4-2022

UX/U-Eye: Designing Graphical User Interfaces for Exclusive Eye Gaze Control

Timothy Curol

UX/U-Eye: Designing Graphical User Interfaces for Exclusive Eye Gaze Control

by

Timothy Curol

Undergraduate honors thesis under the direction of

Dr. Shuangqing Wei

Department of Electrical and Computer Engineering

Submitted to the LSU Roger Hadfield Ogden Honors College in partial fulfillment of the Upper Division Honors Program.

April 2022

Louisiana State University
& Agricultural and Mechanical College
Baton Rouge, Louisiana

Acknowledgments

This work is an extension of the coursework done for my engineering Senior Design project, which is being done alongside Jack Clement, Ewan Robertson, and Emily Vu. I worked alongside Emily specifically to achieve functionality in our software Graphical User Interface design. This thesis work improves the usability of the software beyond the scope of capstone design by more closely studying eye gaze control to make modifications in the elements presented on-screen. I would like to thank Jack, Ewan, and Emily for their incredible teamwork throughout the project, and for their continued support of my thesis work.

Thank you to Elissa McKenzie, St. Lillian Academy, and William's family for their insight and assistance. Thank you to my panel, Dr. Shuangqing Wei, Dr. Ramachandran Vaidyanathan, and Dr. William Ma for their support, guidance, and insight. Finally, thank you to my family and friends for being with me throughout this process.

Table of Contents

Chapter 1 Introduction **1**
1.1 Assistive Painting Device 1
1.2 Modern Eye Tracking Technology 3
1.3 Outline 4
Chapter 2 Using Eye Gaze Control **5**
2.1 Characteristics of Human Eye Gaze 5
2.2 Selection Interaction While Using Eye Gaze Input 7
2.3 Problems Unique to Eye Gaze Tracking Input 8
2.3.1 Failure to Detect Eyes 8
2.3.2 Discriminating User Input 8
2.3.3 Directing User's Fixation 10
2.3.4 The "Midas Touch" Problem 10
2.3.5 Visible Cursor 10
2.3.6 Drawing Attention Away from Important Region 11
Chapter 3 Design Solutions **12**
3.1 Addressing Problems Unique to Eye Gaze Tracking Input 12
3.1.1 Failure to Detect Eyes 12
3.1.2 Discriminating User Input 13
3.1.3 Directing User's Fixation 14
3.1.4 The "Midas Touch" Problem 16
3.1.5 Visible Cursor 17
3.1.6 Drawing Attention Away from Important Region 18
Chapter 4 Conclusions and Future Work **20**
4.1 Conclusions 20
4.2 Future Work 20
References **22**
Vita **23**

Abstract

To make the act of painting accessible to a user with a physical disability, an assistive painting device was designed with accompanying software to run on an assistive computer. Because this computer is controlled with eye gaze input, unique control problems were encountered for the user interface and were addressed throughout the software design. Preliminary testing indicates that the design changes aided the delivery of user-friendly software.

Chapter 1

Introduction

1.1 Assistive Painting Device

In 2021-2022, LSU's engineering capstone design Team 12 was tasked with developing an assistive painting device which is accessible to a user with cerebral palsy (CP). CP symptoms are different for every person, but the intended user of the device is non-verbal (anarthria) and has no voluntary control of their hands and feet. To communicate throughout the day, the customer uses software on a Tobii Dynavox computer controlled through eye gaze and head switches as their augmentative and alternative communication (AAC) device [1].

The team was excited at all the possibilities of using eye tracking technology to map a user's gaze as input for a painting machine, imagining a nearly magic machine operated with perfect accuracy. However, observing the use of a Tobii Dynavox to communicate showed the team that eye tracking was not perfectly reliable in controlling on-screen software. The device was accurate enough that a user was able to eventually select the on-screen actions they desired, but many buttons took multiple attempts because they failed to select or an unintended button was mistakenly selected. In addition, the eye tracking input occasionally tended toward certain spots on the screen or lagged behind. Because the device is used to communicate, repeated inaccuracies are very disruptive and aggravating for the user.

To make the product more adaptive to the current limitations of eye tracking technology, the team's design splits the control and implementation, with a Graphical User Interface (GUI) which runs on a user's assistive computer as the user's direct control and a separate painting device which follows the commands given by the software. This design gives the user the ability to undo digital actions before they are performed with real paint, which avoids permanent effects of

incorrect eye gaze selections. This Thesis work improves the usability of the GUI beyond the scope of capstone design by more closely studying eye gaze control to make modifications in the elements presented on-screen.

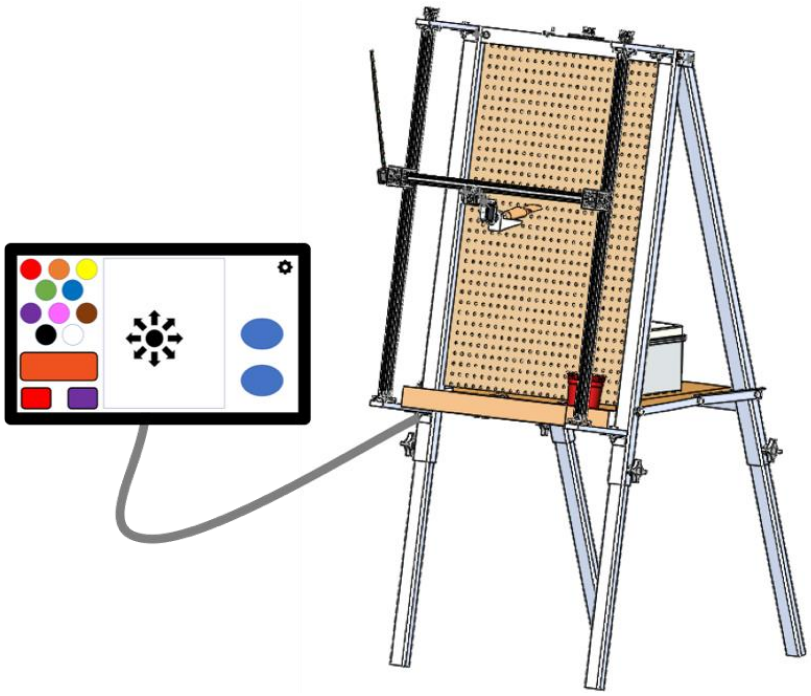

Figure 1.1. *Team 12's Project Design*

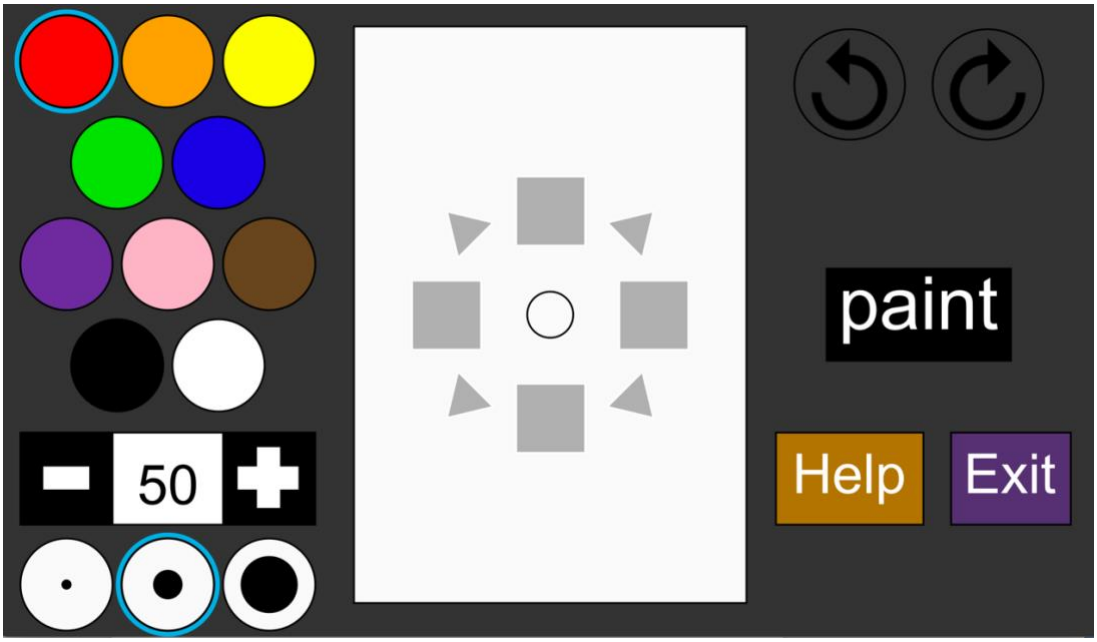

Figure 1.2. *Team 12's Painting Screen GUI*

1.2 Modern Eye Tracking Technology

Current eye tracking works non-intrusively through cameras and light, making it convenient and comfortable for users. The device's light source sends an infrared or near-infrared light out to illuminate the user's pupil. This generates a reflection on the user's cornea, the outside of the eye. The device's infrared camera records the reflection of light and uses the calculated difference between the pupil and the infrared reflection to determine the direction of the user's gaze [3]. The latency time, or the length of time between eye movement and corresponding change on display, ranges from 45 to 81 ms [10].

Because eye gaze is mapped based on a difference between two parts of the eye, the angle of user's head and eyes is an important factor. If the user's head turns away, the camera can no longer spot a user's eyes. If the camera is too close to the user, the eye tracker will no longer support selection on the edges of the screen because it can no longer clearly observe the user's eyes when they turn their heads at an angle to look at the edges of the screen [4].

Full eye gaze control is not included with many standard consumer operating systems. Therefore, utilizing eye gaze for exclusive control of a computer requires purchasing both the eye tracking hardware and relevant computer software. The base version of Windows OS runs a limited Eye Control option which has the user choose each action from a launchpad and then apply that selection to the interface [2]. To properly replace the mouse input with a Tobii eye tracker requires purchasing Windows Control software [12], which effectively locks disabled users into needing to purchase additional software to interact with a device. While this cost helps fund the further development of the company's eye gaze technology, it does put additional strain and financial barriers on a user's abilities to interact with technology.

1.3 Outline

Chapter 2 explains the characteristic of human eye gaze and its interaction with user interfaces, as well as design problems unique to tracking eye gaze as software input. Chapter 3 proposes solutions to the design problems and explains the implementations relevant to Team 12's project design. Chapter 4 summarizes the conclusions and outlines possible future work.

Chapter 2

Using Eye Gaze Control

2.1 Characteristics of Human Eye Gaze

Human eye gaze is a complex process run with many parts of the eyes and the brain. While human sight feels smooth, it is actually made of constant small motions which are corrected by the optic pathway and are therefore largely imperceptible to humans in action. For eye tracking, especially for interactions on a screen, there are two relevant types of eye movement:

- Fixation is the relatively stable action of looking directly at a point. This is the perceived action of looking at something.
- Saccade is the rapid movement between points. These are ballistic movements, meaning they are taken with a trajectory from one point to the next and cannot be altered in the middle of movements. Saccades move the eye gaze from one fixation to the next [8].

The figure below displays the combinations of fixations and saccades which constitute eyesight. Subjects were asked to trace the shapes, and the results were recorded with an apparatus of eye caps and cameras [13]. Note how constant the movement is throughout, even at fixation points.

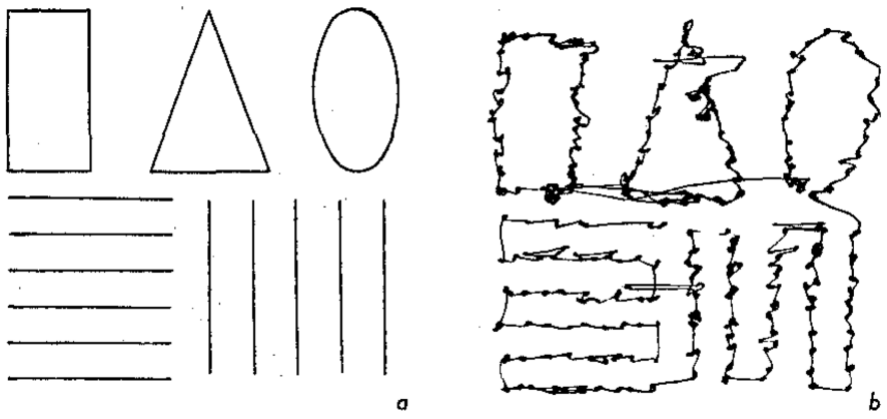

Figure 2.1. *Eye gaze of tracing shapes [13]*

The jittery action of the eye movement seen in Figure 2.1 is likely a primary cause of trouble in eye tracking devices. Human eye movements can get very noisy, and a user can perceive themselves as looking directly at point while their eyes are physically making constant shifts. If the eye tracker takes this information at face value, there is rarely a steady user input.

To gain a better understanding of how an eye tracking device interprets information, Senior Design Team 12 attempted to interact with its GUI design using a Tobii Eye Tracker 4C, available in 1300 Patrick F. Taylor Hall and plugged into the team's personal laptop [11]. The team used a gaze trace feature to visualize where the eye tracker mapped a user's gaze and then modeled that onto the GUI design. Those results are visible below.

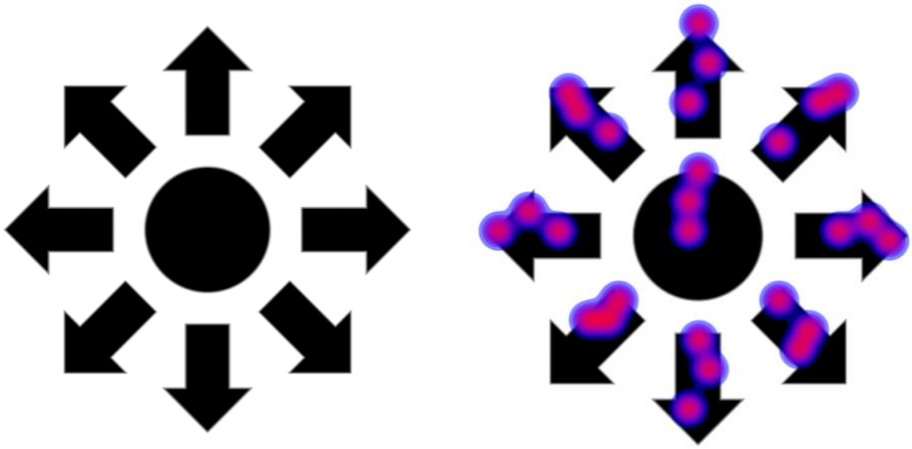

Figure 2.2. *Project GUI and Results of Eye Tracker Selection*

The results of Figures 2.1 and 2.2 indicate that the shapes were perceived as a collection of points, and the user's eye gaze moved around specific points of the on-screen shapes.

Interestingly, an image which perfectly lines up with a viewer's retina as their eyes move around will actually soon disappear from view [9]. This suggests that constant eye movements help a viewer take in new information to gain a fuller picture of what is being looked at while reducing the direct perception of objects deemed unnecessary noise.

2.2 Selection Interaction While Using Eye Gaze Input

In its current basic implementation, eye gaze control works along with other forms of user input to move to a selection faster or more intuitively, while the actual on-screen interaction is performed through another input, such as a mouse click or keystroke. This is the implementation recommended for an able-bodied consumer using an eye tracker connected to a standard computer [11]. With this setup, no change is truly needed to the user interface.

In some cases, input other than eye gaze is not possible or desired. For disabled users, the ability to interact through other methods might not be possible. With this implementation, the interface now needs to use one source (eye gaze) as both a movement and an action. A common solution is to use dwell time to perform an action, allowing eye gaze to both select and interact. Dwell time has the user hold their gaze at the same position for a set amount of time, after which the interaction is chosen.

Because of constant eye movements, determining a "dwell" can be more challenging than timing a selection which stays in the same place for a period of time. Because users are accustomed to always moving their gaze, even slightly and unconsciously, holding eyesight in place long enough to be registered as a full "dwell" is likely to be uncomfortable. Instead, the eye tracker needs to account for eye gaze held in the same general region to allow for shifts in vision through saccades and fixations on similar points [7].

The specific time interval used to determine a dwell is desired to be just long enough, landing at the high end of the natural time for a user to look in the immediate area around one specific point. This implementation allows eye gaze selection to feel natural yet intentional. As a user becomes accustomed to eye tracking technology, the optimal dwell time could likely decrease.

2.3 Problems Unique to Eye Gaze Tracking Input

Because interacting with a screen through eye gaze is unfamiliar to most people and is not widely considered in the design of user interfaces, there are unique problems and design considerations which arise for using exclusive eye gaze control.

2.3.1 Failure to Detect Eyes

Sometimes the tracker will fail to detect an eye gaze at all. In the case of user blinking, glare in the camera, user leaving the eye tracker's field of view, or some other issue, the tracker will momentarily be unable to register an eye gaze. This clearly presents a problem for a system running on being able to accurately be directed with a user's eyes. Unlike a mouse, which requires a physical click to select, eye gaze which runs on dwell time can accidently select an interaction when left hovering on the screen.

When a user leaves the field of view, or even greatly changes their head angle and distance to the eye tracker, the device's calibration is made inaccurate. A user who continues to use a miscalibrated device will encounter more frequent incorrect interactions.

2.3.2 Discriminating User Input

To gather information on user input, Team 12 used a Tobii Dynavox I-12+ device. The testing ran twice, with a rectangular and a circular button. The button was displayed on-screen for the user to select by dwelling over with their eye gaze, while the mapped eye gaze positions were stored in a dynamic array. Once the button was selected, the program saved an output image displaying the input path and then the program ended. The results are visible in Figures 2.3 and 2.4.

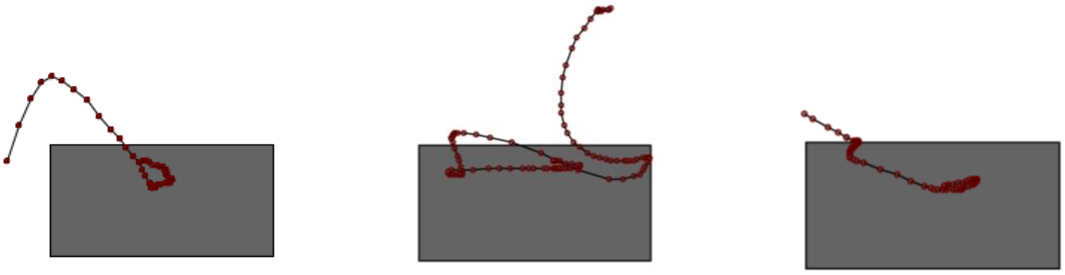

Figure 2.3. *Rectangular button testing*

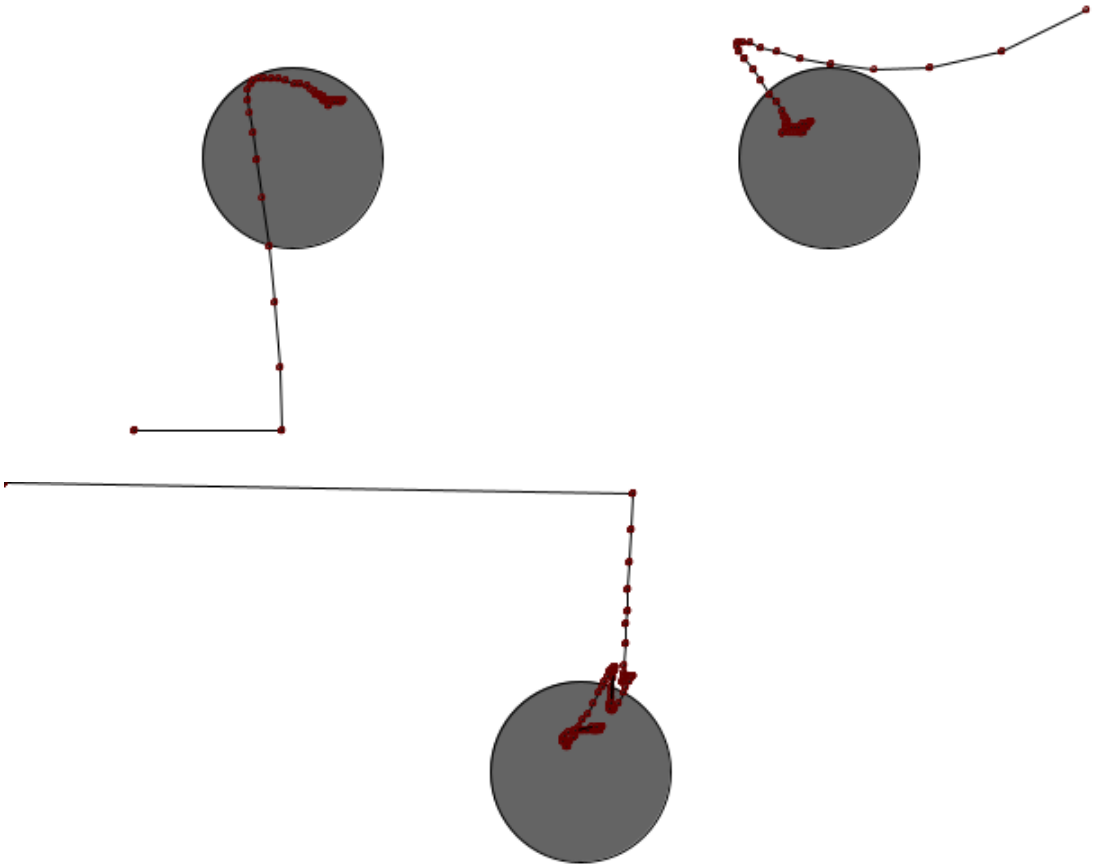

Figure 2.4. *Circular button testing*

The testing reveals that, as the user perceives the shapes on screen, their eye movements sometimes fall outside of the bounds of the button, even when still focusing on the button. This difference in user intention and computer mapping can cause an intended action to go unselected.

2.3.3 Directing User's Fixation

As seen in Figures 2.3 and 2.4, a user's fixations will cause their mapped eye gaze input to occasionally remain in a specific location. To select a button, the user's gaze will need to land within the region. When attempting to not select a button, the user's gaze will need to land in a neutral region. As established in section 2.1, the on-screen elements are perceived as a series of points as the user scans the GUI. Assuming the user is proficient at reading and communicating in the English language, they will be familiar with scanning information from left to right, top to bottom [5]. Careful arrangement of items on-screen and of information within regions can make a GUI easier to scan and select.

2.3.4 The "Midas Touch" Problem

The "Midas Touch" Problem describes the difficulty of eye tracking always being on. For a user accustomed to using input such as a mouse, eye gaze input will feel helpful and novel at first, but selecting with eye gaze will become a problem when the user starts to feel they can never rest their eyes on the screen without causing an action [6]. This result might send a user into increasing panic, causing them to look across the screen more furiously and cause a Rube Goldberg-like series of on-screen actions to take place.

2.3.5 Visible Cursor

An on-screen cursor is essential in using a mouse to visualize the current input, but it is not necessarily essential when operating with eye gaze. If the eye tracking is behaving with a reasonable degree of accuracy, the user does not need to see the current input location because they are already looking at it. In fact, a perfectly accurate eye tracker's cursor would move so perfectly with the user's eye gaze that it would become stationary on the user's retina, and therefore it would disappear from the user's view, as discussed Section 2.1.

Eye tracking is not close to that level of accuracy, so in reality a cursor becomes a massive distraction to the user. Because it constantly moves around, it can easily attract the user's attention. In the best case, this causes the user to look at the cursor as it remains in place, creating a user-software feedback loop of no movement. An unwanted effect of this could be accidentally interacting with some element at that position on the screen. A bad effect of the user looking at the cursor appears if the eye tracker has some noticeable inaccuracy in mapping the user's gaze. If, for instance, the tracker maps the input below the user's eye gaze, the user might look down to the cursor, thus causing the cursor to move further down, which causes the user to look down, and a feedback loop is now in place as the cursor continues to move ahead of the user's gaze which it constantly draws back to itself, until the cursor has moved to the bottom of the screen.

Before writing off a cursor completely, consider that having no visible cursor can leave the user feeling unsure if eye tracking is working, and can cause inaccurate selections if the user has no knowledge of where the input is actually mapped. Because software GUIs display visual location feedback, users have developed an expectation that that will occur [4].

2.3.6 Drawing Attention Away from Important Region

With standard mouse input, eye movement is neutral. Because it does not correspond directly to input, the user is free to move their gaze around without accidentally selecting an unwanted action. Therefore, flashing buttons, popups, and other distracting elements that are additional to the user's intended selection will not immediately affect the user input.

With eye gaze selection, the user's eyes are always actively creating an input. Additional elements that draw the user's attention can potentially cause the user to select an unwanted action or fail to select a desired action.

Chapter 3

Design Solutions

3.1 Addressing Problems Unique to Eye Gaze Tracking Input

To address the problems discussed in section 2.3, a wide set of solutions can be implemented. Because some problems with eye gaze control are connected to specific actions or user ability, the solutions are often tailored to address the particular design problem. This section includes possible solutions for redesigning a user interface or computer operating system, as well as explanations for how Team 12 addressed the problems pertinent to the painting device GUI.

3.1.1 Failure to Detect Eyes

A good solution is the obvious one – if no eye gaze input, then no input to screen. When eye gaze is not detected, set the input value to an error value outside of the normal range of operation (ex: -1). The screen itself will, effectively, receive no input, so there is no danger of a selection being made erroneously [4]. To accommodate cases where the eye tracker has a serious error and cannot detect eye gaze for an extended period of time, a timer can be implemented to track the length of the error and, after a defined period, pause all operation on the computer until the user can regain control. This will minimize the dangers of the eye tracker causing an unwanted series of interactions if it ever has a major tracking issue and works sporadically.

For a wheelchair-bound user, this problem is less likely to occur because their head is supported with a headrest, maintaining the proper distance and angle for use of the assistive device. They are unlikely to fall outside of eye tracker's field of view, so the device calibration should also be maintained throughout use.

3.1.2 Discriminating User Input

To account for the noise of eye movements and the mapped input occasionally falling outside of the button region, buttons need to be reasonably separated. If they are too close, eye gaze input may more frequently be mapped to an incorrect selection. An increase in negative space between elements will make a GUI more readable, though this often comes with a necessary trade-off of reducing the elements on-screen.

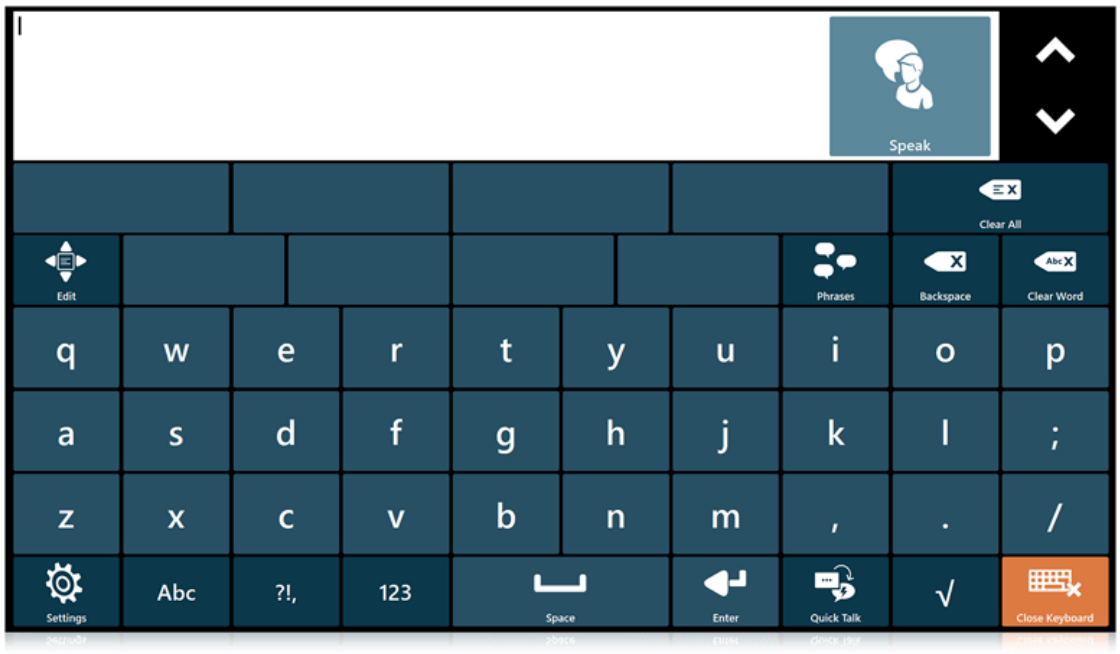

Figure 3.1. *Tobii Communicator keyboard [12]*

Figure 3.1 displays the Communicator AAC software run on a Tobii Dynavox. Because it is intended to offer a full English keyboard and a number of other actions, its design is very busy, with limited space between buttons. Team 12 found that using this software was challenging, as the user's input was often mapped to the wrong letter.

To avoid a design like Figure 3.1, Team 12 avoiding overcrowding the GUI with actions. To account for the issue found in section 2.3.2 of eye gaze falling out of a button's boundaries, the

designed painting GUI offers a selectable area around the buttons. This selectable area corresponds to the region in which a button can be activated, and it is larger than the visible button shapes. In this manner, a user's mapped input can fall outside of a button's shape and still activate the button.

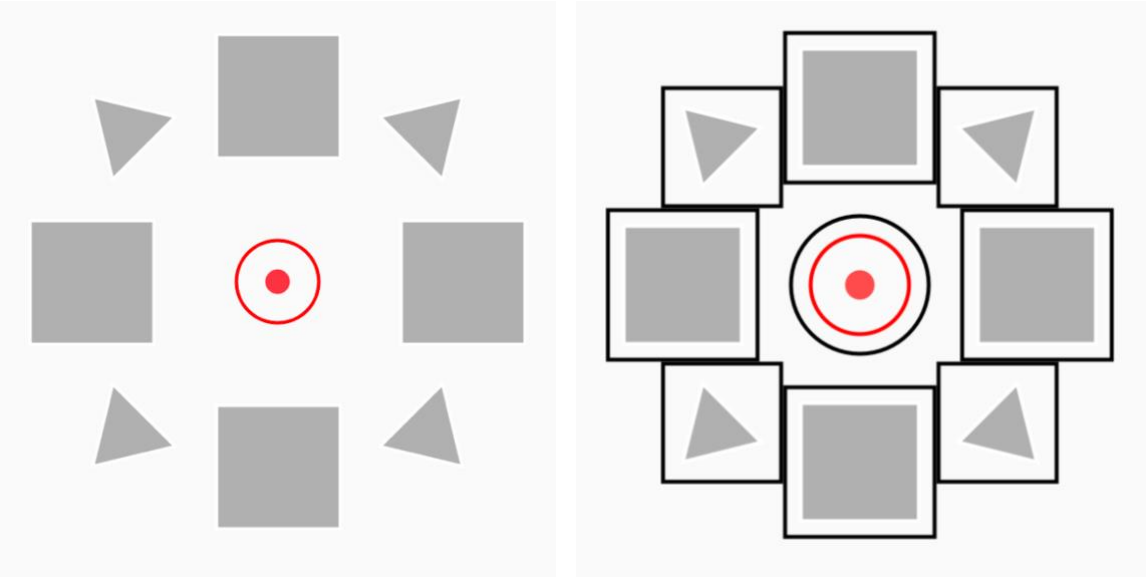

Figure 3.2. *Painting Buttons with Selectable Area*

The selectable areas, such as those visible in Figure 3.2, are achieved by separating the function used to display a button and the function used for activation. With these properties separated, different variables can be applied to specify the displayed button size and button behavior.

3.1.3 Directing User's Fixation

To gather information on user input, a Tobii Dynavox I-12+ device was used in a similar manner to section 2.3.2. The testing ran with a rectangular and a circular button, but this time with a visible dot in the center of the buttons. The button was displayed on-screen for the user to select by dwelling over with their eye gaze, while the mapped eye gaze positions were stored in a dynamic array. Once the button was selected, the program saved an output image displaying the input path and then the program ended. The results are visible in Figures 3.3 and 3.4.

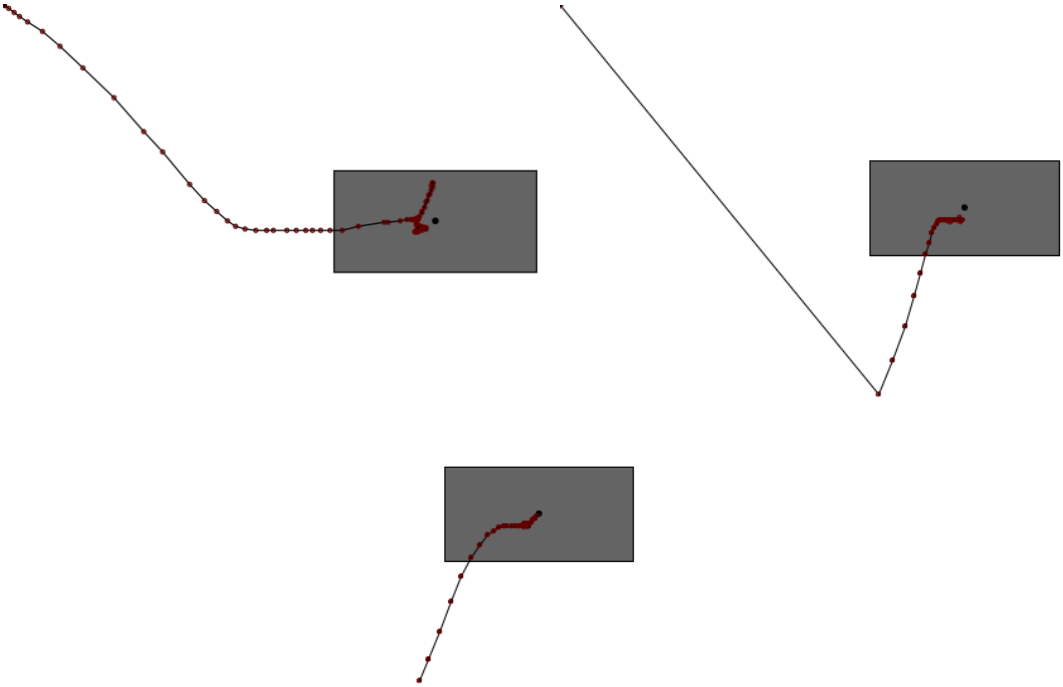

Figure 3.3. *Rectangular button with center dot testing*

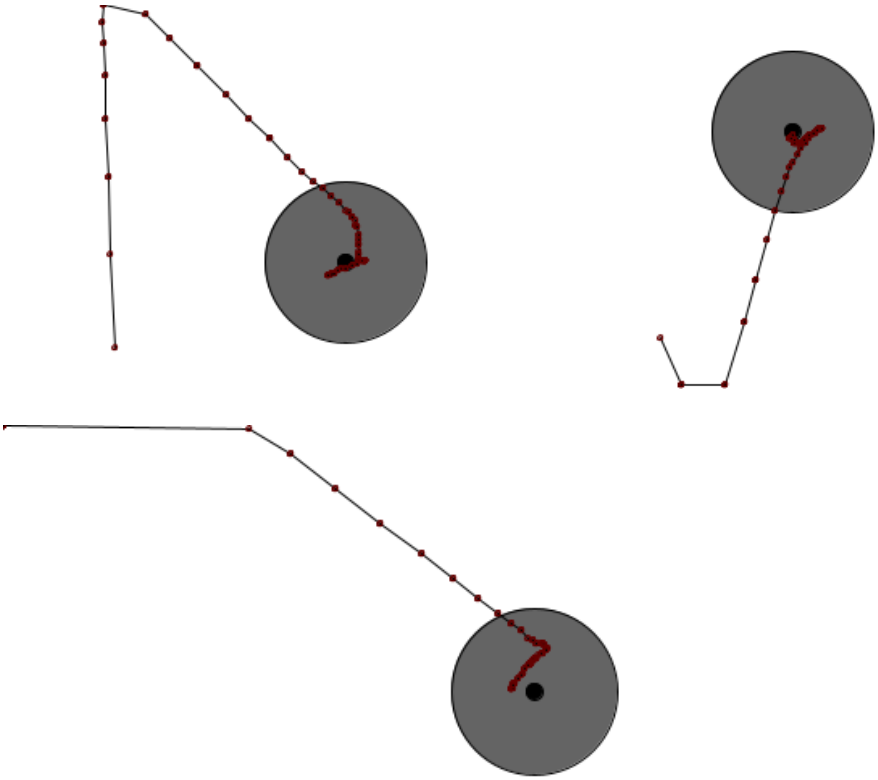

Figure 3.4. *Circular button with center dot testing*

The results show that the user's gaze was directed toward the center of the shape, where the dot was present. This prevents the input from falling outside of the button because the input is much more centered than the selection on the edge visible in section 2.3.2. The results also show the errors in eye tracking calibration and user fixation, as all selections fall slightly left of the center dot. Because the button is sized much larger than that error in mapping and calibration, the user is able to select the button regardless.

3.1.4 The "Midas Touch" Problem

To solve this problem, eye tracking interfaces can implement an on/off button in a clearly identifiable position on screen. Whenever the user would like to dwell longer on content without interacting, such as when reading text or watching a video, they can select the eye tracking button off. This will not turn off the eye tracker, but it will merely shut off on-screen selections except for the on/off button. Now the user is free to look around the screen without causing an undesired interaction (except for the on/off button). When the user would like to begin interactions with the screen again, they can look back at the on/off button. Because the eye tracker is still tracking their eyes, it will be able to select the button and turn full eye tracking back on.

To avoid the instance of selecting multiple actions, new screens should place buttons in positions different from the button used to access the screen. Team 12's design loads a confirmation message with Yes/No options after the Paint button is selected. These Yes/No buttons are placed away from the Paint button, so that the user will not accidentally confirm or deny the actions by lingering in the same position.

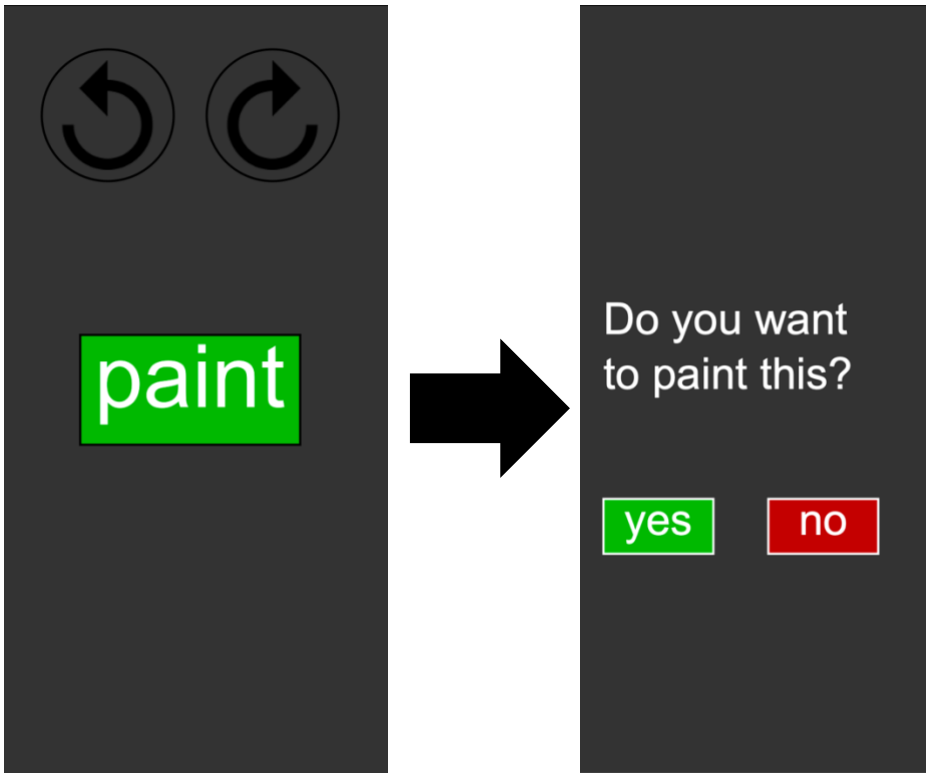

Figure 3.5. *Avoiding Button Overlap*

3.1.5 Visible Cursor

Because users expect visual input feedback, interface designs should include a cursor or equivalent visual information. One good solution is to highlight the element which the eye gaze is currently mapped over. This can be done by having a color appear around the element or by having the element zoomed slightly larger. This method of visual feedback removes the cursor but still maintains the animations often associated with "mousing over" with a traditional computer mouse. The animation can even be modified to have the color or zoom grow throughout the dwell time. Having the selection slowly ramp into place provides good feedback that the system is about to complete the selection and is running properly, as a steady on-screen image could even be confused with a paused or crashed program.

Team 12's GUI design has all buttons react to being "moused over" by having them change color, often to a darker fill color. This is done by constantly checking the location of the mouse, or mapped eye gaze, input and comparing it to the location of all buttons. If the mouse falls within a particular button, a boolean is activated to redraw the button with a different fill. When the input leaves a button region, the boolean reverts to false and the button is redrawn with the standard color.

3.1.6 Drawing Attention Away from Important Region

To prevent drawing the user's focus away from an intended action and toward an unintended action, user interfaces should avoid all extraneous motion that is unrelated to the current user selection. This includes limiting popups, sidebar animations or advertisements, and distracting effects like bevels and zooms. The interface should also allow for a long period of no user selections without having elements flash to invite the user to begin an action.

Team 12's GUI avoids drawing the user's attention to misplaced areas by having very limited behavior triggered by another part of the screen. The only instance of a behavior occurring on the periphery of a user's gaze is the activation of the Paint Confirmation, Undo, and Redo buttons. As seen in Figure 3.6, these buttons are initially darkened to indicate that they are not functional. Only after the user makes changes on the digital canvas are the buttons lit up to indicate that they are ready to perform an action. If user testing indicates that this behavior is problematic or unwanted, the buttons will no longer change color and will instead be set to initially display as the brighter "on" position.

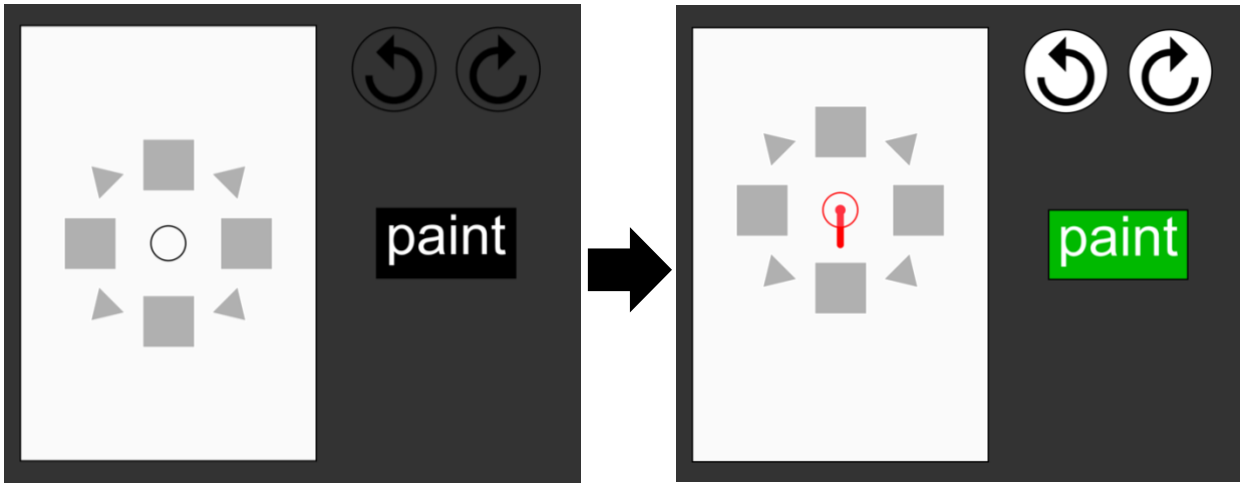

Figure 3.6. *Buttons Activated by Digital Canvas*

Chapter 4

Conclusions and Future Work

4.1 Conclusions

Adapting a user interface to be viewed and controlled by a user's gaze requires careful attention to the intended function of the program and the physical abilities of the user. The best design decisions for optimal eye gaze control go against modern UI design practices by simplifying on-screen designs, removing functionality, motion animations, and ad space.

4.2 Future Work

To gain a wider understanding of eye gaze control and avoid the biased effects of the research, the user testing done in sections 2.3.2 and 3.1.3 could be expanded into a wider study. This study could generate multiple pages of on-screen buttons, in greater variation and combined in different organizational patterns, to better determine how a user will scan across information and ultimately attempt to select an action.

To improve the inclusive design possibilities, the intended userbase, disabled individuals who use eye gaze as an exclusive control method, should be included in every step of the design process. Their insight may assist the designer in making choices best suited for eye gaze, beyond the biases of an able-bodied user familiar with tactile buttons and control inputs. This could include reducing button sizes, creating new button shapes, and implementing new color palettes to better display information for selection.

The design work could investigate additional solutions to account for the mapping or calibration error in the eye tracking camera. Possible solutions are variable button size making items larger in areas of greater mapping error, aligning the selectable area to account for the mapping error, and changing on-screen shapes to direct user's against the error.

This work also fails to account for eyestrain caused by long periods of looking at a digital monitor. Repeatedly holding one's gaze in a long fixation is unnatural to many users, so reducing the frequency of that action would likely improve the comfort of controlling a device with eye gaze. The design proposals in this paper do not address users with strong blue light sensitivity or adapt the designs to use with visual impairments or color blindness.

Beyond adapting the on-screen interface, changes could be made to the eye tracking hardware technology. Higher resolution cameras could generate more precise user input. Pairing eye tracking with machine learning or AI could help the device predict the user's eye movements, reducing latency or noisy eye fixations. These solutions would likely come at the expense of higher processing costs.

References

[1] Berkowitz, S. (2016, July 10). What are the top 3 AAC device manufacturers. *Kidz Learn Language*. Retrieved from https://kidzlearnlanguage.blogspot.com/2016/07/what-are-top-3-aac-device-manufacturers.html

[2] Eye Control. (2022). Get started with eye control in windows. Retrieved from https://support.microsoft.com/en-us/windows/get-started-with-eye-control-in-windows-1a170a20-1083-2452-8f42-17a7d4fe89a9

[3] Eyeware (2019, August 21). Eye tracking 101: What is it & how does it work in real life? Eyeware. Retrieved from https://eyeware.tech/blog/what-is-eye-tracking/

[4] Hartson (1993). Eye Movement-Based Human-Computer Interaction Techniques: Toward Non-Command Interfaces. Retrieved from http://www.cs.tufts.edu/~jacob/papers/hartson.pdf

[5] Human eye (n.d.). Human eye – Movements of the eyes. Encyclopedia Britannica. Retrieved March 4, 2022, from https://www.britannica.com/science/human-eye/Movements-of-the-eyes

[6] Jacob, Robert J. K. (1991). The use of eye movements in human-computer interaction techniques: what you look at is what you get. *Association for Computing Machinery*, 9(2). Retrieved from http://www.cs.tufts.edu/~jacob/papers/tois.pdf

[7] Just, Marcel A., Carpenter, Patricia A. (1980). A Theory of Reading: From Eye Fixations to Comprehension. *Psychological Review*, 87(4), 329-354. Retrieved from https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.677.6396&rep=rep1&type=pdf

[8] Purves D, Augustine GJ, Fitzpatrick D, et al., editors. (2001). Types of Eye Movements and Their Functions. *Neuroscience*. 2nd edition. Sunderland (MA): Sinauer Associates. Retrieved from https://www.ncbi.nlm.nih.gov/books/NBK10991/.

[9] Riggs, LA, Ratliff, F., Cornsweet, JC, & Cornsweet, TN. (1953). The disappearance of steadily fixated visual test objects. *Journal of the Optical Society of America*, 43(6), 495-501. Report #: 6. Retrieved from https://escholarship.org/uc/item/9459b626

[10] Stein, N., Niehorster, D. C., Watson, T., Steinicke, F., Rifai, K., Wahl, S., & Lappe, M. (2021). A Comparison of Eye Tracking Latencies Among Several Commercial Head-Mounted Displays. I-Perception. https://doi.org/10.1177/2041669520983338

[11] Tobii Gaming. (2022). The next generation of head tracking. Retrieved from https://gaming.tobii.com

[12] Windows Control. (2022). Windows eye control. Retrieved from https://us.tobiidynavox.com/products/windows-control

[13] Yarbus. (1967). Eye Movements and Vision. Retrieved from http://wexler.free.fr/library/files/yarbus%20(1967)%20eye%20movements%20and%20vision.pdf

Vita

Timothy Curol was born in Lake Charles, Louisiana. In August 2018, he enrolled at Louisiana State University and began pursuing a Bachelor of Science in Computer Engineering. He anticipates graduating in May 2022.