level applications through the keyboard is considered to be using less time and interruptions on the mouse.

All of the testers were quick learners and knew how to use the prototype within a couple of minutes. The gaze pattern as seen in figure 4.2 is similar to how one would normally press the start menu and select an application or directly select an application from the taskbar. Suggesting that the gaze pattern in figure 4.2 reassembles the pattern for selecting applications in the Windows operating system, which all of the testers are already familiar with. During a interview of one of the testers said: ”If you intent to switch between applications, you would likely look at the icon in the first place anyway”. Being familiar with how navigation works through the keyboard and mouse, he emphasized that we are already doing the work of navigating mentally. Another tester thought it was cool to make things change by using his eyes to do the work.

Depending on the preferences the testers expressed for using the keyboard or mouse would also color their opinion towards the gaze based prototype. The two testers that had adopted good routines using keyboard shortcuts express that the prototype seamed slow to use. Even when more mistakes where introduced using the keyboard, the keyboard would still feel faster to use. Testers that preferred to use the mouse would seem to favor the prototype over the use of the mouse. One of the testers meant that the prototype reduced the necessity to fiddle with the navigation, further saying: ”If you only need to see in order to explain, that would make it easier.”. Another tester argued that the eyes are looking at the application that is desired to show, long before the actual selection is made.

When dealing with feedback from testers, we have to be aware that they can also be fooled by themselves into believing that something went faster because it was fun or slow because it seemed boring. To help support the feedback gained though the interviews and observations, hidden timestamps where recorded in the workflow. When testing the prototype a web application AMCODE (see appendix B) as where used as part of the workflow to receive data updates. Before the test where started the test leader would help entering the testers name and mark the condition being tested in AMCODE. Since the first two tasks were solved in collaboration with the test leader, AMCODE would mark the starting time from the moment the tester requested the third client and the stop time when there were no more clients left. . The testers were never informed about the timestamps being recorded and they were neither encouraged nor discouraged to complete the tests in a timely manner. The timestamps are intended to check the testers perception of time in relation to the actual time.

As mentioned where personal preferences of using the keyboard and mouse would color the testers opinion towards the gaze based prototype. It can be confirmed through the timestamps that the two testers preferring to use the keyboard used the least amount of time to complete the task. The bar graph as seen in figure C.1 shows that tester #2 and tester #4 have the fastest completion time of all of the testers. When interpreting the blue bars in in figure C.1, also titled *"with out"*, means that the blue columns represents the timestamps where the testers used the keyboard or mouse. From the logs both of the testers #2 and #4 are identified as using the keyboard to solve the task. It is therefore confirmed that the strong preferences towards the use of keyboard shortcuts are justified as it makes the testers more efficient.

The testers that improved using the prototype than with their personal preference are identified as testers #1 and #3. When comparing the bar graphs as seen in figure C.1 both of the testers completed the task slightly faster when using the prototype. The logs identify the testers #1 and #3 to use the mouse when solving the task. Moreover it could be explained that the timestamps are afflicted by carryover effects (see Chapter 3.2). The mouse was used in the start condition for testers #1 and #3 as seen table 5.1. It is possible that having solved the first task using the mouse would make the testers more efficient when solving the task the second time when using the prototype. Carryover effects can also be the case for tester #2 and #4 where the prototype was used in the start condition. However having mentioned the possibility of carryover effects, I don't think it is likely that this is the case for tester #2 and #4. Moreover arguing that the use of keyboard shortcuts are more efficient than the mouse and the prototype. Moreover it is promising that the prototype proved to be more efficient for some of the testers whom preferred to use the mouse.

6.2.2 Prototype 2: Gaze Directional Tab Order

The prototype implements eye movement silently, meaning that the prototype does not indicate where the user is looking when using the prototype. Information about the gaze is hard to visualize when considering that our eyes never stop moving rapidly. Further having a visual cue jumping back and forth on the screen representing eye movement will distract the user [31]. The prototype uses an algorithm to determine the direction of the gaze, where the output is left, top, right, bottom or none. Where none indicates that the eyes are looking at the current selection. As discussed in Chapter 6.1.2, inaccuracy may occur often since the input fields do not obtain the optimal sizes for selection. During testing one of the testers experience that the cursor stopped moving when pressing the tab key. He was looking at the input field above the currently selected input field, however the inaccuracy tricked the algorithm into believing that the tester where looking at the current selected input field. Proceeding to press the tab key, nothing would happen because the algorithm would proceed to output none. The tester would think that something was wrong with the equipment or the prototype. Moreover dealing with the mouse or keyboard one can always be certain where the mouse is located and when buttons are pressed, the testers receives feedback by the sound and tangibility of the mouse click or key press. Dealing with gaze data the prototype failed to give the tester feedback, making the tester believe that something was wrong.

Chapter 7

Conclusion

In the journey of this master thesis we have investigated the following: ”How can eye movement be accommodated in business applications?”. Through research, prototyping and evaluation we try to answer RQ 1.1 of how natural eye movement can be implemented in business applications and RQ 1.2 of how eye movement is perceived in relation to the use of keyboard and mouse. To gain knowledge about business applications a transportation company where sought out where employees where observed throughout a workday. From experience gathered by other researchers [30, 32, 20], some of being presented in related work and from knowledge gained through observations of the employees in the transportation company, two prototypes where created. To answer the research questions, students were recruited to test the prototypes under multiple conditions. Were the methodology of within-subject design was used to structure the tests. Further analyzing the results gathered from video recordings, observations and interviews as discussed in Chapter 6.

7.1 Implementation of Natural Eye Movement

Implementing natural eye movements in applications can be quite difficult and within the field of study it is referred to as the Midas touch problem, see Chapter 2.1.5. Researchers have over many years proposed numerous ways to encounter the Midas touch problem in different applications [21, 32] and games [28, 19].

7.1.1 Prototype 1: Navigating Between Top-Level Applications

The prototype is gaze based, meaning that the prototype only uses input from the eyes to operate. By only using input from the eyes also means other input modalities are not introduced. Moreover the prototype is used to navigate between top-level applications by only input from the eyes. The prototype spawns a menu when the gaze is registered within a trigger field in the bottom left corner of the screen. As suggested by researchers [21, 28] a small icon is placed inside of the trigger area, so that the eyes may have something to look at. When the menu is spawned, menu items representing top-level applications are listed horizontally along the bottom of the screen. Each menu item is represented by drawing a large square, where the icon of the application being represented is placed in the center of the square as seen in figure 4.4. The title of the application is rendered beneath the application and also inside of the square. The menu items uses the application icon as recognizable features as suggested to spawn the menu. The size of the trigger area and the menu items are adjusted for optimal selection speed as suggested by Ware and Mikaelian [30]. Since the prototype does not enable dwell time to spawn the menu nor to select a application, Ohno and Mukawa [25] suggests that the users can become more efficient when using the prototype over a longer period of time. When the menu items where spawn, the prototype would show the current top-most application as the first menu item. When the menu is unintentionally spawned from rapid eye movement, the first menu item selected would be the current application. Reducing the potential amount of error selections.

7.1.2 Prototype 2: Gaze Directional Tab Order

The prototype is gaze added, meaning that the prototype uses input from the eyes and the keyboard to operate. The prototype is used to modify the direction of the tab order where multiple input fields are present. Where the position of the gaze is used to determine the direction towards the input field the user is looking at, the tab key is used to move the cursor towards the gazed upon input field. Allowing the keyboard to trigger an event based on the gaze position is also suggested by researchers [21, 32] to encounter the Midas touch problem. Dealing with multiple input fields the prototype does not have the luxury of increasing the size of the input fields as recommended by Ware and Mikaelian [30], arguing that by increasing the size of the input fields will only clutter the interface.

By not increasing the size of the input fields, the gaze prediction algorithm as attached in appendix D was designed to produce the direction of the gaze from the currently selected input field. The algorithm will output one of the following states: left, top, right, bottom or none, where none indicates that the user is looking at the current input field and where the remaining states determine the direction the user is looking. Thinking it would prove difficult to jump directly to the input field the user would look at, the algorithm where developed to only predict one tab stop at a time when the tab key is pressed. This feature would also allow the testers to compensate when lacking the accuracy to move straight to the desired input field. From observing testers encounter the need to compensate for the lack of accuracy, the eye movement cannot be considered natural. However testers that did not encounter the lack of accuracy would look at the desired input field and press the tab key multiple times until the input field was selected. These testers would also suggest that the cursor should jump directly towards the input field when pressing the tab key once.

Since the tab key is introduced to move the cursor one step at a time towards the gaze position for each time the tab key is pressed. The required amount of tab key presses will vary, depending on the amount of tab stops before reaching the desired input field. The amount of tab stops can therefore be compared to dwell time. Where a the need to press the tab key a couple of times can resemble a short dwell time and the need to press the tab key multiple times can resemble a long dwell time. Further drawing the conclusion from Jacob's observations [21] that a short dwell time is natural and a long dwell time is not natural.

7.2 Perception of Eye Movement in Relation to the Use of Keyboard and Mouse

Eye movement can never be the same as the keyboard or the mouse. Both the keyboard and the mouse provides tactile feedback in form of a sound from pressing a button on the keyboard or from mouse clicks. Moreover both the keyboard and the mouse are tangible and providing the testers a sense of control during use.

7.2.1 Prototype 1: Navigating Between Top-Level Applications

When testing the prototype, the testers where tested in two conditions. The order of the conditions is organized after a Latin square. Were one of the conditions involved using the prototype developed to navigate between top-level applications, the other condition would allow the testers to choose between the keyboard or the mouse to complete the tasks. Two of the testers that preferred to use the keyboard would be more insusceptible to the idea of using the prototype.Finding that testers that had invested in learning how to use keyboard shortcuts, would also spend the least amount of time to complete the tasks. Again emphasizing that the testers never knew they were timed. The four other testers that preferred to use the mouse would be more susceptible to the idea of using the prototype on a daily basis. Moreover the testers that were used to the mouse, would have to switch between the keyboard and mouse when navigating. Furthermore acknowledging that the prototype would reduce the necessity to use the mouse.

Where the keyboard and the mouse can provide tangibility for the testers, eye movement can provide an immersive experience [28]. From an interview with one of the testers, he said: "If you intend to switch between applications, you would likely look at the icon in the first place anyway.". The tester further emphasized that when we use the keyboard or the mouse to navigate, we are already providing gaze information to perform the navigation. Why should we bother to use the keyboard or the mouse when the eyes are already doing the job?

7.2.2 Prototype 2: Gaze Directional Tab Order

Comparing prototype 2 with prototype 1, the testers would encounter the disintegration of accuracy earlier because of the small input fields in relative to the larger trigger fields used in prototype 1 where the disintegration was barely noticed. Furthermore one of the testers that encountered the disintegration of the accuracy when navigating from the current input field to the input field above where made to believe that something was wrong with the prototype or the equipment. The prototype had stopped responding when pressing the tab key. Because of the disintegration of accuracy the prototype where made to believe that the tester was still looking at the selected field. As a result of looking at the selected input field the algorithm would proceed to output none. It is therefore suggested for future work that when the algorithm produces none for output, the prototype should pursue other options.

7.3 My contribution

Through the work carried out during this master thesis I have gained knowledge about gaze interaction and its many applications and areas of use. I have further applied that knowledge in order to produce two prototypes with the intention of once being used in a workplace. Both prototypes implements eye movement, where prototype 1 is gaze based and prototype 2 is gaze added. Having created two prototypes that I hope will spur further interests towards research of eye movement in business application, I have furthermore provided answers in the discussion (see Chapter 6.1) on how to enable natural eye movement. At last I provide insight towards how eye movement is perceived in relation to the use of keyboard and mouse (see Chapter 6.2).

7.4 Further Work

Having dived into the field of study to complete this master thesis, new potential research questions have emerged. Further providing recommendations for further research for the two prototypes provided.

7.4.1 Prototype 1

Proceeding to open more applications and added to the list of top-level applications, the prototype will keep adding menu items horizontally along the bottom of the screen. After having opened 7-10 applications depending on the screen resolution, the menu items can no longer be fitted within the screen. How can the prototype structure the menu items when exceeding the width of the screen?

While the prototype is only concerned with handling navigating between top-level applications, it is desirable to be able to navigate folders as well. How should folders be represented in the prototype and how should the menu items represent a recognizable eye feature that identifies the folder.

7.4.2 Prototype 2

Having the directional tab order move one input field at a time can seem unnecessarily when having to tab multiple input fields. How can the prototype manage to jump directly to the desired input field?

When looking at the current selected input field, how should the prototype provide the next tab stop when proceeding to press the tab key?

Bibliography

[1] Enumwindows function (windows) @ONLINE, December 2013. http://msdn.microsoft.com/en-us/library/windows/desktop/ms633497%28v=vs.85%29.aspx.

[2] The eye tribe c# language tutorial @ONLINE, 2013. http://dev.theeyetribe.com/csharp/.

[3] Nuget gallery — json.net @ONLINE, 2013. https://www.nuget.org/packages/Newtonsoft.Json/.

[4] Setforegroundwindow function (windows) @ONLINE, February 2013. http://msdn.microsoft.com/en-us/library/windows/desktop/ms633539%28v=vs.85%29.aspx.

[5] Showwindow function (windows) @ONLINE, February 2013. http://msdn.microsoft.com/en-us/library/windows/desktop/ms633548(v=vs.85).aspx.

[6] The eye tribe @ONLINE, August 2014. http://theeyetribe.com.

[7] Eyetribe/tet-csharp-client @ONLINE, September 2014. https://github.com/EyeTribe/tet-csharp-client.

[8] Lorem ipsum - all the facts - lipsum generator @ONLINE, 2014. http://www.lipsum.com.

[9] Windows design guidelines for keyboard interaction@ONLINE, February 2014. http://msdn.microsoft.com/en-us/library/windows/desktop/dn742465(v=vs.85).aspx.

[10] Tobii eyex @ONLINE, January 2015. http://www.tobii.com/eye-experience/.

[11] Within subjects design @ONLINE, January 2015. http://web.mst.edu/~psyworld/within_subjects.htm.

[12] Within subjects design (what it is and how it's used) @ONLINE, January 2015. http://psychology.about.com/od/windex/a/within-subjects-design.htm.

[13] Arbeidstilsynet. Veiledning til arbeid ved dataskjerm. 2006.

[14] Richard A Bolt. Eyes at the interface. In *Proceedings of the 1982 conference on Human factors in computing systems*, pages 360–362. ACM, 1982.

[15] Andrew T Duchowski. A breadth-first survey of eye-tracking applications. *Behavior Research Methods, Instruments, & Computers*, 34(4):455–470, 2002.

[16] Andrew T Duchowski. *Eye tracking methodology: Theory and practice*, volume 373. Springer, 2007.

[17] Tovi Grossman and Ravin Balakrishnan. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–290. ACM, 2005.

[18] Edmund B Huey. Preliminary experiments in the physiology and psychology of reading. *The American Journal of Psychology*, 9(4):575–586, 1898.

[19] Poika Isokoski, Markus Joos, Oleg Spakov, and Benoît Martin. Gaze controlled games. *Universal Access in the Information Society*, 8(4):323–337, 2009.

[20] Robert JK Jacob. What you look at is what you get: eye movement-based interaction techniques. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 11–18. ACM, 1990.

[21] Robert JK Jacob. The use of eye movements in human-computer interaction techniques: what you look at is what you get. *ACM Transactions on Information Systems (TOIS)*, 9(2):152–169, 1991.

[22] I Scott MacKenzie and Xuang Zhang. Eye typing using word and letter prediction and a fixation algorithm. In *Proceedings of the 2008 symposium on Eye tracking research & applications*, pages 55–58. ACM, 2008.

[23] Paivi Majaranta and IGI Global. *Gaze interaction and applications of eye tracking: advances in assistive technologies*. Medical Information Science Reference, 2012.

[24] Carlos H Morimoto and Marcio RM Mimica. Eye gaze tracking techniques for interactive applications. *Computer Vision and Image Understanding*, 98(1):4–24, 2005.

[25] Takehiko Ohno and Naoki Mukawa. Gaze-based interaction for anyone, anytime. In *Proceedings of HCI International 2003*, volume 4, pages 1452–1456. Citeseer, 2003.

[26] Michael Schiessl, Sabrina Duda, Andreas Thölke, and Rico Fischer. Eye tracking and its application in usability and media research. *MMI-interaktiv Journal*, 6:41–50, 2003.

[27] Susan K Schnipke and Marc W Todd. Trials and tribulations of using an eye-tracking system. In *CHI'00 extended abstracts on Human factors in computing systems*, pages 273–274. ACM, 2000.

[28] J David Smith and TC Graham. Use of eye movements for video game control. In *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 20. ACM, 2006.

[29] Roel Vertegaal, Gerrit van der Veer, and Harro Vons. Effects of gaze on multiparty mediated communication. In *Graphics Interface*, pages 95–102, 2000.

[30] Colin Ware and Harutune H Mikaelian. An evaluation of an eye tracker as a device for computer input2. In *ACM SIGCHI Bulletin*, volume 17, pages 183–188. ACM, 1987.

[31] Susan Weinschenk. *100 things every designer needs to know about people.* Pearson Education, 2011.

[32] Shumin Zhai, Carlos Morimoto, and Steven Ihde. Manual and gaze input cascaded (magic) pointing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 246–253. ACM, 1999.

[33] Xuan Zhang and I Scott MacKenzie. Evaluating eye tracking with iso 9241-part 9. In *Human-Computer Interaction. HCI Intelligent Multimodal Interaction Environments*, pages 779–788. Springer, 2007.

Appendices

Appendix A

Unit Ref

The Unit Ref application is a tool created with regards to test the workflow designed when testing prototype 1. The application is designed to create a fictive numerical reference for a supplied character unit. The character unit will always consist of three characters; however which character or the order of the characters may be completely random. Moreover the Unit Ref application will allow any given input and create a random numerical value between 1000 and 9999, suggesting that the Unit Ref value returned will never exceed a length of four digits. A screenshot of the application is seen in figure A.1.

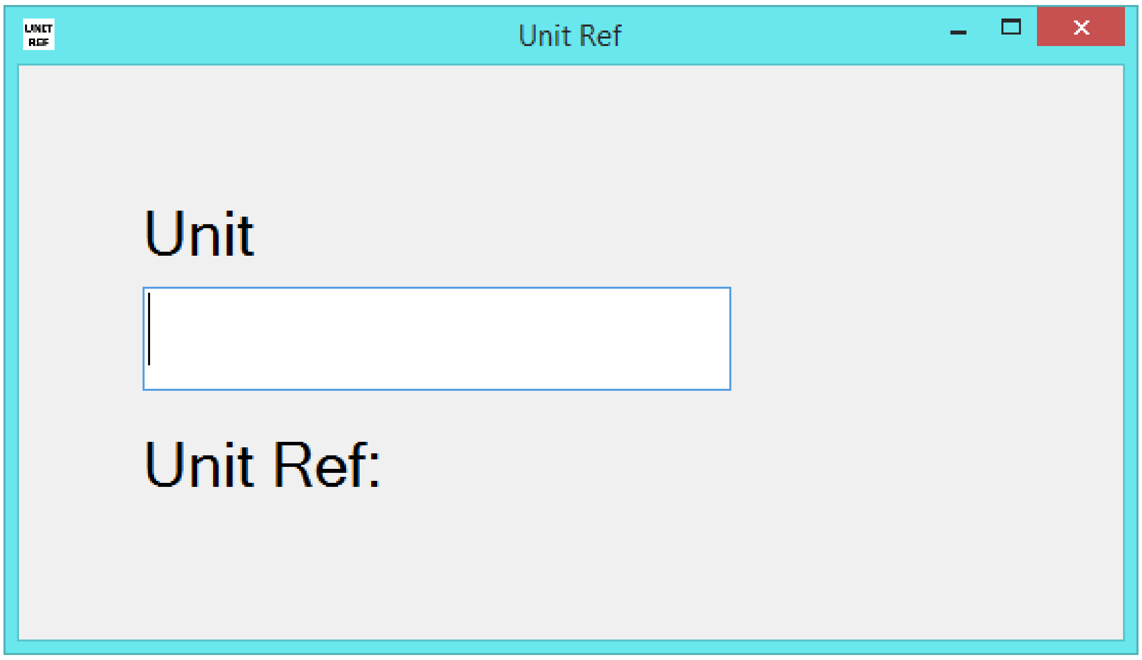


Figure A.1: Screenshot of the Unit Ref application.

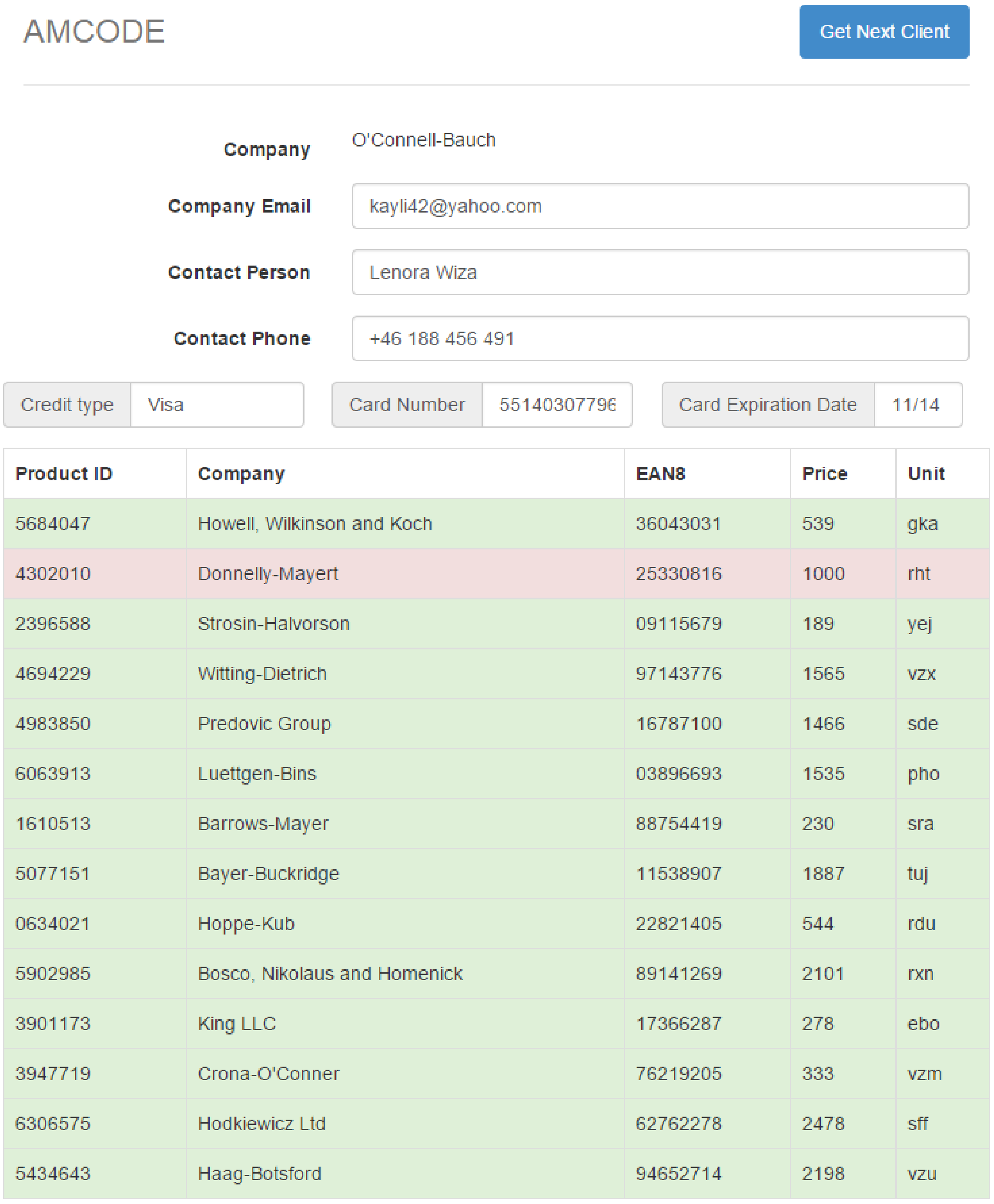
Appendix B

AMCODE

The AMCODE application is a tool created with regards to test the workflow designed when testing prototype 1. AMCODE is a web application and therefore only accessible through the web browser.

B.1 Design

AMCODE is developed to emulate a business application where the employee will update client records that have a faulty unit. By registrating a new session with information for the test condition, the web application will start to serve client information with a list of products as seen in figure B.1. Each product is represented by a product id, company, ean8 for barcode, price and a unit. All of the information is randomly generated and not of interest, with the exception of product lines highlighted in red. The red highlighted product lines indicate that the supplied unit of three alphabetic characters are not registered and will require to be so. As mentioned all of the data are randomly generated as they are not really interesting. However to create a illusion of a minimalistic application used for work, company and product information are randomly generated and used for filler. Similar to the idea behind Lorem Ipsum [8], where historic language is used for filler text in designs and applications in order to not be distracted by recognizing familiar companies and products.



Company	O'Connell-Bauch
Company Email	kayli42@yahoo.com
Contact Person	Lenora Wiza
Contact Phone	+46 188 456 491

Credit type	Visa	Card Number	55140307796	Card Expiration Date	11/14

Product ID	Company	EAN8	Price	Unit
5684047	Howell, Wilkinson and Koch	36043031	539	gka
4302010	Donnelly-Mayert	25330816	1000	rht
2396588	Strosin-Halvorson	09115679	189	yej
4694229	Witting-Dietrich	97143776	1565	vzx
4983850	Predovic Group	16787100	1466	sde
6063913	Luettgen-Bins	03896693	1535	pho
1610513	Barrows-Mayer	88754419	230	sra
5077151	Bayer-Buckridge	11538907	1887	tuj
0634021	Hoppe-Kub	22821405	544	rdu
5902985	Bosco, Nikolaus and Homenick	89141269	2101	rxn
3901173	King LLC	17366287	278	ebo
3947719	Crona-O'Conner	76219205	333	vzm
6306575	Hodkiewicz Ltd	62762278	2478	sff
5434643	Haag-Botsford	94652714	2198	vzu

Figure B.1: Screenshot from web application.

Appendix C

Hidden timestamps from testing Prototype 1

During the tests the testers where timed when solving the tasks three to seven. The first two tasks where excludes as they were also used to train the tester in how to solve the tasks. The testers where not aware that they were timed during the test and there were no indication that they could be timed. While the timestamps can give some indication, it should also be mentioned that the testers could pause when they were asked questions or derail from the tasks when trying to explain what they were doing.

The results from the hidden completion time are shown in figure C.1 where completion time is represented on the Y-axis and the participating testers are represented along the X-axis. The first blue column in each of the column groups represents the time spent solving the tasks without the prototype. This meaning that the blue graph represents the desired keyboard or mouse to navigate between top-level applications. The red column, represented as the second column in the column groups represents the completion time when using the prototype.

Looking at the graph in figure C.1 and data from the interview logs, we know that testers number two and four preferred to use the keyboard as opposed to using the mouse. The strong preference towards the use of keyboard is clearly reflected in the time spent to complete the tasks, as those two testers used the least amount of time when using the keyboard. Reminding that none of the testers were aware that they were timed during the experiment.

Tester #1 and #3 where the only two whom decreased the time spent on solving the task when the prototype was used. The mean time using the prototype is found with little deviation among the testers. The average time spent using the prototype was 169 seconds where the fastest completion time was 118 seconds and the slowest completion time was 227 seconds. Tester number five did have a 60% angle of view impairment to the right, that would explain the extra time spent to complete the tasks.

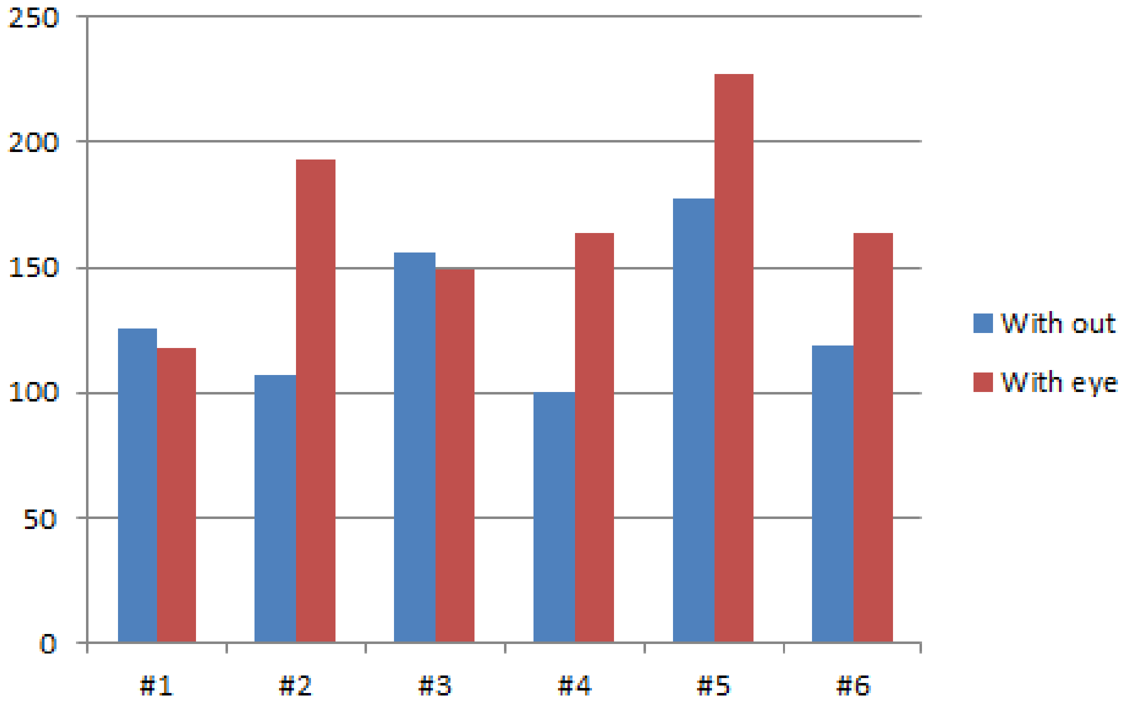


Figure C.1: Bar graph of the hidden completion time. Y-axis is time in seconds and X-axis represents testers. The blue column represents the use of keyboard or mouse and the red column represents the use of eye tracking with the prototype.

Appendix D

Algorithm for Gaze Direction

Listing D.1: Algorithm for determining the direction of the gaze from the selected text field.

```
private GazeDirection gazeDirection = GazeDirection.NONE;
private void tmrSampleDirection_Tick(object sender, EventArgs e)
{
    int dx = 0;
    int dy = 0;
    int thresholdX = 0;     int thresholdMaxX = 30;
    int thresholdY = 0;     int thresholdMaxY = 30;
    gazeDirection = GazeDirection.NONE;

    // Check for directions
    if (xOff > (focusedTextBox.Left+focusedTextBox.Width))
    {
        // right
        dx = 1;
    }
    else if (xOff < (focusedTextBox.Left-thresholdX))
    {
        // left
        dx = -1;
    }

    if (yOff > (focusedTextBox.Top + focusedTextBox.Height))
    {
        // down
        dy = 1;
    }
    else if (yOff < (focusedTextBox.Top-thresholdY))
    {
        // up
        dy = -1;
    }

    if (dy != 0 && dx != 0) // Two directions are given.
    {
        gazeDirection = GazeDirection.TWODIRECTIONS;
        int xSize = 0;
        if (xOff > focusedTextBox.Left)
        {
            xSize = xOff - focusedTextBox.Left;
        }
        if (xOff < focusedTextBox.Left)
        {
            xSize = focusedTextBox.Left - xOff;
        }
        int ySize = 0;
        if (ySize > focusedTextBox.Top)
        {
            ySize = yOff - focusedTextBox.Top;
        }
        if (ySize < focusedTextBox.Top)
        {
            ySize = focusedTextBox.Top - ySize;
        }
        if (xSize > ySize)
        {
            if (dx == 1)
            {
                gazeDirection = GazeDirection.RIGHT;
            }
            else
            {
                gazeDirection = GazeDirection.LEFT;
            }
        }

        if (xSize < ySize)
        {
            if (dy == 1)
            {
                gazeDirection = GazeDirection.DOWN;
            }
            else
            {
                gazeDirection = GazeDirection.UP;
            }
        }
    }
    else
    {
        if (dx == 1)
        {
            gazeDirection = GazeDirection.RIGHT;
        }
        if (dx == -1)
        {
            gazeDirection = GazeDirection.LEFT;
        }
        if (dy == 1)
        {
            gazeDirection = GazeDirection.DOWN;
        }
        if (dy == -1)
        {
            gazeDirection = GazeDirection.UP;
        }

        /*
        // Check for superseding thresholds
        if (xOff > (focusedTextBox.Left + focusedTextBox.Width +
            focusedTextBox.Width + thresholdMaxX) ||
            xOff < (focusedTextBox.Left - focusedTextBox.Width -
                thresholdMaxX))
        {
            gazeDirection = GazeDirection.THRESHOLDMAX;
        }

        if (yOff > (focusedTextBox.Top + focusedTextBox.Height +
            focusedTextBox.Height + thresholdMaxY) ||
            yOff < (focusedTextBox.Top - focusedTextBox.Height -
                thresholdMaxY))
        {
            gazeDirection = GazeDirection.THRESHOLDMAX;
        }

        if (gazeDirection != GazeDirection.THRESHOLDMAX && gazeDirection !=
             GazeDirection.TWODIRECTIONS && gazeDirection != GazeDirection.
            NONE)
        {
            lastKnowGazeDirection = gazeDirection;
        }
        */

    }
    if (GazeDirectionEnabled)
    {
        toolStripStatusGazeDirection.Text = "Gaze Direction: " +
            gazeDirection;
    }
}
```

Appendix E

Intervju guide

Kandidat: **Arbeidsbeskrivelse:**

1) På en skala fra 1 til 10 hvor 1 er lite erfaren og 10 godt erfaren, hvor erfaren føler du deg med datamaskin?

2) Hvilke oppgaver anser du for å være mest tidkrevende?

3) Hvilke oppgaver anser du deg for å være effektiv eller god arbeidsflyt?

4) På en skala fra 1 til 10, hvor 1 er lite og 10 er mye. Hvor stor papirflyt har du som krever at du må bytte mellom å bruke datamaskin og behandle papirer?

4.1) Kan det være andre oppgaver som gjør at du må bytte mellom datamaskin og annet håndarbeid?

5) Har du oppgaver som er repetitive i <u>større</u> eller <u>mindre</u> grad?