COSC460 Honours Report
Improving Users' Command Selection Performance

November 2, 2012

Joel Harrison
Jeh94@uclive.ac.nz
Department of Computer Science and Software Engineering
University of Canterbury, Christchurch, New Zealand

Supervisor: Professor Andy Cockburn
andy@cosc.canterbury.ac.nz

Abstract

Hotkeys have been shown to improve user command selection performance through proving a flat selection hierarchy and fast activation through the keyboard. However, hotkeys are often not learned by users even though command selection performance improvements are possible. Part of the hotkey learning problem is the current hotkey feedback methods which require the user to acquire the command with the mouse in order to display.

In this paper we present ExposeHK, a new interface technique which allows the user to browse, perform and learn hotkeys from within the hotkey modality. ExposeHK encourages users to learn hotkeys through practice thereby providing a seamless transition from novice to expert performance. Our evaluations demonstrate that with ExposeHK users are able to achieve a significantly higher level of command selection performance. Furthermore, our final evaluation demonstrates that when ExposeHK is available in an application users' hotkey usage is extremely high.

Acknowledgements

The author would like to thank his supervisor, Andy Cockburn, for his on-going input, feedback, and support throughout the project. In addition the author would also like to thank Sylvain Malacria for his valuable feedback and input towards the project.

Contents

1 Introduction

Command selection is one of the most common user interactions and is primarily done through interfaces such as menus, toolbars, and the Ribbon. These interfaces tend to group commands into meaningful categories to allow for quick refinement when searching for a command. However, for commands which are used regularly, navigating through the command hierarchy impedes users' command selection performance. For a single command selection the performance cost is small, however when users complete hundreds of command selections daily the performance cost is significant. Therefore, it is important that interfaces provide users with shortcuts to enable expert level of performance [1].

Hotkeys, also known as keyboard shortcuts, offer the user the ability to make command selections without having to navigate through command interface hierarchies. Models of performance, such as Keystroke-level model (KLM) [2], demonstrate that hotkeys can be used to make faster command selections compared to the pointer based method of selection. Hotkeys have been shown to improve users' command selection performance [3] but are not readily learned or used by novice users [4].

Part of the hotkey learning problem is in the way that hotkey shortcuts are displayed to the user. In most interfaces the hotkey shortcuts are displayed through tooltips or as text alongside the command name (see Figure 1.1). However, these display techniques often result in the hotkey shortcut going unnoticed as the user has already performed pointer action to acquire the command. Alternative menu designs [5] have been proposed based on increasing hotkey feedback or increasing pointer selection cost which have been able to significantly improve learnability and usage of hotkeys.

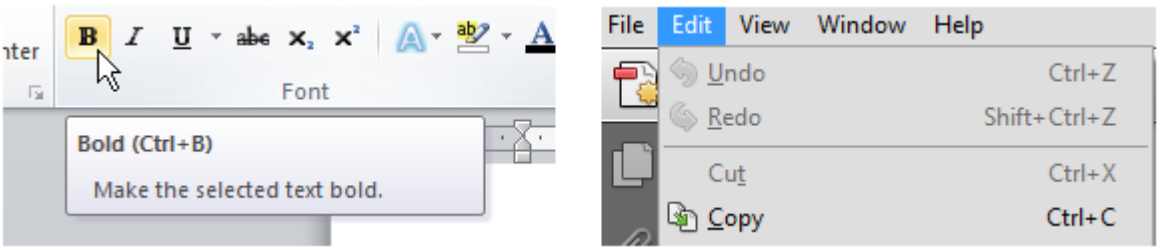

Figure 1.1: Common hotkey display methods. Left, the hotkey is displayed through a tooltip. Right, the hotkey is displayed alongside the command in a menu.

This paper aims to improve users' command selection performance with a new interface technique which allows users to browse, perform and learn hotkeys within the hotkey modality. The new technique called ExposeHK removes the pointer based interaction for finding hotkeys and encourages hotkey learning through practice.

We applied ExposeHK to the Ribbon (ExposeHK_R) and CommandMap (ExposeHK_CM) interfaces. Our evaluations of the two ExposeHK interfaces revealed that users were able to achieve significant improvements in command selection performance with ExposeHK as well as users having a strong preference towards the ExposeHK interfaces. We then evaluated ExposeHK in a real world application to determine if ExposeHK would alter the users' command selection procedure. Our results showed that with an ExposeHK interface, users immediately have high hotkey usage for command selection. Through a combination of a selection method with a high ceiling of performance and a smooth transition from novice to expert interaction; ExposeHK is able achieve a high level of command selection performance.

2 Background

The focus of our research is on improving users' command selection performance. Many alternative selection techniques and interfaces that have been proposed to improve users' command selection performance. In the following chapter we have focused and categorized the background work into three different mechanisms of selection: pointer, keyboard and eye gaze. Although there are many other mechanisms of selection, such as speech, these are not the focus of this paper.

2.1 Pointer

The pointer offers a very simple and intuitive mechanism of selection and is often driven by the mouse.

2.1.1 Conventional Hierarchical Menus

Command selection interfaces, such as menus and the Ribbon, tend to group commands into a meaningful hierarchy to allow for quick refinement when searching for a command. Interaction with these command selection interfaces are generally driven by the pointer and allow users to navigate through a visual search and point technique. As users begin to gain knowledge and experience in making command selections they become no longer reliant on visual search for navigation and selection. However, these interfaces force the user to perform the same hierarchical navigation of the interface for selection, thereby trapping the user in a novice method of selection with a low ceiling of performance.

2.1.2 Priority Menus

Many command selection interfaces have been designed to improve command selection performance through knowing that users will select from a small subset of items most of the time [6]. These interfaces are often designed to allow for quick selection through methods such as reducing pointer movement (Pie menus [7] , Hotbox [8]) or reducing search time (Split menus [9], Morphing menus [10], Temporal menus [11]). Through giving preference to commands which are selected most often they are able to improve users' command selection performance. However, methods such as adaptive menus have been shown to be harmful [12] to user command selection performance through not keeping spatial consistency.

2.1.3 Exploiting Spatial Memory

Spatial memory is the ability to recall the location of an item in a particular environment. User's spatial memory is extremely powerful and has been shown that users are able to recall and retrieve items from a large set based on spatial locations [13]. Numerous research has shown that item selection through utilizing the users' spatial memory is quicker than using convention item selection interfaces such as menus [13, 14]. Spatial memory offers improved performance over other acquisition methods that are governed by Hick-Hyman law [15] which has logarithmic search time compared to spatial memory recollection time which is constant. Spatial memory has also been shown to last for a long time as users were still able to recall the location of items after months without interaction [14]. CommandMaps [16] and ListMaps [17] are examples of interfaces which have been shown to improve user performance through removing spatial overloading by giving each item an unique spatial location. For CommandMaps they showed that for novice users the performance was similar to that of using a hierarchical command interface. However, for expert users, the CommandMap was able to improve command selection performance significantly. This showed that once spatial knowledge is learnt, performance gains can be made.

2.1.4 Gesture Interfaces

Gesture interaction interfaces allow for quick selection of commands through mapping a performable action to a particular command. One of the most famous gesture based menus is Marking Menus [18, 19] which allow for

expert performance with radial based menus. Marking menus allow novice users to perform a hierarchal search and navigation of a radial menu interface to make command selections. Through practice users are able to gain experience in making command selections with the radial menus and are able to learn the gesture shapes for commands. Once these gestures are learnt, users are able to make command selections without the need to display the radial menu. This transfers the user from performing a novice hierarchical navigation for selection to an expert gesture interaction for selection. Other interfaces have been proposed to improve Marking menus including Flower menus [20] which looks to map more commands at each hierarchical level therefore reducing the levels for navigation.

2.2 Keyboard

Human performance models such as KLM demonstrate that the keyboard is a very fast mechanism for interaction.

2.2.1 Hotkeys

Hotkeys, also known as key combinations, map a specific key combination to a command (e.g. Ctrl + C is Copy). Once hotkeys have been learnt they have been shown to improve users' command selection performance [21] and are often applicable across multiple applications (e.g. Ctrl + S is Save in many applications). Hotkeys are able to improve users' command selection performance through providing a flat selection hierarchy by removing the navigation through a command hierarchy. Since hotkeys are activated through the keyboard it removes the need for the user to perform a costly homing action to acquire and use the mouse for command selection. Although, a wide range of commands are accessible through hotkeys, users tend to get stuck in a 'satisficing' [22] mode in which they are reluctant to switch to a different mechanism of selection even through performance gains are possible [23].

Hotkey Display Methods

The conventional methods for finding and learning hotkeys are through tooltips or text alongside command names (see Figure 1.1). These display methods force the user to perform the physical action of acquiring the command with the pointer in order to display a hotkey command. Therefore, as the command selection action has already been performed with the pointer the hotkey command is often overlooked and consequently never learnt by the user. These poor methods of hotkeys display result in the user undergoing a performance dip as they transition to the use of hotkeys [24].

Alternative methods have been used to remove the pointer interaction when learning hotkeys such as cheat sheets and virtual keyboards (see Figure 2.1). Cheat sheets present hotkeys as a list of command name and key combination pairs. While virtual keyboards display a keyboard on the screen with commands appearing on the keyboard. Although, these interfaces allow the user to find and activate hotkeys solely through the use of the keyboard, by not using the users' current interface knowledge they force the user to perform a costly visual search to find the desired command hotkey.

Learning Hotkeys

Learning a hotkey to make a command selection is a mentally harder task to achieve compared to pointer based selection. Grossman et al. [5] proposed and evaluated several menu designs to improve hotkey learning based on increasing hotkey feedback or increasing pointer selection costs. Their evaluation identified audio feedback and disabled menu items as the best menu designs for hotkey learning. The audio feedback menu design increased hotkey feedback through a speech response of the hotkey combination when a command was selected from the menu. For the disabled menu items design, they disabled the users' ability to make a command selection with the pointer, thereby, forcing the user to make the command selection with the associated hotkey. They found that these methods of hotkey learning increased hotkey usage significantly over the current method of hotkey text display in menus. This research showed that hotkey learning can be significantly improved through alternative designs.

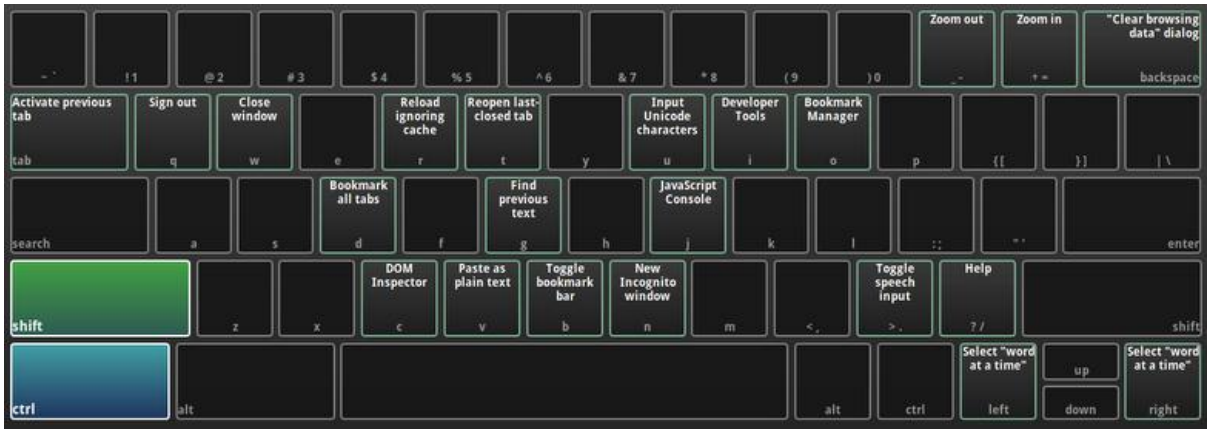

Figure 2.1: Virtual keyboard used in Google's Chromebook for displaying hotkeys.

2.2.2 Alt-keys

Another alternative to hotkeys in applications such as Microsoft Word is Alt-keys. Alt-keys allow the user to browse and select any item from the Ribbon solely through the use of the keyboard. Alt-keys are initiated by pressing the alt key which overlays key tips on the interface to allow the user to navigate the Ribbon interface (see Figure 2.2). This display method has an advantage over cheat sheets and virtual keyboards of using the users' current knowledge of the interface when discovering keyboard shortcuts. However, Alt-keys do have a down side as they require a decision at every level of the command hierarchy. This results in the Alt-key key combinations becoming convoluted and requiring the user to remember longer key sequences (e.g. Copy becomes Alt + H + C instead of just Ctrl + C).

Figure 2.2: Alt-keys interface. Left, key tips displayed at top level of Ribbon. Right, key tips displayed inside a tab once a tab has been selected through a key press.

2.3 Eye Gaze

Currently eye trackers are primarily used by people with disabilities who are unable to use conventional input devices. However, as a result of the decreasing cost [25] and increasing accuracy of eye tracking it is becoming a practical additional input device for able-bodied users. An example of this is Lenovo, who are prototyping a laptop [26] which incorporates a Tobii eye tracker. Therefore, the way to best utilize the users' eye gaze is becoming an area of interest for HCI [27].

2.3.1 Eye Gaze Selection

We have classified eye gaze based selection as interaction techniques which use eye gaze as the primary mechanism of selection. EyePoint [28] uses a combination of eye gaze and keyboard input to offer all the capabilities of the traditional mouse. To select a target with EyePoint the user first looks at the target and presses Ctrl. Pressing Ctrl magnifies the gaze region; the user then reacquires the target and releases Ctrl to select. Through using a combination of eye gaze and keyboard input, EyePoint successfully removes the costly context switch to the mouse required for tradition pointer selection. However, EyePoint does have inefficiencies, as it requires four actions for selection from the user and forces the user to reacquire the command in the magnified interface. The evaluation of EyePoint showed it had similar performance to using the mouse and users had a significant subjective preference towards EyePoint.

2.3.2 Eye Gaze to Assist Selection

Eye gaze has the potential to be a fast and natural selection method, however issues such as tracking accuracy and rapid eye saccades result in eye gaze often not being used as the primary selection mechanism. Therefore, eye gaze is often used as a mechanism for helping the user make a selection with another input device. MAGIC (Manual and gaze input cascaded) [29] is a method to improve pointer performance through utilising eye gaze to quickly transport the cursor to a particular screen location and then the mouse is used for precise selections. MAGIC pointing was able to benefit from the fast movement of eye gaze while overcoming the small inaccuracies through the addition of the mouse for precise interactions. An evaluation of MAGIC pointing showed that after a short period of interaction users were able to achieve a similar performance to mouse based selection. K. J. Räihä et al. [30] proposed an alternative eye gaze supported method for improving pointer performance by using eye gaze to disambiguate between the multiple cursors used with Ninja cursors [31].

3 Design Goals for Fast Command Selection

In this chapter we will outline and explain our goals that we have identified for fast command selection performance.

3.1 Fast Mechanism

One of the key aspects of fast command selection performance is a fast mechanism of interaction. There are multiple mechanisms that can be used for command selection including pointer, keyboard, eye-gaze, and speech. In the following section we will focus on pointer, keyboard, and eye-gaze mechanisms for selection.

3.1.1 Pointer

Selection through a pointer offers a very simple and intuitive mechanism of selection. Pointer based mechanisms are suitable for novice users as they allow for a simple search and point technique. In this paper we are focusing on pointer based interaction with the mouse. Users are very familiar and generally efficient with pointer interaction due to its high use in WIMP (Windows, Icons, Menus, Pointer) interfaces. Pointer selection for most applications has extremely small mental demand and often only requires the user to remember the spatial locations of items.

The users' achievable level of performance with mouse selection is limited by Fitts' law [32]. Fitts' law governs the level of performance achievable for a selection as a measure of the distance and size of the target to select. Methods such as increasing target size or decreasing distance are often employed to improve pointer selection performance.

For applications, such as Microsoft Word, where the user's hands are predominantly placed on the keyboard while working, selection with the pointer requires the user to perform a switch from the keyboard to the mouse. Once the command selection is complete the user will then often move their hands back to the keyboard to continue working. This round trip switch from the keyboard to the mouse and back to the keyboard when performed hundreds of times daily has a significant effect on performance.

3.1.2 Keyboard

Keyboard interaction allows for the ability to make a wide range of command selections quickly. Models of human performance such as Keystroke-level model (KLM) [2] demonstrate that keyboard is an extremely fast mechanism of interaction. Keyboard selection has advantages over pointer selection through not requiring a input device context switch to acquire the mouse and not being restricted by Fitts' law.

Keyboard selection generally has a high mental demand as users often have to blindly recall key combinations for commands. Whereas, pointer selection allows for a much simpler search and point technique with experienced users often only having to learn the spatial location or hierarchical steps for selection. Good mappings between key combinations and commands allow the user to quickly and easily remember the necessary keyboard input. There are also keyboard navigation techniques such as Alt-keys allow the user to browse and make command selections solely through the use of keyboard. Techniques such as Alt-keys should be able to lower the mental demand on the user as they do not have to blindly recall keyboard combinations.

3.1.3 Eye Gaze

Generally for command selection through the conventional interfaces with a mouse the user's eyes locate the command before the necessary mouse movements can be completed. Therefore, if we are able to remove the need for the mouse movement for selection then performance improvements are possible.

Using the eyes as an input device has benefits such as speed, lower physical strain, and natural interaction. Eye gaze offers a natural input which is extremely fast to manoeuvre. Compared to other input devices such as the mouse, eye gaze it does not strain the user's body leading to injuries such as RSI. The eyes also do not have issues with the mouse sensitivity where different mice are calibrated with different control-gain functions.

There is a range of eye tracking systems starting from simple webcam tracking to dedicated hardware such as Tobii eye trackers. High end eye trackers tend to track the user's eye through sending infrared light and tracking the corneal reflection. However, Tobii eye trackers still have inaccuracies of around half a degree of visual angle. This inaccuracy of the user's eye gaze translates to around 0.56cm variability or 21px with the user at 64 cm from the screen [33].

Eye tracking stability suffers from the small involuntary saccades performed by the human eye. These saccades are detected by the eye tracker and lead to sporadic data. These small movements make interactions using the eyes hard for precision target acquisition. The small involuntary movements of the eye were shown by Yarbus [34] in which he performed experiments in which participants were asked to track the outlines of shapes while trying to not perform saccades (see Figure 3.1). Eye tracking accuracy is also harmed by a factor known as Drift. Drift refers to inaccuracy of the eye tracking resulting from factors such as moisture in the eye and changes in sunlight intensity. Calibration is used by eye trackers to negate these effects on a per user basis.

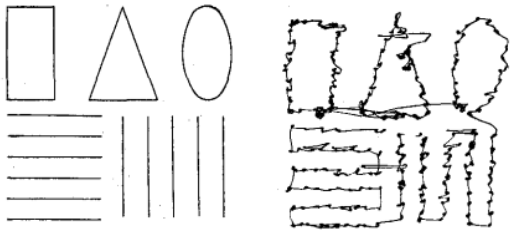

Figure 3.1: Involuntary saccades in the eye demonstrated by Yarbus [34]. Left: the shapes the participants were asked to track with their eyes. Right: The eye movements of a participant trying to trace the shape outlines.

The final issue is the Midas touch problem, where if the eyes are the sole input device it is not possible to deactivate the eyes and therefore you get unintentional selections. To overcome the Midas touch issue, methods such as fixation are used in which the user must fixate their eyes on the target for approximately a second in order to confirm a selection. However, this leads to performance dip due to the fixation waiting time. Often a secondary input device is used [35], such as a keyboard, to remove this lag time brought by fixation.

3.2 Novice to Expert Transition

One important aspect of a new selection technique is ability and encouragement for the user to transition to expert selection performance. Issues arise when a user gets stuck in a 'satisficing' mentality and is reluctant to switch selection technique even through performance improvements are possible. Marking menus [18] are a good example of allowing the user to transition to expert levels of performance by having the user practice the same movement of selection during their novice stages of interaction. Then once the user has had sufficient practice they are able to make the same movements for selection but no longer require the hierarchical searching as they have been able to mentally map a movement to a command. This interaction technique means that users do not get stuck in a selection method with a low ceiling of performance. Therefore, it is important that the new interaction technique allows the user to seamlessly transition from novice to expert performance.

3.3 Few Actions

As well as having a fast mechanism of interaction it is also important to reduce the number of actions needed to make a selection. A fast selection mechanism alone is not going to lead to a fast selection performance if the user has to take excessive steps to make a selection. Interfaces such as the Ribbon

which group commands into meaningful categories require the user to navigate through a hierarchy to make the command selection. Therefore, if the intermediate navigation steps can be reduced, command selection performance can be improved. Interfaces such as CommandMaps [16] and hotkeys are examples of reducing the number of actions needed for selection through flattening the command selection hierarchy. Both of these selection methods have been shown to improve command selection performance.

3.4 Decrease Context Switches

3.4.1 Maintaining Focus

An important aspect for improving user command selection performance is to require minimal attention from the user's current activity. Not only does a task switch incur a time cost but user's actions tend to be more error prone after a task switch [36]. A switch also incurs a longer term cost as users become slower at the task when a task switch has been performed. Therefore, by allowing the user to remain focused on a task with minimal to no task switch we will be able to eliminate any cost in time and errors. Furthermore, the addition of a switch in focus requires the user to put to memory the task that they were performing before the switch in focus. This switch in focus can also lead to the user forgetting the previous task depending on the mental requirement of the task being performed [37, 38].

3.4.2 Input Focus

Models of user performance such as Keystroke-level model (KLM) highlight the cost of switching input devices. In the KLM the cost of switching from the keyboard to the mouse and vice versa, also known as homing, incurs a time cost of 0.4 seconds. Therefore, in applications like Microsoft Word where the users' hands are predominantly placed on the keyboard, pointer selection will incur a 0.8 second penalty just for the switch to and from the mouse. By removing any switching in input devices we are able to remove the homing penalty for command selection.

3.5 Extent

3.5.1 Applied to Different Applications

It is important that the new interaction technique can be applied to multiple real applications. Hotkeys are a good example of a command selection technique that is extendable across applications as they do not require a change in the current interface. This will allow the user to be able to use the interaction technique for multiple applications reducing the performance dip from interacting with multiple interfaces.

3.5.2 Applied to Current Selection Interfaces

Through applying the selection technique to existing interfaces it allows the user to use their current knowledge of the interface when learning a new interaction technique. This allows the user to quickly find and issue commands in the new interface therefore reducing the initial learning curve. By not redesigning or replacing the user's current interface users will be more accepting as it will not be completely removing what the user is familiar with.

4 Design of Eye Gaze Based Interfaces (Failure)

Our first attempt at improving users' command selection performance was through utilizing the users' eye gaze. Eye gaze was selected as it has the potential to improve command selection performance through fast movement, requiring no input device switch, and natural interaction. We designed two different interaction techniques aimed improve user command selection performance; HotkeyInSight and CommandInSight.

Both of the InSight interfaces (HotkeyInSight and CommandInSight) look to remove the hierarchical navigation of selection through, when requested, displaying commands which normally reside in the spatial vicinity of the user's eye gaze. This removes multiple steps in the selection process through removing the hierarchal navigation to find commands. The InSight interfaces can be applied to almost any command selection interface as it is able to cope with spatial overloading of commands. By using the current layout of the commands in an interface it allows the intermediate users, who have spatial knowledge of commands, to easily transition and become efficient in using the InSight interfaces.

4.1 HotkeyInSight

HotkeyInSight aims to help users transition to hotkeys without the user experiencing a performance dip in the hotkey learning process. When activated, HotkeyInSight presents to the user the hotkey shortcuts to the commands that normally reside in the spatial vicinity of the users' current eye gaze. This allows the user to quickly find and complete a hotkey shortcut through the use of eye gaze and keyboard input. Though allowing the user an easy way to find and activate hotkeys we are able to efficiently train the user in the use of hotkeys. Therefore, we are able to remove any chance of the user being stuck in a 'satisfying' mentality with a low performance selection method.

Selection Procedure

To make a command selection the user first looks at the spatial screen location where the command the user wants to select normally resides. The user then presses and holds a modifier key (Ctrl, Alt, Windows key) which then overlays on top of the interface the commands which reside in the vicinity of the user's eye gaze along with their associated hotkey combination (see Figure 4.1). Once displayed, the HotkeyInSight interface will not update as a result of the user changing their gaze position. This is to allow the user to move their eyes to read the hotkey combination. When the user releases the modifier key the HotkeyInSight interface is hidden. There is also no pointer or dot to show the user current gaze position as we felt this would intrude on normal workflow.

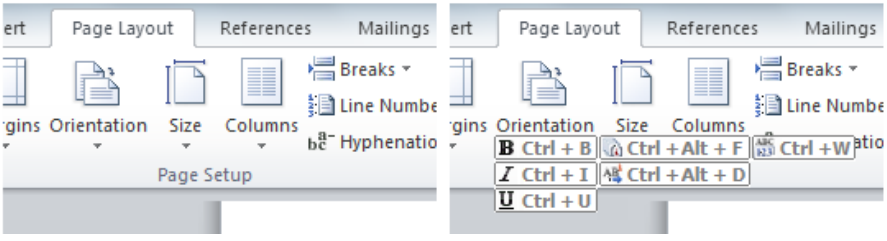

Figure 4.1: HotkeyInSight command selection procedure: 1) Look at screen location 2) Press modifier key to activate HotkeyInSight 3) Find command on overlaid interface 4) Perform hotkey action.

4.2 CommandInSight

CommandInSight is a global hotkey which, when activated through the keyboard, will overlay on the interface all the commands that reside in the spatial vicinity to the user's eye gaze. CommandInSight has an advantage over HotkeyInSight by allowing selection of commands that do not have an associated hotkey. CommandInSight also overcomes the Midas touch issue for selection through the user pressing the spacebar to confirm selection.

Selection Procedure

The activation process is the same as HotkeyInSight, with the user first looking at a particular screen location and pressing a modifier key to display commands that reside in the vicinity of the user's eye gaze. When the CommandInSight interface is shown a red dot is displayed to show the user's current eye gaze position. The user is able to make a selection of a command from the CommandInSight interface through gazing at the command's icon and pressing the spacebar to confirm (see Figure 4.2).

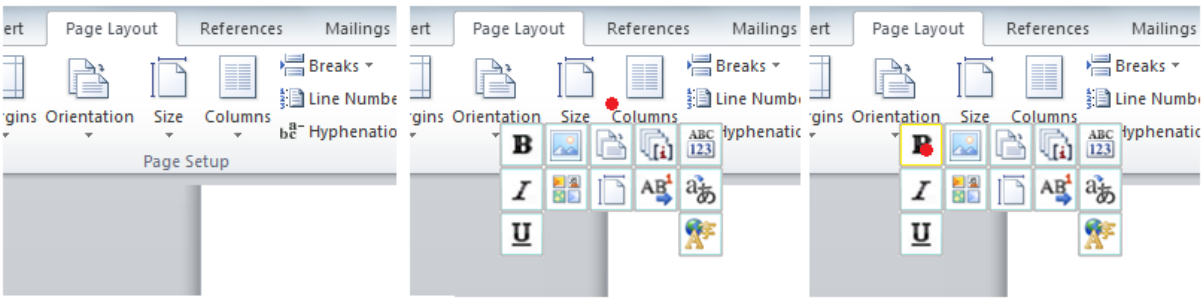

Figure 4.2: CommandInSight command selection procedure. 1) Look at screen location 2) Press modifier key to activate CommandInSight 3) Gaze at command on overlaid interface 4) Press space to confirm

4.3 Design and Implementation

4.3.1 Eye Tracking

To track the user's eye gaze a Tobii TX300 eye tracker was used. The Tobii API allows for listening of various eye events including gaze data, eye distance and number of eyes detected. Since we are only interested in the eye gaze data we wrote appropriate methods to be called when new gaze data was made available. Due to the small involuntary saccades in the user's eye it was necessary to implement a smoothing algorithm to allow for stable eye control. Simple smoothing methods such as moving average were tested but lead to a noticeable lag when eye movement was large. Therefore, we implemented a One Euro Filter [39] for our smoothing algorithm as it had been shown to improve smoothing accuracy and also lag, over the moving average and Kalman Filter. The One Euro filter resulted in a responsive and accurate gaze marker that was used for the CommandInSight interface.

4.3.2 Activation and Confirmation

We decided to use a combination of eye gaze and keyboard input to display and interact with the InSight interfaces to remove any input context switch to the mouse. Using a modifier key for the activation of HotkeyInSight means that it places the user part way to completing a hotkey selection. For the CommandInSight interface we decided to use keyboard input for command confirmation to overcome the Midas touch issue of accidental selection. This solution was chosen over an alternative such as fixation for selection as it does not introduce a lag time which would be detrimental to command selection performance.

4.3.3 Distance and Size Considerations

The activation radius refers to the distance value for which commands that within this distance from the gaze point are shown in the overlaid interface. The decision on the value for the activation radius relies on the accuracy of the eye tracker and importantly the accuracy of the user's spatial memory. The Tobii eye tracker being used had an average error rate of 21 pixels when the user was 64 cm from the screen [33]. Scarr et al.[16] performed an experiment to determine to what level of accuracy users were able to recall the locations of

commands in Microsoft Word's Ribbon. They found that the median error distance was 92 pixels. However, in their experiment users were required to point to the location on a blank Ribbon, so did not have visual aids which would have helped in determining the relative position of a command in the Ribbon interface. For this reason we used a activation radius of 80 pixels as users will be more accurate if they have some visual landmarks [40, 41]. For the CommandInSight interface we decided to have the icons size of 50px x 50px. This was chosen as it balances the distance the eyes must travel while also being large enough for quick selection and invulnerable to small inaccuracies in the eye tracker.

4.3.4 Command Layout

The initial idea we had for the overlaid command layout was a spring layout. A spring layout is a force based graph layout algorithm in which graph nodes are attracted towards their original locations but are repelled by other nodes which reside in the same location. However, displaying many commands led to command icons having a large displacement from their original location. Furthermore, the spring layout led to random positioning of commands so users were unable to quickly tell from which category an icon belonged, leading to users performing a costly visual search through all commands.

Our final command layout design was a grid layout in which the commands are grouped together according to their Ribbon tab, making it easy to search for a command in the interface. This design was used by both the InSight interfaces. This interface layout was chosen as it has a simple mapping metaphor from the categories to the overlaid interface, allowing the user to quickly refine their visual search for a given command. However, this interface layout results in commands that belong to the first or last categories being significantly displaced from their original location. It also does not give information as to the command's original spatial location like the spring layout algorithm.

4.4 Why it failed

Both HotkeyInSight and CommandInSight were able to provide responsive and interactive interfaces. However, there were accuracy issues which resulted in both of the interfaces being difficult and slow to use for command selection.

4.4.1 Eye Tracker Inaccuracy

The Tobii eye tracker has an error level of 0.5 degree of user sight which translates to approximately 21px when the user is 62 cm from the screen. Although, this inaccuracy seems small it still lead to a noticeable difference in the actual gaze position and the reported gaze position. The error distance also tended to increase when the user's eyes gaze was close to the edges of the screen.

4.4.2 Inaccuracy in Recalling the Spatial Location of Items

When testing the interface we found it was hard to determine the normal spatial location of commands without them being displayed. Solutions such as increasing the activation radius resulted in too many commands being displayed due to the heavy spatial overloading and closeness of commands. Therefore, the inaccuracies in the user's spatial knowledge along with the tracking inaccuracies resulted in large errors in trying to expose the desired command.

5 Design of Keyboard based Interfaces – ExposeHK

Following the failed eye gaze interfaces we looked to develop an alternative hotkey display method. From the InSight interfaces we were able to identify some design aspects that were helpful in the learning of hotkeys such as removal of pointer interaction for hotkey feedback, modifier activation and using the user's current interface knowledge.

Figure 5.1: ExposeHK technique allowing the discovery of hotkeys through keyboard interaction.

We developed ExposeHK as a new interface technique that helps users browse, perform and learn hotkeys within the hotkey modality to improve users' command selection performance. By using hotkeys as the command selection method it allows users to achieve a high level of performance through advantages such as a flat selection hierarchy, fast keyboard interactions, and removal of mouse interaction. Hotkeys also achieve our goal of maintaining focus through allowing the user to make command selections with minimal switch in task focus.

ExposeHK is activated by a modifier key and removes the need to use the pointer to activate hotkey feedback (see Figure 5.1). Therefore, allowing users to browse, inspect, confirm and issue hotkey commands within the hotkey modality. When activated ExposeHK displays the hotkey combinations for commands above the command icons in an interface. This allows intermediate users the ability to use their knowledge of the interface to help them quickly find a command's hotkey combination.

ExposeHK allows the user to perform a seamless transition from novice to expert performance through providing novices an easy method for hotkey discovery. This allows novice users the ability to learn hotkeys through practice while avoiding the user falling into a 'satisficing' mode with a low performance selection technique. As users begin to repeatedly make command selection with hotkeys they will start to learn and become faster at activating hotkeys until finally they will no longer require the assistance of ExposeHK.

ExposeHK has the ability to be easily applied to interfaces which use icons. In the following sections we have applied ExposeHK to the Ribbon and CommandMap interface. Similarly, it can be applied to alternative interfaces through allowing browsing with the keyboard.

5.1 ExposeHK_R – ExposeHK for the Ribbon

ExposeHK was applied to the Ribbon interface to create ExposeHK_R (see Figure 5.2). Since the Ribbon is made up of many tabs leading to spatial overloading, it is necessary to allow the user to switch tabs in order to find commands. To achieve the design goal of allowing command selection with no input context switches we added the ability for the user to switch tabs through the use of the arrow keys.

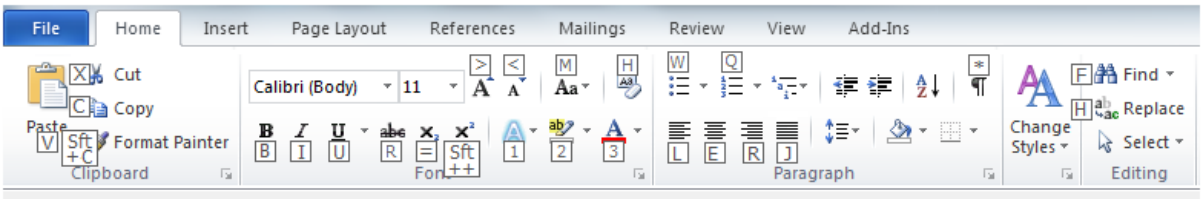

Figure 5.2: Ribbon interface with ExposeHK_R activated.

5.2 ExposeHK_CM – ExposeHK for the CommandMap

We looked to improve the ExposeHK_R interface by removing the need to tab switch in order to find commands. The CommandMap is an example of removing the intermediate step of tab switching by flattening the Ribbon hierarchy. Therefore, we applied ExposeHK to the CommandMap to create ExposeHK_CM. When the user presses a modifier key a CommandMap containing all the commands and associated hotkeys are shown to the user.

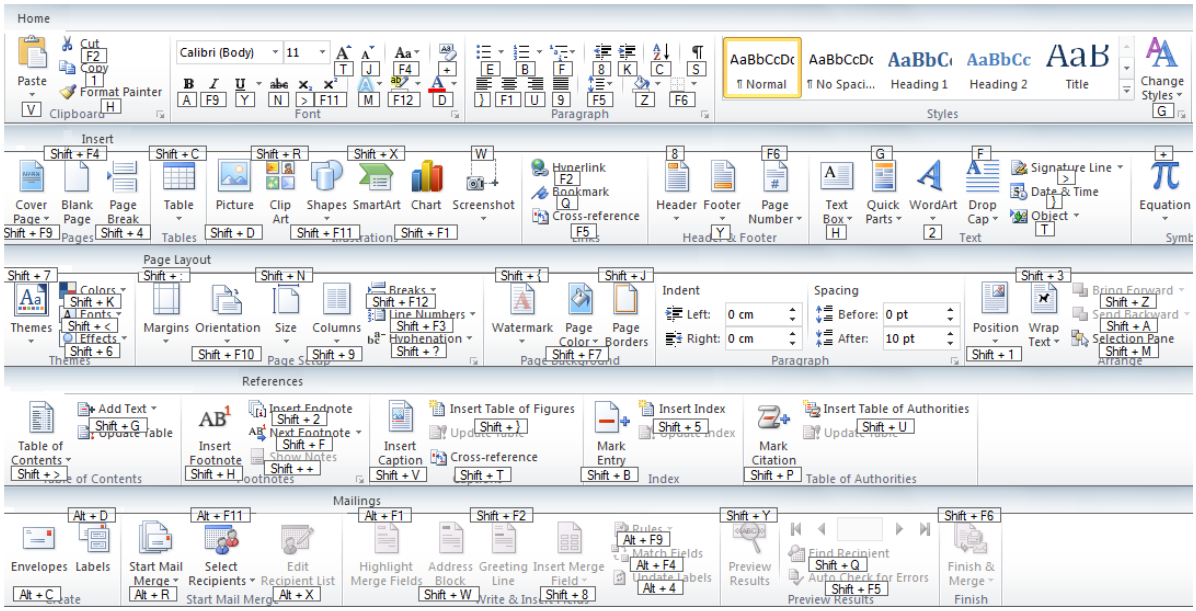

Figure 5.3: ExposeHK_CM interface.

6 Evaluation of ExposeHK_R

The focus of this evaluation is to compare the command selection performance of ExposeHK_R to current selection techniques in the Ribbon; Pointer and Alt-keys. Pointer selection was chosen as it is a common method of selection in the Ribbon interface. While Alt-keys was chosen as it is the current method of selection that allows browsing and selection in the Ribbon through the use of the keyboard.

We are also interested in the level of performance that is obtainable with each selection technique as, with everyday usage, users would be using the technique to make thousands of selections. In addition, we are also interested in the subjective preference of users for the different techniques.

6.1 Design, Participants and Apparatus

The design of the experiment is a 3 x 8 x 2 analysis of variance with the factors *technique (Pointer, Alt-keys, ExposeHK_R), block (1–8)* and *tab switch (true, false).* The factor *block* looks at the users' performance as they gain experience in making command selections. The factor *tab switch* looks at the difference between selections which are on the current Ribbon tab and those that require the user to switch tab to find. The *technique* factor was counter-balanced across the participants. 18 participants (ages 21 – 45, 1 female) were recruited from the computer science department and were given a $10 voucher for their time. The experimental system was developed in C# and was run on Windows 7 computer. The screen used was 1280 x 1024 22" display along with an external QWERTY keyboard and optical mouse.

6.2 Procedure

The experiment task was to select items from a Ribbon like interface. Each participant completed the tasks with each technique; Pointer (mouse selection), Alt-Keys, and ExposeHK_R. For each technique the user had to complete the selection with the given technique (e.g. If for Alt-keys, the user clicked on the item no action was taken). The experimental Ribbon interface consisted of 7 tabs with 18 items on each tab. Each item was 30px x 30px and would highlight a border when the mouse was moved over it.

For each technique, 3 items from the 1st tab and 3 items to select from the 4th tab were randomly chosen for selection. It was made sure an item to select did not appear as a selection for a previous technique. For each technique participants completed 8 blocks of selections, with a block of selections consisting of 2 selections of each of the 6 items. The item selection order was randomly generated at the start of each block and was made sure that half of the selections would require a tab switch and consecutive items were not the same. Across the 8 selection blocks the 6 items to select remained consistent, allowing users to gain experience in making those selections. Before each technique participants completed 8 selections on a practice interface which consisted of 3 tabs with 8 items on each tab.

To start a selection the participant would have to move the cursor to a 70px x 70px square in the center of the screen and then press space. Once space was pressed, text was displayed showing the name of the item to select and the timer for the selection was started. Upon making a correct selection a sound was made and text was displayed to tell the user to move back to the centre square and press space (see Figure 6.1). Once space was pressed the timer for that item selection was stopped. Requiring the user to press space to start and end a selection was used to simulate a round trip command selection; where initially while a user is working, their hands are placed on the keyboard and then once the selection is made they return to working on the keyboard.

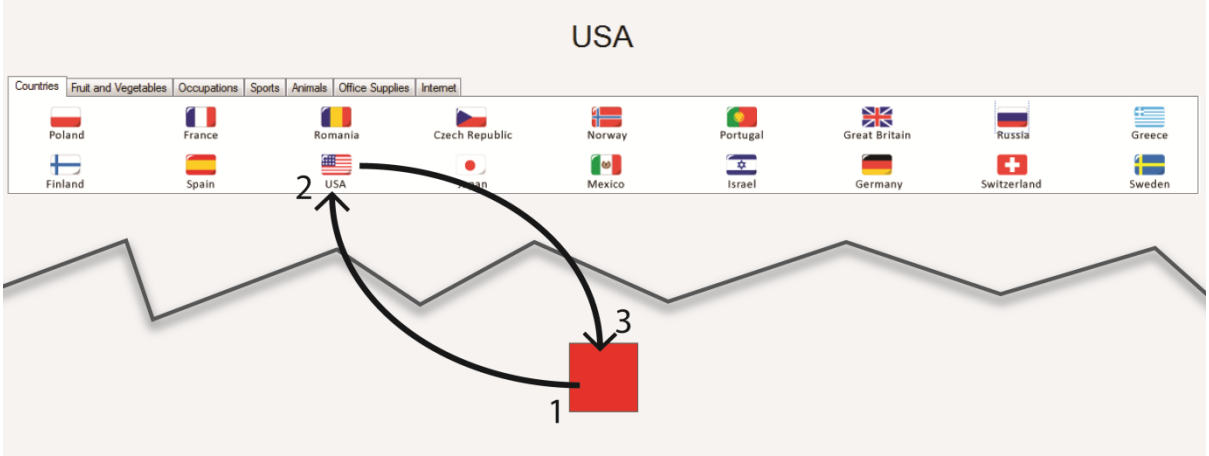

Figure 6.1: Procedure for a selection. The numbers indicate the steps in making a selection.

For each technique the user was able to switch tabs using the arrow keys, mouse scroll wheel or pressing the tab title on the Ribbon interface. For the ExposeHK_R when a user successfully made a selection of an item that was located on a different tab, the current tab was switched to the tab for which the selected item belonged.

For the ExposeHK_R technique the hotkey for an item consisted on the Ctrl key plus a letter key. The letter keys were chosen from the left most 12 letter keys on the keyboard (Q,W,E,R,A,S,D,F,Z,X,C,V). It was made sure that the letter key did not match the first or last letter in the associated item. This was to remove any association that could make the hotkey easy to remember. Similarly for Alt-keys technique at both the tab and item level the associated key did not match the first or last letter of the particular tab or item.

Participants were encouraged to complete the item selection as fast and accurately as possible. When a user made an incorrect command selection it resulted in a 3 second lockout form the interface. This was to discourage users form guessing hotkeys or Alt-keys when they were not completely certain. A sound was also made when an incorrect selection was made that was different form the correct selection confirmation sound.

The time taken, item selected, tab switch, key presses and errors were recorded for each selection. Upon completing each interface participants completed a NASA-TLX and at the end of the experiment completed a questionnaire ranking the interfaces for 3 different questions.

Figure 6.2: ExposeHK_R technique when Ctrl was held down

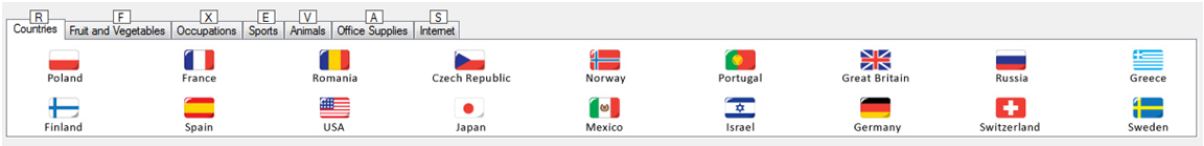

Figure 6.3: Alt-keys with keys tips displayed for each tab

Figure 6.4: Alt-keys when a tab as second level of selection with key tips displayed above each item

6.3 Results

6.3.1 Selection Times

The average selection time per block is summarised in Figure 6.5. Selections with errors were removed from the timing results. From the results there was a significant main effect for *technique* ($F_{2, 34} = 34.55$, $p < 0.001$) with mean selection times of 3133.53ms (s.d. 975.08), 4491.98ms (s.d. 2048.92) and 2862.86ms (s.d. 1624.64) for Pointer, Alt-keys and ExposeHK_R respectively. A pair-wise comparison found that there was a significant difference between Alt-keys and Pointer also Alt-keys and ExposeHK_R but no significant difference between Pointer and ExposeHK_R.

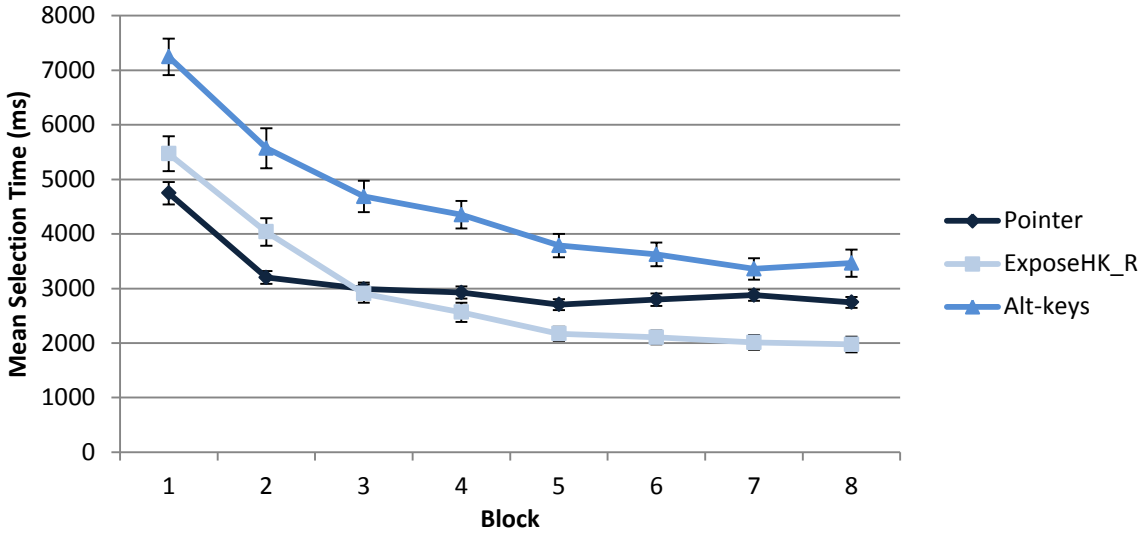

Figure 6.5: Mean selections time per block. Error bars show standard error.

As mentioned previously, we are interested in the selection performance that is achievable with each technique. Therefore, we performed an analysis of variance on the final block data. We found that there was a significant main effect for *technique* ($F_{2, 34} = 13.73$, $p < 0.001$) with mean selection times of 2765.12ms (s.d. 608.84), 3382.02ms (s.d. 1487.72) and 1950.83ms (s.d. 856.77) for Pointer, Alt-keys and ExposeHK_R respectively. A post-hoc analysis showed that there was a significant difference between all pairs of techniques for block 8.

There was a significant main effect for *tab switch* ($F_{1, 17} = 282.50$, $p < 0.001$), as items which require an intermediate tab switch to find are going to take longer to select. There was a significant main effect for *block* ($F_{7, 119} = 111.20$, $p < 0.001$), as users were able to become faster in their command selection as they gained experience through the blocks.

There was a significant *technique X block* ($F_{7, 14} = 9.69$, $p < 0.001$) interaction. This interaction is illustrated in Figure 6.5 with Pointer selection performance improving significantly less than both Alt-keys and ExposeHK_R from block 2 onwards. This slowdown in improvement with Pointer selection consequently results in ExposeHK_R becoming faster than Pointer selection by block 3.

There was a significant *technique x tab switch* ($F_{2, 34} = 10.61$, $p < 0.001$) interaction, which can be explained by Alt-keys requiring a heretical navigation of the interface to select a command regardless of if the user is already on the correct tab. This is shown in Figure 6.6 with Alt-keys having a lower improvement in performance compared to Pointer and ExposeHK_R when no tab was necessary. A more detailed view of tab switch performance per block is shown in Figure 6.7 and 6.8.

There was a significant *tab switch x block* ($F_{7, 119} = 8.46$, $p < 0.001$) interaction. This interaction can be explained by users being able to gain experience in tab switching and learning that the items were only being selected from two tabs. There was a significant *technique X block X tab switch* ($F_{14, 238} = 2.81$, $p < 0.005$) interaction. This interaction is most probably due to users learning hotkeys and as a result there becomes no performance dip due to tab switching with ExposeHK_R.

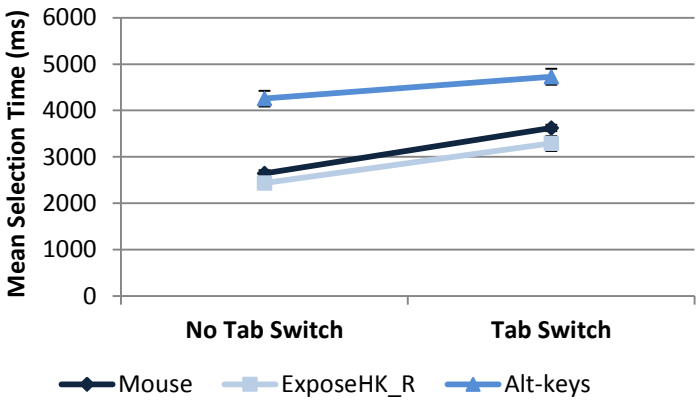

Figure 6.6: Mean selection time for tab switching. Error bars show standard error.

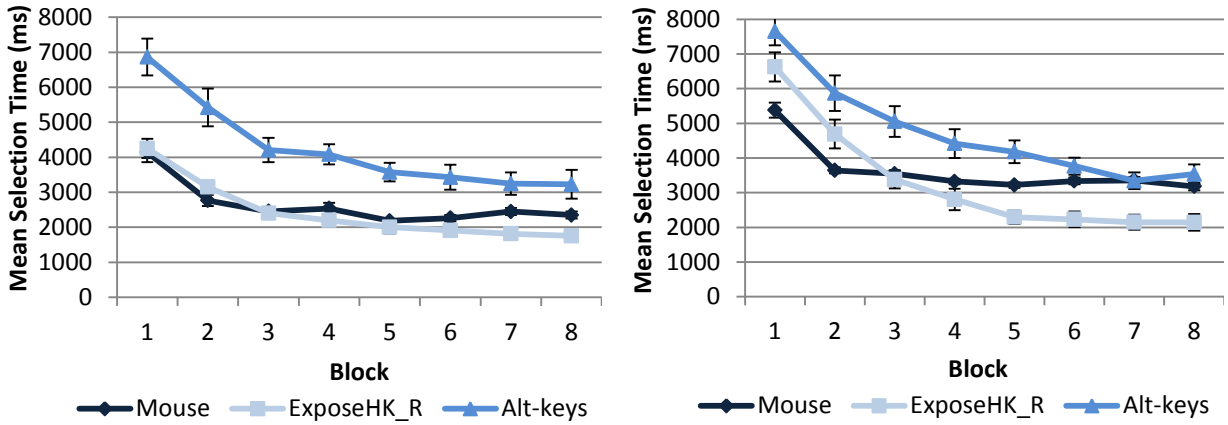

Figure 6.7: Mean selection times for selections which do not (left) and do require a tab switch (right). Error bars show standard error.

6.3.2 Errors

The error results showed a significant main effect for *technique* ($F_{2, 32} = 13.48$ $p < 0.001$). The mean error rates were 0.21% (s.d. 1.32), 1.62% (s.d. 3.94), and 2.15% (s.d. 4.05) for Pointer, ExposeHK_R and Alt-keys respectively. The higher error rates for ExposeHK_R and Alt-keys can be explained through issues such as incorrect recall for key combinations or finger slips for the key combinations selections. Overall there were a very small number of errors with total error sums of 7, 53, and 70 out of 1728 selections for Pointer, ExposeHK_R and Alt-keys respectively. There was no significant main effect for *block* ($F_{7, 119} = 0.59$ $p = 0.764$) which demonstrated that errors occurred randomly across the blocks.

6.3.3 Subjective Responses

NASA-TLX

The NASA-TLX (Task Load Index) results from the experiment are summarized in Figure 6.8. A Friedman test of the NASA-TLX responses showed that there was significant difference in the responses for all of the NASA-TLX categories except for temporal demand.

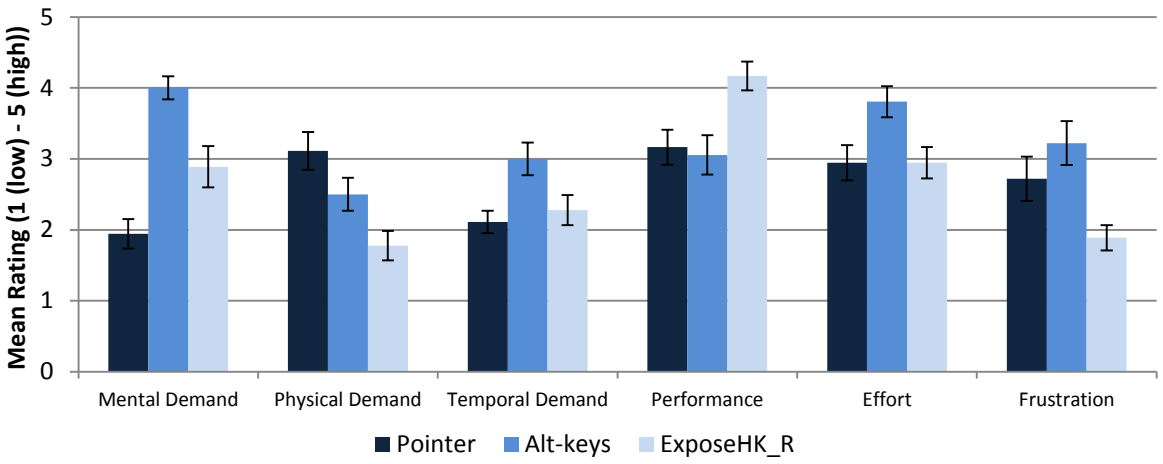

Figure 6.8: Mean NASA-TLX responses. Error bars show standard error. Lower scores are best except for performance.

For mental demand there was large difference in the mean responses with Pointer selection having the lowest mental demand. ExposeHK_R has very low physical demand and frustration compared to both Pointer selection and Alt-key selection. The effort results for ExposeHK_R and Pointer were very similar with Alt-keys having significantly worse results. ExposeHK_R had significantly higher mean performance compared to Pointer and Alt-keys.

Preferences

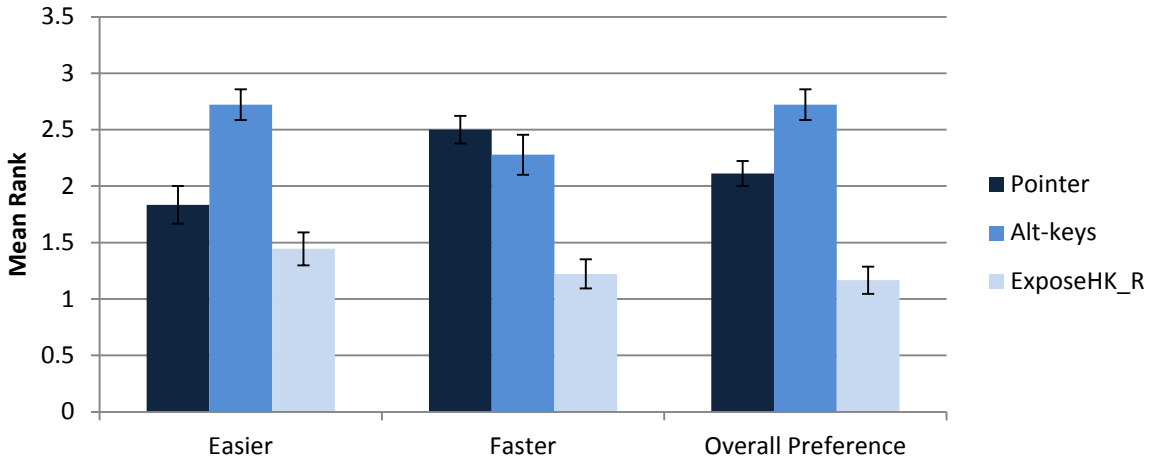

Figure 6.9: Mean preference ranking responses. Error bars show standard error. Lower scores are better.

After the experiment each participant was asked to rank the three selection techniques for three questions. The three questions were: which technique they felt was easier, which was faster and which they preferred overall. The results of the questionnaire are summarized in Figure 6.9. Friedman tests showed that there was significant difference in the responses for all 3 questions with ExposeHK_R having the best average ranking for all three questions.

6.4 Discussion

The main goal of this evaluation was to compare the performance of ExposeHK_R to the selection techniques which are already implemented in the Ribbon interface. Our evaluation showed that ExposeHK_R is significantly faster than Alt-keys, which is the current method in the Microsoft's Ribbon for browsing and performing selection with the keyboard. Overall ExposeHK_R was shown to have comparable performance to Pointer selection. However, looking at the obtainable performance of each technique, through analysing the final

selection block data, ExposeHK_R was able achieve significantly faster selection performance, with ExposeHK_R achieving 28% and 44% faster selections than Pointer and Alt-keys respectively.

There were mixed results for the NASA-TLX responses with Pointer selection having a lower mental demand but ExposeHK_R having significantly higher performance along with lower physical demand and frustration. Our preference ranking responses showed that users preferred the ExposeHK_R technique for selection. Overall this evaluation demonstrated that ExposeHK_R not only leads to significantly faster selection performance but users also prefer it to the current selection techniques.

Why ExposeHK_R was able to achieve faster command selection performance?

ExposeHK_R was able to achieve fast selection performance through allowing users easily and efficiently practice to learn hotkeys. This allowed users to seamlessly transition to the expert selection performance of hotkeys. This is in comparison to Pointer selection in which the user's ultimate performance is limited by their mouse movement speed. With Pointer selection, the low ceiling of performance is illustrated in Figure 6.5 with Pointer performance improvements slowing down at block 5. This suggests that users knew the tab and spatial location of items but were just limited by mouse movement speed.

The tab switching results further demonstrated the benefits of learning hotkeys with ExposeHK_R. Once users learnt hotkeys they were able to benefit from the flat selection hierarchy with users achieving selection improvements over the Pointer of 32.5% for selections which required a tab switch in the final block. ExposeHK_R was also significantly faster than Alt-keys for both selections which required a tab switch and selection which did not, as Alt-keys required the hierarchal navigation for command selections regardless of if the user was in the correct tab to begin with.

Mental Demand of Key Combinations

The NASA-TLX responses showed that Pointer selection had a very low mental demand compared to both ExposeHK_R and Alt-keys, with ExposeHK_R also having a far lower responses compared to Alt-keys. The low mental demand for Pointer selection is due to participants only needing to learn spatial locations of items and parent tab while the other selection techniques required participants to learn and recall key combinations. Many participants commented that it would have been easier to learn key combinations if there was an easy association between the command and the hotkey. This suggests that in real applications where commands have hotkeys with meaningful associations the mental demand of ExposeHK_R and Alt-keys would decrease. Furthermore, once past the initial stages of learning key combinations mental demand may decrease as the participants have sufficient experience and can use their muscle memory for activation.

Hotkey associations

In the experiment we removed any easy association that would be possible for a key combination of an item for both ExposeHK_R and Alt-keys. However, in a real application, the hotkeys are made easy to remember through having an easy association between the key combination and the command (e.g. Ctrl + B for Bold or Ctrl + C for Copy). Therefore, in our experiment we were experimenting with a worst case scenario for both ExposeHK_R and Alt-keys. Even with no obvious association ExposeHK_R, was still able to lead to a significant difference in obtainable performance. The use of hotkeys with easy association will only improve the performance of ExposeHK_R as it should improve the initial stages of learning hotkeys.

Hand positions

For the evaluation item selection procedure we required the user to press space at the start and end of a selection to simulate a round trip command selection. However, in the experiment for Pointer selection users tended to place one hand on the space bar and the other on the mouse; thereby never having both hands on the keyboard. By both hands not being on the keyboard it does not fully simulate the normal user hand placement for applications such as Microsoft Word. We could have emphasised the need for both hands to be placed on the keyboard at the start and end of a selection by requiring two buttons to be pressed that would have required the participant to use both hands. Doing this would have decreased the performance level of the Pointer selection while ExposeHK_R and Alt-keys performance would have only been slightly decreased if at all. We decided not to do this as we wanted to perform the same experimental method that Grossman et al.[5] used in their study of hotkey learning techniques.

7 Evaluation of ExposeHK_CM

The focus of this evaluation is to compare the selection performance of ExposeHK_CM to the pointer based CommandMap. Additionally, like the evaluation of ExposeHK_R we are also interested in the obtainable selection performance of each technique and the subjective preferences.

7.1 Design, Participants, and Apparatus

The design of the experiment was a 2 x 8 x 3 analysis of variance with the factors *technique* (ExposeHK_CM, CommandMap), *block* (1–8) and *distance* (close, medium far}. The factor *block* looks at the users' performance as they gain experience in making the command selections. The factor *distance* looks at the difference between selections which are at different screen distances from the user's current cursor position. 18 participants (ages 21 – 45, 1 female) were recruited from the computer science department and were given a $10 voucher for their time. The order of the factor *technique* was counter balanced across the participants. The experimental system was developed in C# and was run on Windows 7 computer. The screen used was 1280 x 1024 22" display along with an external QWERTY keyboard and optical mouse.

7.2 Procedure

The experimental task was exactly the same as in the evaluation of ExposeHK_R with the user making items selections form a CommandMap like interface. The CommandMap interface consisted of 7 tabs and 18 items per tab with each item being 30px x 30px in size.

The procedure for item selection, stimulus, and error penalties were consistent with the evaluation of ExposeHK_R. The 6 items to select for a technique were chosen based on their distance from the screen center point. The 3 distances used were close (200px – 250px), medium (400px – 450px) and far (600px – 650px), with 2 items to select from each distance. For each technique the user completed 8 blocks of selections with each block having 2 selections for each of the 6 items allowing the user to gain experience across blocks.

For both of the selection techniques the CommandMap was displayed when the participant pressed Ctrl and was hidden when the Ctrl key was released (see Figure 7.1). For the CommandMap interface the selection of an item was done through the mouse. While with ExposeHK_CM the selection was done through completing the associated hotkey. Participants were instructed to make the command selections as quickly and accurately as possible.

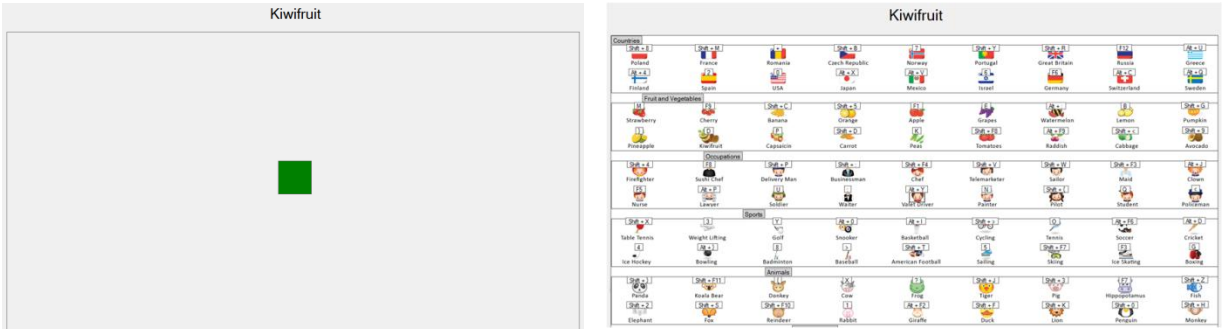

Figure 7.1: Experimental interface without (left) and with ExposeHK_CM activated (right).

Similar to the ExposeHK_R evaluation, the hotkeys for the ExposeHK_CM technique consisted on the Ctrl key plus a letter key. The letter key was chosen randomly form the 12 left most letter keys on the keyboard but like the ExposeHK_R evaluation was made sure it did not match the first of last letter of the related item.

The time taken, item selected, key presses and errors were recorded for each selection. Upon completing each interface participants completed a NASA-TLX and at the end of the experiment completed a questionnaire asking for their preference of technique for 3 questions.

7.3 Results

7.3.1 Selection Times

The mean selection times per block are summarized in Figure 7.2. Selections with errors were removed from the timing results. There was a significant main effect for *technique* ($F_{1,17} = 8.332$ $p < 0.05$) with mean selection times of 2539.41ms (sd 1752.72) and 2924.26ms (sd 1147.05) for ExposeHK_CM and CommandMap respectively. This showed that there is a significantly faster selection performance with ExposeHK_CM compared to the pointer based CommandMap.

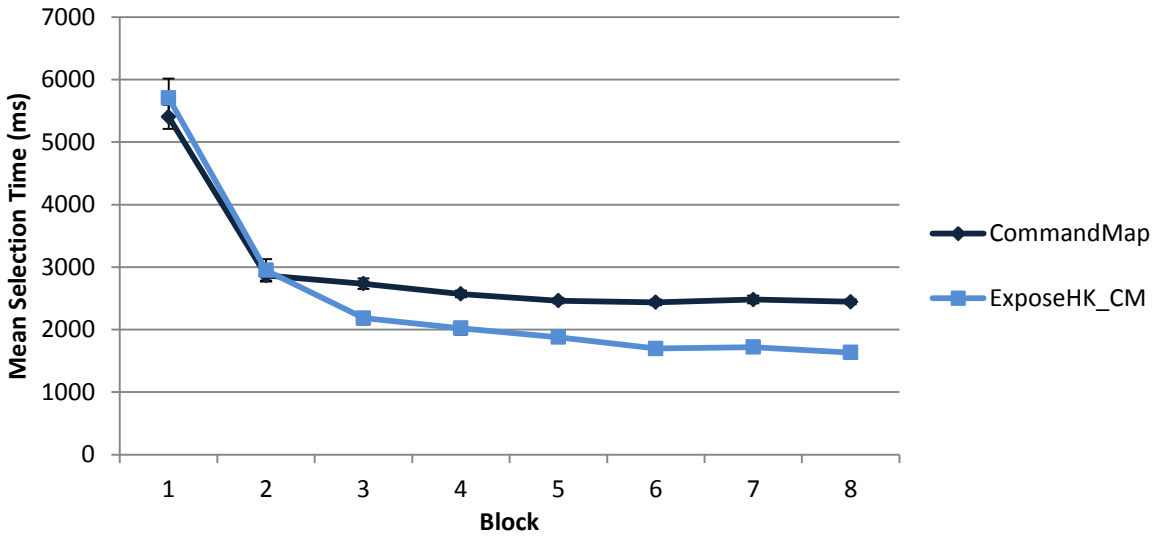

Figure 7.2: Mean selections time per block. Error bars show standard error.

Similar to the evaluation of ExposeHK_R, we are interested in the level of performance that is obtainable with each technique. Through analysing the final block data we found that there is a significant main effect for *technique* ($F_{1, 17} = 62.91$, $p < 0.001$) with mean selection times of 1674.09ms (sd 490.44) and 2443.46ms (sd 301.73) for ExposeHK_CM and CommandMap respectively.

There was a significant main effect for *block* ($F_{7, 119} = 206.48$, $p < 0.001$) as users were able to gain experience and become faster in their selections as they completed more blocks. There was also a significant main effect for *distance* ($F_{2, 34} = 4.66$, $p < 0.05$), showing that the distance to a target did have an effect on selection times.

There was a significant *technique x block* ($F_{7, 119} = 6.44$, $p < 0.001$) interaction. This interaction is illustrated in Figure 7.2 by the CommandMap being slightly faster than ExposeHK_CM for the first two blocks but at block 3 ExposeHK_CM has a significantly larger improvement in performance compared to the CommandMap.

There was a significant *technique x distance* ($F_{2, 34} = 4.80$, $p < 0.05$) interaction. This is explained by hotkey selections not effected by screen distance while for pointer based selection the user has to move the cursor further in order to make a selection.

There was no significant *block x distance* ($F_{14, 238} = 1.57$, $p = 0.09$) interaction, suggesting that the time difference between distances remained consistent over blocks. There was also no *technique x block x distance* ($F_{14, 238} = 1.256$, $p = 0.236$). This can be explained by ExposeHK_CM not being affected by distance in the hotkey learning process while with the CommandMap distance still has an effect no matter the experience.

The mean selection times at each distance is summarized in Figure 7.3. There was a significant *technique x distance* ($F_{2, 34} = 10.61$, $p < 0.001$) interaction, this can be explained by the ExposeHK_CM technique being unaffected by distance due to the keyboard selection. An interesting feature of the distance results is that the far distances mean selection time was fastest for ExposeHK_CM and second fastest for CommandMap. The far distance would be expected to be the worst performing distance for the pointer based CommandMap due to the extra movement for selection. However, this result can be explained by the far items being around the corners of the CommandMap which would have served as landmarks therefore helping the user learn spatial locations.

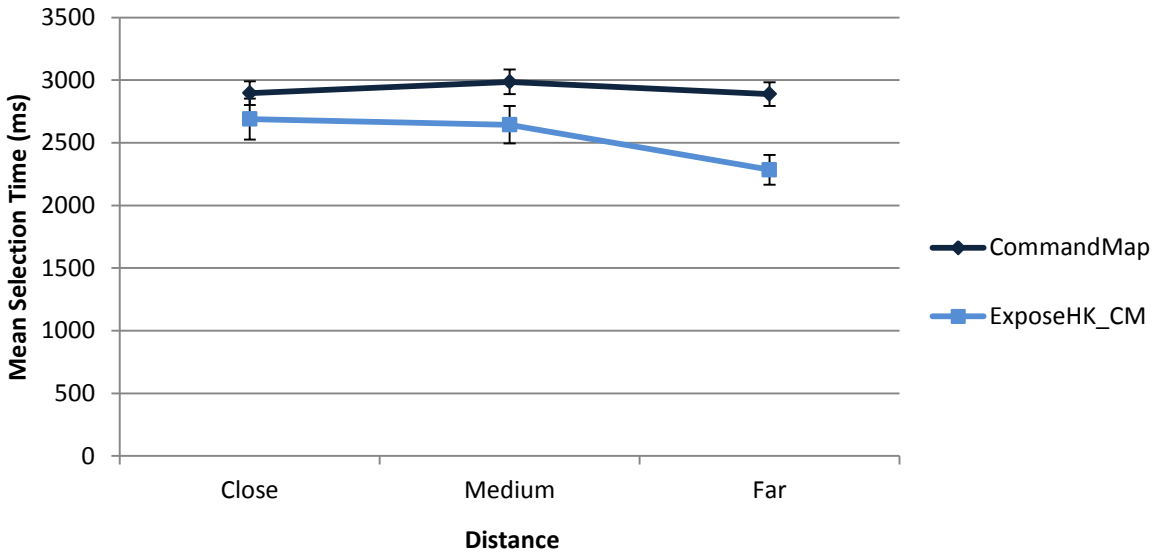

Figure 7.3: Mean selections time per distance. Error bars show standard error.

7.3.2 Errors

For the error rates there was a significant main effect for *technique* ($F_{1, 17} = 9.122$, $p < 0.01$), with mean error rates per selection of 0.12% (sd 0.98) and 0.96% (sd 3.06%) for CommandMap and ExposeHK_CM respectively. Even though there is a significant difference between the error rates ExposeHK_CM was still able to achieve very low mean error rate below 1% per selection. There was no main effect for *block* ($F_{7, 199} = 0.645$, $p = 0.718$), demonstrating that errors occurred randomly over the blocks.

7.3.3 Subjective Responses

NASA-TLX

The NASA-TLX results are summarised in Figure 7.4. There were significant difference in the Physical Demand and Performance rating between the CommandMap and ExposeHK_CM. There was no significant difference for any of the other categories.

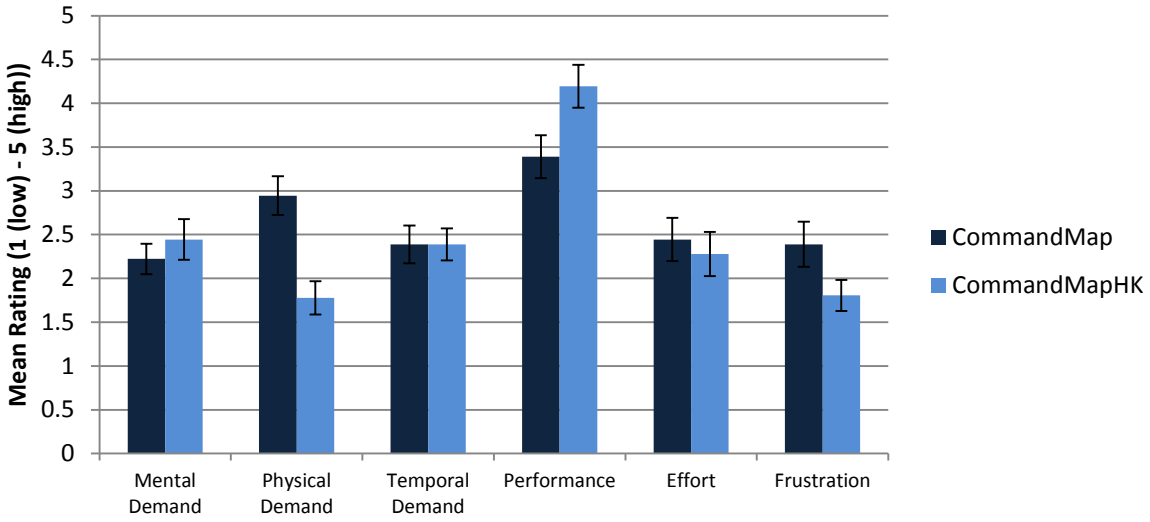

Figure 7.4: Mean NASA-TLX responses. Error bars show standard error. Lower scores are best except for performance.

Preference

Table 7.1: Preference Responses.

	CommandMap	ExposeHK_CM	$\chi 2$	p
Easier	4	14	4.5	< 0.05
Faster	1	17	12.5	< 0.0005
Overall Preference	1	17	12.5	< 0.0006

At the end of the experiment participants were asked which of the techniques they thought was easier, faster and which they preferred overall. The results of the questionnaire are summarised in Table 7.1. For all three questions users had a significant preference towards ExposeHK_CM.

7.4 Discussion

The main goal of this evaluation was to compare the performance of ExposeHK_CM to the pointer based CommandMap. Our evaluation showed that ExposeHK_CM was significantly faster than the CommandMap. Looking at the obtainable performance of each technique, through analysing the final selection block, ExposeHK_CM was 33% faster than the CommandMap.

The NASA-TLX results showed a significant preference towards ExposeHK_CM for both physical demand and performance. Our preference ranking responses showed a very strong preference towards ExposeHK_CM for being faster, easier and overall preferred. Overall this evaluation demonstrated that ExposeHK_CM is not only faster than the CommandMap but users also prefer ExposeHK_CM.

The issues of hotkey associations and participant hand placement that were mentioned in the discussion of the evaluation of ExposeHK_R remain consistent in this evaluation.

Why was ExposeHK_CM faster?

As discussed in the evaluation of ExposeHK_R, ExposeHK_CM was able to achieve fast command selection performance through allowing users easily and efficiently practice to learn hotkeys. This is in comparison to the pointer based CommandMap in which the user's ultimate performance is limited by their mouse movement speed. For the CommandMap we saw a similar effect to the Pointer selection in evaluation of ExposeHK_R by which the user's performance improvements slow down considerably at block 5 (see Figure 7.2)

The effect of distance demonstrated the difference between the pointer based CommandMap and keyboard based ExposeHK_CM. Although both techniques flatten the command selection hierarchy the CommandMap was affected by item distance. The speed difference between mouse movement and keyboard input is highlighted by the ExposeHK_CM mean time for far selections items being a 20% improvement on the CommandMap.

Deploying ExposeHK_CM

One issue of the CommandMap is that it requires a large screen to display. Scarr et al. highlighted this issue in their evaluation of the CommandMap [16]. From our evaluation we have shown that if a CommandMap can be implemented in an application then an ExposeHK_CM should also be considered as an alternative due to the subjective preferences and performance improvements demonstrated in this evaluation.

Many participants commented on the CommandMap being overwhelming with its instantaneous display of many commands. If ExposeHK_CM was to be implemented in an application it may have to have a delay for display to remove the ExposeHK_CM being displayed each time the user activated a hotkey. Therefore, ExposeHK_CM would only become visible when the user was uncertain of a hotkey combination.

8 Evaluation of ExposeHK in a Real World Application

From the previous two evaluations we have shown that ExposeHK is able to improve the selection performance of the current selection methods in the Ribbon as well as the CommandMap. However, in these studies the user was constrained to complete the selection with the chosen selection technique. Therefore, we are unable to determine if given the choice users would use ExposeHK.

In order to analyse the effect ExposeHK would have on user's command selection method a longitudinal study is necessary. From this study we would be able to determine if given an ExposeHK interface users would increase their hotkey usage and number of hotkeys learnt. Additionally, if given a new interface with the traditional hotkey feedback methods, would users attempt to learn hotkeys and at what rate do users learn and use hotkeys.

As longitudinal studies are a big commitment and take a lot of resources to complete, it is necessary to complete a pilot study to gather some initial results. Therefore, this evaluation is a pilot study focusing on the short term effect of ExposeHK. For this study we wanted to simulate a real world task which required the selection of multiple commands. We decided to implement a simple text editor with the task being text formatting as this requires multiple selections of many commands.

8.1 Design, Participants, and Apparatus

The design of the experiment is a within-subjects design with participants completing the task with both normal and ExposeHK interfaces. 12 participants (ages 21 – 30, 2 female) completed the experiment. Each participant completed the task with the no ExposeHK interface first followed by the ExposeHK interface and finally the no ExposeHK interface again. The experimental system was developed in C# and was run on Windows 7 computer along with an external QWERTY keyboard and optical mouse.

8.2 Procedure

The experiment involved using a simple text editor to format text to a desired format (see Figure 8.1). The whole text editing interface was set to a fixed size of 1310px x 700px. Participants first completed the text formatting task with a simple text editor. The task did not involve editing the text by adding or deleting text. The text editor had hotkeys for each command and the hotkeys were consistent with those used in Microsoft Word (e.g. Bold was Ctrl + B, Italics was Ctrl + I). This allowed participants to use their current knowledge hotkeys for commands and would not confuse the participants when learning new hotkeys. The hotkeys were able to be discovered through tooltips which would appear when the user hovered the mouse over a command icon. The tooltips had a 500ms delay before appearing and would display the command name and associated hotkey key combination.

Once users completed the text formatting task with the normal text editor they completed the exact same task with a text editor which had ExposeHK. Before completing the text formatting task with ExposeHK users completed a practice session with ExposeHK. The practice session consisted of a very simple text formatting task to allow the user to understand how ExposeHK worked. When the user pressed and held Ctrl the hotkeys key combination would appear above each of the commands in the toolbar (see Figure 8.2). Participants were told that ExposeHK was a new method for displaying hotkeys and that they were still able to complete command selections with the pointer.

After completing the task with the ExposeHK text editor interface they then completed the same task with a text editor without ExposeHK. This text editor was the exact same interface used for the first text formatting task.

Upon completing the text formatting task with a particular interface, participants completed a NASA TLX. After completing all of the text formatting tasks participants completed a questionnaire asking the participants to rank the interfaces. For each interface key presses, mouse movements, command selections, tooltip displays were logged to a log file. Participants were told not to try and go as fast as possible but to just complete the task as they normally would with a text editor.

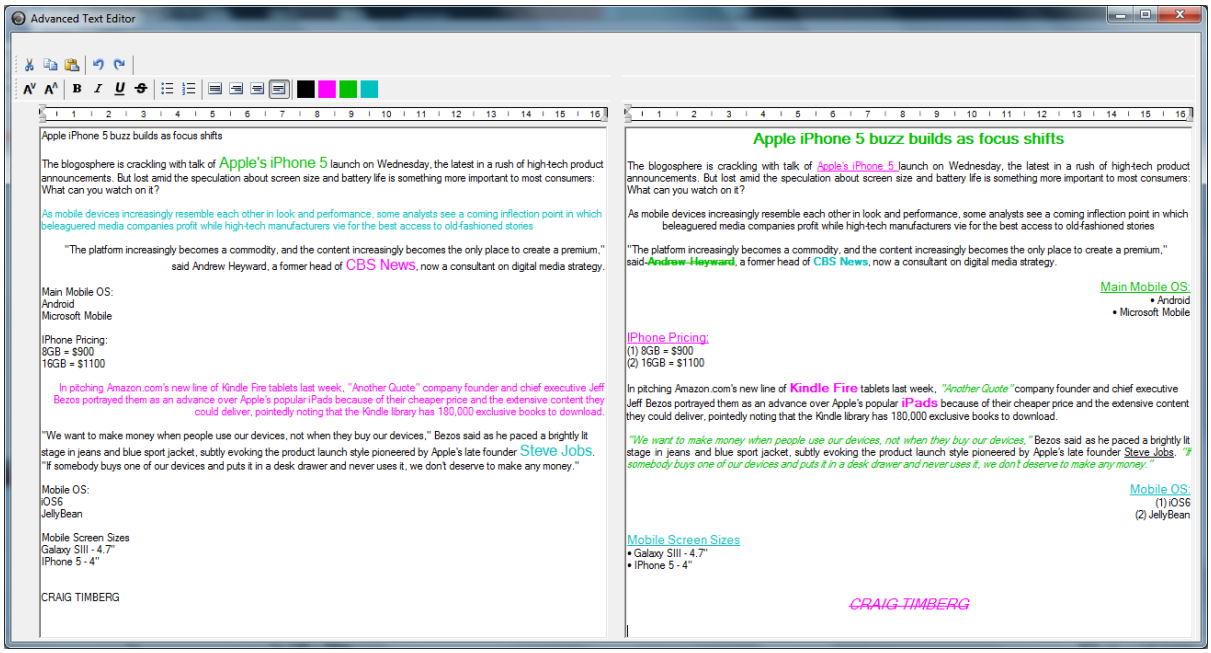

Figure 8.1: The Experimental Interface with a text editor on the left hand side consisting of a text area and a toolbar of commands. The desired final format is displayed in the right hand side of the text editor.

Figure 8.2: Toolbar with ExposeHK activated

8.3 Results

8.3.1 Hotkey Use

Interfaces

There was a large difference in the hotkey usage for each interface with mean hotkey usage of 9.79% (s.d. 19.67), 86.20% (s.d. 12.87) and 73.09% (s.d. 21.15) for interface 1, ExposeHK and interface 3 respectively. However, we are unable to determine how much the increase in hotkey usage from interface 1 to interface 3 was due to ExposeHK and how much is due to learning over time. To better understand the hotkey usage over time we split each interface task into 10 blocks over time (see Figure 8.3). This shows that the hotkeys usage for interface 1 does not increase drastically overtime suggesting that users are not trying to learn new hotkeys but

are just continuing with the ones that they currently know. For the ExposeHK interface the graph demonstrates that the hotkey usage is instantly very high and is consistent through the task.

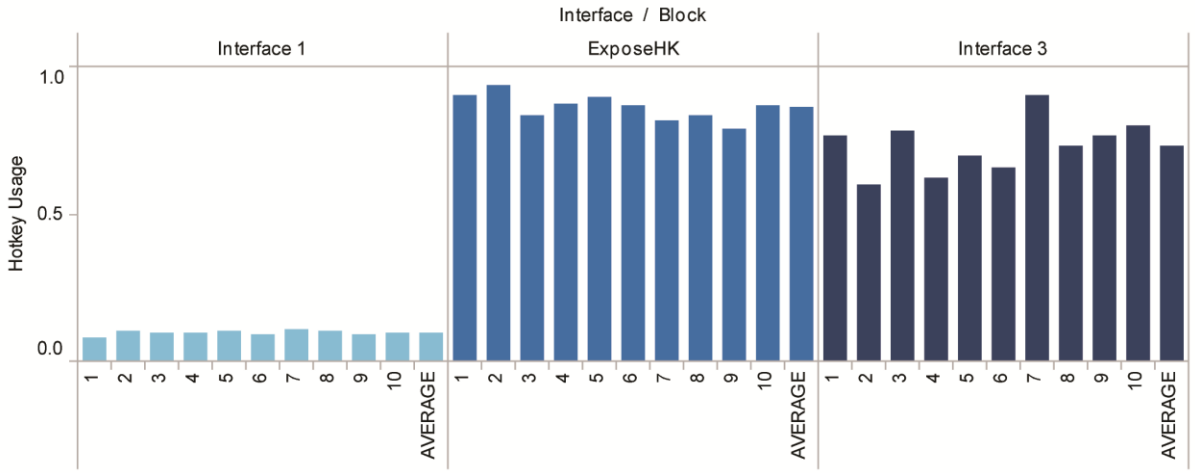

Figure 8.3: Mean hotkey usage per interface split into 10 blocks

Commands

The hotkey usage for each command is summarized in Figure 8.4. A more detailed view of the command hotkey usage for each participant is summarized in Figure 8.5. For these results commands which were not necessary in the formatting task but were still selected a very small amount such as undo and redo were excluded.

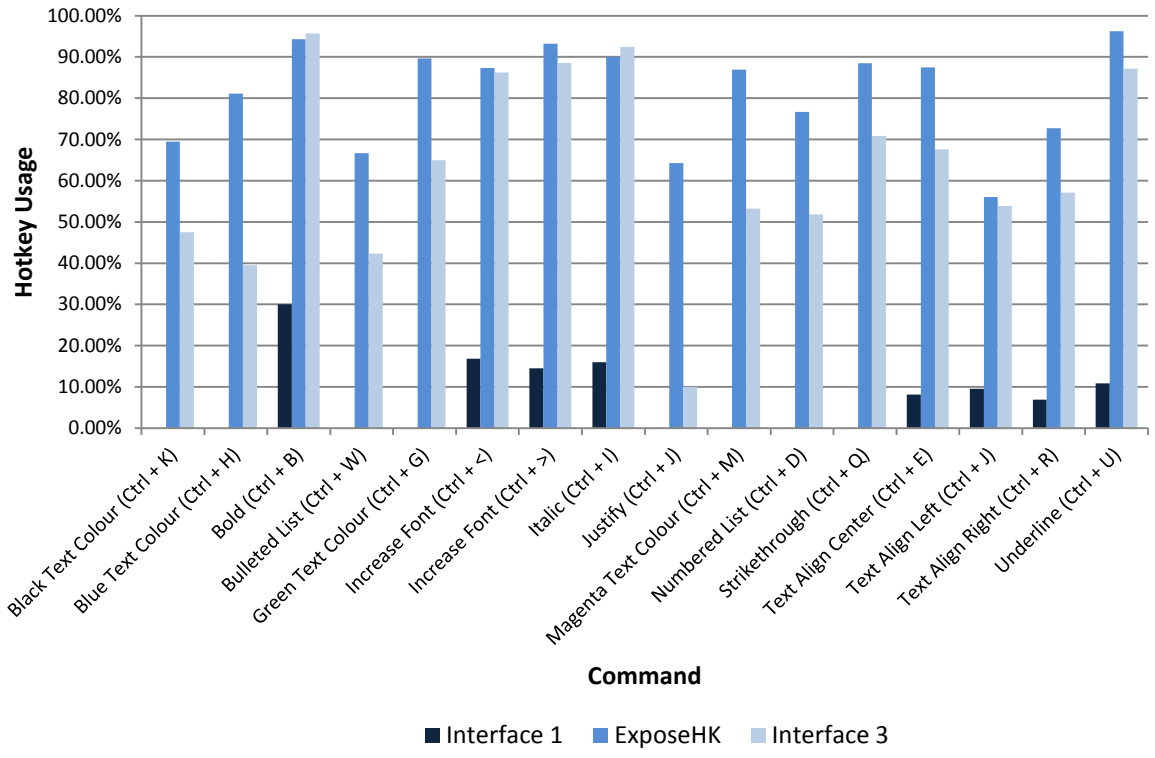

Figure 8.4: Hotkey usage per command

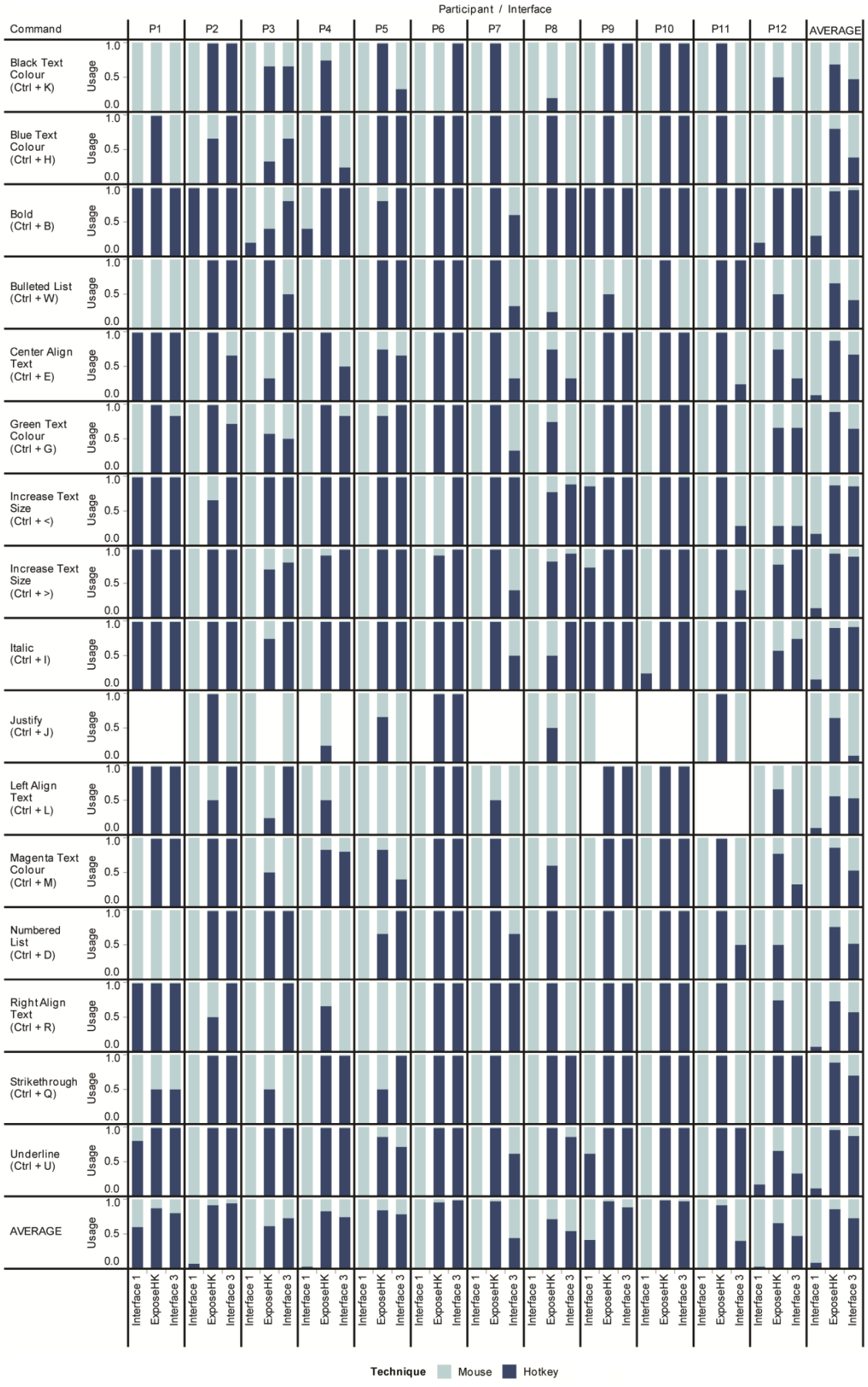

Figure 8.5: Command hotkey usage for each participant

All of the hotkeys used for interface 1 are hotkeys which exist in text editors such as Microsoft Word. Commands that do not have hotkeys in Microsoft Word such as bulleted list, strikethrough and the text colours have no hotkey usage for interface 1. This suggests that users are not trying to learn new hotkeys but are just using hotkeys for commands which they already know. For the ExposeHK interface there is high hotkey usage for most commands with the lowest hotkey usage being 56%. For the final interface there are mixed hotkey usage results. For commands which are existent in Microsoft Word with easy associations such as Bold (Ctrl + B) and Italics (Ctrl + I) there was high hotkey usage. Whereas, commands which are not existent in Microsoft Word and do not have an easy association such as text colour blue (Ctrl +H) there is a large drop of in hotkey usage from the ExposeHK interface to interface 3.

Figure 8.5 demonstrates the large variation in the hotkey usage for each participant. In particular for interface 1 the hotkey usage for most of the participants is close to zero with only participant 1 and 9 having substantial hotkey usage. For the ExposeHK interface and interface 3 participants 6, 9 and 10 have extremely high hotkey usage close to 100%.

8.3.2 ToolTip Exposures

For Interface 1 the total number of tooltip exposures across all participants was 749 (mean 62.42, s.d. 14.74) with only 2 hotkey combinations being activated after the tooltip exposure. This demonstrates that although the tooltips were exposed a great deal the user took little notice with only twice activating the hotkey directly after an exposure. This further suggests that for interface 1 hotkeys were not being learned but users were using their prior knowledge for hotkey command selections. For interface 3 there were a total of 210 tooltip exposures (mean 17.5, s.d. 11.37) with 15 hotkeys being activated after an exposure. The drop in tooltip exposures can be explained by the increase in hotkey usage.

8.3.3 Subjective Responses

NASA-TLX

The NASA-TLX results are summarized in Figure 8.6. There was no significant difference between the interfaces for any of the NASA-TLX categories.

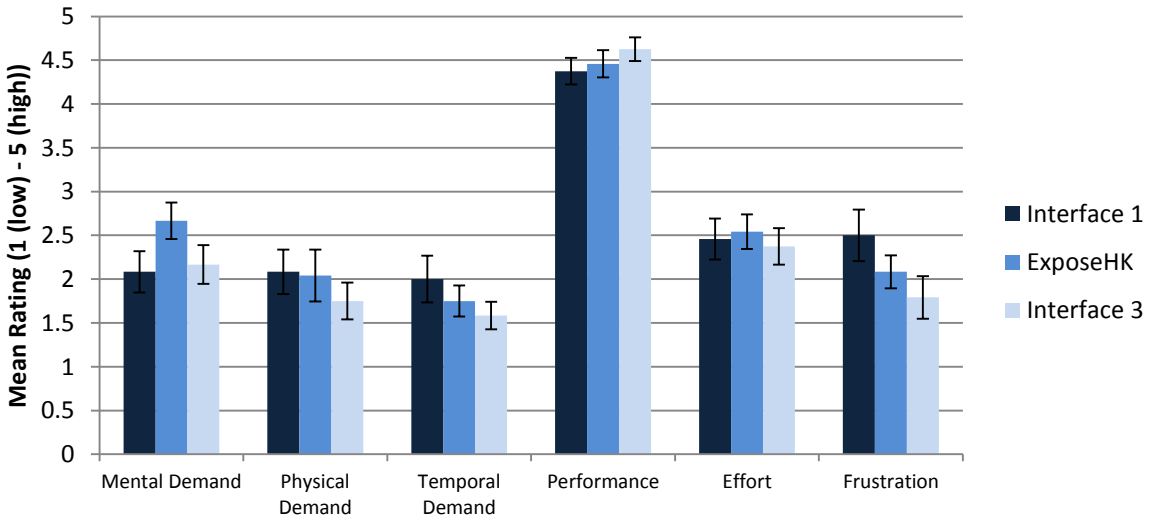

Figure 8.6: Mean NASA-TLX responses. Error bars show standard error. Lower scores are best except for performance.

Preference

At the end of the experiment participants completed a questionnaire to rank the interfaces (1^{st} to 3^{rd}) for which interface they felt was easier, faster and preferred overall. The results are summarized in Figure 8.7. Freidman tests showed a significant difference for the faster and overall preferred responses. The ExposeHK interface and interface 3 had very similar rankings for all three questions with interface 1 having the worst ranking for all 3 questions.

Participants also completed a questionnaire regarding hotkey usage. The participants were asked to rank the interfaces for 3 questions; which interface did they feel they used the most hotkeys, learned the most hotkeys and which they preferred for learning hotkeys. The results are summarized in Figure 8.8. Freidman tests showed a significant difference for all three questions. For all three questions the ExposeHK interface had by far the best results, only being ranked 2nd twice. Interface 3 had the next best ranking for all questions with interface 1 having far worse results.

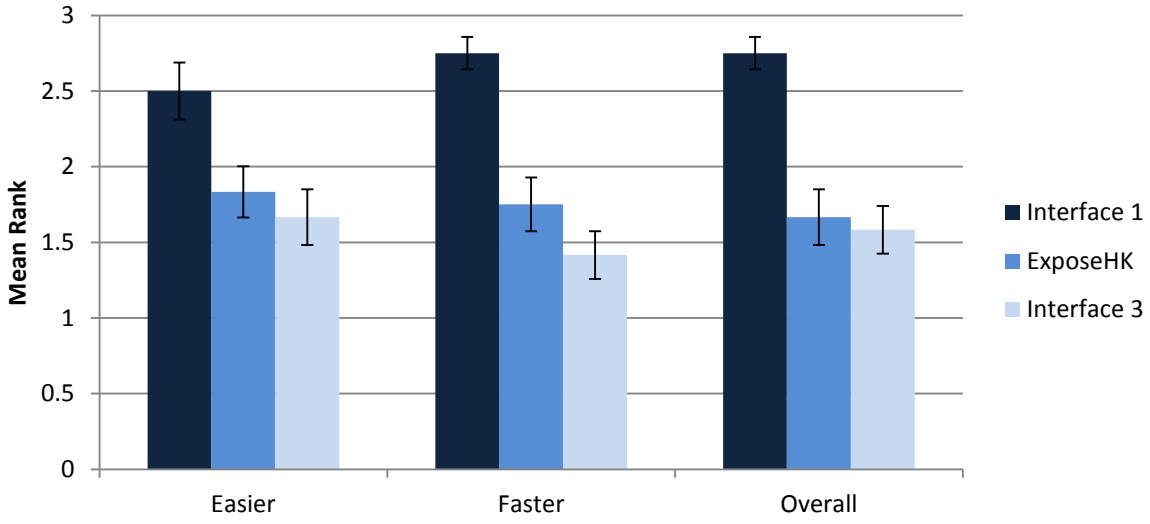

Figure 8.7: Mean preference ranking responses. Error bars show standard error. Lower scores are better.

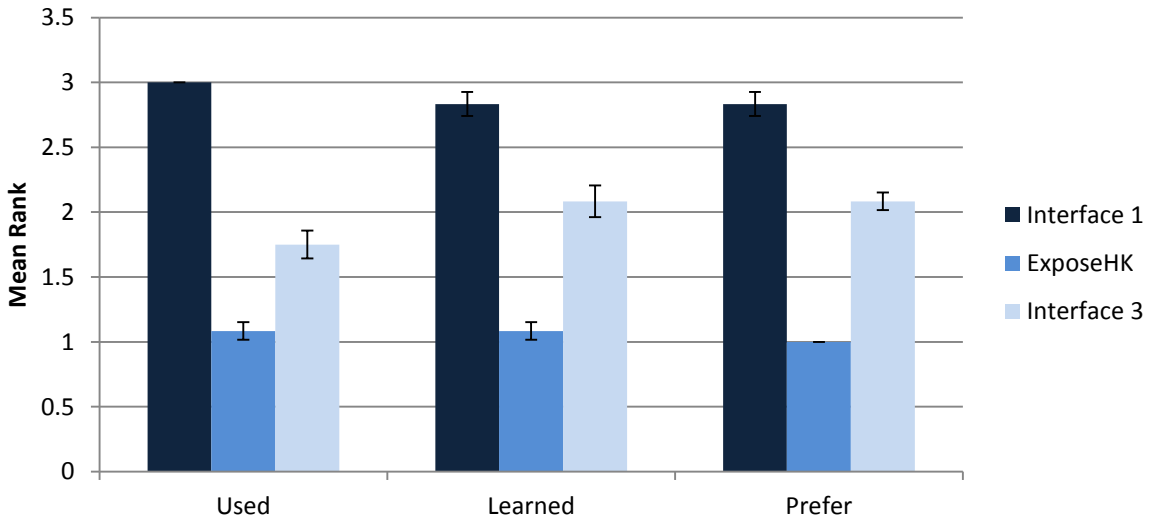

Figure 8.8: Mean hotkey preference ranking responses. Error bars show standard error. Lower scores are better.

8.4 Discussion

This evaluation was designed as a pilot study to look at hotkey usage short term with and without ExposeHK. We were able to show that instantly with the ExposeHK interface users had an extremely high level of hotkey usage. Furthermore, our subjective results showed that users preferred ExposeHK for learning hotkeys.

Future Evaluation

To accurately look at the effect ExposeHK has on hotkey learning and usage a more comprehensive evaluation is needed. One possible evaluation could be a between subjects experiment with half of the participants using an ExposeHK interface. It will also be necessary to allow users the opportunity to learn hotkeys over time so would

involve the users completing a text editing and formatting task each day over a period of time. We would then be able to compare the hotkey usage for each interface as well as the usage over time. The hotkey learnability of each interface could be compared through a pre and post hotkey knowledge test.

With the text formatting task we tried to simulate a real world task of editing a document once it had been written. However, this may not represent the real way in which users create documents. In order to analyse ExposeHK for a text editing applications we would need to allow the user to create and format the document in their normal procedure. This would involve the user starting with a blank text document and requiring to write and format a document to a specific target.

An alternative to a between subjects evaluation would be to look at the normal usage patterns of users with ExposeHK through implementing an add-in for a Microsoft Office application such as Excel or Word. Once implemented the ExposeHK add-in would be available to anyone to install. The add-in would log interactions which would allow us to analyse the real world usage patterns of users with ExposeHK.

Tooltips for Hotkey Display

The poor hotkey usage results in interface 1 of just below 10% suggest that tooltips are not a good method of exposing the existence of hotkeys. The results demonstrated in Figure 8.5 of the hotkey usage for each participant further demonstrates this with only 2 participants having a sizable hotkey usage.

The high tooltip exposure count with extremely low hotkey activation of interface 1 demonstrates that they were ignored by users even through many of these exposures would have been accidental. Through users ignoring the tooltips it results in new hotkeys never being learnt. The hotkey usage for the commands in interface 1 further supports this idea of no hotkeys being learnt in interface 1 due to the only hotkeys being activated being hotkeys that are existent in Microsoft Word.

One possible reason for tooltips always being ignored is that tooltips are not primarily used for displaying hotkeys. Tooltips are often used to just provide the user with further information about a particular item. Therefore, since tooltips do not always mean a hotkey shortcut is available, users will just ignore the tooltip if they are not interested in finding out more information about an item. This supports the idea of ExposeHK as it is a method that is dedicated solely to displaying hotkeys.

9 Discussion and Conclusion

9.1 Discussion

In this paper we have presented ExposeHK as a new method for allowing the user to seamlessly transition to the expert performance of hotkeys. ExposeHK quickly reveals to the user the existence of hotkeys and allows the user to browse, perform, and learn hotkeys within the hotkey modality.

Our evaluations of ExposeHK revealed some key results:

- Both ExposeHK_R and ExposeHK_CM allows users to obtain a significantly higher level of command selection performance;
- Users had a strong preference towards both ExposeHK_R and ExposeHK_CM techniques for being easier, faster and preferred overall;
- With the addition of ExposeHK to an interface, hotkey usage was immediately very high.

Why was ExposeHK able to improve command selection performance?

In our first two evaluations we demonstrated that the ExposeHK technique allowed users to obtain a significantly higher level of command selection performance. The ExposeHK technique was able to achieve the fast command selection performance through utilizing hotkeys which take advantage of fast keyboard interaction, flat selection hierarchy, and removing any context switch in input device. Furthermore, hotkeys allow the user to maintain focus on their current work as users do not have to navigate through hierarchy interfaces. Through maintaining focus the user is also able to remove any reduction in performance or increase in errors that result from task switching.

ExposeHK allowed users to become efficient in hotkeys through providing a fast and simple method of display to encourage novice users to practice to learn hotkeys. By utilising the keyboard, a fast mechanism of interaction, it allows the user to quickly activate ExposeHK while also being within the hotkey modality. Overlaying hotkeys on the current interface allowed intermediate users the ability to quickly find hotkeys through using their current interface knowledge. The combination of an ultimately quick selection method with a smooth transition from novice to expert allows ExposeHK at achieve its high level of command selection performance.

Why was ExposeHK able to achieve a high level of hotkey usage?

Our third evaluation of ExposeHK focused on determining if ExposeHK would alter the user's method of command selection. The results of the evaluation showed that instantly with the ExposeHK interface users' hotkey usage was extremely high. ExposeHK was able to achieve a high level of hotkey usage through a simple activation and display method which quickly reveals to the user to existence of hotkeys in an interface.

The third evaluation also highlighted that they current method of hotkey display through tooltips is ineffective. The high tooltip exposure and low hotkey usage of interface 1 strongly suggested that users were ignoring the hotkey feedback and therefore not learning new hotkeys. Therefore, with users not actively learning hotkeys they will most likely be stuck in in a method of selection with a low ceiling of performance.

Hotkey Associations

For the first two experiments we used key combinations which had no obvious association to the related command. The participant's comments in the first two evaluations further demonstrated this with many saying

that hotkeys would be easier to recall with easy associations. Through easier association we would expect that the hotkey learning stages performance would increase and result in faster initial selection performance for ExposeHK. The third study has demonstrated this idea of hotkeys have no obvious association being harder to learn. An example of this was the command usage for interface 3 for which blue text colour (Ctrl + H, no obvious association) command had a hotkey usage of 39% compared to the green text command (Ctrl + G, easy association) which had hotkey usage of 65%.

ExposeHK for New Interface Training

A possible use of ExposeHK could be as a training interface for learning and expressing hotkeys in a new interface or application. This would allow users to become aware of the existence of hotkeys in addition to allowing users to learn hotkeys in an efficient manner. This is important issue as hotkeys are often applicable and consistent across multiple applications (e.g. Ctrl + S is Save). Therefore, users may have a wide knowledge of hotkey combinations but without the awareness that they are applicable they may never be attempted. The issue of users not identifying the existence of hotkeys was highlighted in evaluation three in which only two participants were using hotkeys semi-regularly for the first interface.

Deploying ExposeHK

The strong preference scores for ExposeHK in the first two evaluations along with the high hotkey usage for the ExposeHK interface in the third evaluation strongly suggest that if given the chance users would use the ExposeHK technique regularly. Therefore, the way in which ExposeHK can easily be added and implemented in interfaces is a key issue.

Future work for ExposeHK would be to allow developers to easily implement ExposeHK in their applications and interfaces. Similar to the way in which tooltips are associated with GUI components, ExposeHK key tips should be able to be added to any GUI component. This would allow developers to easily attach key tips to the GUI components which have associated hotkeys. Through implementing ExposeHK in many applications user would become expectant of ExposeHK in the same way users expect there to be tooltips for commands.

Hotkey Command Coverage

One shortfall of ExposeHK as a command selection method is that not all commands have an associated hotkey. Therefore, when a user activated ExposeHK only to find there is not hotkey they will then be forced to switch to an alternative method of selection. For ExposeHK to be effective in an interface there needs to be a high number of hotkey available to the user. Apart from hard coding in hotkeys, one alternative is to allow the user to specify hotkeys or automatically generate hotkey for commands which are used regularly. This would allow the user to use ExposeHK for most of their command selections. There will also be the poor hotkey command coverage in applications like Microsoft Word where there are a very large number of commands available. To allow for high hotkey coverage for many commands it may be necessary to use multiple modifiers and non-modifiers for hotkey key combinations.

Future Studies

As mentioned in the discussion in the third evaluation of ExposeHK further evaluations are needed in order to determine the true ability of ExposeHK for learning and encouraging hotkey usage. Further studies could also be conducted into the aspects of learning hotkeys. In particular we would be interested in determining the upper limit of the number of hotkeys that can be learnt as well the speed in which multiple hotkeys can be learnt. These studies could also look into the mental demand of issuing hotkeys as users transition from the hotkey learning stages to the expert muscle memory recalling of hotkeys. The mental demand of learning hotkeys was demonstrated in the evaluation of ExposeHK_R with the mental demand of ExposeHK_R being higher than pointer selection.

9.2 Conclusion

In this paper we have presented ExposeHK as a new interface technique which allows the user to browse, perform and learn hotkeys from within the hotkey modality. Hotkeys have been shown to improve user command selection performance through proving a flat selection hierarchy and fast activation through keyboard

input. ExposeHK allows users to become efficient in hotkeys through providing a fast and simple method of hotkey display which encourages novice users to learn hotkeys through practice.

We evaluated two ExposeHK implementations designed for the Ribbon and CommandMap interfaces; ExposeHK_R and ExposeHK_CM. The evaluation of ExposeHK_R showed that users were able to obtain command selection performance improvements of 28% and 44% over to the Pointer and Alt-keys selection methods respectively. Our evaluation of ExposeHK_CM showed that users were able to obtain command selection performances which were 33% greater than Pointer based CommandMap. In both of these evaluations there was also strong user preference towards both the ExposeHK interfaces.

Our third evaluation of ExposeHK focused on determining the effect ExposeHK would have on users command selection technique in a real world application. The results of the evaluation showed that with the ExposeHK interface users' hotkey usage is extremely high. Furthermore, the strong user preference towards the ExposeHK technique across our evaluations suggests that users would use ExposeHK if implemented in an application. Our evaluations have shown that ExposeHK not only leads to faster command selection performance but is also well liked.

Bibliography

[1] J. Nielsen, "Finding usability problems through heuristic evaluation," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1992, pp. 373-380.

[2] S. K. Card, T. P. Moran, and A. Newell, "The keystroke-level model for user performance time with interactive systems," *Communications of the ACM*, vol. 23, pp. 396-410, 1980.

[3] D. L. Odell, R. C. Davis, A. Smith, and P. K. Wright, "Toolglasses, marking menus, and hotkeys: a comparison of one and two-handed command selection techniques," presented at the Proceedings of Graphics Interface 2004, London, Ontario, Canada, 2004.

[4] D. M. Lane, H. A. Napier, S. C. Peres, and A. Sandor, "Hidden Costs of Graphical User Interfaces: Failure to Make the Transition from Menus and Icon Toolbars to Keyboard Shortcuts," *International Journal of Human-Computer Interaction*, vol. 18, pp. 133-144, 2005/02/01 2005.

[5] T. Grossman, P. Dragicevic, and R. Balakrishnan, "Strategies for accelerating on-line learning of hotkeys," presented at the Proceedings of the SIGCHI conference on Human factors in computing systems, San Jose, California, USA, 2007.

[6] Zipf, "Review of "Human behavior and the principle of least effort"," *Journal of Consulting Psychology*, vol. 13, pp. 224-224, 1949.

[7] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman, "An empirical comparison of pie vs. linear menus," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1988, pp. 95-100.

[8] G. Kurtenbach, G. W. Fitzmaurice, R. N. Owen, and T. Baudel, "The Hotbox: efficient access to a large number of menu-items," in *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, 1999, pp. 231-237.

[9] A. Sears and B. Shneiderman, "Split menus: effectively using selection frequency to organize menus," *ACM Trans. Comput.-Hum. Interact.*, vol. 1, pp. 27-51, 1994.

[10] A. Cockburn, C. Gutwin, and S. Greenberg, "A predictive model of menu performance," presented at the Proceedings of the SIGCHI conference on Human factors in computing systems, San Jose, California, USA, 2007.

[11] D.-S. Lee and W. C. Yoon, "Quantitative results assessing design issues of selection-supportive menus," *International Journal of Industrial Ergonomics*, vol. 33, pp. 41-52, 2004.

[12] L. Findlater and J. McGrenere, "A comparison of static, adaptive, and adaptable menus," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2004, pp. 89-96.

[13] G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. v. Dantzich, "Data mountain: using spatial memory for document management," presented at the Proceedings of the 11th annual ACM symposium on User interface software and technology, San Francisco, California, United States, 1998.

[14] M. P. Czerwinski, M. Van Dantzich, G. Robertson, and H. Hoffman, "The contribution of thumbnail image, mouse-over text and spatial location memory to web page retrieval in 3D," in *Proceedings of INTERACT'99 - Human Computer Interaction, 30 Aug.-3 Sept. 1999*, Amsterdam, Netherlands, 1999, pp. 163-70.

[15] W. E. Hick, "On the rate of gain of information," *Quarterly Journal of Experimental Psychology*, vol. 4, pp. 11-26, 1952/03/01 1952.

[16] J. Scarr, A. Cockburn, C. Gutwin, and A. Bunt, "Improving command selection with CommandMaps," presented at the Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems, Austin, Texas, USA, 2012.

[17] C. Gutwin and A. Cockburn, "Improving list revisitation with ListMaps," presented at the Proceedings of the working conference on Advanced visual interfaces, Venezia, Italy, 2006.

[18] G. Kurtenbach and W. Buxton, "User learning and performance with marking menus," in *Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence*, 1994, pp. 258-264.

[19] G. Kurtenbach and W. Buxton, "The limits of expert performance using hierarchic marking menus," in *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, 1993, pp. 482-487.

[20] G. Bailly, E. Lecolinet, and L. Nigay, "Flower menus: a new type of marking menu with large menu breadth, within groups and efficient expert mode memorization," in *Proceedings of the working conference on Advanced visual interfaces*, 2008, pp. 15-22.

[21] D. L. Odell, R. C. Davis, A. Smith, and P. K. Wright, "Toolglasses, marking menus, and hotkeys: a comparison of one and two-handed command selection techniques," in *Proceedings of Graphics Interface 2004*, 2004, pp. 17-24.

[22] H. A. Simon, "Theories of decision-making in economics and behavioral science," *The American Economic Review*, vol. 49, pp. 253-283, 1959.

[23] J. M. Carroll and M. B. Rosson, *Paradox of the active user*: The MIT Press, 1987.

[24] J. Scarr, A. Cockburn, C. Gutwin, and P. Quinn, "Dips and ceilings: understanding and supporting transitions to expertise in user interfaces," presented at the Proceedings of the 2011 annual conference on Human factors in computing systems, Vancouver, BC, Canada, 2011.

[25] M. Kumar, "Reducing the Cost of Eye Tracking Systems," *Technical Report CSTR* vol. 2006-08, April 2006.

[26] C. Ziegler. (2011). *Tobii and Lenovo show off prototype eye-controlled laptop, we go eyes-on*. Available: http://www.engadget.com/2011/03/01/tobii-and-lenovo-show-off-prototype-eye-controlled-laptop-we-go/

[27] "Eye tracking -- A new interface for visual exploration," *BT Technology Journal*, vol. 24, pp. 57-57, 2006.

[28] M. Kumar, A. Paepcke, and T. Winograd, "EyePoint: practical pointing and selection using gaze and keyboard," presented at the Proceedings of the SIGCHI conference on Human factors in computing systems, San Jose, California, USA, 2007.

[29] S. Zhai, C. Morimoto, and S. Ihde, "Manual and gaze input cascaded (MAGIC) pointing," in *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, 1999, pp. 246-253.

[30] K. J. Räihä and O. Špakov, "Disambiguating ninja cursors with eye gaze," in *Proceedings of the 27th international conference on Human factors in computing systems*, 2009, pp. 1411-1414.

[31] M. Kobayashi and T. Igarashi, "Ninja cursors: using multiple cursors to assist target acquisition on large screens," in *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, 2008, pp. 949-958.

[32] P. M. Fitts, "The information capacity of the human motor system in controlling the amplitude of movement," *J Exp Psychol*, vol. 47, pp. 381-91, 1954.

[33] T. Technology, "An introduction to eye tracking and Tobii Eye Trackers," *Tobii Eye Tracking*, January 27, 2010 2010.

[34] A. L. Yarbus, B. Haigh, and L. A. Rigss, *Eye movements and vision* vol. 2: Plenum press New York, 1967.

[35] S. Stellmach and R. Dachselt, "Look & touch: Gaze-supported target acquisition," in *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, 2012, pp. 2981-2990.

[36] S. Monsell, "Task switching," *Trends in cognitive sciences*, vol. 7, pp. 134-140, 2003.

[37] R. E. Smith, "The cost of remembering to remember in event-based prospective memory: investigating the capacity demands of delayed intention performance," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 29, p. 347, 2003.

[38] A. L. Cohen and P. M. GOLLYVITZER, "The Cost of Remembering to Remember Cognitive Load and Implementation Intentions Influence Ongpfng Task Performance," 2008.

[39] #233, r. Casiez, N. Roussel, and D. Vogel, "1 € filter: a simple speed-based low-pass filter for noisy input in interactive systems," presented at the Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems, Austin, Texas, USA, 2012.

[40] S. Ghani and N. Elmqvist, "Improving revisitation in graphs through static spatial features," presented at the Proceedings of Graphics Interface 2011, St. John's, Newfoundland, Canada, 2011.

[41] S. Leifert, "The influence of grids on spatial and content memory," presented at the Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems, Vancouver, BC, Canada, 2011.