UNIVERSITY OF CALIFORNIA, SAN DIEGO

EYEHOME 3.0 A Multimodal Input System with Adaptive User Interfaces for the Severely Motor Impaired

A Thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Angie Ngoc Anh Nguyen

Committee in charge:

Professor Nadir Weibel, Chair
Professor William Griswold
Professor Scott Klemmer

2017

The thesis of Angie Ngoc Anh Nguyen is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2017

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Many thanks to Amado and Cinthya for their help and all of their hard work, and the Moxie foundation for providing funding for the equipment needed. I would also like to thank Dr. Nadir Weibel for the opportunity to work on this project and for all of the guidance and advice he has given me throughout the process of building this system. Many thanks also to Mary Perkins for her work in developing the machine learning back-end for the Eyehome 3.0 system, and to Wesley Chan and Colleen Emmenegger for their work in conducting data collection. Finally, I would like to thank Ruchika Shivaswamy, who contributed to the devices and machine learning back-ends for Eyehome 2.0, which eventually evolved into Eyehome 3.0.

ABSTRACT OF THE THESIS

EYEHOME 3.0 A Multimodal Input System with Adaptive User Interfaces for the Severely Motor Impaired

by

Angie Ngoc Anh Nguyen

Master of Science in Computer Science

University of California, San Diego, 2017

Professor Nadir Weibel, Chair

Individuals affected by severe motor impairment lose the ability to move and interact effectively. The severity of motor impairment varies across affected individuals, and conditions like LIS and ALS can fluctuate and deteriorate over time, sometimes within the same day. Despite the increasing availability of augmentative and assistive technologies, existing solutions do not easily adapt to these changing capabilities. In this paper, we present Eyehome 3.0, an extensible multimodal input system which builds user interfaces that adapt to the user via recurrent neural networks. In particular, eye gaze data is used to inform the layout of these user interfaces. We also present a framework for

easily developing additional layouts and applications for the system, and how additional devices can be incorporated. We discuss how our participatory design work with Amado, a person with LIS, helped drive the development of the system.

Chapter 1

Introduction

Severe motor disorders like Amyotrophic Lateral Sclerosis (ALS) or Locked-In Syndrome (LIS) can cause affected individuals to lose the ability to move or interact with their environment effectively. LIS is the most severe case in which a person with LIS is cognizant, but has total paralysis of almost all voluntary muscles. Typically, a person with LIS is still able to retain some eye movement, but due to their inability to move, most are completely dependent on their caretakers [15]. This can be mentally and emotionally taxing on the individual, with many developing depression [14].

For those with LIS, interacting with their environment and with others becomes more difficult. However, if a person with LIS has reasonably good oculomotor abilities, caretakers will oftentimes create their own system of communication with the individual through eye gestures like blinking [18]. As most individuals with severe motor impairments are unable to interact with a computer as an able-bodied person would, many look to assistive technologies and in particular, Augmentative and Alternative Communication (AAC) devices and systems. These devices and systems often have user interfaces designed to help those with physical disabilities and are used to supplement communication for those with speech impairments [4][17].

Even though research in accessible computing has been around for more than twenty years, challenges still remain for creating a user interface for people with disabilities because severity in motor impairments can vary. For some people with LIS, using their eyes is their sole mode of expression. However, even oculomotor ability can fluctuate and deteriorate over time, sometimes within the same day. This fluctuation makes it difficult to have one system or user interface design that works for every individual at all times. For example, if a user becomes more exhausted throughout the day, the user interface cannot adapt into one that requires less effort to use. Current AAC systems would require a manual reconfiguration to adjust for changing levels of ability, which shows an important and key limitation in current AAC systems, especially with respect to the work presented in this paper. It is especially important that the system conform to the user rather than the other way around, highlighting the idea of ability-based design and adaptive user interfaces [8].

This paper builds on our prior work with previous user interfaces and systems designed for people with severe motor impairment. We first discuss related work and their limitations. Next, the paper will introduce Eyehome 3.0, giving a brief overview of how the system works. Then it will introduce the system as a whole in more detail in terms of interaction, features, architecture, and implementation. We will then discuss the participatory design approach we used while working with Amado, a person with LIS, in order to drive the development of Eyehome 3.0. Finally, we conclude by discussing limitations and future work.

1.1 Related Work

There are a variety of Augmentative and Alternative Communication (AAC) devices available to help individuals affected by motor impairments. Tobii, a company

that develops and sells eye-tracking and eye control products, produces the AAC device Tobii Dynavox[1]. While the Dynavox incorporates eye-tracking and a grid-like layout for users, the customization and setup process is difficult, and the user interface does not adapt to the user automatically. Tobii also produces the Tobii Eye Tracker 4C[2], a standalone eye-tracking device that is more affordable. Retailing at $149, the 4C has a more accessible price point, and proves to be just as good as high-end eye-tracking devices for many users [18].

Another line of work recently investigated creating adaptive eye-tracking by using neural-network-based user profiles to reduce jitter and improve accuracy for people with motor impairments. While the eye-tracking accuracy is improved, the limitation for this work does not account for people who only have certain oculomotor ranges, like vertical movement only [2].

Moreover, while there exists work in creating adaptive user interfaces, few have implemented adaptive user interfaces for individuals with severe motor impairments. Many rely on user profiles or customization processes that require outside assistance. An example of such is Gazespeaker[3], a free software with a grid-like user interface that is integrated with eye-tracking. The grids contain images, and the software also let the users modify and create new grids. However, like Dynavox, Gazespeaker does not adapt the user interface to the user automatically.

Finally, an example of an adaptive user interface system is SUPPLE, which can generate user interfaces, but only after the user goes through two phases of questionnaires where the user had to choose their preferences [10]. However, SUPPLE is not geared towards eye-tracking for severely motor impaired individuals and its intended user typically still has enough motor control to use a mouse or trackball. Additionally, the

[1] https://www.tobiidynavox.com
[2] https://tobiigaming.com/eye-tracker-4c
[3] https://www.gazespeaker.com

user interface of SUPPLE does not change as user capabilities change.

Chapter 2

Eyehome 3.0

Our main goal with Eyehome 3.0 was to create a general interactive system for all degrees of capabilities. The key feature of Eyehome 3.0 is the adaptive user interface that learns about the user through multimodal input. By first calibrating itself to the user, it assigns the user a pre-built layout. With further usage, the system continually adapts to the user, refining the layout by manipulating the layout components.

Eyehome 3.0 also includes a development framework to promote extensibility that makes it easy for developers to create applications and layouts specific to Eyehome 3.0. Additionally, more devices can be added to the system, like a microphone for detecting voice commands, to enhance the capabilities of the user.

The current Eyehome 3.0 system runs on a Microsoft Windows device and is integrated with eye gaze from the Tobii Eye Tracker 4C. For convenience, it also supports mouse input and keyboard input, which can be used as an additional modality of interaction by the caregivers. We chose to use this eye-tracking device because we determined that it was accurate enough for Eyehome 3.0 and due to its relatively low cost compared to high-end eye trackers [16].

To evaluate and test our system, we have implemented four pre-built applications,

such as a keyboard application and a text-to-speech voice application. We have also implemented two basic layouts: a four-sides layout and a eight-grid layout. Figure 2.1 shows the Voice application in both layouts.

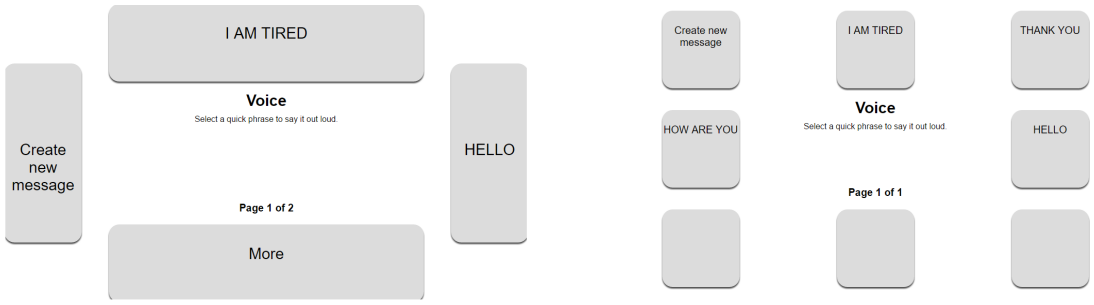

Figure 2.1: The Voice application displayed with the four-sides layout (left) and the eight-grid layout (right).

To give a better overview of Eyehome 3.0, in the next section, we discuss the evolution of the software system from its predecessors. We also detail the system interaction to show how a user would use Eyehome 3.0, as well as the key features of discovery, adaptability, and extensibility. We provide a description of the system architecture and implementation, and finally conclude with our evaluation of the system.

2.1 Predecessors

The development of Eyehome 3.0 was iterative and was based on previous efforts to implement an eye-tracking interface for locked-in patients. The previous versions, namely OcuHub, Eyehome, and Eyehome 2.0, were all developed with the same stakeholders in mind.

OcuHub was developed as part of a CSE 218 class and served as a prototype of an application home screen which used eye-tracking as its main input [3]. It had two modes: a vertical mode, and a radial mode. The vertical mode displayed application buttons vertically. The radial mode showed the application buttons in a circle around a

central button.

OcuHub then became Eyehome 1.0 once we were able to work with Bob, a person with LIS, and collect data on his eye gaze capabilities. The two layouts in OcuHub proved to be too difficult, and we developed a new mode where the layout had a center button and four buttons aligned to the sides of the screen. We also incorporated blinking as a mode for interacting with Eyehome 1.0.

However, we found that Eyehome 1.0 was integrating too many different software components, causing installation to become hard and creating a large learning curve for development. We developed Eyehome 2.0, a cleaned-up version of the Eyehome 1.0 system in C#, HTML, CSS, and JavaScript. With the new foundation, we were able to create an application framework that was easy to develop with. It has since evolved, finally introducing extensibility and adaptive user interfaces with Eyehome 3.0.

2.2 System Interaction

The user interacts with Eyehome 3.0 primarily through eye gaze and fixation so that the user interface is optimized for eye gaze. For convenience, especially for family and caregivers, users can also interact with the system through the keyboard and mouse. In this section, we give a brief, high-level, overview of how a user would interact with the Eyehome 3.0 system.

The user first goes through a short calibration phase in which a user follows a sequence of boxes on the screen with their eyes. Afterwards, the user is presented with a pre-defined layout consisting of buttons and a view panel that was determined to be the best fit during the calibration. This layout is also associated with a trained network to predict user interface interactions like button selection. We will describe how the system determines the best fit layout and the trained network more in depth in the following

sections.

The user can make a button selection by fixating their eyes on the button they would like to select. This eye-tracking data is sent to the machine learning back-end every second and fed as input to a user's model. This machine learning model then sends back its interpretation of the eye-tracking data to the front-end where the UI is updated. During this fixation, the button will pulse in order to give the user visual feedback. Once the system has determined that a user wants to make a button selection, the application performs the action associated with that button. For instance, fixating on a button labelled "HELLO" in Eyehome 3.0's voice application would make the computer say "hello." If this action happens to be a page change such that the content of the view panel or the buttons change, a period of rest time, will execute in which all buttons are locked so that the user is able to scan the new button content without selecting them. Locking the buttons is meant to prevent the "Midas touch", a problem that occurs when the system mistakes the user's intention to scan a button rather than select it, or vice-versa [11].

As the user continues to use the system, the system uses machine learning to adapt its behavior to the user by refining the basic layout and manipulating different aspects of it. When user capabilities change enough that the current layout is no longer optimal for the user, the system will trigger a recalibration, which aims to find a different basic layout that better fits the user, and starts refining the new layout instead.

2.3 Features

There are three main features that are unique in Eyehome 3.0: Discovery, Adaptability and Extensibility.

2.3.1 Discovery of User Capabilities

Because capabilities vary across individuals, calibration is important to discover the capabilities of the specific user at the specific time. Continuously understanding these capabilities informs the system's initial user interface settings. To meet this need, we created a calibration application with the goal of discovering the user's eye gaze capabilities. We focused on using this calibration application to find the optimal layout, rest time, and intent or selection time for the user.

On first-time use of the calibration application, each new user is assigned to a unique model in the machine learning back-end. Calibrating the user on each button for each layout informs the front-end about which of the available layouts is the best fit for the user. To this extent, the calibration application displays each button for each layout one at a time at random for a random duration time, with a random rest period in between buttons. In other words, a button will appear on the screen for a random duration time, then disappear for a random rest time, and then reappear on a different part of the screen. Eye-tracking data is stored as (x, y) coordinates during this calibration period and the resulting layout is selected.

To decide which layout is the best for the user, the data from the calibration application is sent to the machine learning back-end. The data is used to create multiple training set patterns each corresponding to a specific layout. During the training phase, the neural network finds the optimal parameters (i.e. weights and biases) by iterating through the calibration data and moving the parameters closer in the direction that reduces error on the training set. The trained networks are then saved in the shared database along with their optimized parameters for later use. Next, each trained network is given the calibration data. The network performs button predictions based off of the gaze data of the calibration, and checks this with the actual displayed buttons. The number of correct predictions is then divided by the number of total buttons displayed, and the quotient

indicates how accurate the network's output actually is. The layout associated to the network which performed best, or the one with the highest accuracy, is finally sent to the front-end to display to the user.

The machine learning back-end also parses the calibration data to find if certain buttons were never reached by the user. In those cases, the machine learning back-end will inform the front-end that these buttons should be removed from the layout since it is assumed that the user cannot reach them.

We found that it was also useful to also discover optimal intent timing and optimal rest timing. Intent timing is how long a user must fixate, or dwell, on a button before the button is selected, while rest timing is the time after a button is selected in which all the buttons are locked. In the calibration application, the time in-between each button appearance when no button appears, is randomized between two to four seconds. The length of time that a button is displayed to the user is also randomized between two and four seconds. Randomizing the length of time informs the system of the best rest time and the best timing for selection. We chose to start this range at two seconds because previous research has found that able-bodied people performed more accurately when dwell time was set to approximately one second, which we thought might be too fast for Amado [1]. We then chose to limit the range at four seconds because it is faster than Amado's Dynavox, which he said was "slow" [18].

Once the layout, rest time, and selection time has been determined to be a good fit, the user is taken to the Eyehome 3.0 home page which will present all of the available applications, each of them displayed using the determined optimal user interface settings.

2.3.2 Adaptive User Interface

After the calibration process, Eyehome 3.0 has adapted to the user by using the user's trained model to predict button selection intent, making the user interface more

effective. The system has also adapted to the user by assigning a base, or starting, layout. In addition, as the user continues to use the system, the system continues to automatically adapt the user interface without the need for a frustrating manual customization process [18]. This is achieved via the machine learning back-end, which keeps track of unusual eye gaze to button offsets to inform the layout of necessary adjustments, or refinements.

To optimally account for the needs of our users (see later section on participatory design), we broke down our adaptive user interface layout into two main components for simplicity: buttons and a view panel (see Figure 2.2). Buttons contain text and have actions associated with them, which the user interacts with using selection. The view panel is the current display that the user can read. When using eye-tracking as an input, the view panel also serves as a place for the user to rest in order to reduce strain. With having the simplicity of only two different components and the freedom to choose placements, sizes, and number of buttons, layouts are open-ended and able to evolve. Different basic layouts can be created for Eyehome 3.0 which can contain any number of buttons. The adaptive calibration model can then determine the sizes and locations of the buttons and the view panel interactively and depending on the current user's conditions.

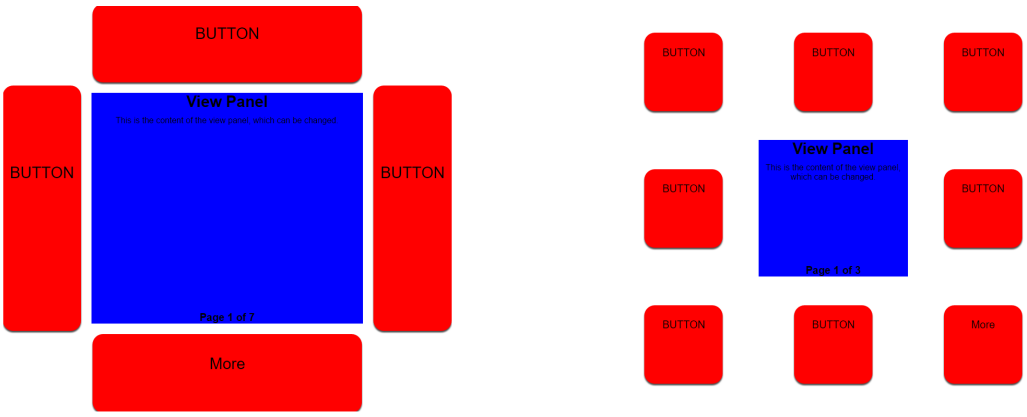

Figure 2.2: Two pre-defined layouts that can be presented to the user with buttons (in red) and the main view panel (blue). Left: the placement for the four-sides layout; Right: the eight-grid layout.

We have built two basic layouts for Eyehome 3.0: the four-sides layout and the

eight-grid layout. The four-sides layout consists of a view panel in the center and four rectangular buttons along the sides of the screen as detailed in Figure 2.2 on the left. We chose to keep this layout, which we first implemented in Eyehome 1.0, because of previous research and positive response to the layout by another research participant [3]. Additionally, the four-sides layout does not have corner buttons, an interaction that has been shown to be difficult for people with limited eye gaze capabilities [9]. As the system continues to learn about the user, Eyehome 3.0 can change the layout by removing the buttons that are proving difficult to reach or changing the sizes of the buttons. In case the user shows improvement, the system might choose to adjust to the eight-grid layout, which is more complex than the four-sides layout and uses more of the screen real-estate for more buttons, as seen in Figure 2.2 on the right. Like the four-sides layout, the eight-grid layout can be further refined over time.

As explained in the previous ”Discovery” section, the adaptive aspect of the layout begins by automatically choosing the basic layout with respect to the highest calibration accuracy for the user, and then refining this layout. Once a layout is chosen and the associated model selected, user interaction predictions are made by the system. This serves to make the user experience more efficient regardless of the user's accuracy. As the gaze data is continuously stored, the machine learning back-end can find any peculiar offsets that can trigger refinements for improvements in accuracy and usage efficiency. This process of refinement allows the basic layout to be adjusted over time with layout manipulations like removing buttons or resizing buttons.

For Eyehome 3.0, button sizes are determined by the basic layout, which should set a small, medium, and large-sized button. On first use of the layout, the medium-sized button is displayed, and our refinement process can choose to increase or decrease the button size. When the button size limits are exhausted, the machine learning back-end can choose to trigger recalibration using the calibration application to switch to an entirely

different layout. All layout changes are triggered by events sent from the machine learning back-end over the websocket as it learns about the user, allowing layouts to be dynamic in real time.

2.3.3 Extensibility

Eyehome 3.0 has evolved significantly from its predecessors in that we have focused on building a system that allows for easy extensibility in three different directions: adding devices, creating applications, and creating layouts.

Devices

In preparation for this work, we have worked with other people with LIS and found residual abilities that could be used to interact with the system [18]. Because there are many different degrees and variations of motor disabilities, it is important to consider more input devices than just the mouse, keyboard, and eye tracker. Integrating more devices allows the system to better resolve ambiguity in selection such as the issue created by the ”Midas Touch”. Thus, in our system architecture (see Figure 2.3) we tried to decouple the integration of devices as much as a possible. The goal is to avoid a ripple effect of changes to the rest of the code base when new tracking devices are added. By allowing easy extensibility of devices, more device options become available, in turn, allowing for a wider range of options given the user's current ability [12].

Applications

Eyehome 3.0 provides a framework for developers to easily create new applications by simply programming the content of what should appear in the view panel and the content and actions of the buttons. This gives developers the freedom to create applications without having to additionally write the same application multiple times to fit

each of the different layouts. The applications are also added automatically to the home page of Eyehome 3.0, where all the other applications are. In unburdening application developers, we hope that Eyehome 3.0 can grow and host a wider variety of applications. Additionally, there is potential for an ”app-store model” similar to the Android Play Store[1] where Eyehome 3.0 users can explore and install a variety of applications that are built for the system by third-party developers.

Layouts

The framework built into Eyehome 3.0 also allows developers to easily create new basic layouts for which new models are trained. The developer simply creates a number of buttons with their locations and sizes, and the location and size of the view panel. Once a new layout is created or changed, the system automatically stores each button position and size for the calibration application, and a model is created for the layout. Loose coupling of the applications to the layouts also allows for layout code changes to happen without having to change every single Eyehome 3.0 application.

2.4 Architecture and Implementation

Eyehome 3.0 is a more extensible and adaptive system than its predecessors, and we have achieved this by building a very loosely coupled architecture. It is implemented using JavaScript (AngularJS and node.js), HTML, CSS, Python, and C#. The system also uses a document-oriented database with JSON-like documents. The front-end portion of Eyehome 3.0 is built as a web application, while the back-end computation is handled on a separate server, meaning Eyehome could also perform as a cloud computing application. This also allows the front-end client to run on any device that can support

[1] `https://play.google.com/store/apps`

local eye-tracking like the Tobii Eye Tracker 4C.

Since Eyehome 3.0 is a system that is built in an iterative way, we selected technologies with the idea of longevity in mind for future development. We also designed an architecture of loosely coupled components to follow separation of concerns for better software maintainability. The architecture is divided into three components: (1) A web application front-end, (2) a local back-end component for processing device input, and (3) a back-end for machine learning computation on a separate server. Each component communicates in an event-driven way via websockets. Additionally, the three components can read from and write to a shared database. Figure 2.3 shows an architecture diagram of the Eyehome 3.0 system.

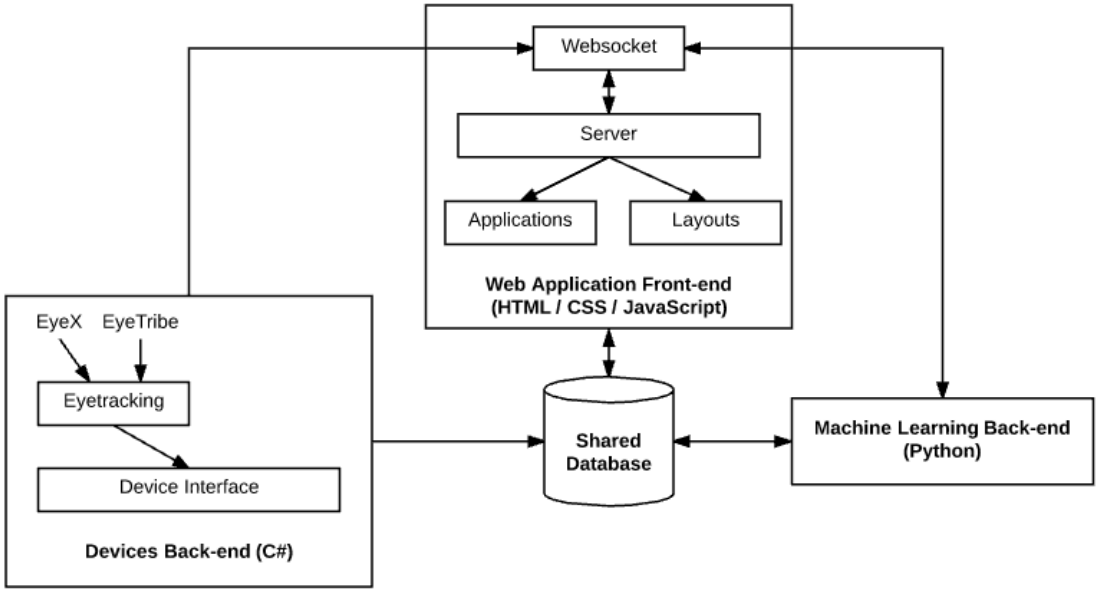

Figure 2.3: The architecture diagram for Eyehome 3.0, depicting the three components of the system. The devices back-end is local to the machine where Eyehome 3.0 is running, while the machine learning back-end can run on a separate server.

2.4.1 Inter-system Communication

While the separation of the three components of Eyehome 3.0 leads to loose coupling, there is still a need for inter-system communication. The components communicate via the websocket server using the Socket.IO framework, an open-source framework that has drivers for .NET, Node.JS, and Python. Communication using websockets allows for real-time event triggering, such as sending real-time selection predictions from the machine learning back-end to the front-end.

2.4.2 Shared Database

Each of the three components are able to write to and read from the shared MongoDB, a NoSQL database. Documents in this database are fetched and stored in a JSON format. We chose to keep this shared database on top of the websocket communication because of the persistence of the data and for processing of data over time.

2.4.3 Web Application Front-End

The front-end is a web application that follows the MEAN stack, which means that it uses MongoDB, ExpressJS, AngularJS, and NodeJS. This web application also acts as the server for the websocket engine, Socket.IO, which is used to communicate between the three components of Eyehome 3.0. We chose to use these languages and frameworks because of current trends in many web and cloud applications providing RESTful APIs to their services, such as Twitter and IFTTT.

The placement of the buttons and the view panel is done separately from the applications themselves, meaning layouts disregard any specific application and vice versa. To create a new basic layout with the framework, layout developers must follow a

set JavaScript and/or jQuery template. A layout developer must implement methods such as `getSmall()`, `getMedium()`, `getLarge()`, which return pixel values for the desired button sizes, and `placeButtons()` and `placePanel()`, which programmatically places the buttons and view panel for the layout. These button and view panel sizes and pixel locations are stored into the shared database programmatically to be used in the machine learning back-end.

The layout developer must also implement `loadButtonContent()`, which loads the application button content onto the layout buttons themselves. This method can become complex as layouts must determine how to load the button content onto the buttons. For example, the text-to-speech Voice application (see Figure 2.1) has more than the four buttons of the four-sides layout. In this case, the method `loadButtonContent()` is written so that it will automatically move buttons to the next page, and dynamically create a ”More” button and additional button pages. On the other hand, the eight-grid layout has more buttons than the Voice application would need. In this case, there will be buttons left empty when displaying the Voice application with the eight-grid layout. The application developer can chose to hide these buttons if they are not needed.

For the Eyehome 3.0 applications to exist separately from the basic layouts, all applications are written as Angular controllers attached to HTML templates that follow a specific format. These controllers must extend a baseController which already handles websocket communication, initialization of the current layout, and general application interactions. In these controllers, developers are able to manipulate the markup that is attached to the view panel, making the application’s view panel content changeable. To initialize the application buttons, developers must implement `initButtonContent()`, which creates an array of mapped button IDs to button content and the corresponding method to be called on button click. The end result will look like the following JSON definition.

```
[{
    "id": 1,
    "label": "Previous Page",
    "onClick": "previousClicked()"
}, {
    "id": 2,
    "label": "Next Page",
    "onClick": "nextClicked()"
}]
```

With this mapping, application developers only need to implement the button's corresponding method, e.g. `previousClicked()`, within the controller. Additionally, because layouts place buttons using their ID regardless of the button content, this mapping is important for button selection predictions from the machine learning back-end. The machine learning back-end attempts to determine which button ID the user is trying to select based on the current layout configuration, sends that ID to the front-end to make the call of the function the button ID is mapped to.

2.4.4 Machine Learning Back-End

The machine learning back-end lives on a separate host server from the front-end and the devices back-end, which both run on the user's local machine. We chose to separate this component because of the potential required processing power needed, which we did not want to necessarily account for on the user local machine.

We chose to write the machine learning back-end in Python due to the extensive amount of machine learning libraries that are available for this language. The machine learning back-end parses data from the shared database after pre-processing in which data from the devices is paired with the data from the front-end by timestamps.

The machine learning back-end implements a recurrent neural network (RNN) because RNNs are capable of modelling sequential and time dependencies [6]. The RNN

is built and run using TensorFlow[2], an open-source software library for machine learning, which has an API for Python. The data parsing is done using Pandas[3] and NumPy[4], which are both data analysis and computation packages in Python.[5]

Additionally, the Socket.IO package for Python is also used for message passing and event triggering. Depending on the triggered event, the network is used in different ways. For example, during calibration, the model trains each copy of the model separately in a user's unique RNN model and determines best layout configurations. During regular Eyehome 3.0 use, the RNN model is used to make intent predictions.

2.4.5 Devices Back-End

The devices back-end uses .NET and C#, because most devices, like the Tobii Eye Tracker 4C, have a .NET SDK. This eliminates the need for creating additional software or adapters for managing programming language incompatibility when trying to incorporate new devices. The devices back-end also connects to the Socket.IO server and the shared MongoDB in order to log device data.

In order to be extensible for more devices, the devices back-end follows the Factory Pattern, where each newly introduced device must implement the Device interface, including the methods `startUp()`, which starts the device, and `write()`, which will write the data from the devices along with a timestamp, deviceName, and deviceType [5]. Because of the JSON-like documents in the shared database, any type of data stream can be stored and later processed by the machine learning back-end. To fully incorporate these new devices in adapting user interfaces or improving user interface interactions, changes need to be made to the machine learning back-end to use new device data

[2] `https://www.tensorflow.org`

[3] `http://pandas.pydata.org`

[4] `http://www.numpy.org`

[5] It is outside of the scope of this paper to describe the machine learning algorithms in details. These have been developed in collaboration with another student and are not part of this thesis.

streams being stored. For example, the user selection event sent to the front-end is currently determined by the machine learning's processing of eye gaze fixation. If we were to incorporate another device that should be used to determine selection instead, the machine learning back-end would need to be changed to incorporate this new device and how to use the data from that device for selection.

The current implementation of Eyehome 3.0 supports collection of eye gaze data from the Tobii Eye Tracker 4C as well as the Tobii EyeX. Eye gaze data is stored as (x, y) coordinates at 90 Hz for the Tobii Eye Tracker 4C.

Chapter 3

Testing and Evaluation

We wanted to know what the best user interface would be for a person with severe motor impairments, and also how to discover the optimal starting configurations for this user interface. In addition to answering these research questions, we iteratively worked with Amado, living with LIS, and his wife Cinthya to better inform development of Eyehome 3.0. Throughout our work we followed participatory design practices that allowed us to tailor our development to their needs, while maintaining a general-purpose system in mind [13][18].

Amado is a 56-year-old former CAD drafter who lives with his wife, Cinthya, and their two children. After an intracerebral hemorrhage, Amado was diagnosed with LIS. He has limited head and facial muscle movement. Cinthya is Amado's primary caretaker. Amado currently uses a low-tech AAC system where looking up indicates "Yes", looking down indicates "No", and closing his eyes indicates "I don't know" (Figure 3.1). He also has a Tobii Dynavox, which he finds to be slow [18].

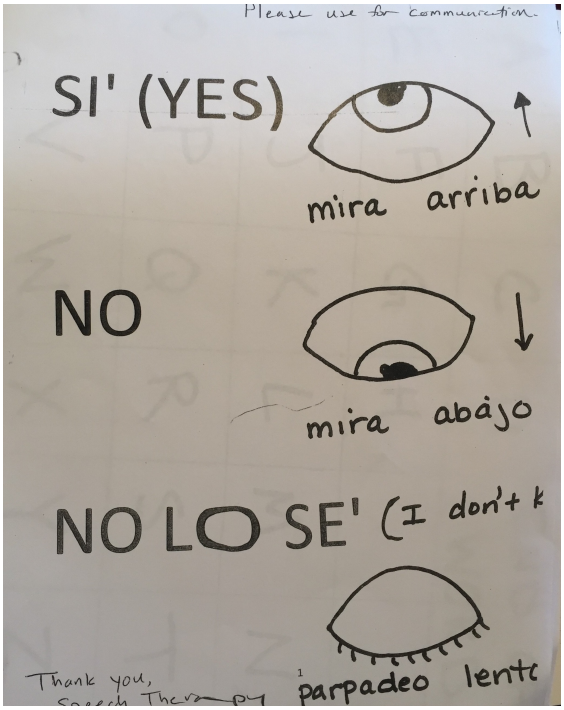

Figure 3.1: Amado's low-tech AAC system, in Spanish. The directions, in order: Look up to say "Yes". Look down to say "No". Close your eyes to say "I don't know".

3.1 Data Collection

We collaborated with Amado and Cinthya primarily to drive the development of the calibration application. We were also able to gain insight on creating basic layouts and button size ranges.

Based on previous data collection setups, we attached a Tobii Eye Tracker 4C to a Microsoft Surface Pro 3 tablet on a gooseneck (see Figure 3.2). This was then attached to a rolling, elevated table, allowing the tablet and eye tracker to be repositioned quickly for Amado's gaze. We had a separate laptop for controlling our data collection application so that we were not reaching over Amado to manage the application and monitor Amado's progress, after starting.

Prior to visiting Amado, we created a manual data collection application that displayed light blue boxes at randomized locations with randomized duration times, on a

Figure 3.2: Setting up the Eyehome 3.0 data collection application on a tablet with an eye tracker, attached to a gooseneck clipped to a side table.

black screen. The intent was to convert this application into the automatic calibration application that would drive discovery and adaptivity of the user interface described above. The person running the data collection application was able to see everything that Amado could see in addition to a control panel and a trail of white dots which mirrored Amado's current eye gaze (see Figure 3.3). These were both invisible on Amado's side to reduce any cognitive burden on him.

We started the data collection application with boxes that aligned to the sides of the screen so that we could determine whether or not Amado could reach all sides of the tablet. Afterwards, we collected data following Amado's eye gaze for three sessions. The first session sampled boxes of size 200 pixels, the second session 150 pixels, and the last 100 pixels. Each of these sessions displayed twenty boxes at completely random locations on the screen, with a random duration time anywhere from two to four seconds, and a constant rest time of two seconds. The randomized duration time was set between two and four seconds. In total, each session was around 20 minutes long.

In between each session, we were also able to ask Amado questions about his

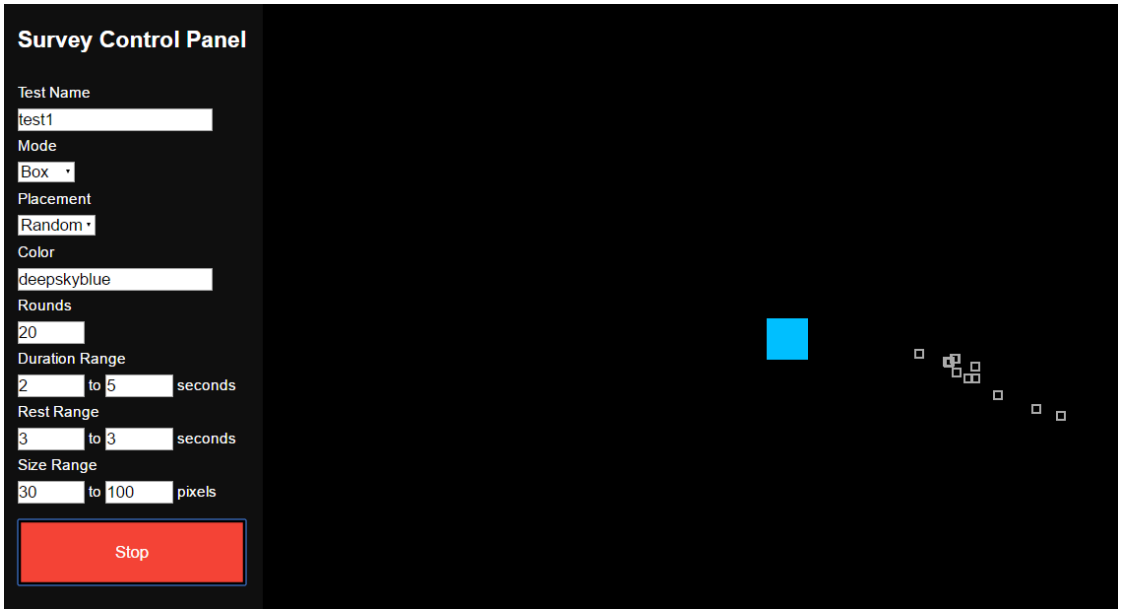

Figure 3.3: The data collection application on the controller side. This is different from what Amado could see on the client side. It has a control panel for changing the settings on the left. The blue box is the current box Amado must fixate on, and the small white dots are Amados current eye gaze x and y coordinates.

preferences with the rest time and box sizes using Amado's current form of communication system of looking up, down, or closing his eyes, for confirmation. This gave us insight into his preferences relative to the results of the data collection application results.

3.2 Data Collection Evaluation

Based on feedback from Amado, he liked the box of size 150 pixels the best out of the three sizes. Additionally, he confirmed that the two second rest period in between each box appearance was enough rest. As he is capable of using the Tobii Dynavox, which has buttons approximately the same size as the 200 pixel boxes, we were able to assess that he could use buttons smaller than what the Dynavox provides.

However, we found that we were randomizing too many parameters during the data collection for us to get good layout evaluations. With completely randomized

locations, only a few boxes that were displayed landed within any of the pre-built layouts. This meant that determining which layout had the highest accuracy was less efficient and the majority of the boxes sampled could not be used for the training sets of each layout.

This resulted in our decision to create a calibration application that sampled gaze on already existing layout buttons but at random instead, keeping the randomization on duration and rest time. This also means that calibration would be faster and more efficient, since we avoid throwing out data. On the other hand, we found that the data collection application was still useful for layout designers to inform the creation of new layouts based on the specific capabilities of the individuals.

Chapter 4

Discussion

Based on our participatory design approach as well as the simplicity of creating new applications and layouts for Eyehome 3.0, the overall prospects for Eyehome 3.0 are encouraging. Additionally, we feel that our research and implementation of the system provided insights on how other adaptive user interfaces can be created, especially in how to incorporate machine learning into creating these adaptive user interfaces.

4.1 Limitations

While we believe that Eyehome 3.0 is a step in the right direction for adaptive user interfaces, we recognize some of the limitations of our work.

Aside from Amado, we have worked with two other people affected with LIS in preparation for this work. We acknowledge that the number of participants with severe motor disabilities that we were able to work with is quite small. This means that our work with Eyehome 3.0 may not necessarily be a solution that works for other people affected by LIS or severe motor disabilities.

When implementing more complex adaptive user interface layouts, we risk confusing the user by violating learnability and predictability of the user interface. In general,

good user interface design practices describe predictability, or the ability for the user to anticipate displays based on previous knowledge or experience [7]. The user interface might be too volatile, giving the illusion of inconsistency and increasing the amount of user mistakes. This is an important point that further research needs to investigate.

Having pre-built layouts may also limit the capabilities of the user, especially when the user can interact with more complex layouts than the ones implemented. In general, Eyehome 3.0 may not be a good solution for more able-bodied people.

Lastly, the expansion of Eyehome 3.0 relies on having other developers to add applications and layouts and integrating more devices. While the process is easy for developers, it is not as simple for those who do not know how to program. In the future we hope to be able to extend the development of Eyehome applications through end-user programming based on simple drag-and-drop paradigm and application development.

Chapter 5

Conclusion

By working with individuals affected by severe motor impairments and studying how they used their current AAC devices and systems, we were able to use eye gaze to inform how user interfaces can be adapted to the user. As a prototype, the system shows that adaptive user interfaces that are extensible are possible.

Individuals with motor impairments experience a loss of effective movement and effective interaction with their surroundings. Additionally, severe motor impairments vary and capabilities can change over time, sometimes within the same day. Current assistive technologies and AAC devices and systems do not support adaptability or have other limitations, like troublesome manual configurations or difficult set ups.

We have built a system that adapts to the user through changing layouts and discovering optimal rest and intent timing. Eyehome 3.0 is a system that is also extensible by being open-ended enough that new layouts, new applications, and new devices could be introduced.

5.1 Future Work

Currently, only a few applications and two basic layouts are integrated in the system. The number of applications and layouts is therefore currently very limited. We are working on developing more applications and more layouts, especially to improve communication for people affected by severe motor disabilities. We are working on incorporating more social media access, as well as IFTTT[1], which will open up a lot more capabilities with its easy-to-use API and vast connections to other software. For example, IFTTT can be used to control home automation devices like light switches. We are also planning to build more complex layouts containing more buttons in different locations.

Additionally, the current Eyehome 3.0 devices being used as input are mouse and keyboard for the caregivers, and the Tobii Eye Tracker 4C for the motor impaired individual. We are working to incorporate other modalities such as the Myo armband[2], a wearable device that detects activity in the forearm muscle. When we last met with Amado, he showed us that he was able to consciously move his left hand, though very slightly. We hope that incorporating this will allow for more accurate intent detection, as well as solving the Midas touch problem in a more effective way, using this interaction for ”selection”. We plan to incorporate a galvanic skin response sensor to detect stress and frustration levels, potentially to alert the system of a potential layout change or a necessary recalibration [19]. We would also like to incorporate a brain computer interface as another mode of input for those affected by severe motor impairments [12].

More work is needed in the development and implementation of the system. Optimizations need to be made, especially in the machine learning back-end so that processing is done faster and more efficiently. Improvements can be made on the

[1] https://ifttt.com
[2] https://www.myo.com

calibration application to reduce the amount of time needed to collect data. Additionally, the current machine learning back-end can only classify eye-tracking data into discrete categories, or buttons. In the future, we would also like to explore the possibility of using machine learning to create new and customized layouts for the user without having to use refined pre-built layouts.

All future work necessitates further involvement from Amado as well as other individuals affected by severe motor disabilities. We will continue to incorporate participatory design in the development of Eyehome 3.0.

5.2 Contributions

My work on this system started at Eyehome 1.0 in creating applications. Since Eyehome 1.0, I have contributed ideas on how the adaptive user interface system should be built. For Eyehome 2.0, I created a variety of applications such as Twitter, eBooks reader, and the keyboard, for the four-sides layout. I also began development on the framework for having extensible applications. Finally, I introduced and implemented the new process of inter-system communication via websockets. With Eyehome 3.0, I created the new architecture, separating the devices layer from the machine learning back-end, while redesigning the devices layer to be extensible. I also introduced and implemented the adaptive UI framework, creating new applications, transitioning the four-sides layout to be adaptive, and implementing the new eight-grid layout.

Dr. Nadir Weibel serves as a project adviser for the EyeHome 3.0 system. Members of the EyeHome team are: Mary Perkins, Wesley Chan, and Colleen Emmenegger. Mary Perkins developed the machine learning back-end for the system. Wesley Chan and Colleen Emmenegger helped with the data collection.

Bibliography

[1] ABDUL MOIZ PENKAR, C. L., AND WEBER, G. Designing for the eye: design parameters for dwell in gaze interaction. In *Proceedings of the 24th Australian Computer-Human Interaction Conference* (2012), ACM, pp. 479–488.

[2] ANAELIS SESIN, MALEK ADJOUADI, M. C. M. A., AND BARRETO, A. Adaptive eye-gaze tracking using neural-network-based user profiles to assist people with motor disability. *Journal of rehabilitation research and development 45*, 6 (2008), 801–818.

[3] ANDERSON, B. Eyehome: An eye-tracking system to assist people with neurological conditions, 2015.

[4] D JEFFERY HIGGINBOTHAM, HOWARD SHANE, S. R., AND CAVES, K. Access to aac: Present, past, and future. *Augmentative and alternative communication 23*, 3 (2007), 243–257.

[5] ERIC FREEMAN, ELISABETH FREEMAN, E. R., BATES, B., AND SIERRA, K. *Head first design patterns*. O'Reilly Media, Inc., 2004.

[6] FUNAHASHI, K., AND NAKAMURA, Y. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks 6*, 6 (1993), 801–806.

[7] GALITZ, W. O. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. John Wiley & Sons, Inc., New York, NY, USA, 2007.

[8] JACOB O. WOBBROCK, SHAUN K. KANE, K. Z. G. S. H., AND FROEHLICH, J. Ability-based design: Concept, principles and examples. *ACM Trans. Access. Comput. 3*, 3 (Apr. 2011), 9:1–9:27.

[9] KOREY L. STADING, A. S. N. Eye-gaze aac access for children with complex communication needs, 2011.

[10] KRZYSZTOF Z. GAJOS, J. O. W., AND WELD, D. S. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of*

the 20th annual ACM symposium on User interface software and technology (2007), ACM, pp. 231–240.

[11] MAJARANTA, P., AND RÄIHÄ, K.-J. Twenty years of eye typing: systems and design issues. In *Proceedings of the 2002 symposium on Eye tracking research & applications* (2002), ACM, pp. 15–22.

[12] MATTHEW ERHART, ANGIE NGUYEN, R. S. B. A. C. E., AND WEIBEL, N. Personalized multimodal computer interfaces for the motor impaired, 2016.

[13] SCHULER, D., AND NAMIOKA, A. *Participatory design: Principles and practices*. CRC Press, 1993.

[14] SMITH, E., AND DELARGY, M. Locked-in syndrome. *BMJ: British Medical Journal 330*, 7488 (2005), 406.

[15] STEVEN LAUREYS, FRÉDÉRIC PELLAS, P. V. E. S. G. C. S. F. P. J. B. M.-E. F. K.-H. P. F. D., ET AL. The locked-in syndrome: what is it like to be conscious but paralyzed and voiceless? *Progress in brain research 150* (2005), 495–611.

[16] SUNE ALSTRUP JOHANSEN, JAVIER SAN AGUSTIN, H. S. J. P. H., AND TALL, M. Low cost vs. high-end eye tracking for usability testing. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems* (2011), ACM, pp. 1177–1182.

[17] WALSH, T. J. Utterance-based systems: organization and design of aac interfaces. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility* (2010), ACM, pp. 327–328.

[18] WESLEY CHAN, BONNIE CHINH, J. T. B. C. E., AND WEIBEL, N. What do you want to say? human-centered design of adaptive and affordable communication for severe motor disabilities. *ACM Transactions on Accessible Computing, Under Review* (2017).

[19] YU SHI, NATALIE RUIZ, R. T. E. C., AND CHEN, F. Galvanic skin response (gsr) as an index of cognitive load. In *CHI'07 extended abstracts on Human factors in computing systems* (2007), ACM, pp. 2651–2656.