Requirements Document

iTrack

Adrian Berding, Thomas Huang, Sheily Shah, Kristen Reault

System Description

System Purpose

People who are paralyzed from the neck down have difficulties using their computers, since they are unable to utilize their hands. While they are able to interact using voice commands, this requires much effort from the users just to browse the web. Additionally, in public settings, using voice commands might disturb others. There is a need for a tool that helps these people use their computers in a way that is more natural and discreet.

To this end, we developed an eye-tracking software to enable people who are paralyzed from the neck down to interact with their computers using their eyes. Since they are already using their eyes to look at their computer screens, this interaction is much more natural. It is also much more discreet, since the interaction is basically silent. Users of our software will be able to utilize the movements of their eyes to perform certain commands on the computer, such as increasing the font size or scrolling down a page. Through our software, we hope to enable those who are paralyzed from the neck down to more easily use their computers.

Project Scope

Base Functionality

The core functionality of our project will be to recognize simple eye movements and execute commands corresponding to those movements. For example, we plan on having winking the right eye correspond to scrolling down a web page. The core eye movements that we will look for and are confident we can track are looking left, looking right, look up, winking the right eye, winking the left eye, closing eyes, and any combination of the movements (looking down is not included). We understand that blinking is a very common and unintentional movement, so that is why we will have closing eyes correspond to re-calibrating the eye center, a non disruptive function if accidentally triggered. The eye center is essentially a coordinate position of a user's eyes when looking straight forward and is what is used to determine what direction a user is looking. So re-calibration after closing eyes will essentially create a new position for the center of the eye on the next frame that one's eyes are open. For all other eye movements, the commands possible through our software that we expect as base functionality will incorporate anything that can be done using a keyboard. Another form of calibration that will be available to the user is the ability to change the sensitivity threshold to trigger commands, essentially adjusting how far left one has to look to trigger a look left command or how long a user has to keep their eyes closed.

Since we foresee a limitation on the number of unique eye movements we can reasonably recognize, we will implement “modes.” Different modes will essentially modify what command corresponds to the core eye movements. Modes that are planned to be implemented as default are video watching and reading mode. On top of that, we will have a way for users to create a custom mode with their own commands. To help keep track of all the commands and modes, we will have a help menu that will act as a hub for switching modes, viewing command lists, and creating new modes. We also hope to enable users to input commands that pull up certain webpage links. For example, users can add a command that when executed would open the front page of *The New York Times* on a web browser.

Extended Functionality

Outside of our base functionality, we have some reach goals for our project that might not be possible due to time or other limitations. A specific reach goal we have for our project is the ability to control a mouse with one's eyes. This would entail having the mouse move to where one's point of gaze is on the computer screen. A combination of lack of time and the inconsistency of camera-based pupil tracking might make this goal hard to accomplish. On top of the core eye movements, we would like to potentially capture more complicated movements, like eye rolling and diagonal movements.

Use Case Diagram

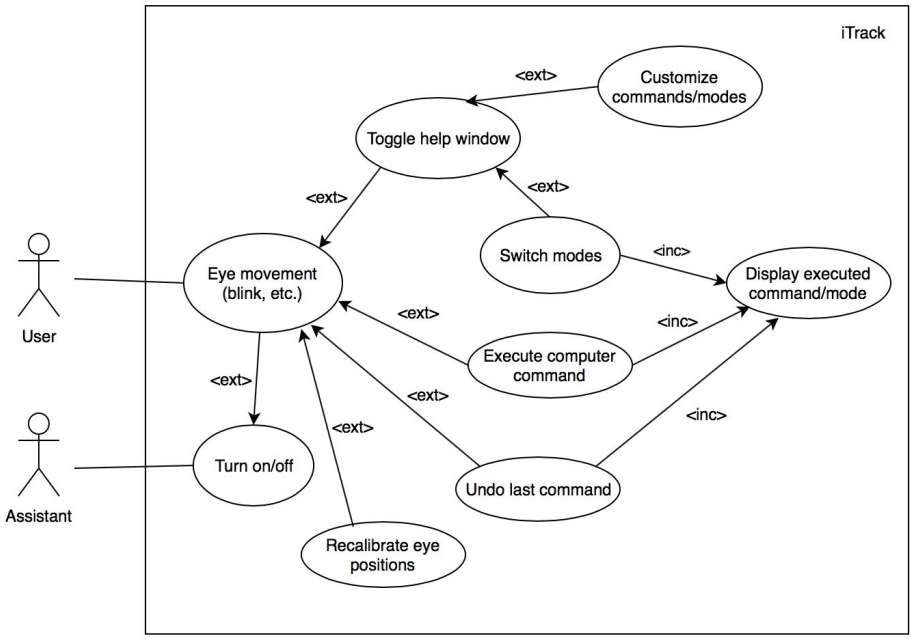

Figure 1 Use Case Diagram

The use case diagram below assumes that the user is paralyzed from the neckdown and needs some form of a “Assistant” to start our program, this could be another person or another tool that the user uses as accessibility like voice commands. Once the program is running, eye movement from the user can trigger 1 of 4 things: re-calibration of the eye center (closed eyes), toggling of the help window (look up), undoing of the last command (look left then right), and execution of computer commands (all other movements).These are the four core eye movements. In the help menu users will be able to switch modes or access the customization settings by using eye movements. For any action that switches modes, executes a commands, or undoes a commands we will display said action as a widget on the computer screen.

Project Flow

For a standard use of our program, there will be 5 objects interacting with each other during the session. The first of these objects is the “User,” which can interact with the system through eye movements. The next object is the “Pupil Detection”; this is the open source library that will be used to find and track the users pupils. The next object “iTrack coordinate handler,” which is the part of our software that will communicate with the “Pupil Detection” and determine if there was any new core eye movements. When an eye movement is detected, the “iTrack coordinate handler” will call the “iTrack interface handler,” which will execute the corresponding commands and update the final object, the “Display”. The "Display" represents the computer display of the user.

The scenario represented in the sequence diagrams below starts with our application already running on the user's computer. The user is expected to be directly in front of their device's camera. The user wants to read a *New York Times* article that is open on their computer about a recent event, which includes news broadcasts of the event. The user will try to scroll through the article to read it and watch the broadcasts, using just the movements of their eyes. The user is also in a public scenario, so there could be various people in the background that are also in the frame of the camera.

Expected Use Sequence Diagram

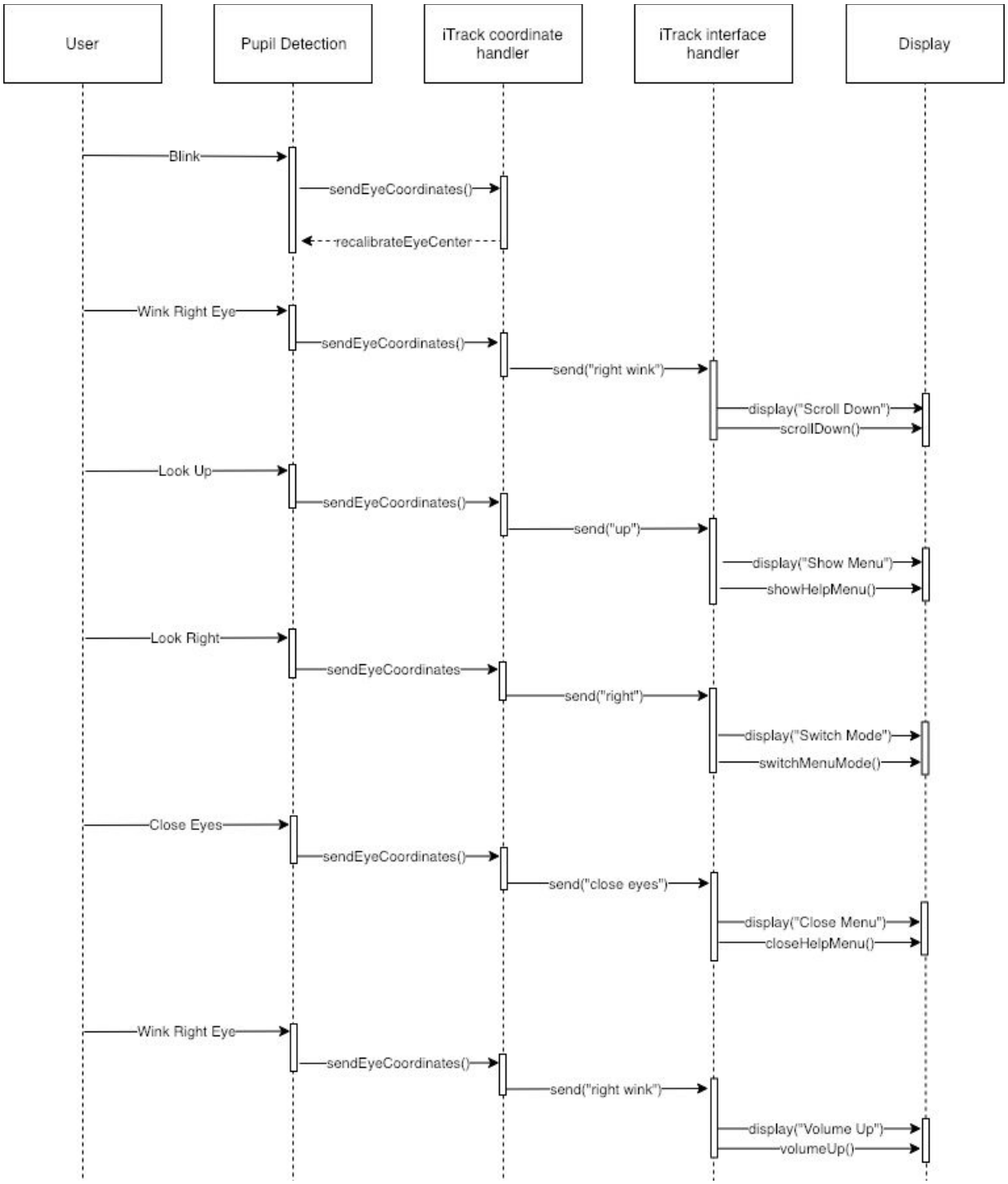

Figure 2 Expected Usage Sequence Diagram

When the user first uses our software, they will blink their eyes, which will notify our coordinate handler to recalibrate the pupil of the user's eyes. This will make sure that our software is accurate and will perform as expected. Next, while on a web page, the user can wink their right eye, which will notify our interface handler to scroll the web page down. A widget display will also pop up briefly to notify the user of the executed command. If the user forgets what commands are available or what eye movements map to what commands, they can look up, which will notify our interface handler to show the help menu. This menu contains commands particular mode that the user is on as well as which one of the four core eye movements that will trigger each command. In this menu, the user can also look right, which will notify the interface handler to switch to a different mode. The user will start in the "Read" mode, and will be in the "Video" mode after switching. To close the menu, the user will close their eyes, which will notify the interface handler to close the help menu. Now, in "Video" mode, the user can wink their right eye again, but this time it will increase the volume of the video.

Unexpected Usage Sequence Diagram

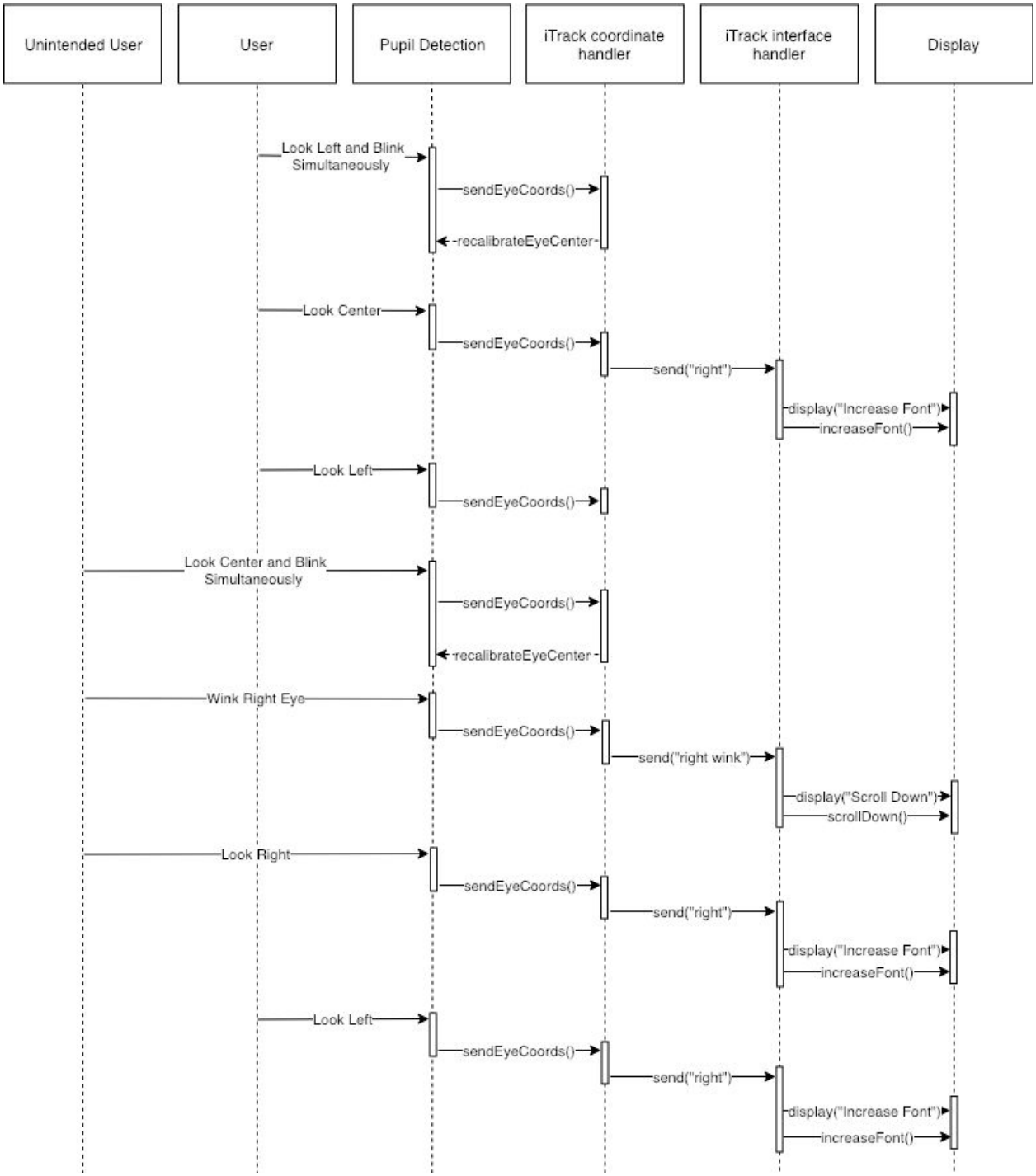

Figure 3 Unexpected Usage Sequence Diagram

When reading an article, the user might accidentally look left and blink simultaneously, which will notify the coordinate handler to recalibrate the center of the eye. Since the user is looking left, this will incorrectly calibrate the position of the eyes. Now, when the users look straight, the

coordinate handler will think the user is looking right, and send that command to the interface handler. This will incorrectly trigger the increase font size command, even though the user did not intend for that to happen. If the user wants to decrease the font size to change it back to normal, the coordinate handler will think the user is looking straight, and will not send any commands to the interface handler.

Additionally, the camera might pick up another pair of eyes from an unintended user. When this unintended user blinks, it will notify the coordinate handler to recalibrate the center of the eyes with respect to this user. Whenever this user winks their right eye, it will notify the interface handler to trigger the scroll down command, even though the original user did not intend for that to happen. When this unintended user looks right, it will increase the font size of the article. Now, when the original user looks left to decrease the font size to get back to the original size, the coordinate handler might think the user is looking right, since it is calibrated to the other user's eyes. This will again increase the font size incorrectly.

Enumeration of Requirements

Navigation Requirements

Number: NVG-1	**Name:** Scrolling	**Functional**	**Priority:** α
Description	Enable users to scroll web pages using their eyes.		
Source	We wanted users to be able to read an article with our software. Scrolling would be an essential feature for such a task.		
Notes			

Number: NVG-2	**Name:** Moving Mouse	**Functional**	**Priority:** Ω
Description	Enable users to control their mouse cursor with their eyes.		
Source	As part of background research, we looked into many existing eye tracking devices and software and found this to be a common function.		
Notes	This is a reach goal for the project as it might be limited by the accuracy of the pupil tracking we use.		

Number: NVG-3	**Name:** Clicking	**Functional**	**Priority:** Ω
Description	Enable users to click with their mouse cursor with their eyes.		
Source	This goes along with the mouse control of NVG-2.		
Notes			

Number: NAV-5	**Name:** Command to Webpage	**Functional**	**Priority:** Ω
Description	Enable users to input a webpage link as a custom command		
Source	We wanted the ability to quickly go to web pages without much effort. Mimics “Favorites” on web browsers.		
Notes			

Media Requirements

Number: MD-1	**Name:** Volume	**Functional**	**Priority:** β
Description	Enable users to control the volume of a video with their eyes.		
Source	When listening to Brad talk about what he most often does on his computer, he said one of the things he does a lot is watch Hulu. While our project isn't targeted towards Brad, we thought this was a good thing to consider and wanted to have video watching based commands.		
Notes			

Number: MD-2	**Name:** Play/Pause Video	**Functional**	**Priority:** β
Description	Enable users to play/pause a video with their eyes.		
Source	Same as MD-1.		
Notes			

Display Requirements

Number: DSP-1	**Name:** Font Size	**Functional**	**Priority:** α
Description	Enable users to increase/decrease the font size of a web page with their eyes.		
Source	We wanted users to be able to read an article with our software. When reading on a computer, enlarging the font size can be a useful tool to have.		
Notes			

Meta Requirements

Number: META-1	**Name:** Undo	**Functional**	**Priority:** β
Description	Enable users to undo the previous command. For example, undoing an increase font size command should decrease the font size.		
Source	When accidentally triggering a command, we wanted users to have a quick, standard way to undo it.		
Notes	Not all commands will be undoable.		

Number: META-2	**Name:** Lock	**Functional**	**Priority:** β
Description	Enable the users to lock the device so their eye movements will not trigger any commands.		
Source	If a user is at a point where they don't plan on using any commands for a while, we want them to be able to lock the software so they can proceed without worrying of any accidental command triggers.		
Notes			

Mode Requirements

Number: MODE-1	**Name:** Reader Mode	**Functional**	**Priority:** β
Description	Enable users to switch to reader mode, which will allow them to use reading centric commands like scrolling.		
Source	Since we will likely be limited in the number of unique eye movements that we can link to commands, we plan to have different modes that are oriented towards certain tasks.		
Notes			

Number: MODE-2	**Name:** Video Mode	**Functional**	**Priority:** β
Description	Enable users to switch to video mode, which will allow them to use commands that interact with videos.		
Source	Same as MODE-1.		
Notes			

Number: MODE-3	**Name:** Custom Commands and modes	**Functional**	**Priority:** Ω
Description	Enable users to customize their eye movement to command mappings.		
Source	We think there will be additional commands that the users want that we did not account for, so we want to give users a way to make their own custom commands and modes.		
Notes			

Settings Requirements

Number: S-1	**Name:** Calibration	**Functional**	**Priority:** α
Description	Enable users to calibrate the pupil tracking to their specific setup.		
Source	Different variables affect how the tracker works, so we want to give users the ability to calibrate the tracking to their specific setup.		
Notes	What would be required for calibration is not yet totally known, but we do know we would need users to choose a sensitivity threshold (how far left they have to look to trigger a left command).		

UI Requirements

Number: UI-1	**Name:** Help Menu	**Functional**	**Priority:** α
Description	Enable users to bring up a help menu of available commands.		
Source	In case users get confused about what movement corresponds to what command, we want to have a help menu that lists all commands in the current mode.		
Notes	The help menu will later on also act as a hub to switch to different modes.		

Number: UI-2	**Name:** Show Executed Command	**Functional**	**Priority:** α
Description	Display recently executed commands in some fashion.		
Source	We wanted to give users some idea when things happen so they never get confused on whether something did/didn't trigger.		
Notes			

Non Functional Requirements

Number: N-FUNC-1	**Name:** Keyboard and mouse not needed	**Non-Functional**	**Priority:** ∞
Description	The use of our software should not require use of anything but ones eyes. This does not apply to turning it on/off and setting it up.		
Source	Our problem statement specifies that our target client are people paralyzed from the neck down. They would not have ability to use a mouse or keyboard.		
Notes	Exception: need keyboard/mouse for setup and customization.		

Number: N-FUNC-2	**Name:** 1 second response	**Non-Functional**	**Priority:** ∞
Description	Following a user action, our software should complete all linked actions to that command within a second.		
Source	Many of the planned commands (like scrolling) would likely be executed often, therefore we want to make the execution process quick.		
Notes			

Number: N-FUNC-3	**Name:** Works with immobilized head	**Non-Functional**	**Priority:** ∞
Description	Moving one's head should not be required for any of the actions in the software.		
Source	Our target client are people paralyzed from the neck down and would have limited to no mobility of their head.		
Notes			

Number: N-FUNC-4	**Name:** Works without external help	**Non-Functional**	**Priority:** ∞
Description	Users should be able to use our software without the assistance of another person.		
Source			
Notes	**Exception: Setup and customization**		

Number: N-FUNC-5	**Name:** Actions discreet	**Non-Functional**	**Priority:** ∞
Description	All eye movements/commands should be discreet and not noticeable or disruptive to others a room.		
Source	One of the advantages of using eye movement for accessibility rather than voice commands is that they have the potential to be quick and discreet.		
Notes			

Number: N-FUNC-6	**Name:** Inexpensive	**Non-Functional**	**Priority:** ∞
Description	We want to make our software require no additional components except a inexpensive webcam.		
Source	To the suggestion of Dr Chesney, eye trackers a quite expensive but almost everyone has access to a webcam, so if we were to make a webcam based software it would be much more accessible to the public.		
Notes			

Number: N-FUNC-7	**Name:** Easy to install	**Non-Functional**	**Priority:** Ω
Description	We are hoping to make our software easy to install, such that there's no additional plug-ins, packages, or hardware except for a webcam.		
Source	We want setup of this software to be easy and that begins with the installation process.		
Notes			

Number: N-FUNC-8	**Name:** Intuitive Interface	**Non-Functional**	**Priority:** β
Description	The interface for switching modes, seeing commands, and visualizing various settings should be easy to understand and intuitive as possible.		
Source	With any software it is good to have an intuitive interface.		
Notes			

Number: N-FUNC-9	**Name:** Works across all Macs	**Non-Functional**	**Priority:** α
Description	The software should work on all macOS systems.		
Source	All mac laptops have a webcam that is good enough quality to run the pupil tracking, so we felt it would be a good initial target platform.		
Notes			

Number: N-FUNC-10	**Name:** Natural eye movements must not trigger commands	**Non-Functional**	**Priority:** α
Description	Simply looking at any location on the computer screen should not trigger any commands. Other natural actions like blinks should also not trigger any commands.		
Source	We want to avoid any accidental triggering of commands.		
Notes			

Number: N-FUNC-11	**Name:** Minimal Re-calibration	**Non-Functional**	**Priority:** β
Description	The calibration process for the pupil tracking should not be a common occurance and users should be able to easily use the software without re-calibrating either during their session or at the beginning of a session if they have previously used the software in the same computer setup.		
Source	The calibration process would likely require the usage of a mouse therefore we want users to be able to use our software without re-calibrating every time.		
Notes	What would be required for calibration is not yet totally known, we do know we would need users to choose a sensitivity threshold (how far left they have to look to trigger a left command). Actions that would likely require a re-calibration would be changing sitting distance to the computer and changing cameras.		

Prototype User Interface (UI)

Help Menu View

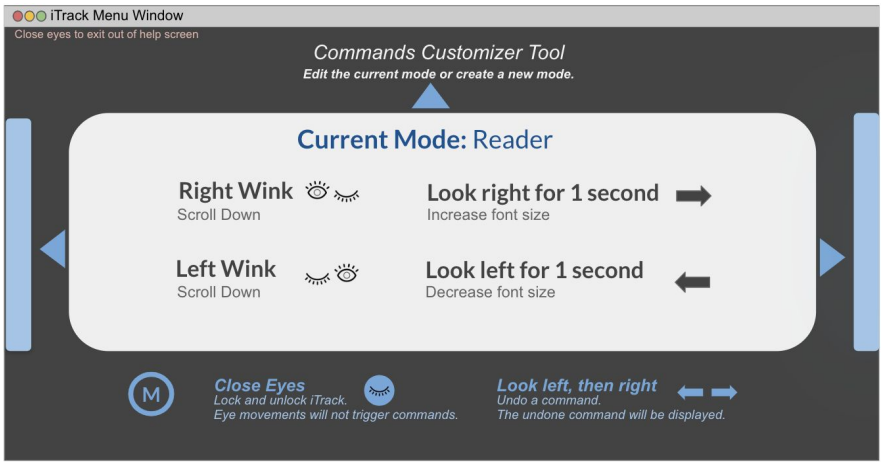

Figure 4 Help Menu- Default Reader Mode

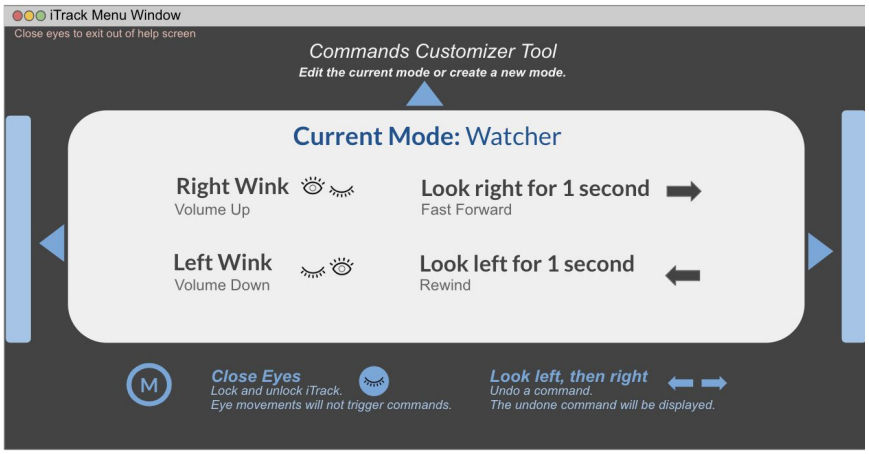

Figure 5 Help Menu - Default Watcher Mode

When a user looks up and holds that position for a couple seconds, a help window will pop up. This help window will, upon opening, show the current mode and the corresponding commands for that mode. Below the mode, we have the two default meta commands, lock/unlock and undo, that will always be available to the user in any mode. The reader view enables simple features like scrolling and increasing/decreasing font size. The watcher view enables the functionality required to watch a Youtube video – like being able to play/pause, forward, and rewind a video. The triangles on either side of the mode indicate the ability to switch to another mode by looking left or right. Similarly, looking up will bring the user to the customization tool.

Commands Customizer View

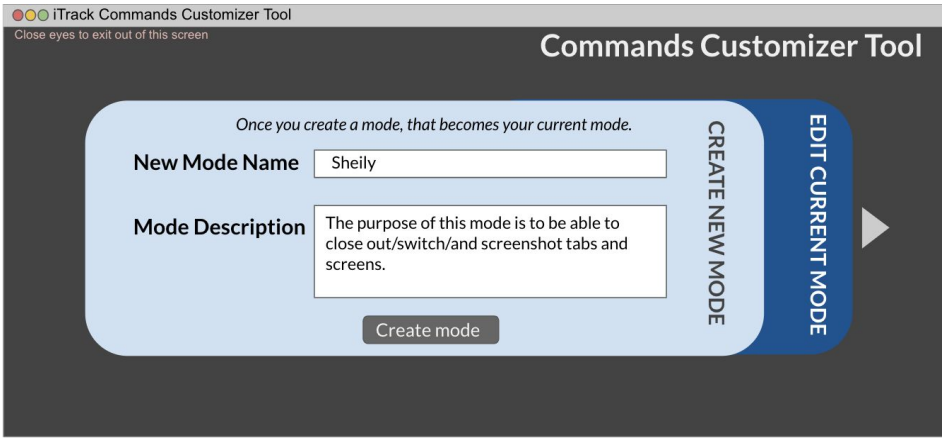

Figure 6 Creating a new mode, "Sheily", on the Commands Customizer Tool

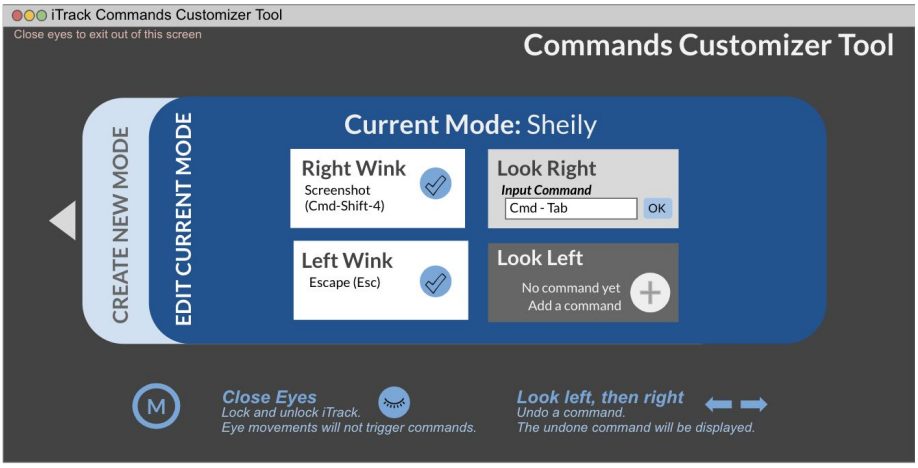

Figure 7 Editing the current mode, "Sheily"

When a user looks up from the help menu view, they would be able to launch the Commands Customizer Tool. For this tool, the user might need the assistance of an assistant to be able to do everything. A user can choose to create a new mode, which would involve setting a mode name and brief description. Once a user creates a new mode, that becomes the current mode. Looking right would allow users to edit the current mode. The "Edit Current Mode" view allows users to write custom keyboard-based commands or webpage links to correspond to the four core eye movements, and always has the Meta Commands on display as well.

In the example above, this user has created a "Sheily" mode. The white boxes with the check marks represent commands that have been set. If a user clicks on a set command, they can change the command. The light gray box shows how the user would input a keystroke command to set a command. The user can put in the keyword "goto" followed by a URL to create a

command that opens up the corresponding website instead of a standard keystroke command. The bottom right box shows an eye movement that has not been set.

Widgets View

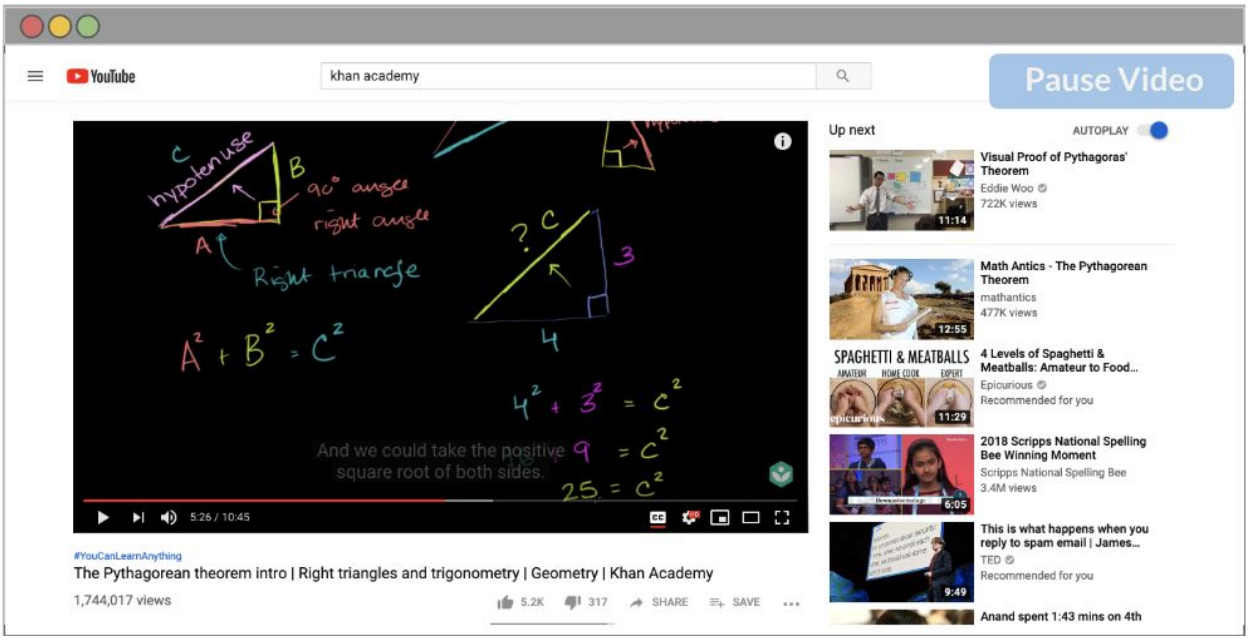

Figure 8 Widget view during regular usage

This view is what the user would be spending the most time on. Suppose a user is on a webpage and on the watcher mode, like the example in Figure 8. Upon performing a command, a widget would show up to show what the executed command was (pause video) and disappear after a few seconds until the next command is executed. Since a lot of core eye movements involve looking away from the computer screen, this widget will be a useful way for users to confirm that an action was triggered when they recenter their gaze.

Environment Description

Development Environment

We will be developing on macOS, since the majority of our team members own laptops from Apple. These laptops come with built-in cameras, which makes testing our software a lot easier and consistent across team members.

Target Environment

Since we are developing on macOS, our main target platform is all Mac computers with built-in cameras. If ample time remains after finishing development of the main functionality of our software, we will consider porting our software to Windows and Linux.

Software Packages and Tools

For tracking pupils, we are using an open source library in C++ called eyeLike [1], which utilizes openCV [2] for face and eye region recognition. For enabling our software to interface with commands on computers, we are using PyAutoGUI [3], a Python module for controlling the mouse and keyboard. We are also considering using WebGazer.js [4] for predicting the gaze of the users.

References

1. eyeLike, https://github.com/trishume/eyeLike, accessed 10 February 2019.
2. OpenCV, https://github.com/opencv/opencv, accessed 10 February 2019.
3. PyAutoGui, https://github.com/asweigart/pyautogui, accessed 12 February 2019.
4. WebGazer.js, https://github.com/brownhci/WebGazer, accessed 12 February 2019.