Design 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. 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're developing 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

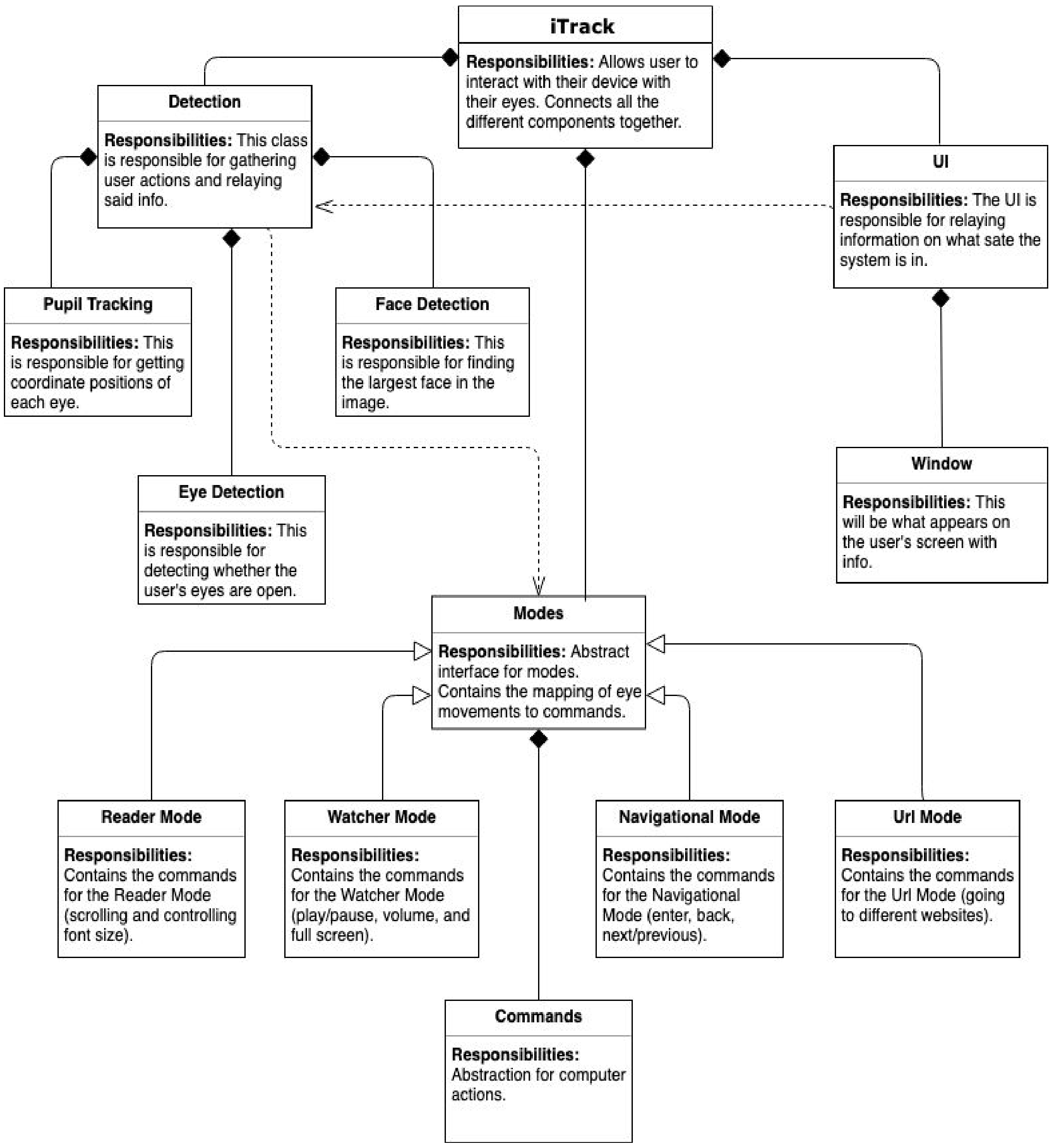
Base Functionality

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, looking 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. In addition to 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.

System-level Class Diagram
In the class diagram in Figure 1, the iTrack class is responsible for communication between the three main components of our software (Detection, Execution/Modes, and UI). The Detection class is responsible for collecting all user actions (pupil and eye movements) and sending that information to the iTrack class when specific actions are detected. The Pupil Tracking, Eye Detection, and Face Detection classes capture the specific user information and send it to the Detection class. When a specific action is sent to iTrack, it will then relay that information to either the Modes class or the UI class depending on the action. The Modes class is responsible for executing a command based on what action was detected and what the current mode is. Reader Mode, Watcher Mode, Navigation Mode, and Url Mode are the default instances of the Modes Class and each contain four unique commands. The UI class displays information regarding the current state of the software to the user, either through pop-up windows or audio cues.


iTrack
Responsibilities: Allows user to interact with their device with their eyes. Connects all the different components together.
Detection
Responsibilities: This class is responsible for gathering user actions and relaying said info.
UI
Responsibilities: The UI is responsible for relaying information on what sate the system is in.
Pupil Tracking
Responsibilities: This is responsible for getting coordinate positions of each eye.
Face Detection
Responsibilities: This is responsible for finding the largest face in the image.
Window
Responsibilities: This will be what appears on the user's screen with info.
Eye Detection
Responsibilities: This is responsible for detecting whether the user's eyes are open.
Modes
Responsibilities: Abstract interface for modes. Contains the mapping of eye movements to commands.
Reader Mode
Responsibilities: Contains the commands for the Reader Mode (scrolling and controlling font size).
Watcher Mode
Responsibilities: Contains the commands for the Watcher Mode (play/pause, volume, and full screen).
Navigational Mode
Responsibilities: Contains the commands for the Navigational Mode (enter, back, next/previous).
Url Mode
Responsibilities: Contains the commands for the Url Mode (going to different websites).
Commands
Responsibilities: Abstraction for computer actions.
Figure 1: System-level Class Diagram

System-level Statechart Diagram

Figure 2 shows the Statechart Diagram for iTrack. Initially, iTrack is idle. iTrack's idle state will execute commands based on corresponding eye movements if the system is not locked. If the system detects that the user's eyes are looking up, it changes state to helpMenuOpen and opens the help menu. In this state, if iTrack detects a left wink, it will either lock or unlock the system, depending on whether it was previously locked/unlocked. When the system is locked, none of the user's eye movements will trigger any response from iTrack, besides looking up (opening and closing help menu) and left wink (unlocking the system). If the system is not locked and detects a left or right movement, it will switch the current mode. While in helpMenuOpen if the system is not locked and detects a right wink, it will change state to customSettingsOn and open the custom settings window. In this state, the user can create new modes or add new commands. If the system detects a right wink in this state, it will close the custom settings window and go back to the help menu.

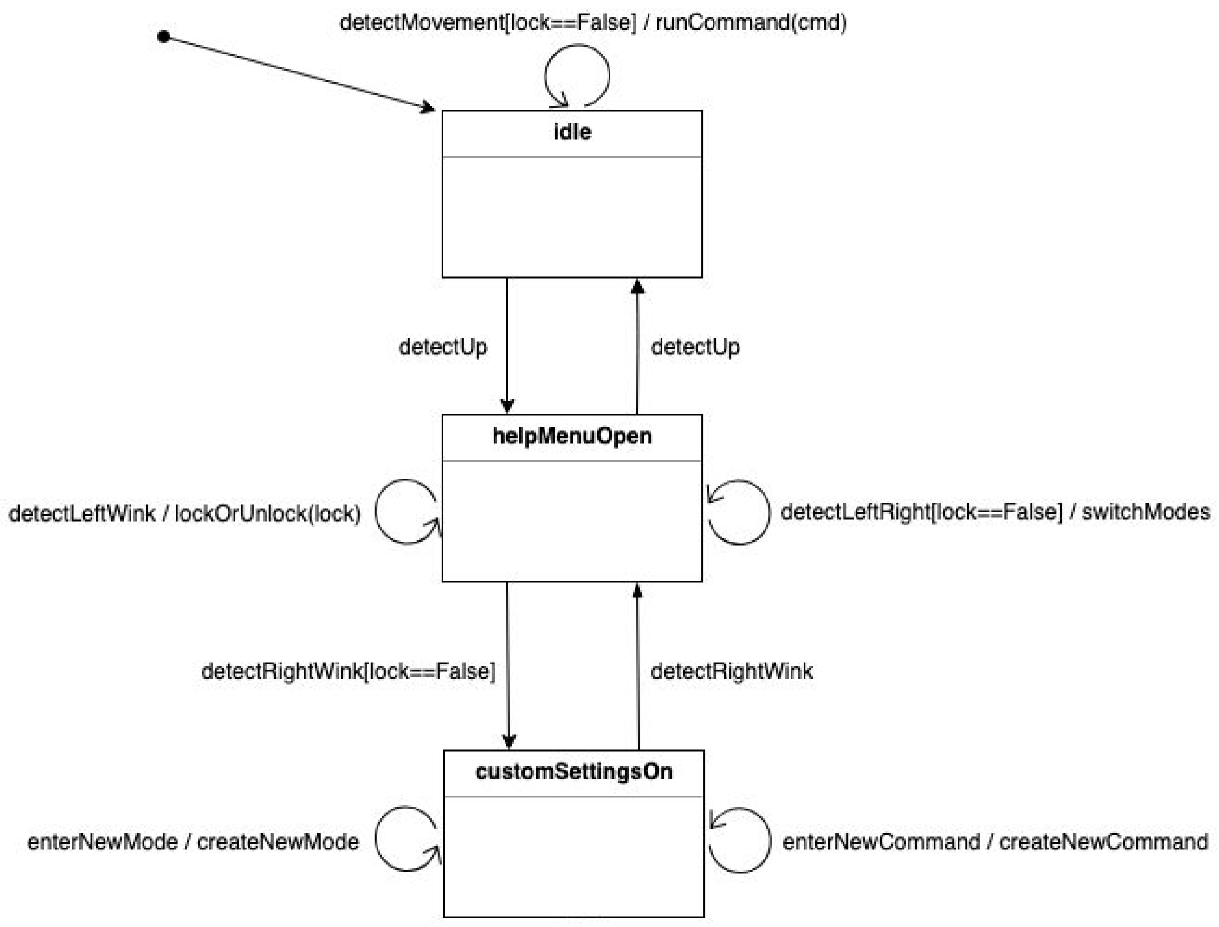

detectMovement[lock==False] / runCommand(cmd)
idle
detectUp
detectUp
helpMenuOpen
detectLeftWink / lockOrUnlock(lock)
detectLeftRight[lock==False] / switchModes
detectRightWink[lock==False]
detectRightWink
customSettingsOn
enterNewMode / createNewMode
enterNewCommand / createNewCommand

Figure 2: Statechart Diagram

Class Level Design

Info

Name	Description
Mode	This class is a container that contains mappings from the four core eye movements to four commands.

Statechart Diagram

Figure 3 shows a zoomed in statechart diagram of our Mode class. Here, the outer states all represent built-in default modes within the system. Every mode has corresponding commands that will occur if a eye movement is detected while not in the help menu. Switching between modes is possible by looking left or right while in the menu. The center state represents our custom mode editor, where new modes and new commands can be created.

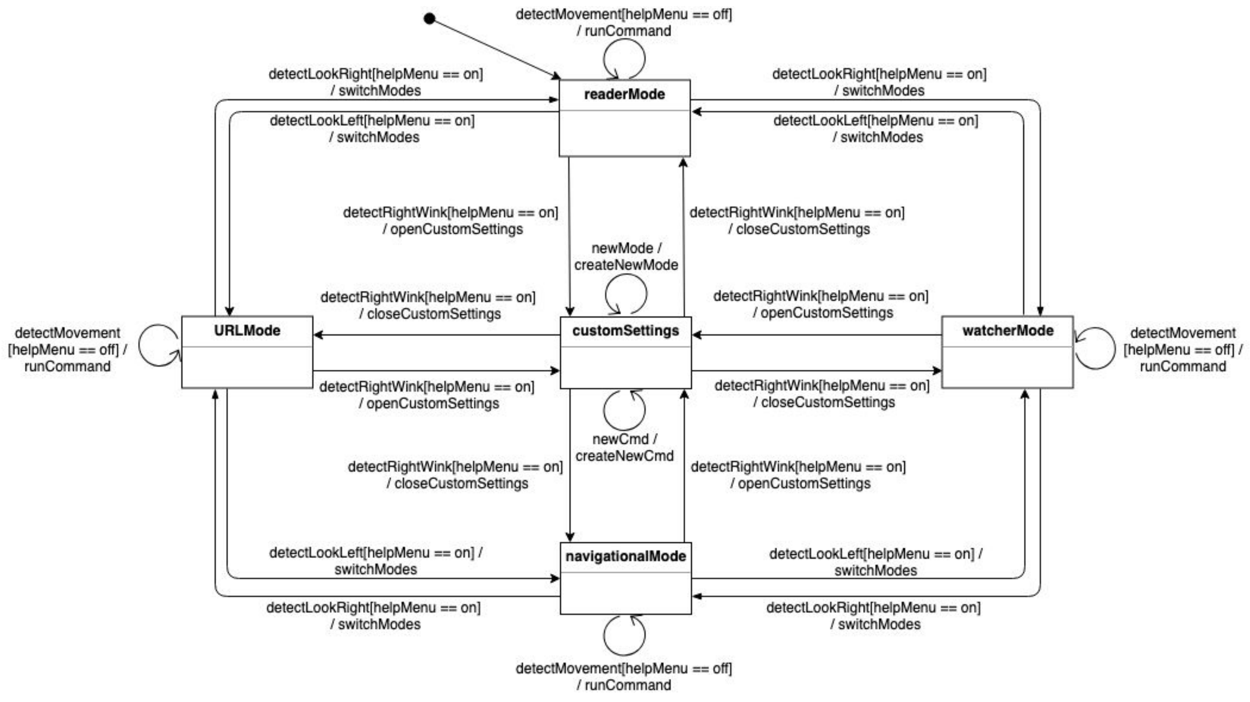

detectMovement[helpMenu == off] / runCommand
readerMode
URLMode
customSettings
watcherMode
navigationalMode
detectLookRight[helpMenu == on] / switchModes
detectLookLeft[helpMenu == on] / switchModes
detectRightWink[helpMenu == on] / openCustomSettings
detectRightWink[helpMenu == on] / closeCustomSettings
newMode / createNewMode
newCmd / createNewCmd
detectMovement [helpMenu == off] / runCommand

Figure 3: Mode Statechart Diagram

Attributes

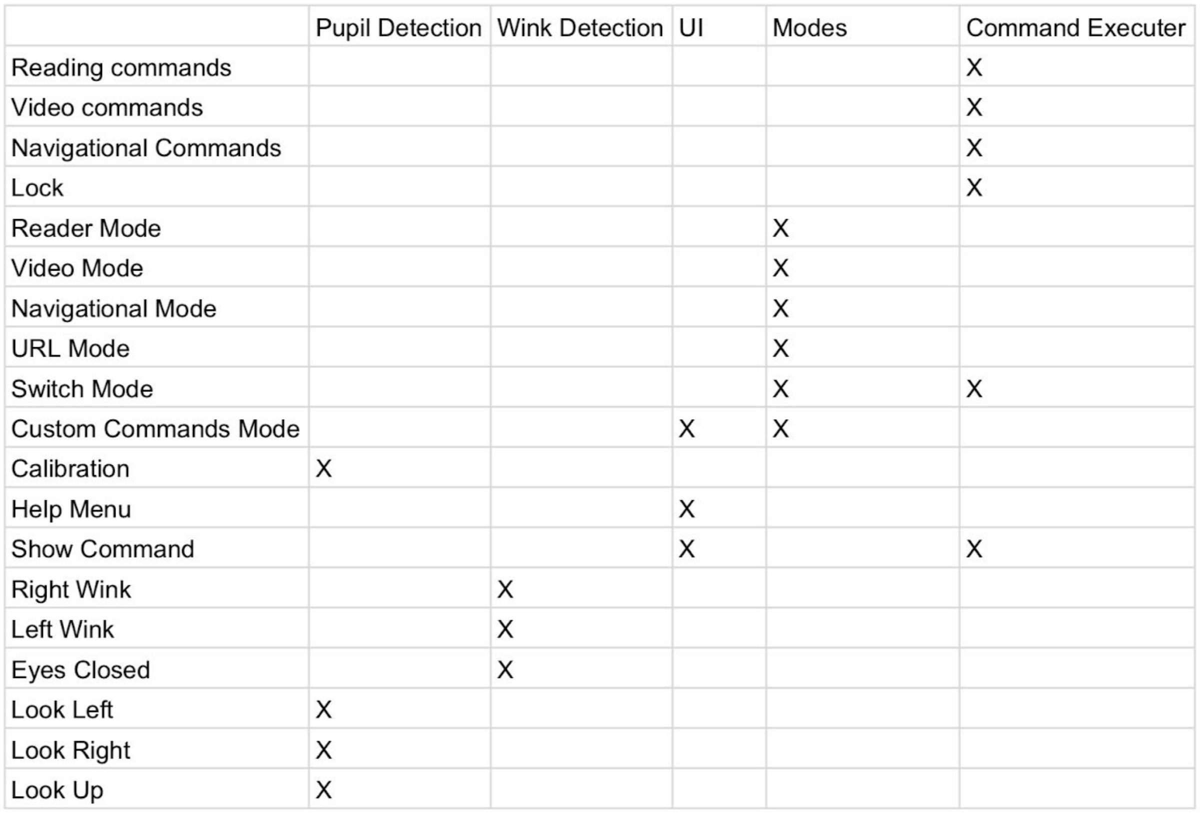
Attribute	Name	Description	Type
Mode Name	mode_name	Name of the mode.	string
Commands	commands	Dictionary of movements to commands for the mode. Each eye movement is mapped to a certain command.	dict
Command Function	command_func	The function to execute when a certain eye movement is detected.	function
Command Name	command_name	Name of the command.	string
Command Args	command_func_arg s	Tuple of arguments to input into the command function.	tuple

Operations

Operation	Name	Description	Output	Input
Set Command	set_command	When creating a new mode or editing an existing mode, set_command maps the input movement to a new command.	N/A	eye_movement_type, command_func, command_name, command_func_args
Execute Command	execute	Takes in an eye movement and executes the corresponding command.	N/A	eye_movement_type

Traceability Matrix

An enumeration of our requirements cross-reference can be seen in Figure 4. The Command Executor Class, not surprisingly, is responsible for executing all commands on the computer. All types of commands and the locking functionality are incorporated in the Command Executor. The Mode class is responsible for remembering, switching, and creating modes. The UI class is responsible for displaying information to the user, this includes the custom mode creation, help window, and last executed command windows. Our wink detection class handles detection of right/left winks as well as detecting closed eyes. Finally, our Pupil Detection class handles tracking the user's eyes and detecting if they look left, right, or up.



	Pupil Detection	Wink Detection	UI	Modes	Command Executer
Reading commands					X
Video commands					X
Navigational Commands					X
Lock					X
Reader Mode				X	
Video Mode				X	
Navigational Mode				X	
URL Mode				X	
Switch Mode				X	X
Custom Commands Mode			X	X	
Calibration	X				
Help Menu			X		
Show Command			X		X
Right Wink		X			
Left Wink		X			
Eyes Closed		X			
Look Left	X				
Look Right	X				
Look Up	X				

Figure 4: Traceability Matrix

Design Alternatives and Rationale

Eye Tracker Hardware

Our first alternative that we considered was using a designated eye tracking device, such as a Tobii Eye Tracker 4C [1]. Eye tracking devices are extremely accurate, since they use a combination of infrared sensors and cameras. This will allow us to recognize more complicated eye movements. However, the eye trackers are very expensive and not a regularly available item. This would mean that the user would have to buy an eye tracker to use our product, which could make iTrack inaccessible to many people.

Mouse Control

Our second alternative design was enabling users to control their mouse with their eyes, similar to what is provided by software like WebGazer.js [2]. Being able to accurately control the computer mouse with only eyes would enable our users to more easily interact with the computer. However, we decided we wanted to use webcams and not designated eye trackers, due to reasons mentioned above. The less accurate eye tracking with webcams means that the mapping from eye positions to mouse positions will be fairly inaccurate, making this design infeasible.

Additional Movements

Our final alternative design was incorporating movements other than eye movements, such as facial expressions. This would allow users to map more movements to different commands, which could make our product easier to use. However, we found that the four eye movements were already enough, especially with the modes. Furthermore, it would make it harder for the user to memorize the movements if we introduce too many.

References

1. Tobii Eye Tracker 4C, https://gaming.tobii.com/products/, accessed 31 March 2019.
2. WebGazer.js, https://github.com/brownhci/WebGazer, accessed 31 March 2019.