Controlling file systems with gaze

Thiago Chaves de Oliveira Horta

University of Tampere
Department of Computer Sciences
Interactive Technology
M. Sc. thesis
Supervisor: Oleg Špakov
June 2011
In recent years, free and open-source gaze tracking systems have become available, making the traditionally expensive technology affordable. Such systems have a range of applications, from research in fields as diverse as psychology, neuroscience and advertisement, to the development of a new interaction modality to reduce the physical strain in mundane activities, to providing control of a computer to severely disabled people. Despite that, gaze tracking technology remains largely unknown to the general public.

This thesis discusses some reasons for the lack of popularity of gaze tracking systems, and proposes that developing a fully gaze-controllable operating system and developing gaze-added interfaces are two good paths to spread the technology. This discussion is enriched by a review of techniques developed to overcome the shortcomings of gaze tracking systems.

In order to make a small contribution to the goal of developing a gaze-controlled operating system, this thesis provides four different file manager designs, which are evaluated in an experiment with the help of twelve participants. The results indicate that a Dasher-like interface may be a poor choice for a file management environment, while a gesture-activated may be a more popular choice.
Summary

1. Introduction ................................................................................................................. 5
2. Designing for gaze interaction ..................................................................................... 7
   2.1. Gaze interaction considerations .............................................................................. 7
       2.1.1. Precision ............................................................................................................. 7
       2.1.2. Speed ................................................................................................................. 8
       2.1.3. Comfort .............................................................................................................. 8
       2.1.4. Desktop behavior is not designed for gaze interaction ......................................... 9
       2.1.5. The “Midas touch” problem ............................................................................... 9
   2.2. Interaction techniques ............................................................................................. 9
       2.2.1. Gaze-only interaction ......................................................................................... 10
       2.2.2. Gaze-added interaction ................................................................................. 10
       2.2.3. Dwelling ............................................................................................................ 11
       2.2.4. Blinking ............................................................................................................. 11
       2.2.5. Eye gestures ...................................................................................................... 12
       2.2.6. Analysis of non-command gaze information ......................................................... 12
       2.2.7. Context-aware interfaces ................................................................................. 13
       2.2.8. Cursor positioning optimization ...................................................................... 13
       2.2.9. Non-discrete target boundaries ....................................................................... 14
       2.2.10. Joystick-like cursor positioning ................................................................... 14
       2.2.11. Expanding targets ......................................................................................... 15
       2.2.12. Zooming functionalities ................................................................................. 15
       2.2.13. Graphical non-WIMP interfaces .................................................................... 16
   3. File manager designs ................................................................................................. 18
       3.1. Orthodox file managers ...................................................................................... 18
       3.2. Navigational file managers ............................................................................... 19
       3.3. Spatial file managers ........................................................................................ 20
       3.4. Three-dimensional file managers ....................................................................... 21
       3.5. Other file manager paradigms ............................................................................ 22
4. Designing gaze-controlled file managers .................................................................. 24
   4.1. Browserous ........................................................................................................... 24
       4.1.1. Dwell-activated interaction mode: Browserous (Dwell) ...................................... 26
       4.1.2. Gesture-activated interaction mode: Browserous (Gesture) .............................. 27
       4.1.3. Continuous interaction mode: Browserous (Continuous) .................................. 28
   4.2. Dasherous ............................................................................................................. 30
5. Experiment design ....................................................................................................... 35
   5.1. Experiment setup .................................................................................................... 35
   5.2. Participants ............................................................................................................ 35
   5.3. Tasks ..................................................................................................................... 35
       5.3.1. Task descriptions ............................................................................................... 35
   5.4. Data collection ....................................................................................................... 37
       5.4.1. Background questionnaire ............................................................................... 37
       5.4.2. Program logs ................................................................................................... 37
5.4.3. Usability questionnaire ................................................................. 37
5.4.4. Interview ................................................................................... 38
6. Experimental results .................................................................................... 39
   6.1. Participants background information .................................................. 39
   6.2. Task completion times ........................................................................ 39
   6.3. Task errors ....................................................................................... 41
   6.3.1. Usability questionnaire .................................................................. 44
   6.3.2. Interview ...................................................................................... 47
7. Discussion ............................................................................................... 49
   7.1. Experiment, data and hypotheses' validation ......................................... 49
   7.2. Software design considerations ............................................................ 50
   7.3. Software quality considerations .......................................................... 51
8. Conclusion ................................................................................................ 52
9. References ................................................................................................ 53
1. Introduction

In recent years, there has been much interest in utilizing information related to where a user directs his or her gaze when interacting with a computer. Techniques that allow one to collect and utilize such information are called gaze tracking techniques. The ability to know precisely the location where a person is looking has several uses, ranging from analyzing gaze paths and evaluating layouts to implementing hands-free control of a computer.

The presence of such technology, as well as its usefulness make gaze tracking a compelling area of research. With the assistance of gaze tracking equipment and brain imaging equipment, a neuroscientist may discover links between visual attention and neural activity. A psychologist may use information of the gaze fixations of a person reading a text to identify confusion. Vehicle producers can provide systems to watch a driver’s eye movements for signs of distraction. Software developers can investigate new interaction possibilities. A survey describes such examples and others in greater detail [Duchowski, 2002].

Examples of currently existing applications of gaze tracking in HCI include utilizing gaze fixation information to assist in the reading of technical texts, such as iDict [Hyrskykari, 2006]; gaze trajectory analysis to refine search results [Kataja, 2011], gaze-controllable text input interfaces, such as Dasher [Ward, 2000]; mouse and keyboard emulation for control of different elements of the desktop and even control of computer games, such as Snap Clutch [Istance, Hyrskykari, Vickers and Chaves, 2009].

There are many different methods to discover a person’s point of visual attention. The methods may barely differ, such as software-only differences using similar algorithms, or they may have a bigger number of differences, including hardware differences, such as requiring one to use head-mounted equipment, often shaped like eyeglasses, or placing cameras beneath the computer screen. Despite the possibility of changing the software while using the same equipment, this thesis will refer to each hardware and software combination as a single gaze tracking system.

Gaze tracking equipment that is worn on the head, such as eye glasses with mounted cameras or helmets are called head-mounted gaze trackers. Head-mounted gaze trackers are often capable of providing precise gaze information at a low outlay in equipment [Dongheng, Babcock, Parkhurst, 2006]. If the interaction is being done with a fixed computer screen, head-mounted trackers will often be very sensitive to head rotation and translation, unless they also gather positioning and orientation information through other means, for example by using scene information from another camera that faces forward rather than facing the user’s eyes.

If the gaze tracker functions by using information from cameras next to a computer screen, then it is called a remote gaze tracker. Remote gaze tracking can be done either with regular web cameras [Zieliński, 2011] or with specialized equipment [Tobii, 2011a]. These trackers may be less sensitive to head rotation and translation, although the positioning of their cameras may either result in lower precision overall, or require higher resolution to obtain the same precision as head-mounted gaze trackers.

The act of utilizing a gaze tracking system to control a computer is called gaze
interaction. Numerous studies have been done on developing methods for gaze interaction to input text into a computer, as well as selecting objects or even browsing websites. Some commercial gaze tracking systems include a number of specialized applications or interface modifications to existing programs in order to allow out-of-the-box interaction with the computer to some extent, such as mouse cursor positioning by eye and automatic scrolling of websites in a web browser [Tobii, 2011b].

Despite the richness of research in the area of gaze interaction, very few gaze-controlled applications as yet exist and, at the time of writing this thesis, there is no operating system equipped with enough support for gaze interaction to provide the same range of functionalities provided by keyboard and mouse-controlled systems. Some of the challenges in the field are understanding how to compensate for accuracy limitations in gaze interaction, developing efficient interaction methods for the various graphical environments' widgets, and providing comfortable interfaces.

The purpose of this thesis is to conduct a preliminary examination of a single problem in the large universe of problems involved in fully controlling the computer: graphical file management.

As the object of study, I chose to design and compare different aspects of four different modes of interaction for file management. three of them inspired from WIMP interfaces, and one inspired from Dasher.

In order to compare the four interfaces, an experiment was conducted. The test measures both quantitative and qualitative aspects of the interfaces, and allowed me to test two hypotheses about how the interfaces compare against each other. The nature of such experiment led to much extra information being gathered during the test, allowing the identification of new research questions for the future.

The hypotheses tested in this thesis were:

- **H1**: A continuous mode of interaction will feel more natural for navigation than a discrete mode of interaction.
- **H2**: A discrete mode of interaction will offer a better sense of control for action activation than a continuous mode of interaction.

In this context, the mode of interaction of an interface is called continuous if the interface produces a gradual change from one state to another as the process of confirming an action. An interface with a discrete mode of interaction is one that produces immediate changes from one state to another once an action is confirmed.

This thesis is divided into six chapters. Chapter 2 makes a review of current gaze interaction techniques and their characteristics. Chapter 3 describes file managers and the different design paradigms that they follow. Chapter 4 describes the design of four gaze-controlled file managers. Chapter 5 describes the experiment set-up and the data collected. In chapter 6, we view the experimental results. Chapter 7 discusses the meaning of the experimental results and makes a decision on the validity of the hypotheses. Finally, in chapter 8 a review of the thesis, as well as the final thoughts on the subject are presented.
2. Designing for gaze interaction

There has been some progress in the design of interfaces for gaze interaction, but there is still progress to be made. At the time of writing this thesis, one cannot simply order an operating system fully controllable by gaze, or at least not without paying several thousand euros for the system.

In order for full gaze tracking operating systems to emerge and become affordable, there needs to be greater diversity of gaze interaction software.

Traditional challenges in the development of gaze interaction software have been the complexity of gaze tracking software, the high cost of gaze tracking equipment, the lack of computing power in personal computers, the unfamiliarity of using one's eyes to control things, and the lack of support for gaze trackers as input devices on standard operating systems.

Fortunately, some of these challenges have already been met. There are solid open source gaze tracking algorithms and equipment designs available on the Internet, such as openEyes [Dongheng, Babcock, Parkhurst, 2006], Opengazer [Zieliński, 2011], and ITU Gaze Tracker [Agustin, Skovsgaard, Mollenbach, Barret, Tall, Hansen, Hansen, 2010], greatly reducing the complexity and cost of developing for a gaze-controlled environment. In addition, computing power on personal computers is quickly catching up with the requirements for running a gaze tracking system.

When designing an interface for gaze trackers, one must take into consideration a number of issues that do not arise in traditional keyboard and mouse interfaces, or, when they do, they arise in a different way. The following sections describe such gaze interaction issues and propose a categorization of interaction techniques that have previously developed for gaze interaction.

2.1. Gaze interaction considerations

This section lists some of the main issues to be considered when developing gaze interaction interfaces. Some information is given on the inherent lack of precision of gaze trackers. Also briefly discussed here is gaze interaction speed and comfort. Finally, this section describes the limitations of traditional desktop environments for gaze interaction, and introduces the Midas touch problem.

2.1.1. Precision

Precision in eye interaction is a combined result of the accuracy of the gaze tracking system and the design of the software being utilized. A precise interaction is one that results in few or no errors. The level of accuracy of a gaze tracking system describes the level of certainty that the system is able to estimate of the user's gaze point, measured in degrees of an arc. The accuracy limit for gaze tracking systems is estimated to be 1° [Jacob, 1991].

The variety of gaze tracking systems and their differences in accuracy result in an interesting trade-off problem, which weighs the level of precision that one will assume the gaze tracker will provide against the range of gaze tracking systems that one desires
to support.

Assuming that the gaze tracking system is highly accurate allows the designer of a software program to include smaller selectable objects on the screen or use narrower gaps between objects. There is a risk, however, that making such design choices will reduce the user's choice in gaze tracking equipment, or induce the user to commit a larger number of errors.

If a software developer wants to support a larger number of gaze tracking systems, then s/he must allow for larger objects and wider gaps between objects. This will make it easier for users to unambiguously select objects, at the cost of utilizing a larger area of the screen or displaying fewer objects at once. The choice for larger objects and wider gaps between objects may also be motivated by a desire to improve the precision of the interaction with the software regardless of the gaze tracking system utilized.

2.1.2. Speed

Another important aspect to be considered when designing an interface is how fast it allows a user to accomplish a task. This may depend on many factors, such as how fast the interface reacts to user input, how precise the interaction with the interface is, and how quickly one can correct errors.

Much research in the design of interfaces for gaze tracking provides some interaction speed comparison between different eye interaction input methods and traditional input methods [Hansen, Johansen, Hansen, 2003], [Bednarik, Gowases, Tukiainen, 2009].

More often than not, it is the case that eye interaction methods are significantly slower than traditional keyboard and mouse methods. However, predictive systems may be able to reduce the speed gap between the modalities for certain domain-specific tasks. Dasher [Ward, 2000] is one example of the utilization of predictive systems in a text-entry system compatible with gaze tracking systems.

2.1.3. Comfort

An aspect closely related with the speed of interaction is how comfortable the interface is. Does the utilization of the interface feel smooth and easy, or is it hard to use, cumbersome or otherwise tiring? Making sure an interface is easy and comfortable to use may be key in promoting its adoption. The comfort of an interface is, however, difficult to measure.

Typically, interaction speed is an important factor that defines how straining an interface is. If interaction with the interface requires many steps or long steps to provide little amount of input, the interface will usually be considered cumbersome. In order to offer the user a comfortable interaction, gestures need to be designed to allow efficient input, and keep movements to a minimum.

The relationship between interaction speed and comfort may be different in gaze-only systems, however. While it is straightforward to estimate a user's object of interest from the gaze fixation, it is harder to estimate the user's intentions towards the object. A gaze-only interface that reacts too quickly to user input may cause many unintended actions to take place, reducing the feeling of comfort.
2.1.4. Desktop behavior is not designed for gaze interaction

Finally, one must decide what kind of desktop environment is being used. Traditional desktop environments are ignorant of gaze information and provide widgets and behavior that are suboptimal for gaze interaction.

In a standard desktop environment, the activation area of some objects may be too small, or tightly packed together. Moreover, much of the functionality may be accessible only by right clicks, double clicks, or click-and-drag gestures.

Unfortunately, developing a new desktop environment for gaze interaction is a very long process. The development of a desktop environment is very costly and takes a long time to be completed, so performing an adaptation of existing environments might be a more realistic goal.

In order to adapt a desktop environment for gaze interaction, a developer needs to be able to configure the size of objects in the desktop, and decide how will the user's intentions be translated into actions by the system, and, obviously, prepare the environment to receive a gaze tracking system as one of its primary methods of input.

2.1.5. The “Midas touch” problem

A further issue confronting the developer is the “Midas touch” problem. This is the act of involuntarily interacting with the interface when one simply desires to look at the interface or things in the interface [Jacob, 1991].

This is a problem characteristic of some interfaces designed for pointing devices, but it is an especially serious concern with regard to the design of gaze interaction interfaces. When the eyes are allowed control over the computer, there is the possibility of accidentally producing non-intended actions by simply looking around in the screen.

A person developing gaze interaction software must consider carefully the methods of action activation chosen for the interaction, and balance between the ease of producing them intentionally, and avoiding producing them unintentionally.

Some methods have been developed to minimize, or even avoid the problem. In Snap Clutch [Istance, Bates, Hyrskykari and Vickers, 2008], a method was developed where the interface would switch between different modes when the user would look away from the screen. The interaction is chosen according to the direction in which the gaze escaped from the screen area. One of the modes included was a look-only interaction mode, where no action activation was possible.

2.2. Interaction techniques

There is a wide set of techniques that are currently applied in gaze interaction with the desktop. The techniques range from different ways to perform actions, to methods to improve object targeting, to utilization of predictive systems that utilize context information to facilitate the execution of common or probable tasks.

A preliminary study of those techniques, their advantages and limitations may significantly improve the gaze interaction system or software. Many of the techniques can be combined in order to provide a better interaction experience.
2.2.1. Gaze-only interaction

The most common concern with gaze interaction is making it fully gaze controllable. That is, no interaction with a mouse or keyboard would ever be needed. In this context, there is no keyboard and mouse to activate an action. These forms of interaction have necessarily to compensate for the lack of an activator button in the physical world.

Usually, that results in selections clicks being generated by dwelling and gesturing, or keyboard behavior being emulated by gestures or on-screen keyboards. Usually, the fact that the user's gaze is used for pointing, examining and initiating actions results in this type of interface being particularly vulnerable to the Midas touch problem. For this reason, gaze-only interfaces must be carefully planned.

Despite its vulnerability to the Midas touch, gaze-only interaction makes for a compelling area of research. For such reason, much research has been done developing gaze-only interfaces.

EyeChess, a gaze-controlled chess application, provides the ability to perform selections either by dwelling, performing eye gestures, or blinking. Although all these interaction modes are examples of gaze-only interaction, they operate differently [Špakov, 2005].

IGO, Intelligent Gaze-added Operating system, is a file manager prototype that provides three modes of interaction, one of which is a gaze-only interface [Salvucci and Anderson, 2000].

More examples of gaze-only interaction are presented further in this chapter, as they are relevant to other interaction techniques discussed here.

2.2.2. Gaze-added interaction

An alternative to designing gaze-only interfaces is the utilization of gaze information in combination with traditional keyboard and mouse control methods. Gaze-added interfaces may either split the interaction responsibilities between the eyes and the keyboard and mouse, or allow some overlap between what is possible to do with each modality.

Utilizing the gaze in conjunction with keyboard and mouse provides gaze interaction methods with a greater robustness to the Midas touch problem. If a user needs to confirm actions by pressing buttons on the mouse or keys on the keyboard, there is very little chance that actions will be initiated without the user's intention to start it.

Alternatively, gaze-added modes of interaction may be utilized in order to improve the quality of standard keyboard and mouse interaction. In such a case, the user's gaze may be utilized to reduce the amount of work required to manipulate the mouse, thus reducing the physical strain on the mouse hand [Zhai, Morimoto, Ihde, 1999].

IGO [Salvucci and Anderson, 2000], mentioned in the previous section, provides a gaze-added interaction mode, in which gaze is utilized as a pointing device, and the ctrl key on the keyboard is utilized as an activator. IGO will still be discussed further, as it provides other interesting features for gaze interaction.

Kumar and Winograd utilized information about the user's gaze point before performing a page-up or page-down operation in order to indicate to the user the position in the text.
that her eyes were. In addition to creating a method to help users not lose track of their position in a text, they experimented with three different gaze-controlled scrolling methods. Their method allows for the toggling of the gaze scrolling via the Scroll Lock key on the keyboard. [Kumar, Winograd, Paepcke, 2007]

Since this interaction mode may improve interfaces already in widespread use, it offers the greatest opportunity to popularize gaze tracking systems.

2.2.3. Dwelling

The most common form of interaction for performing selections or activation of widgets is the dwell interaction. The user looks directly at the object s/he wants to interact with, and if the user's gaze lingers long enough on any particular object, that object is selected or activated.

This method is simple in both usage and implementation. A new user will typically need very little instruction in the interaction mode before s/he understands how to use it, and the programming required in order to enable such a form of interaction is relatively basic. These characteristics make this interaction method popular among many studies [Ohno, 1998], [Miniotas, Špakov, and MacKenzie, 2004], [Barcelos and Morimoto, 2008].

Despite its simplicity in understanding and implementation, there are some pitfalls in the usage of the dwell interaction. Namely, this method is particularly vulnerable to the Midas touch problem. This is due to the fact that, to the software, there may be no difference between the user's actions of examining and activating. This aspect of the interaction mode may make it uncomfortable to the user.

The Quick Glance Selection Method [Ohno, 1998] separates an object from its selection area in order to reduce the Midas touch problem in dwell interaction. The method allows for a two-step, gesturing activation of objects, or a one-step, dwell-only activation of objects. The one-step activation of objects can be performed by dwelling on an object's selection area, rather than the object itself.

2.2.4. Blinking

Utilizing blinking as a method of activation is one of the earliest ideas in gaze interaction. Blinking-activated methods utilize deliberate blinks in order to confirm an action. This introduces the problem of distinguishing between involuntary and voluntary blinks.

The distinction between voluntary and involuntary blinking is often made by defining a longer duration for the activation blink than for a common involuntary blink. Long blinks, however, causes the eyes to lose convergence [Huckauf, Urbina, 2008]. In addition, the user cannot examine the screen while blinking, thus reducing the opportunities for using any techniques for facilitating target acquisition that requires the user's attention, such as expanding targets and zooming functionalities. Expanding targets and zooming functionalities are discussed in further sections of this chapter.

While blinking as selection method does not seem promising, there may be other uses for blink detection, such as calibration correction. A study by Ohno et al. proposes a
gaze tracking system that utilizes both a gaze tracking unit and an eye positioning unit. The eye positioning unit detects user blinking in order to obtain an up-to-date location of the user's eyes. This up-to-date information is utilized to perform automatic calibration correction. [Ohno, Mukawa, Kawato, 2003].

2.2.5. Eye gestures

By utilizing gesture recognition methods, a gaze interaction interface may utilize eye gestures, rather than dwelling or blinking in order to initiate actions. In this context, gestures are represented by paths that, once correctly traced by the user, activate actions. Gestures can be local, performed in a particular area such as a gesturing window, or global, performed across the whole screen in order to activate an environment action or confirm an action after an object was selected.

A study [Isokoski 2000] investigated a gaze interaction implementation of the Minimal Device Independent Text Input Method (MDITIM). The interface allowed characters to be written by performing off-screen fixations in sequence. This technique allows for text entry to be performed without covering any regions of the screen with an on-screen keyboard.

EyeWrite, a gaze-controlled text entry application, is operated by local gaze gestures. In order to write, the user must perform unistroke character-like gaze gestures within a small window. The interface is able to operate without covering a large region of the screen with an on-screen keyboard. [Wobbrock, Rubinstein, Sawyer, Duchowski, 2007]

The utilization of gaze gestures to initiate actions has the benefit of providing the system with robustness to the Midas touch problem. The act of following a gesture path needs the user's explicit intent, and is hardly performed by accident.

In addition to being resilient to the Midas touch problem, eye gestures make it possible to support a multitude of actions without the requirement of an interaction mode switching system, or other input devices, such as keyboard and mouse. Snap Clutch, for example, utilizes quick off-screen glances to switch between interaction modes [Istance, Bates, Hyrskykari and Vickers, 2008].

The possibility of utilizing diverse gestures to enable a multitude of actions to be performed comes at a price. Large quantities of gestures may impose a significant load on the user's memory and complicate the training process for the interface, as well as reduce the system's reliability in correctly interpreting the user's gesture.

2.2.6. Analysis of non-command gaze information

Sophisticated analysis of gaze data may provide the interface with useful information about the user's intentions and expectations. The user's gaze paths and fixations may offer hints of the user's interest, confusion or other states of mind.

The utilization of non-command gaze data allows for interaction improvements, such as removal of irrelevant items from an image search, or implementation of an automatic help system.

GaZIR [Kozma, Klami and Kaski, 2009], a gaze-based interface for image retrieval,
makes a statistical analysis of the user's gaze patterns while s/he searches for a particular image and utilizes the information to refine the results of the search.

iDict [Hyrskykari, 2006], a text reading software, provides help to a user who is reading a text in a foreign language. When iDict detects that the user is confused by a certain expression, the system presents a translation to the user. The confusion is inferred from gaze fixations longer than what is necessary to read the word.

2.2.7. Context-aware interfaces

Similarly to non-command gaze information, context information may also be utilized to improve interaction with an interface. The word context here may mean a large number of things, from the history of the user's actions, to knowledge of intrinsic characteristics of a particular problem, to information about the widgets on the interface and their state of activation.

Utilization of a history of the user's commands is widespread in keyboard and mouse applications. Text editing programs may provide quick access to recently opened files, music players may provide a list of songs most often selected by the user, an internet browser may offer to automatically complete an address being typed at the address bar if the address resembles one of the pages the user has visited.

In certain areas, knowledge of intrinsic characteristics of the language allow the design of input methods that greatly outperform an interface that is agnostic to such characteristics. Text entry systems, for instance, benefit greatly from the utilization of language models.

Finally, a developer may be able to identify action sequences that make little sense, and prevent the user from doing them, or alternatively warn the user about it. Examples of such action sequences are: trying to utilize deactivated objects, trying to delete system files, or trying to copy more data into a removable media than fits in its storage space. The interface may helpfully display disabled objects in a different manner to differentiate them from the active ones, or provide warning dialogs for actions that may prevent the system from operating.

GiNX, Gaze based Interface Extensions, utilized information about widget position, task and user behavior [Barcelos and Morimoto, 2008]. The system distinguishes between active and inactive widgets in order to improve mouse cursor positioning. In addition to that, the system utilizes a user model in order to predict expected targets.

IGO in its gaze-only interaction mode utilized context information in order to give probable menu options and certain widgets a higher probability of being selected. For example, the file menu increased its probability of being selected right after a file was opened or selected, modeling the fact that a user will normally be interested in the menu after selecting or opening a file or folder. [Salvucci and Anderson, 2000]

2.2.8. Cursor positioning optimization

Another problem that may arise from utilizing dwell to generate selections and activations is the difficulty of fixating the gaze at a single point. This may be due to a number of reasons: the gaze tracking system may be inaccurate, the objects may be too
small, or the dwell time too long.

When the gaze tracking system is inaccurate, the gaze cursor may produce jitters that are distracting to the user. In some cases, a user may feel tempted to follow an inaccurate gaze cursor and drift away from the intended target.

Small targets, due to being intrinsically difficulty to target, may be a problem independently of how accurate a gaze tracking system is. Many applications, however, rely heavily on small icons in order to save screen space for other more important widgets.

Some problems may be alleviated through the use of gaze cursor smoothing algorithms. In their simplest form, a gaze cursor smoothing algorithm takes an average of the n most recent gaze samples and uses that value as the position for the gaze cursor.

In addition to that, techniques such as snapping the gaze cursor's position to the center of a nearby object lessens the user's effort of concentrating on the intended target [Tien and Atkins, 2008].

2.2.9. Non-discrete target boundaries

Another idea for improvement of targeting of objects in gaze interfaces is the utilization of non-discrete boundaries for the objects' active areas. One peculiarity of this concept is the fact that active areas of different objects may overlap.

When the user's gaze enters the active region of one or more objects, a probability function may be utilized to decide whether or not the user is focusing on a particular target.

Alternatively, each one of the objects whose active areas contain the gaze cursor may accumulate gaze score based on how likely it is that the object is the user's intended target. In a dwell-activated interaction, an object could be activated if its gaze score reaches a threshold before the other objects.

A recent study by Špakov compares the accuracy of a number of algorithms designed to disambiguate the user's intended target, given the inherent inaccuracy of gaze tracking. The study suggests that two methods named Fractional Mapping and Dynamic Competing have good potential in facilitating selections, although both algorithms may need to be refined in order to reduce their rate of incorrect selections [Špakov, 2011].

2.2.10. Joystick-like cursor positioning

In most gaze tracking systems and interfaces, the user's gaze point is the approximate position of the gaze cursor, but there are other possibilities in making the conversion from gaze point to a gaze cursor position. Some gaze tracking systems may provide a joystick-like behavior for the gaze cursor instead.

A gaze tracking system in this category will not convert the user's gaze point directly into a gaze cursor's location, but rather utilize that information to set a gaze cursor's heading and speed in that direction.

Joystick-like behavior for gaze cursors in gaze tracking systems is very rare. This is due to the fact that pointing directly to the location that the user is looking is very intuitive
and allows for very quick pointing. This interaction method remains largely unexplored for gaze interaction, however, and research on it may reveal domains where a joystick-like behavior is adequate or even preferable.

I4Control [Fejtová, Fejt, Štěpánková, 2006] is a gaze tracking system that utilizes a head-mounted eyeglasses hardware design, equipped with one single camera and one control unit that performs simple image analysis to identify the general direction of the user's gaze. The setup is intended to allow for out-of-the-box utilization and a very low price for a gaze tracking system.

### 2.2.11. Expanding targets

In a context like gaze interaction, where pointing at small objects is difficult, the idea of making objects larger soon comes to mind. Some gaze interaction methods may change the size of objects dynamically in order to facilitate target acquisition. Such techniques may be able to reduce the compromise between ease-to-point and screen space utilization.

Target expansion can be utilized to disambiguate the user's intentions. For example, if the user fixates her gaze in an area with many smaller objects packed together, expanding the probable targets will increase the distance between object. This action can make identification of the correct target easier and faster [Miniotas, Špakov, and MacKenzie, 2004].

A system of dynamically expanding targets has been utilized in menus [Špakov, Miniotas, 2005]. In this system, the expansion occurs in such a way that the most probable target stays in place, and the other menu options are shifted up and down to give space to the target. In the occasion that the probable target is not the intended target, the user may shift her gaze to the target's new position in order to correct the selection.

### 2.2.12. Zooming functionalities

Another method to tackle the problem of pointing at small targets or compensating for lack of precision is the use of a zoom functionality. By zooming into an area of interest, an interface will magnify probable targets, making the intended target easier to acquire. [Bates, Istance, 2002].

Despite the distinction in this thesis between expanding targets and zooming interfaces, there is an intersection between the categories. In order to differentiate, this work will consider techniques that result in an unequal share of expansion among probable target candidates as part of the expanding targets category.

Zooming into a region of the interface makes selection easier by expanding the objects' selection area and the distance between objects, at the potential cost of occluding other regions of the interface [Hansen, Skovsgaard, Møllenbach, 2008].

In StarGazer [Hansen, Skovsgaard, Møllenbach, 2008], a gaze-controlled text-entry program, the letters on the alphabet and some special function keys are organized into concentric circles. Initially the targets may be too small to acquire, but once the user looks in the direction of the intended key the interface will start panning and zooming.
This process increases the size of all targets, while occluding non-relevant objects and bringing the more probable candidates to the center of the screen.

Snap Clutch provides a magnifying glass tool to facilitate the selection of small icons in games [Istance, Hyrskykari, Vickers and Chaves, 2009]. In order to access the magnifying glass tool, the user must first dwell on a box containing the magnifying glass icon in order to acquire the magnifying glass. A second dwell will drop the magnifying glass at the user's point of gaze. After that the user may continue interacting within the magnified area, or outside it. Dwells outside the expanded area will cause the magnifying glass to shift its center to the new gaze location.

2.2.13. Graphical non-WIMP interfaces

Research in gaze interaction sometimes leads to graphical interfaces that do not fit the well-known WIMP (Window, Icon, Menu, Pointing device) design paradigm. Instead, those researchers utilize different designs, with different sets of basic graphical building blocks.

Research that utilizes non-WIMP interface designs may be motivated by the desire to break limits imposed by WIMP interfaces, explore new paradigms of interface design that may supplement and improve interaction with WIMP interfaces, or provide an environment that is more adequate for gaining insight on user behavior.

Non-WIMP interfaces may inspire from metaphors such as "navigating in a tunnel", or "flying in a classroom", or they may have more abstract design concepts, leading to more unconventional interaction methods. The non-conventional operation method of such interfaces may come at a cost of intuitiveness and an increased difficulty of adapting to the interface. In order for a new paradigm of interface to survive, it must often outperform conventional interfaces in efficiency, convenience, comfort, or possibly in all of those characteristics.

Dasher [Ward, 2000], a text-entry system with support for gaze-only interaction, is operated via a stream of characters, that move from the right corner of the program's window to the left corner. Characters move closer to the vertical center as they are looked at, and once they cross the horizontal center of the screen, they are selected. The program utilizes a language model to predict probable next characters and prioritizes those over less likely ones, making them easier to choose.

In Dasher, the speed of interaction may be controlled by the user, by moving the gaze away from the center of the screen. The further the user looks to the right, the faster the typing can be done. In order to erase text and undo actions, the user may look at the left corner of the screen, which causes letters to return to the right side of the screen.

The unusual appearance and interaction protocol of Dasher result in a longer learning process than on-screen keyboards. Despite its disadvantages, Dasher may be one of the fastest methods for gaze-only text-entry [Ward, MacKay, 2002]. The interface, however, might benefit from adjustments when utilized with languages other than English [Tuisku, Majaranta, Isokoski, Räihä, 2008].

GaZIR [Kozma, Klami and Kaski, 2009], a system for retrieving images, shows the user three concentric rings of images. The images in the rings are scaled up when they move...
away from the center, and scaled down when they are closer to the center. The user can cause the interface to zoom in and out. Zooming in will cause more image results to be fetched and displayed into a new ring in the center, and zooming out will cause the interface to restore image rings that were zoomed off-screen.

The tunnel-like interface of GaZIR is designed to modify the user's behavior. While a user would traditionally search for images from top-to-bottom, and left-to-right manner, the circular arrangement of images in GaZIR makes such searching method unfeasible. This design choice was made in an attempt to increase the importance of the images' contents in relation to the images' arrangement in the user's search patterns, and provide better gaze path information to a predictive system.

StarGazer [Hansen, Skovsgaard, Møllenbach, 2008], described in the previous section, is another example of non-WIMP interface.

Non-WIMP designs for gaze interaction interfaces have a potential of introducing game-changing software. Their development, however, requires insight, and the development of their graphical side may be fairly more complex, given that they may lack support from well-known programming libraries for widget interaction.
3. File manager designs

A file manager offers a rich environment for examining the comfort and efficiency of gaze interaction techniques.

The tasks that one may perform with a file manager offer a number of challenges to the developer. A typical file manager allows one to create, erase, rename, copy and move folders and files, as well as setting read/write permissions and sharing options. Such tasks vary in the amount of searching and actions required, as well as the nature of the actions required.

A file manager may vary in its appearance and modes of interaction. Before graphical user interfaces were commonplace, there were text-based shells. Currently there are several graphical file managers utilizing varying interface designs.

An on-line article [Wikipedia: File manager, 2011] classifies file managers in seven categories: directory editors, file-list file managers, orthodox file managers, navigational file managers, spatial file managers, three-dimensional file managers, and web-based file managers. Due to the lack of popular graphical standalone file managers in the first two categories, and the fact that we are mostly interested in the desktop environment in this thesis, only orthodox, navigational, spatial, and three-dimensional file managers will be considered.

This chapter of the thesis should be read with the understanding that this section of the text is not based on a formal study. The purpose of this chapter is merely to offer a glimpse in the variety of file manager designs, and provide an intuition-based analysis of the potential of such designs for use with gaze tracking systems.

3.1. Orthodox file managers

File managers in this category may also be called command-based file managers, or twin-panel file managers. This type of file manager offers a list-view of two locations of the file system at once, displayed side-by-side in two panels at the top of the window. A panel at the bottom of the screen provides help and the means for the user to issue commands to the interface.

This file manager offers the benefit of allowing multiple-location operations such as copy and move to be performed without a need to issue a "paste" command. This characteristic of the orthodox file managers might be useful in improving the interaction speed on keyboard and mouse interaction, as well as gaze interaction, though no studies on this topic were found by the author of this thesis at the time of its writing.

The two-panel solution comes at the cost of screen space. Since two locations are shown at the same time, there is less space on the screen to display each view. While desktop computer display size and resolution has increased significantly since the creation of the first orthodox file managers, their popularity has diminished since the advent of navigational file managers.

An example of a current orthodox file manager, GNOME Commander, is given in Figure 1. In addition to the two file-system view panels and the command panel at the bottom, icons at the top are provided to promote the possibility of performing mouse-
only interaction with the program.

Figure 1: GNOME Commander, an example of an orthodox file manager. This image was retrieved from https://secure.wikimedia.org/wikipedia/en/wiki/File:GCMD-screenshot.png, and used with permission

3.2. Navigational file managers

Navigational file managers are often called Explorer-like file managers, due to Windows Explorer being the most popular of the navigational file managers. This style of file manager only showed up after the advent of graphical user interfaces (GUIs).

Interfaces in this category emphasize on the ease of navigation between folders, often offering multiple ways to navigate. A user may revisit a previously seen location by pressing a "back" button, access a parent folder via the current path display widget, or go to other file-system locations by utilizing a tree-view located on the left hand side of the window. It may be also possible to visit a file or folder by directly typing the address on a location bar.

The emphasis on ease of navigation that is given on this type of file manager has the potential of offering a good quality of interaction in gaze-controlled systems. A user may appreciate an interface that allows him or her to undo steps in navigation, go to distant locations of the file system with a single gesture, or type a location or query via through a text-entry method optimized for file searching.

Another characteristic often shared by navigational file managers is the utilization of icons to represent the files and folders in the section of the window that displays the current location. This representation may offer as little distinction as an icon to represent folders, and another to represent all other files, or utilize a large collection of
icons to distinguish between folder and file types, or even previews to certain files.

File managers that are built on the navigational paradigm may vary greatly in appearance. Not all of the characteristics listed in the two previous paragraphs are necessarily available in every navigational file manager.

Figure 2 shows an example of navigational file manager, Dolphin. A tree view representation of the file system can be visualized at the bottom-left corner of the image, at the center-left a quick access to bookmarked folders can be seen. Images files are displayed as previews, rather than icons.

![Figure 2: Dolphin, an example of navigational file manager. Image retrieved from https://secure.wikimedia.org/wikipedia/en/wiki/File:Dolphin_FileManager.png, and utilized with permission.](image)

3.3. Spatial file managers

Spatial file managers are characterized mainly by their behavior of tying each window of the program to a single location in a 1:1 relationship. In other words, a single window cannot display more than one location's contents at any given time, and no location can be displayed by more than one window at a time.

The second main characteristic in spatial file managers is their ability to preserve user-defined location and scale information for any object's icon. This serves the purpose of allowing the user to organize objects within a single folder independent of their alphabetical, date of last alteration, type information or other common automatic organizations that other file managers may provide.

In principle, the characteristics of this type of file manager do not seem particularly interesting or useful for gaze interaction software design. The personalization
capabilities of such file systems, however, may provide interesting challenges for gaze interaction. How to properly provide drag-and-drop operations or user-defined scaling of objects in a gaze-controlled environment?

An example of spatial file manager is given on Figure 3. Two different locations on the file system are displayed in two different windows. The user would need to open a third window to be able to visualize three locations at once.

![Figure 3: ROX Filer, an example of spatial file manager. Image retrieved from https://secure.wikimedia.org/wikipedia/en/wiki/File:Rox-desktop-2004.png, and utilized with permission. The image was cropped and its background erased for improved readability.](image)

While spatial file managers are not as popular as navigational file managers in desktop computers, their customization capabilities are often incorporated on the background area of environments such as Microsoft Windows, Mac OS, Gnome and KDE, in the form of quick access icons for applications, files, folders, and links. Often these quick access icons may be resized and reorganized individually at will.

### 3.4. Three-dimensional file managers

The article that served as basis for most of this chapter [Wikipedia: File manager, 2011] reserves a single category for three-dimensional file managers. Despite this categorization choice, file managers in this group may vary greatly, from an alternative evolution branch on navigational file managing, to physical space metaphors of the office desktop.

Currently, three-dimensional file managers are not popular. The reason for this may either be their higher graphic processing requirements, or a lack of maturity in design of such file managers. There are, however, a few current interesting projects. If they succeed, they might set the standard for the next most popular file manager type. Some notable three-dimensional file managers are BumpTop [BumpTop, 2011b], NavScope [NavScope, 2011], and Real Desktop [Real Desktop, 2011].

BumpTop, recently acquired by Google [BumpTop, 2011a], and Real Desktop are based on a physical space metaphor of the office desktop environment. In a sense, these file managers are an extension of the spatial file managers: they support spatial organization of files and folders, grouping of objects, scaling of objects, as well as physics effects such as collision detection. BumpTop is currently non-available to the general public.

While the first two examples utilize a spatial metaphor, NavScope is closer to the
navigational file managers. Its interface provides tree views, and an address bar, as well as the display of the contents of more than one location at once. Its preview of a folder's contents differs from that of a two-dimensional navigational file manager, however. Rather than occupying the whole folder view pane with the preview of a single folder, many folders can be seen at once, and their views can be adjusted by scrolling, zooming, or changes in the layout. The folder previews may be either stacked or organized in lines and columns.

While three-dimensional file managers are not popular yet, they might become in the future. So far, the most mature file managers in this category are strongly based on the spatial or the navigational metaphors. A three-dimensional environment, however, may provide opportunities for new file manager paradigms. The ease of implementation of certain operations, such as scaling, zooming and panning in three-dimensional environments may turn out to be useful for the development of gaze-controlled file managers.

3.5. Other file manager paradigms

Despite providing a good starting point for reviewing existing file managers and the different paradigms. The Wikipedia article in its current form fails to provide a category that would include certain experimental file managers such as VennFS, a Venn-Diagram file manager [Chiara, Erra, Scarano, 2003], seen in Figure 4.

Figure 4: VennFS provides organization of documents according to categories and displays them in Venn-diagrams, rather than folders. This image is taken from the article by Chiara et al.
VennFS was developed with the intent of breaking one single tradition of file managing that pervades all file managers previously mentioned in this chapter: hierarchical organization of the file system.

In order to distance from the hierarchical structuring of the file system, VennFS provides a tagging system, which allows users to organize documents according to categories, and place documents in more than one category at once. The categorization of documents is visualized in the form of Venn-diagrams.

The interface of VennFS seems in some aspects similar to spatial file manager interfaces. Specifically, objects are organized in a user-defined manner around a two-dimensional space, and its categories may differ in size. Gaze interaction with such kind of interface could, therefore, benefit from advances made in spatial file managing environments.
4. Designing gaze-controlled file managers

The motivation of this thesis is evaluating advantages and disadvantages of a number of interaction techniques in the context of file management. For this purpose, two different gaze-controlled file managers were developed for the experiments: Browserous, a file manager with three different modes of interaction, and Dasherous, a file manager with one mode of interaction inspired from Dasher.

The gaze-controlled file managers were developed in the Python programming language, and with support of the Pygame, OpenGL, Cairo and GIO programming libraries. GIO was used to collect MIME-type of files from their path information. Cairo and Pygame were used to load MIME-type icons for the Gnome desktop environment. OpenGL was used to render the graphics.

The two file managers use the concept of tree and nodes to model the file-system. On top of that, nodes are also used to represent actions menus and actions. Each folder and each file has its own actions menu. Interaction with actions menu nodes and action nodes is the same as interaction with file and folder nodes.

The actions menu is a special node that contains action nodes. Which actions it contains depends on what is the parent node of the menu. If the parent node is a file, then Open, Delete and Copy are available. If the parent node is a folder, then Copy, Paste and Delete are available. The Delete option is not available for the root folder. The parameter of the actions is dependent on the file or folder node that contains the actions menu where the action is found. In order to delete Text1.txt you have to go to Delete in Text1.txt's actions menu, while you would need to go to the actions menu in Text2.txt if you wanted to delete that file instead.

The actions menu has a target node. The target node of its actions is its parent node. In other words, the node it belongs to. When a user selects “ExampleFile.exe -> Actions -> Open” the program understands “Open ExampleFile.exe”.

4.1. Browserous

Browserous is inspired both by navigational and spatial file managers. Like spatial file managers, it displays files and folders as objects to be interacted with, and allows the visualization of the contents of one single folder at any given time. Figure 5 shows an image of Browserous (Dwell) displaying the contents of a node.

The contents of a folder node are displayed in the following order: A “go back” button, an actions menu for the current folder, the folders contained in the current position, and the files that the current node contains. The “go back” button, when activated, brings the user to the parent node of the current node.

The actions menu of a folder offers a set of actions that can target the current folder. Those actions are “new file”, “new folder”, “copy”, “paste” and “delete”. “Copy” places a copy of the current folder in the clipboard, and “paste” pastes the contents of the clipboard onto the folder. Each action is displayed along with a specific icon, as seen in Figure 6. The “go back” button allows the user to leave the actions menu. The “copy” and “delete” actions are not shown for the root folder's actions menu.
Figure 5: Browserous (Dwell) displaying the root node's contents.

Similarly to folders, files also have their own actions menu. Unlike folders, there is no explicit "actions menu" button for a file. Instead the actions menu for a file is displayed as soon as the user navigates inside a file. Figure 7 displays the actions menu of a file. Available actions for a file are "open", "copy" and "delete". The "go back" button allows the user to leave the file and see the contents of its parent node.

Figure 6: A folder's path and its actions menu in Browserous. The "Paste" action is shown as disabled because the clipboard is empty.

Like many navigational file managers, Browserous provides a visualization of the user's current position in the file-system as a path at the top of the screen. Though unlike many of the keyboard-and-mouse file managers, interaction with the visualized path is not possible. One example of path can be visualized at the top-left corner of Figure 6, which shows root/Program Files. Another example can be seen in Figure 7, where the path is root/Program Files/LibreOffice.zip.

Opening a file in Browserous does not cause the file to be opened, instead it just provides a visual feedback. The word "opening" followed by the file's icon and name are displayed at the center of the screen while the program exits the current file to display its parent node. Figure 8 shows an image of a music file being opened in Browserous.
Browserous can operate in three different modes: dwell-activated, gesture-activated and continuous. Both the dwell-activated and gesture-activated provide a "Go back" action, while the continuous mode doesn't need one.

In order to explain the usage of Browserous, we must first clarify that there is a distinction between select and activate. In the context of Browserous, selecting does not necessarily mean activation.

4.1.1. **Dwell-activated interaction mode: Browserous (Dwell)**

In the dwell-activated interaction mode, Browserous requires the user to dwell on an object before it is activated. As long as the object has the user's focus, it will expand slowly. After half a second from the moment it is gazed upon the object will be highlighted. The object is activated once it receives a gaze fixation longer than one second. A highlighted item can be seen on Figure 9. Browserous (Dwell)'s appearance was shown previously in Figure 5.
4.1.2. **Gesture-activated interaction mode: Browserous (Gesture)**

In the dwell-activated interaction mode, the activation process is divided in two steps: selection and activation. Selection happens by dwelling on an object. The focused item will grow slightly and then it will get highlighted. The selection procedure takes half a second. This selection is preserved unless the user glances on another object long enough to select it.

In order to confirm the selection and activate the item, the user needs to glance at the activation area at the bottom of the screen. When this happens, the program will zoom into the object unless it is an action or "go back" button, in that case it will zoom out of the current node. Figure 10 shows an image of Browserous (Gesture) displaying the root folder's contents, and the activation area can be seen at the bottom of the image.
4.1.3. Continuous interaction mode: Browserous (Continuous)

The last mode of interaction with Browserous is the continuous interaction mode. In this mode, activation and cancellation of actions are continuous actions. A node's selection starts as soon as the node is glanced upon, producing an expanding view of its contents. If this expanding view is allowed to reach full-screen size, then an item's selection is considered to be complete. In order to cancel an already started selection process, the user can glance away from the expanding view of the node, and it will retract until it disappears.

In order to keep a consistency in the program's behavior that glancing at an item immediately causes that item to expand, the "go back" button was removed from this interface. Instead, two "zoom out" areas were created at the left and right sides of the window. If the user's glance lays upon those sections, even a node view that takes the whole screen will shrink, revealing the parent node's contents beneath it. The interface can be seen at Figure 11.

![Figure 11: Browserous (Continuous) shows a view of the root node.](image)

In some aspects, this form of interaction resembles the dwell-activated interaction mode. A selection and activation of an item are performed when the user glances for long enough at the item, and looking away from the item cancels its activation process. In addition, the time required to fully activate or deactivate an object is one second. Despite the similarities, it is important to take notice of a few differences between the methods.

The first difference is that a preview of the result of performing a navigation is immediately visible, as shown in Figure 12. In classical dwell-activated modes of interaction, an item may expand slightly before selection, but no preview of the next node will be shown before the selection is complete. Indeed, the possibility of presenting a preview of an incoming node is easier to conceive in a navigational
Figure 12: Focusing on an item in Browserous (Continuous). The expanding target can be seen at the top right corner of the image.

The second difference is the quantity of expansion an item produces before it is finally being considered as selected or activated. Items in Browserous (Continuous) expand in an accelerating fashion until they achieve full screen. This causes the target area of the focused item to expand, and hides other potential targets behind the expanding area.

Figure 13: Expanding action in Browserous (Continuous). Image cropped to improve readability.

This behavior of Browserous (Continuous) is well-defined for a navigational context.
However, file management tasks also involve activation of actions. Since an action does not contain the contents of the file tree nor other collection of actions, it was decided that an expanding view of the action's icon would be displayed instead. The behavior is portrayed in Figure 13. Again, reaching full-screen size causes the item to be activated.

4.2. Dasherous

Experiments with Dasher have showed good results in eye-typing speed and accuracy before. In order to evaluate the viability of a Dasher-like interface in a file management context, Dasherous was developed.

Nodes in Dasherous are represented as colored rectangles. Files are represented with magenta rectangles. Folders are represented in either yellow or cyan, alternating in color according to depth in the file tree. The choice of alternating the colors is to provide the user with contrasting colors between levels of folders to help distinguish between folders and empty regions of the new node. Files and folders' actions menus are represented in white and actions in light gray if enabled, and dark gray if disabled.

Following the standard utilized in Browserous, each node in the system is displayed as a pair of icon and text. Moreover, the same set of icons was utilized in Dasherous. This was done as an attempt to reduce the feeling of inconsistency caused by how much a Dasher-like interaction mode is different to the interaction modes implemented in Browserous.

![Figure 14: A view of the root folder in Dasherous. This image has been manipulated to improve the visibility of the regions. A - Rewind region; B - Scroll-up region; C - Neutral region; D - Scroll-down region; E - Move forward region. In the actual program, the thickness of the lines at the center region of the screen is only 1 pixel wide.](image)

In order to interact in a Dasher-like manner, 5 regions were defined on the screen: A
move-forward region, a rewind region, a scroll up region, a scroll down region and a neutral region. A manipulated image of Dasherous to improve visibility of the regions is shown in Figure 14.

When the user looks at the move-forward region, labeled with E in Figure 14, the nodes in the file system expand. In addition to that, looking at the region may cause the view to scroll up or down. The speed at which both the expansion of the nodes and the scrolling of the view happens is dependent on how far from the center of the screen the user looks. These movements allow the user to navigate into new folders, files and, eventually, reach actions.

The rewind region, labeled with A in Figure 14, produces the opposite effect than the move-forward region. Looking at this region will cause all nodes to shrink and retract back to the right corner of the screen. Just like in the move-forward region, looking up or down inside that region allows you to scroll the view. The behavior of this region allows the user to navigate out of files, folders and actions menus.

Expansion and shrinking of nodes in Dasherous follow different criteria. For expansion of nodes, a node that contains the screen coordinates of user's point-of-attention will grow faster than other nodes in the same level of the file system. For shrinking of the nodes, the bigger nodes in a single level will shrink faster than the smaller ones until they reach the same size. An example of a node growing faster than its sibling nodes is given in Figure 15.

The relationship of vertical space between a parent node and all of its child nodes is 1.6:1. In other words, a node with fewer child nodes will reserve more vertical space per node than a node with many child nodes. This relationship is fundamental in the process of expanding and shrinking nodes, as the parent nodes are resized first, then child nodes are resized to fit the new available space. In situations where a child node expands or shrinks quicker than its siblings, that node is responsible for 50% of the total size difference among the group, while the rest of the child nodes share equally the remaining 50% of the total size difference.

Just like Browserous (Continuous), the interaction mode utilized in Dasherous provides the user with a preview of nodes that haven't yet been visited by the user. Differently to Browserous (Continuous), however, Dasherous provides a preview of multiple neighboring nodes at once, rather than one preview at a time. This behavior can be viewed most clearly in Figure 14 and Figure 15.
Figure 15: Approaching a folder in Dasherous. Image cropped for improved readability.

A node's left boundary is strictly dependent on the node's size. Every node is square-shaped and will have its right boundary fixed on the right corner of the screen. This creates the impression that a node is approaching the cursor at the center of the screen when the node grows, or that a node is moving away from the cursor when it shrinks.

The scroll-up and scroll-down regions, labeled as B and D in Figure 14, provide the user the means to scroll the view up and down. While such functionality is provided already by the move-forward and rewind regions, the scrolling regions will not cause any changes to the nodes. These areas were provided as a complement to the neutral region.

Finally, the neutral region, labeled as C in Figure 14, is an area of the screen which the user can glance upon without causing any activity to occur in Dasherous. This region serves two purposes: First, it provides a resting area for the eyes. Second, it displays the current path to the user, allowing them to review their current location in the file system. Figure 16 and Figure 17 offer a good view of the visualization of the current path in the neutral area.

The actions that can be performed on files and folders in Dasherous are the same as the ones provided in the Browserous interfaces. A user can create new files and folders inside a folder, as well as copying the folder into the clipboard, pasting the contents of the clipboard onto the folder, and deleting a folder. A file can be opened, copied and deleted. The actions menus for files and folders can be viewed in Figure 16 and Figure 17, respectively.
A node is considered to be focused if the node contains the center of the screen and it does not have any child nodes that also contain the center of the screen. In the case of actions, being focused also means being activated.

While Dasherous is inspired on Dasher and aims to mimic its functionality, interaction
with Dasherous differs from interaction with Dasher in many aspects. Those differences go beyond eye-typing vs file management.

The reason for the first difference in behavior is caused by the fact that Dasherous is an interface to a finite structure which is organized hierarchically, while Dasher is an interface to a possibly infinite text input area. The basic units in Dasherous, the nodes, are strictly organized in a tree-like manner and are unique, while the basic units in Dasher, the character, can be arranged in any possible way and may be repeated indefinitely.

At the selection of each character, Dasher presents its whole set of characters over again. While this works in eye-typing, presenting the whole file-tree at each level of selection and interaction is simply not possible, or at least not convenient. Because of that, it was decided that when actions were performed in Dasherous, the program would retreat its nodes and return control to the user somewhere near the place where he performed his latest action.

Another difference is that Dasherous provides really only one mode of operation, while Dasher allows the user to utilize different modes of interaction, like constant speed forward or joystick mode. While these interaction modes have not been implemented in Dasherous, it is possible to add them to the program later.

Finally, the last intended difference in design is that Dasherous displays both icons and text for each one of its nodes. This was done in order to give the user a visual feedback as similar as possible as the one in Browserous.

Browserous and Dasherous are part of the EyeFM project, hosted on https://code.google.com/p/eyefms/. This thesis refers to the version released on December 7, 2010, named EyeFM-2010-12-07.zip.
5. Experiment design

The main motivation in this thesis was to make a comparative investigation of the performance of different gaze-controlled interfaces in the context of file managing. It was hypothesized that the continuous interaction interfaces, Browserous (Continuous) and Dasherous, would be better for navigation, and the discrete interaction interfaces, Browserous (Dwell, and Gesture), would give a better sense of control when engaging actions.

In order to test the designed interfaces and the hypotheses, an experiment was conducted using one gaze tracker, two file managers with four modes of interaction in total as well as four similar sets of four tasks each, a questionnaire and an interview.

5.1. Experiment setup

The file managers were run on a Packard Bell EasyNote laptop with 1 GB of RAM memory, Athlon Turion 64 X2 processor running Ubuntu 10.10 AMD64. The gaze tracker was a Tobii T60 device and it was communicating with a computer running Windows XP. In order to make the communication between both computers we utilized an ETU-Driver version 1.27 testing application with the NetServer plug-in and a direct ethernet connection.

This seemingly awkward setup was necessary at the time of writing the thesis, due to a restrictive policy in the access to the official programming tools to communicate with Tobii devices. In addition, ETU-Driver, an alternative to the utilization of the official programming tools, was a Windows-only program at the moment of writing the thesis, while the file managers were programmed to run on Ubuntu 10.10.

In order to reduce gaze cursor jitter, a simple cursor smoothing algorithm was utilized. The algorithm averaged the position of the last 10 gaze samples in order to determine the position that the cursor should be displayed.

In order to minimize the number of different mouse actions that needed to be emulated for a dwell-based interaction, the tasks were limited to opening files, copying files, deleting files and creating new folders and files. Each program provided menus to allow the player to perform the actions needed to accomplish a task.

5.2. Participants

Twelve participants took part in the experiment. Half of the participants were students who volunteered once an invitation to participate in the experiment was made in an Interactive Technology course. A call for participants was sent to an international student e-mailing list in order to gather the remaining six participants, this call for participants offered one cinema ticket per participant.

5.3. Tasks

Each participant performed a set of four tasks for each interaction mode, resulting in a total of 16 tasks performer per participant. Each set provided a task for each one of the following actions: find and open a file, find and copy two files and paste them
elsewhere, delete a folder at the root level, create a new folder at the root level and a new file inside the new folder.

The sets of tasks were highly similar to each other, with minor differences only in the first two tasks. Each set of tasks presented its own unique targets to be interacted with, though they were always present at the same level in the hierarchy, and the actions expected by the user were the same. These small differences were designed in order to create conditions that would require the user to search for new locations, rather than navigate from memory.

Before each set of tasks, the moderator would run the program on a mouse-controlled setting, and demonstrate the usage of the programs. The moderator demonstrated the different active areas of the interface, as well as opening file, copying, deleting operations, and file creation operations. Before terminating the demonstration, the moderator asked whether any aspect of the interface needed to be clarified.

During the test sessions, and before running a task, the description of a task would be displayed on the screen. The participants were requested to read the task's description out loud. At this point, the participants had the opportunity to ask for clarifications on the instructions. After it was confirmed that the participant understood the instructions, the moderator pressed a key on the computer and the task was initiated.

The tasks were terminated once the participants announced that it was done, rather than when the users performed the necessary actions to complete it. The reason for this was to measure whether, and when the participants would realize that a task was completed.

5.3.1. Task descriptions

In order to better illustrate the tasks performed by the participants, a list of the tasks present in the task set 1 is provided. Since all four sets are similar, it suffices to briefly discuss the differences between the sets.

The first task requires a user to search for a particular file, always contained within a folder within the Music folder. In the different sets of tasks, the music file requested may be a Blues song, Disco song, Jazz song, or Samba song.

In the second task, both the type of file and their location differ. The different task sets require that the user back-up document files, image files, movie files, or compressed files.

Task 1 (search task)

You are a Blues music lover and you want to listen to some music. Luckily, you just happen to have some Blues music on your hard drive. **Find a Blues song and open it.**

Task 2 (copy task)

You have some documents you want to backup: CV.odt and thesis.odt. **Mark both files from the folder “Documents” for copying and then paste them at the folder “Backup”.**

Task 3 (delete task)

You have some unnecessary files in the Trash folder. In order to save disk space, you
can delete the whole folder at once. To save time, **delete the folder named “Trash” by using the Trash folder's actions menu.**

**Task 4 (create task)**

Now that your hard drive has enough space, you can create more folders and more files. **Create a New Folder** in the root folder, and then **navigate into the New Folder** and **create a New File in the New Folder.** (Use the root folder's actions menu to accomplish those actions).

**5.4. Data collection**

In order to gather information about the usage of the interfaces and the participants' experiences with them, a number of different types of data were collected. Background questionnaires provided some basic information about the user themselves, program logs provided a detailed trace of the participants' activity with the programs, usability questionnaires and interviews provided subjective feedback on the experience.

**5.4.1. Background questionnaire**

Some very basic information about the participants was collected via a background questionnaire prior to the execution of any tasks in each session. The questionnaire asked the participants about their profession, field of expertise, familiarity with gaze tracking technology and familiarity with Dasher.

The participant expressed their familiarity with gaze tracking technology and Dasher by picking one out of the following options:

1. I have never heard of it.
2. I have heard of it, but I have never used it.
3. I have tried it, but not often.
4. I regularly have the opportunity to use it.
5. Using and/or understanding it is an essential part of my job.

**5.4.2. Program logs**

Every file manager program logged its activities. The logging system kept track of the programs' command-line arguments, task set selections, task selections, task start and end times, as well as node visits and performed actions, both with timestamps.

Using these log files, it was possible to collect precise information about how many actions, and which actions each participant performed in each task. Unfortunately, due to a difference in the logging of node visits between Dasherous and Browserous, any comparisons regarding number of node visits was found impractical. This difference in the logging system is due to the very different nature of the Dasherous interface in comparison to the other interfaces.

To better shine light on the issue, two example log files depicting the author of the thesis performing Task 2 in an error-free fashion are provided in the Appendix section. The example log files were recorded with Dasherous and Browserous (Dwell).
5.4.3. Usability questionnaire

Once a participant was done with all the tasks, a usability questionnaire was applied. While the traditional design of usability questionnaires utilize a 7-point Lickert scale, this study opted for a preferential voting system instead. The goal was to force the participants to pick the “best” interface and the “worst” interface regarding different aspects of the interfaces.

The interfaces were ranked according to 4 characteristics not specific to any task: appearance, intuitiveness, ease of navigation, and ease of control. They were also ranked according to which would be best suited to each one of the four tasks in a set.

5.4.4. Interview

In addition to the log files and the questionnaires, each participant was also interviewed at the end of each session. The participants were asked seven questions which they were allowed to answer freely. In case some information was ambiguous, the moderator asked for clarifications.

Interview questions:

1. How did it feel to use your eyes to control the files on the computer?
2. Did any of the tasks feel difficult to you?
3. Was any of the tasks made easier or more difficult once you tried it with different interfaces?
4. Did you have any favorite interface?
5. Was that interface better than the other interfaces in every aspect?
6. Did you have any least favorite interface?
7. Was that interface worse than the other interfaces in every aspect?

The 2nd question was clarified to mean understanding the task itself, not the performance of the task with a particular interface. The participants were allowed to give indecisive answers in the 4th and 6th questions, in those cases the answer was interpreted as “no” and their follow-up questions were skipped.
6. Results

This chapter presents the data collected in the experiment. Each type of data collected in the experiment has its own subsection where its data is presented. Each section also presents an analysis of the data obtained.

6.1. Participants background information

Before any instructions on the experiment were given, every participant was asked to fill a background questionnaire. This questionnaire inquired about the participant's field of expertise and their familiarity with gaze tracking software and Dasher. The results are illustrated in Figure 18.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Profession</th>
<th>Field</th>
<th>Familiarity with gaze tracking</th>
<th>Familiarity with Dasher</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Researcher</td>
<td>Human-computer interaction</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Researcher</td>
<td>Human-computer interaction</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Researcher</td>
<td>Human-computer interaction</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Student</td>
<td>English translation</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Researcher</td>
<td>Exercise and health</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Student</td>
<td>Computer sciences</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Datanomic</td>
<td>IT</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>Student</td>
<td>Adult education</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>Student</td>
<td>International relations</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>Student</td>
<td>Social sciences</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Student</td>
<td>Translation sciences</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Student</td>
<td>Journalism</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

*Figure 18: Participants' background information*

The background questionnaire revealed that seven participants have tried some form of gaze tracking technology, two of which are researchers with a relatively deep understanding in the field. Meanwhile five participants have tried Dasher, and the rest of them were completely unfamiliar to its interface. Seven of the participants were professionals or academics in the Information Technology field, while the rest of the participants came from diverse fields.

6.2. Task completion times

In order to measure the performance of the participants, in relation to how quickly they were able to perform the tasks, time stamp information from the actions was utilized. The completion time of a task was calculated from the moment the task start was logged, to the moment the task end was logged.

Since the participants were allowed to take as long as they wanted to finish the task, there were a number of successful runs with an unusually high task completion time, due to confusion or other circumstances that could have caused other participants to not finish the task. In order to deal with a more normal distribution of information, I applied the Grubbs' test for outliers recursively and removed the outliers from further analysis of the task completion times.

As Figure 19 shows, for all 4 tasks the relationship between each interface's average
task completion time is similar. Browserous (Dwell) and Browserous (Continuous) are the fastest interfaces, and Browserous (Gesture) and Dasherous are the slowest interfaces. Dasherous being the slowest one in all cases.

Figure 19: Task completion times by task and interface. The information represented in the graphs does not include the task completion time for outlying task runs.

In order to refine the comparison between the interfaces, a paired t-test was performed. Each pair was comprised of a set of task completion times from all participants performing the same task with the same interface. In case a participant had an outlying performance with any of the two interfaces being compared, or failed to complete the task with any of the two interfaces being compared, the participant was eliminated from both sets being compared. The resulting set of comparisons can be viewed in Table 1.
Table 1: paired t-test task completion times relationships between the interfaces

<table>
<thead>
<tr>
<th>Task 1 completion time comparisons:</th>
<th>Task 2 completion time comparisons:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>p</strong></td>
<td><strong>p</strong></td>
</tr>
<tr>
<td>Continuous &lt; Dasherous</td>
<td>Continuous &lt; Dasherous &lt; 0.0001</td>
</tr>
<tr>
<td>Continuous ~ Dwell</td>
<td>Continuous &gt; Dwell</td>
</tr>
<tr>
<td>Continuous ~ Gesture</td>
<td>Continuous ~ Gesture 0.2145</td>
</tr>
<tr>
<td>Dasherous &gt; Dwell</td>
<td>Dasherous &gt; Dwell &lt; 0.0001</td>
</tr>
<tr>
<td>Dasherous &gt; Gesture</td>
<td>Dasherous &gt; Gesture 0.0003</td>
</tr>
<tr>
<td>Dwell &lt; Gesture</td>
<td>Dwell ~&lt; Gesture 0.0081</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task 3 completion time comparisons:</th>
<th>Task 4 completion time comparisons:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>p</strong></td>
<td><strong>p</strong></td>
</tr>
<tr>
<td>Continuous &lt; Dasherous</td>
<td>Continuous &lt; Dasherous 0.0005</td>
</tr>
<tr>
<td>Continuous ~ Dwell</td>
<td>Continuous ~ Dwell 0.2167</td>
</tr>
<tr>
<td>Continuous ~ Gesture</td>
<td>Continuous ~&lt; Gesture 0.3671</td>
</tr>
<tr>
<td>Dasherous &gt; Dwell</td>
<td>Dasherous &gt; Dwell &lt; 0.0001</td>
</tr>
<tr>
<td>Dasherous &gt; Gesture</td>
<td>Dasherous &gt; Gesture 0.0001</td>
</tr>
<tr>
<td>Dwell &lt; Gesture</td>
<td>Dwell ~&lt; Gesture 0.0059</td>
</tr>
</tbody>
</table>

Browserous (Continuous) was significantly faster than Dasherous in every task. The interface didn't seem to have a significant task completion time difference with Browserous (Dwell) for most tasks, with the exception of task 2, in which it is significantly slower. In other tasks no significant difference between Browserous (Continuous) and Browserous (Gesture) was found.

Browserous (Dwell) was significantly faster than Dasherous in every task. The difference is significant in all of the tasks. Browserous (Dwell) is also significantly faster than Browserous (Gesture) for tasks 1 and 3, and less significantly faster in tasks 2 and 4.

Browserous (Gesture) was significantly faster than Dasherous for tasks 1, 2 and 4. There was no statistically significant difference between the two interfaces for task 3.

6.3. Task errors

Since a fast interface that causes its users to commit several errors cannot without controversy be called better than a slower interface that causes users to commit few or no errors, we were also interested in analyzing errors made while performing the tasks.

In order to analyze the presence and nature of errors, we used action information stored in the log files. Each action stored there included a time stamp, name of the action, and target or location of the action. With this information, 6 categories of errors were created and counted. The categories are as follow, from the most serious to the least serious:

- **Task failed**: The most serious of the errors. Whenever a participant mistakenly declared a task to be complete or agreed to stop the task when prompted with a question, this error was counted. Whenever this error was counted, all other errors in the same run were discarded, since it was considered that committing those errors were all just part of failing to complete the task.

- **Failure to acknowledge the end of the task**: Similar to the first error category, but slightly less serious. This error was counted whenever a participant
performed all the necessary actions correctly to complete the task, but failed to announce the end of the task.

- **Wrong action**: This was counted whenever a counterproductive action was performed in place of a productive one. For example, marking a file to be copied when you really intended to open the file.

- **Wrong target**: This was counted whenever the target for the correct action was the wrong one. The most common example of this is marking the whole folder to be copied rather than the files inside the folder.

- **Excessive file opening**: A milder variety of the failure to acknowledge the end of the task error. This error was counted whenever the participant activated the "open" action several times, rather than once, for a target file before announcing the completion of the task. This error was counted only on the first task.

- **Excessive steps**: This type of error had meaning only in the second task. This error was counted whenever the participant copied and pasted one file at a time, rather than marking both files to be copied first, and then pasting both at once at the desired location.

Some successful task runs contained more than one type of error. Moreover repeated cases of one error were not counted separately within a run. It was common, for instance, that one would open the intended file three or more times before declaring the task to be over.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Interface</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dasherous</td>
<td>Excessive file opening</td>
</tr>
<tr>
<td>2</td>
<td>Gesture</td>
<td>Task failed</td>
</tr>
<tr>
<td>6</td>
<td>Gesture</td>
<td>Excessive file opening</td>
</tr>
<tr>
<td>10</td>
<td>Gesture</td>
<td>Wrong action</td>
</tr>
<tr>
<td>11</td>
<td>Continuous</td>
<td>Excessive file opening</td>
</tr>
<tr>
<td>12</td>
<td>Dwell</td>
<td>Excessive file opening</td>
</tr>
<tr>
<td>12</td>
<td>Gesture</td>
<td>Wrong action</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Participant</th>
<th>Interface</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Gesture</td>
<td>Wrong target</td>
</tr>
<tr>
<td>6</td>
<td>Dasherous</td>
<td>Wrong target + Wrong action</td>
</tr>
<tr>
<td>8</td>
<td>Dasherous</td>
<td>Task failed</td>
</tr>
<tr>
<td>10</td>
<td>Gesture</td>
<td>Wrong action + Excessive steps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Participant</th>
<th>Interface</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Dwell</td>
<td>Failure to recognize end of the task</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Participant</th>
<th>Interface</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gesture</td>
<td>Wrong action</td>
</tr>
<tr>
<td>3</td>
<td>Dwell</td>
<td>Wrong action</td>
</tr>
<tr>
<td>8</td>
<td>Dasherous</td>
<td>Wrong action</td>
</tr>
<tr>
<td>9</td>
<td>Dasherous</td>
<td>Wrong action</td>
</tr>
</tbody>
</table>

*Table 2: List of errors by task and participants*

Table 2 shows that the errors were most present in the first task, which may be result of
a learning process with a new interface. Also, it's worth noting that the most common error in that task is the excessive file opening. Which suggests that the visual feedback for opening a file is deficient in some aspects and should be redesigned. This problem is evident in the participant's feedback in Transcript 1.

| Participant | Do you open it here? Oh, okay. |
| Participant | Should I look somewhere else? It doesn't open. |
| Moderator   | Would you like to try it again? Once you feel that the task is done, you should tell me. |
| Participant | Did I do the task? Yes, I opened the file, but it didn't work. |
| Moderator   | No music is going to be played, just some text displayed on the screen. |
| Participant | So, there was some text then? |
| Moderator   | Did you see the text? |
| Participant | Not exactly. |
| Moderator   | Would you like to try it again? |
| Participant | Yes. |
| Moderator   | It is important to me to know this, because I need to if you see whether the task is completed or not. |
| Participant | I look at the file and then what? |
| Moderator   | You open the file and then it's done. |

*Transcript 1: Participant 12, Task 1, Browserous (Dwell)*

It is also worth mentioning that, while there is only one registered case of a *failure to recognize end of the task* error, a number of participants expressed uncertainty over whether a task was complete or not in other situations. Examples of this are given in Transcript 2, Transcript 3, and Transcript 4.

| Participant | Did I manage to do this [task]? I think I was able to. |
| Moderator   | You can check it. |
| Participant | Okay, I'm done. |

*Transcript 2: Participant 3, Task 2, Dasherous*

| Participant | And I go to "Delete". No? Did I go somewhere wrong? I don't wanna be here. Not here either. Can I...? |
| Moderator   | What are you looking for at the moment? |
| Participant | I'm trying to go back to find the "Trash" thing. I don't know where I am. |
| Moderator   | Should we stop [the task] there? |
| Participant | No, hold on. That's not doing anything, okay. Okay. I have no idea where I am anymore. I can't go back. Did I delete the "Trash" already? |
| Moderator   | I can't answer that. |
| Participant | Okay, I'm gonna say I deleted the whole "Trash" thing. Done. |

*Transcript 3: Participant 8, Task 3, Browserous (Dwell)*
Due to such situations, the number of failure to recognize the end of the task errors might be understated. However, it would be hard to quantify and measure how much uncertainty a participant has over whether the task is indeed complete apart from the participant giving up on the task despite it being complete.

One last thing to note in the error analysis is the frequency of task runs with errors per interface, as well as the frequency of each type of error. This is illustrated in Figure 20.

![Task errors per interface](image)

**Figure 20: Task errors per error type and interface**

The most common error is wrong action, occurring mostly in the Dasherous and Browserous (Gesture) interfaces. Those interfaces also have the highest concentration of task errors. The continuous interface, on the other hand, registered the fewest number of errors of all interfaces, with only one task error.

### 6.3.1. Usability questionnaire

The usability questionnaire provided subjective information about the participants’ preferences of the interfaces regarding different aspects, both general and task-specific. The questionnaire’s purpose was to force participants to rank interfaces by picking their favorites and least favorites. The questionnaire provided a list of the interfaces utilized in the order the participant utilized them during the test session.

Figure 21 shows the distribution of votes for best, 2nd place, 3rd place and worst for the different interfaces, in the general aspects. The missing vote for Browserous (Continuous) and Browserous (Dwell) is due to a participant being unable to choose between 2nd and 3rd place for those two interfaces regarding the appearance aspect.

What is noticeable in Figure 21 is the rejection of Dasherous in relation to the other interfaces in the general aspects, marked by the concentration of worst votes it received,
and the relative acceptance of Browserous (Dwell) and Browserous (Gesture).

Surprisingly Gesture ranked well on navigation and control despite being the interface with which participants made errors most often. This could be indication that users feel more comfortable with the utilization of an explicit action, in this case glancing at the OK button at the bottom of the screen, though this should be further tested against different dwell time durations for dwell-activated interfaces.

Also noteworthy in Figure 21 is the significantly greater concentration of best votes given to Browserous (Dwell) in Appearance, regardless of the fact the Browserous interfaces all follow the same basic design. This could be indication that one should consider carefully before adding graphical overlays in an interface and avoid doing so unless needed.

![Distribution of ranking per interface for each general characteristic.](image)

**Figure 21:** Distribution of ranking per interface for each general characteristic.

In the task-specific comparison, there was a higher uniformity of voting among the
Browserous interfaces, as shown in Figure 22. The higher uniformity could be indication that the tasks did not feel, after all, much different from each other despite the intention to design them so.

While Dasherous has clearly the worst ranking in both Navigation and Control, Browserous (Continuous) does not unambiguously hold the 3rd place in any of these characteristics. Instead, it features a smaller amount of strong approval, but also a smaller amount of strong rejection in both characteristics. Such a phenomenon makes this data inappropriate to judge whether continuous modes of interaction is better for navigation (H1), and whether discrete modes of interaction give a better sense of control (H2).

The continued rejection of Dasherous in the ranking of the interfaces for the tasks reinforces the idea that the implementation of Dasherous used in the experiment is indeed weaker than the other interfaces.

Figure 22: Distribution of ranking per interface for each task.
6.3.2. Interview

After the sets of tasks and the questionnaires, the participants were interviewed as the last part of the test session. The purpose of the interview was to gather subjective feedback on the file manager interfaces.

During the interview, the participants were asked to consider whether they perceived a change in difficulty when trying to perform the same task with different interfaces. They were also asked to pick a single interface as their favorite and another for least favorite, and elaborate the reasons for the choice. The participants were allowed to pick as many interfaces as they wanted as their favorite or least favorite.

The participant's decisions of having to pick an interface to call best and one interface to call worst is represented in Figure 23. The total sum of opinions expressed there exceed 24, because some participants chose to declare two interfaces as their favorite, rather than only one. No participant picked more than one interface that they were willing to call “worst”.

![Participant opinion by interface](image)

**Figure 23: Participant preference of interfaces, by interface. Negative values represent rejection of the interface.**

The data represented in Figure 23 reveals two interesting bits of information. The first bit of information is that the participants did indeed feel strongest about Dasherous, in a relatively negative manner, and Browserous (Gesture), in a mostly positive manner.

The second bit of information is that, despite the fact that Browserous (Dwell) was ranked best in most general characteristics and tasks more often than Browserous (Continuous), people still displayed a higher rejection of the Dwell-activated interface than Browserous (Continuous). This could indicate either the participants didn't perceive much difference between the two interfaces when performing the ranking, or that the questionnaire failed to measure important aspects of the interfaces.

Despite the fact that it is easy to plot a graph from the feedback given by participants during the interview, one should be more cautious about taking these data as accurate. A
number of participants showed some level of ambivalence regarding the interfaces. An example of an ambivalent quote is given in Transcript 5. A better analysis for the presence of ambivalence in the participants could be done by crossing the answers given in the questionnaires with the answers given in the interview, but doing so would be outside the scope of this thesis.

<table>
<thead>
<tr>
<th>Moderator</th>
<th>Did you have any least favourite interface?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant</td>
<td>Browserous (Dwell)</td>
</tr>
<tr>
<td>Moderator</td>
<td>Was it worse than the others in every aspect?</td>
</tr>
<tr>
<td>Participant</td>
<td>No, I also thought that it was easy. It was kind of fast. When I didn't make any mistakes it was Ok. There was too much pressure.</td>
</tr>
</tbody>
</table>

*Transcript 5: Participant 3, example of ambivalence. The interface was easy to use, and there was too much pressure.*

While those questions are rather vague and can hardly enable a method to study what makes an interface better or worse in an organized manner, they might reveal new interesting research questions, bring to light design flaws not perceived by the software developer, and offer insight on potentially useful new experiment designs.
7. Discussion
This work involved the development of four file management interfaces. Three of them designed in a WIMP style, with varying methods of interaction: dwell-activation, gesture-activation, and a method which we called continuous activation. Those three interfaces shared a great deal of code and were placed under the name Browserous. A fourth interface was designed with inspiration on Dasher, and it was named Dasherous.

This thesis' experiment aimed to compare four different file managers in performing four different tasks. In addition to performing a total of 16 tasks, each participant still had to answer a questionnaire with 8 questions, requesting them to rank the interfaces and then go through an interview. All of that, together with the briefing on how each interface worked made for a tight schedule for a one-hour experiment, but provided a large amount of data on the interfaces.

7.1. Experiment, data and hypotheses' validation
Despite the large amount of data obtained, the only ones which allowed for a more straightforward examination were the task completion times and task errors. Questionnaire and interview data, while fuzzy, brought to light some important things, and helped raise a set of questions regarding comparing the file managers.

Browserous (Dwell) was clearly the interface which allowed for the fastest interaction, while Dasherous provided the worst. Browserous (Gesture) and Dasherous suffered the highest task error rates, while Browserous (Dwell) and Browserous (Continuous) had very few occurrences of task errors. Despite having the highest error rate, Browserous (Gesture) received the best ranking for the control characteristic and was considered the best interface overall by slightly more than half of the participants. Finally, Browserous (Continuous) received mostly neutral rankings in the questionnaire, but it was the only interface which didn't get classified as worse overall by any of the participants.

Two hypotheses were raised before the experiments began. The first hypothesis, H1, suggested a continuous mode of interaction, in this case Browserous (Continuous) and Dasherous, would feel more natural in navigation than the discrete modes of interaction, in this case Browserous (Dwell) and Browserous (Gesture). After the first test few sessions, it became clear that Dasherous would need to be considered an outlier when testing these hypotheses.

Looking at the questionnaire results, H1 certainly does not hold true regarding Dasherous, but whether or not it is true for Browserous (Continuous) it is harder to tell. It did not get ranked as worst for that characteristic by any participant, but on the other hand, Browserous (Gesture) was ranked best for that characteristic by slightly more than half of the participants.

This data hardly validates hypothesis H1. On the contrary, it either suggests that H1 is false, or that the questionnaire design was inappropriate for measuring such a thing. For this reason, the hypothesis is better considered not validated.

Hypothesis H2 proposed that the discrete modes of interaction would give the user a better sense of control than the continuous modes. This time, the hypothesis clearly
holds true for Dasherous. The hypothesis, however, is untrue for Browserous (Continuous), as the questionnaire data shows that the score in the control characteristic is very balanced between the Browserous interfaces. Hypotheses H2 is, thus, also considered not validated.

There was no trade-off between interaction speed and error rates in this experiment. The slowest interfaces were also the ones that amassed most task errors. Dasherous obtained the worst rankings in all subjective evaluations, and obtained the worst interaction speeds, it was, however, slightly better than Browserous (Gesture) regarding error rates.

Among the Browserous interfaces, Browserous (Gesture) obtained the worst time completion times and worst error rates. Despite that, Browserous (Gesture) received the strongest level of overall acceptance. Browserous (Dwell) provided the fastest interaction speed, and Browserous (Continuous) the lowest error rates.

### 7.2. Software design considerations

The Browserous interfaces looked very similar to each other, with the exception of the presence of overlay regions on Browserous (Gesture) and Browserous (Continuous) and the absence of it in Browserous (Dwell). The presence of the overlays, or possibly the appearance of the overlays turned out to be a design mistake in an aesthetic sense. The participants' opinions on the appearance of the interface clearly favored Browserous (Dwell) over the other interfaces.

The participants' showed an overall preference for the gesture-activated interface over the two other modes of activation. They also expressed comfort in the fact that the confirmation of the action by glancing at the OK button at the bottom of the screen made the interaction more familiar. The high error rate, however, may be revealing that the activation gesture needs to be redesigned.

One possible research direction on the Browserous interfaces is combining the different modes of interaction in order to maximize the task completion times and subjective evaluation of the interfaces, and minimize the error rates. For example, having dwell or continuous activation for consequence-free actions, such as navigating, and gesture activation for actions with a consequence, such as opening files, copying, deleting, and pasting.

Dasherous' development was less straightforward than the Browserous interfaces. Dasher is already in a state of maturity, and it is specialized in text entry, meanwhile Dasherous lies on very immature code, and a very experimental design which attempts to adapt the text entry interface to a file management environment.

Applying a method designed specifically for text entry in a file management environment proved to be harder than expected, and the results show that either the method is inappropriate for file management, or that Dasherous was a deficient attempt at doing so. One fundamental difference between the interfaces is that Dasherous forces the nodes to retract to the right once actions are activated, while in Dasher allowing nodes to retract would mean an undo action.

There were a few design problems in Dasherous. First, each node's icon and text were located in the top-left corner of the node. Such a design choice draws the user's attention
away from the center of the node they intend to reach. Probably the best location for the
text and icon would be at the center-left spot of the node.

The second design issue identified in Dasherous is that the program allows the text and
icon to scroll while the user is looking at it. The problem was especially hard to deal
with when the users tried to read small text of not-yet-expanded nodes. While this
behavior is not a problem in Dasher, in which each node represents a single character, it
makes reading whole file and folder names hard in Dasherous. One possible research
direction is to compare the current implementation with one that delays the shifting of
the text for the first few moments in which a node is focused, another one is to expand
the text of focused nodes to facilitate the reading.

Finally, the last design flaw in Dasherous is its interaction speed. While Dasher is one
of the faster gaze interaction text input methods, Dasherous clearly provided the worst
interaction speeds of the gaze interaction file managers. One limiting factor is the speed
at which the interface scrolls back once an action is activated. Perhaps an experiment
with longer tasks with multiple action activations and different scrolling speeds could
help identify a more optimal rewind speed which allows for fast interaction without
causing the user to lose reference of their current location in the file system.

7.3. Software quality considerations

In addition to design issues, poor code quality is another risk of running experiments
with software prototypes. Even in a controlled experiment, users may produce
conditions that the developer did not foresee, or did not test properly.

Browserous (Dwell) and Browserous (Gesture) exhibited during the experiments a test-
interrupting bug, which would cause the programs to freeze when a participant
attempted to copy a whole folder. Such bug was distracting to both the participant and
the moderator, and certainly harmful to the experiment.

In Dasherous, the frame rate dropped when many nodes were displayed on the screen.
The impact of this frame rate was relieved with some basic graphics cropping algorithm,
though participants still experienced slowdowns occasionally.

In addition, rewinding after activation of actions in Dasherous was not always smooth
and sometimes it scrolled the window up or down excessively. This was due to the fact
that it was hard to predict how much vertical movement was needed to send the screen
center to the right position while every node was shrinking at possibly very different
rates. This issue was more visible when deleting nodes than when performing other
actions.

These examples of poor quality in the software code reveal the necessity of running
more pilot tests when the experiment involves prototypical software of considerable
complexity.
8. Conclusion

In order to tackle file management tasks with gaze interaction, four different file managers were developed: Browserous (Continuous, Dwell, and Gesture), and Dasherous.

Browserous follows the navigational and spatial paradigms of file managers. It displays one single node of a file tree at a time, and its contents are displayed as a combination of icon + name. Dasherous takes inspiration from Dasher and displays the file tree in a Dasher-like fashion.

The development of Dasherous proved to be a challenge. Since file management and text entry are very different problems, many design decisions needed to be made in order to adapt a Dasher-like interface to the file management tasks domain. The resulting interface proved to be very immature. Many design refinements need to be performed before the interface is ready to be used.

An experiment was run where the participants were able to try all the interfaces to perform file management tasks. Results showed, unsurprisingly, that a Dwell-activated mode of interaction was the fastest method of interaction, followed by Browserous (Continuous). The second unsurprising result is that a Gesture-activated mode of interaction is generally preferred for its ability to give users time to think before acting.

One interesting finding is that Browserous (Gesture) received the best results in the subjective evaluation of ease of control, regardless the fact that it was the interface with which most mistakes were made. Also, it was surprising that the fastest methods of interaction were also the most error-free methods in this context.

Much data was gathered in the experiment, which allowed for some new research questions to be raised. One could study interaction methods that utilize context information to decide between a dwell-based activation or a gesture-based activation, to speed up navigation and make the activation of actions that alter the file system feel more comfortable. Another possibility is studying design modifications in Dasherous, or another Dasher-like interface in order to improve its performance and further analyze the viability of the paradigm on a file management context.

Finally, parts of this work serve as a cautionary tale to other human-computer interaction researchers and students. Trying to compare a large number of programs at once is going to reduce the amount of time a researcher has to test each one of the programs. In addition straying away from the well-known path of utilizing Likert scale tests for measuring subjective aspects of a program should be done with care, as the resulting data may be very difficult to analyze.
9. References


[Hyrskykari, 2006] Aulikki Hyrskykari, Eyes in Attentive Interfaces: Experiences from creating iDict, a Gaze-Aware Reading Aid, *University of Tampere, Department of Computer Sciences*, 2006.


Background information questionnaire

Participant #: ______

1) What is your profession?

___________________________________________________________

2) If an academic, which is your field of expertise?

__________________________________________________________

3) How familiar are you with eye-tracking technology?
   [ ] I have never heard of it.
   [ ] I have heard of it, but I have never used it.
   [ ] I have tried it, but not often.
   [ ] I regularly have the opportunity to use it.
   [ ] Using and/or understanding eye-tracking technology is an essential part of my job.

4) How familiar are you with Dasher?
   [ ] I have never heard of it.
   [ ] I have heard of it, but I have never used it.
   [ ] I have tried it, but not often.
   [ ] I regularly have the opportunity to use it.
   [ ] Using and/or understanding Dasher is an essential part of my job.
Questionnaire

Participant #: _______

A - Dasher-inspired interaction: files, folders and controls approaching from the right corner of the screen.
B - Interaction based on dwell only: objects highlight, then activate when you stare at them.
C - Interaction based on eye gesture: objects highlight when you stare at them. To activate them you must look at the bottom of the screen.
D - Continuous interaction: zooming in and out, no "previous folder" button.

Please, rank the file manager interface prototypes based on the following criteria:

**General characteristics:**

**Appearance**
which interface was most aesthetically pleasing:

```
<table>
<thead>
<tr>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
</table>
```

**Intuitive**
which interface was easier to understand how to use:

```
<table>
<thead>
<tr>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
</table>
```

**Navigation**
which interface allowed you to navigate in the file system with most ease:

```
<table>
<thead>
<tr>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
</table>
```

**Control**
which interface allowed you to activate actions with most ease:

```
<table>
<thead>
<tr>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
</table>
```
**Task-specific characteristics:**

**Search task**
which interface was best when you were looking for the song you wanted:

<table>
<thead>
<tr>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
</table>

**Copy task**
which interface was best at letting you make backup copies of your files:

<table>
<thead>
<tr>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
</table>

**Delete task**
which interface was best at letting you erase the trash folder:

<table>
<thead>
<tr>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
</table>

**New folder + New file task**
which interface was best at letting you create new folders and new files:

<table>
<thead>
<tr>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
</table>
Sample Dasherous log file

# New session start: ./run_dasherous.py root
# Fields: ticks(ms)  action_name  action_target
4913  selected task set 1
9537  Task 2 start.
9540  current node: root
15925 current node: root/Documents
19411 current node: root/Documents/CV.odt
20914 current node: root/Documents/CV.odt actions menu
24899 copy "root/Documents/CV.odt"
24901 current node: copy CV.odt
24919 current node: root/Documents/CV.odt actions menu
25970 current node: root/Documents/CV.odt
26507 current node: root/Documents
30337 current node: root/Documents/thesis.odt
31519 current node: root/Documents/thesis.odt actions menu
35227 copy "root/Documents/thesis.odt"
35229 current node: copy thesis.odt
35248 current node: root/Documents/thesis.odt actions menu
36300 current node: root/Documents/thesis.odt
36702 current node: root/Documents
38848 current node: root
42057 current node: root/Backup
42850 current node: root/Backup actions menu
46132 paste "root/Backup"
46135 current node: paste Backup
46148 current node: root/Backup actions menu
47196 current node: root/Backup
48704 current node: root/Documents
50586 Task 2 end.
Sample Browserous (Dwell) log file

# New session start: ./run_browserous.py root dwell
# Fields: ticks(ms)  action_name  action_target
2252  selected task set 1
2828  Task 2 start.
5194  Enter folder "root/Documents"
7203  Enter file "root/Documents/CV.odt"
9691  copy "root/Documents/CV.odt"
11662 Enter file "root/Documents/thesis.odt"
13681 copy "root/Documents/thesis.odt"
15663 Return one level.
17663 Enter folder "root/Backup"
19679 Enter folder menu.
21837 paste "root/Backup"
23302 Task 2 end.