TRACE: Eye-Tracking Keyboard Application

Sadaf Amouzegar *
California Institute of Technology
samouzeg@caltech.edu

ABSTRACT

The ability to track human eye gaze is not at all a new invention. Different eye-tracking devices have been available for nearly a decade. This technology has been used not only for research in the fields of psychology and marketing but also to improve existing technology for disabled individuals. In recent years, new portable and affordable eye-tracking technology has become available and has made it possible for users to plug into their own machines. In turn, this has allowed developers to start building creative eye-tracking applications for computers and tablets.

In this paper, I present TRACE, a virtual keyboard application that uses eye input for writing textual messages. TRACE uses the Eye Tribe tracker to detect where on the keyboard interface the user is looking at, and routes the keyboard input associated with that location to the application, which then displays the "typed" message on the screen.

While the motivation and research for this work has been focused on disabled users, TRACE makes gaze-based typing effective and simple enough for able-bodied users to use for everyday writing tasks. Since the Eye Tribe is the smallest and most affordable eye-tracking module in the world, it is possible for such gaze-based technique implemented in this program to be used as a viable alternative for users who choose not to use a keyboard depending on their abilities, tasks, and preferences.

Categories and Subject Descriptors

H.5.2 [**User interfaces**]: Input devices and strategies; I.2.7 [**Natural Language Processing**]: Speech recognition and synthesis; K.8.1 [**Personal Computing**]: Application Packages; I.5.5 [**Pattern Recognition**]: Implementation; D.2.m [**Software Engineering**]: Miscellaneous; I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Eye Tracking, Input Devices, Gaze-enhanced User Interface, Virtual Keyboard, Auto-completion, Speech Recognition, Machine Learning, Neural Network, Hidden Markov Model

1. MOTIVATION

Approximately 5,600 people in the United States are diagnosed with ALS (amyotrophic lateral sclerosis) every year. ALS is a progressive neurodegenerative disease that debilitates the nerve cells in the brain and the spinal cord. It impacts the motor neurons that run from the brain to the spinal cord, and from the spinal cord to the muscles in the body. When an individual's motor neurons are damaged or destroyed, he/she loses control of the muscles entirely and may become paralyzed. ALS gradually robs individuals of their physical abilities including the ability to speak. In recent year, eye-tracking technology has helped those with ALS communicate with others. One way people with ALS are living better lives is through such assistive technology. Eye-tracking technology such as the Tobii devices or Eyegaze Edge help narrow the gap between those who have lost the ability to communicate and the world around them. Even when patients with ALS or with other diseases that impact movement, are unable to speak, they can "communicate" with the outside world using their eye movements. By tracking their eye gaze, some programs using eye-tracking technology can generate speech on a screen or select completed phrases from the screen for communication.

The Eye Tribe is another recently developed eye tracking technology sold specifically to software developers in order for them to incorporate the device into their applications and programs. The device has broken the record for the smallest eye-tracking device in the world. It does not require an external power source and is extremely portable as it also runs with most computers and tablets. The tracker uses a camera and high-resolution infrared LED in order to track the user's eye movement. The camera captures images of the user's pupils and runs them through computer-vision algorithms that determine the user's on-screen gaze coordinates, the location on the screen where the user is looking.

Taking into account my skills as a computer scientist and my desire to help patients with ALS, I sought to implement an eye-tracking keyboard software using the Eye Tribe hard-

[0]The student is currently pursuing a B.S. degree in computer science at the California Institute of Technology. The project was overseen by Professor Kenneth Pickar.

ware module.

2. RESEARCH AND GOALS

The first step prior to the implementation phase was to conduct careful research and interview the important parties in order to determine the needs of the potential users and the fundamental goals the product should achieve. I contacted the Eye Tribe team to find out more about the capabilities and limitations of the hardware module. The module can only be purchased from their online store and is not offered at any other location. The device registers any eye movement as small as the size of a fingertip. The module has not been tested with patients and therefore, there is a probability of having unforeseen issues. However, the tracker should ideally work for anyone with normal eyes. Certain medical conditions such as strabismus, exotropia, or esotropia may limit the capability of the module. Also there have been known performance issues with bifocals, special lenses, or polarized glasses. I interviewed with the Care Service department of the ALS Associations to find out more about ALS and what the needs of the patients would be with respect to using a keyboard. They informed me that most patients use the Tobii devices and complain about the settings they must deal with in order to configure the device. It is difficult for a lot of patients to customize the controls. Of course, the cost of the Tobii devices (priced at thousands of dollars) has been a major concern as well. I interviewed two employees at the Rancho Los Amigos National Rehabilitation Center to discuss in detail what the needs and expectations are, who the product should serve, and what basic functionalities should be implemented by the end of the 10-week period. It was decided the minimum viable product should be a communication application with an on-screen keyboard which allows a person to type out a message using only their eyes. The primary focus should be on keeping the cost of the product as low as possible rather than improving upon the features of existing similar products out there. Once that is achieved, additional advanced features such as auto-completion, speech recognition, media sharing, and customized settings may be implemented.

Thus, according to online research and in-person interviews, the following conclusions were made:

- Most ALS patients have difficulty using a standard keyboard and require electronic communication aids such as an on-screen eye-tracking keyboard software.
- The cost of existing eye tracking technology is very high
- Similar technology does not support the Mac operating system or smart phones.
- Other eye trackers in the market are bulky (software bundled with hardware) and can become outdated very fast.

Thus, the primary focus of my project was to design a product for ALS patients in order to type a message on a computer by tracking eye movements. I aimed to create a portable, OS-independent software program that can be easily modified to work on a smart phone architecture. While developing the product, I made the assumptions that the Eye Tribe tracking module functions well and is not merely a prototype. My program relies on the API provided by the Eye Tribe team. The API is code written to communicate directly with the device. If this code were to change, my program would be rendered useless. However, I implemented my program such that if the "logic" of the Eye Tribe API were to be modified, my software product would continue to function or could be changed to adapt very easily. In order to enhance the application as well as user experience, the following components were integrated into the program:

- Auto-completion: It takes patients a long time to type out each word entirely. Regardless of how advanced the technology is, there is a very fine line between a simple glance at a key versus what is meant to be taken as an input if we reduce this "registration" time. Therefore, a real-time auto-completion component integrated into the keyboard would be very helpful. Of course, the program would learn from the user's previous history and improve on future auto-completion recommendations.
- Adjustable interface: Different users have different skills and needs. Certain adjustments that the keyboard would ideally allow the user to make should include but are not limited to:
  - Changing the size of the keys
  - Changing the layout of the keyboard (ANSI, alphabetically ordered, etc.) For some people with learning, memory, or intellectual disabilities as well as individuals who are simply not familiar with the more common computer keyboard layout, an arrangement where the keys are in alphabetical order makes it easier to find the right key.
  - Customizable recognition algorithm to allow the user to adjust the "time" it takes to distinguish a gaze as an input. Some users may need to look at a key for only a second or two in order to select the key while others may need to gaze at keys for longer periods of time without intending to select them.
- Sharing on Social Networking sites: The users will be able to login with Facebook account information and post the "eye-written" messages to their profiles directly. This feature allows the user to make textual status posts to the popular social networks without having to deal with the websites directly and of course without needing to manually type anything.
- Speech-to-Text: In order add to the hands-free experience, a speech recognition component enables users to dictate a message by gazing at the microphone button and recording their voice.

3. DESIGN APPROACH

3.1 Architecture Overview

The Eye Tribe module requires no external power source and can be connected to any computer via a USB port. Once the tracker software and user interface are installed, the tracker can be placed below the monitor for a quick

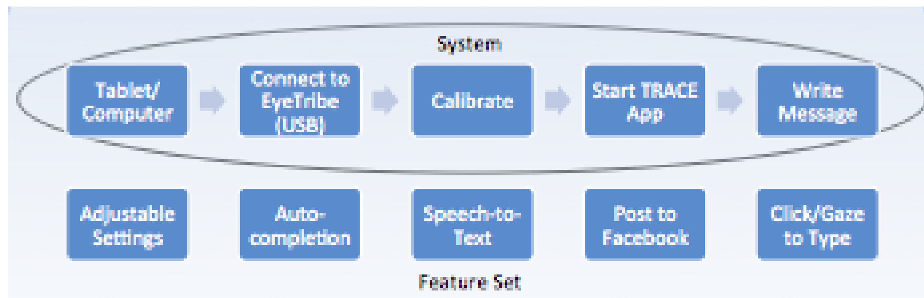

Figure 1: TRACE Application Components

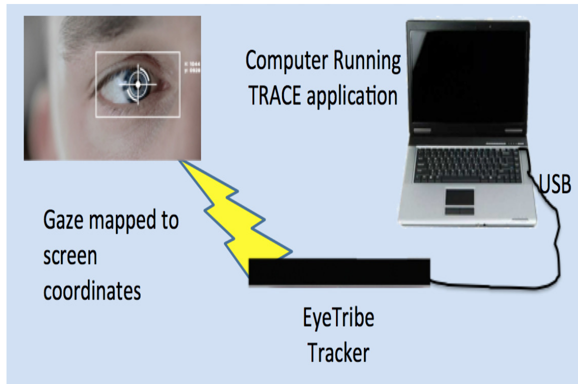

Figure 2: System Architecture

calibration. If successful, the user can proceed to run the TRACE application until a manual quit or alternatively, a loss of socket connection to the tracker.

3.2 Eye Tribe Hardware Module

The Eye Tribe Tracker is an eye tracking system that can calculate the location where a person is looking by means of information extracted from person's face and eyes. In order to track the user's eye movements and calculate the on-screen gaze coordinates, the Tracker must be placed below the screen and pointing at the user. The user needs to be located within the tracker's trackbox. The trackbox is defined as the volume in space where the user can theoretically be tracked by the system. The tracker uses a camera and high-resolution infrared LED in order to track the user's eye movements, captures images of the pupils, and runs them through computer-vision algorithms to determine the user's on-screen gaze coordinates.

The Eye Tribe tracker represents the location on the computer screen where a person is looking by a pair of (x, y) coordinates. The tracker software is based on an Open API design that allows the TRACE application to communicate with the tracker server to get gaze data. This communication relies on JSON messages exchanged via TCP sockets. The TRACE application receives tracker data in the form of one frame per specified time interval. The content of each frame is parsed to extract gaze coordinates, which is then mapped to a key on the virtual keyboard. A connection must be established between a client implementation and the tracker server before messages can be exchanged using the tracker API. Establishing such a connection involves opening a TCP socket in the Python script and connecting to localhost on port 6555. At this point the API protocol proceeds to exchange JSON formatted messages over the socket consisting of the latest frame data. This data contains the raw gaze coordinate in pixels, the pupil size, normalized pupil coordinates, along with other useful metadata.

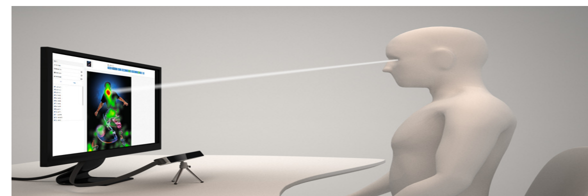

Figure 3: Eye Tribe Setup

Figure 4: Eye Tribe Gaze Coordinate Mapping

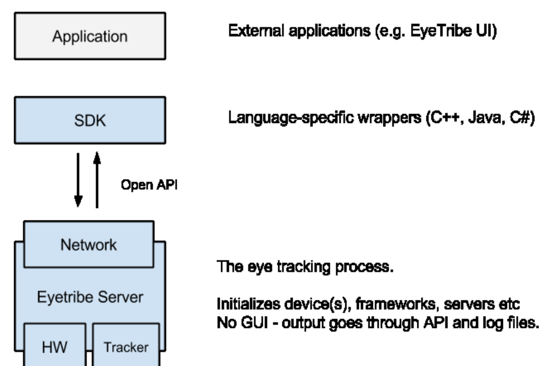

Figure 5: Program Design Overview

3.3 TRACE Keyboard Application

I initially wrote a wrapper in the Python programming language in order to retrieve JSON formatted messages from the device over a TCP socket. The application uses the

pygame library to display a virtual on-screen keyboard with a built-in textbox. The user's eye coordinates are mapped to keys on the screen and according to some fixation time (adjustable in settings), the input is registered and executes the respective action on the text displayed. More advanced features such as auto-completion, speech-to-text conversion, media sharing, and adjustable settings are integrated into the application. These components are explained in further detail in later sections.

3.4 Front End Design

I designed the graphical user interface for the TRACE keyboard application using the pygame library. Pygame offers a set of Python modules designed for writing games and is therefore an ideal choice for writing GUIs as well. It is extremely portable and runs on nearly every platform and operating system, which is the main reason why I chose it over the more popular Tkinter library, Python's standard GUI package.

The GUI features a virtual keyboard and textbox which expand the entire size of any screen. In order to optimize the accuracy of gaze coordinate data, the size of each key is maximized according to the size of the screen. The user can always tell where on the screen he is looking because the key mapped to real-time gaze coordinates is highlighted. If a key is "selected", a ripple effect is displayed on top of the button to notify the user that the character is displayed in the textbox. Further, the user may expand the key options by selecting the SHIFT button which displays capital letters or the &123 button which displays a set of numbers and symbols.

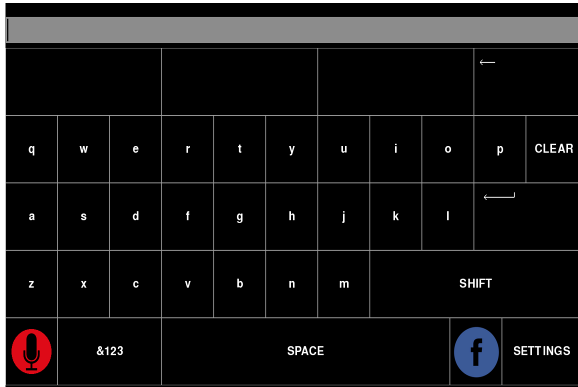

Figure 6: TRACE keyboard user interface

3.5 Auto-Completion Feature

The auto-completion component uses Hidden Markov Models trained on words extracted from Webster, Project Gutenberg, and the user's previously saved messages in order to render the top 3 suggestions as the user "types" out a word. The user has the option of manually "updating" the auto-completion component with a set of typed words from the settings menu. The HMM is re-trained upon each update:

- Transforms training data into ngrams of word-length and normalizes each word
- Builds frequency distribution
- Uses distribution to predict the final states of a word in progress

The corpus data extracted from Webster, Gutenberg, and the user's saved messages is first converted into a single string. The string is then converted into a list of words (ngrams of word-length) and each word is normalized such that character case or symbols are ignored. At this point, the model can determine the frequency of each word in the corpus data. The frequency distribution contains information about the most common words and their frequencies in the corpus data. The next step in the process is the "conditioning" step during which the model creates a probability space of possible values of the next word conditioned under the event that the previous word has occurred. This concept can be summarized as:

P(word A and word B) = P(word B | word A) * P(word A)

P(word B | word A) refers to the probability of word A given word B while P(word A) points to the probability distibution represents all of the words that follow word A.

The auto-completion component is implemented in Python and uses a simple parser to read the initial training data and build the frequency distribution, the Pickle module to serialize the information, and the scikit-learn library in order to train and get predictions from the Hidden Markov Model.

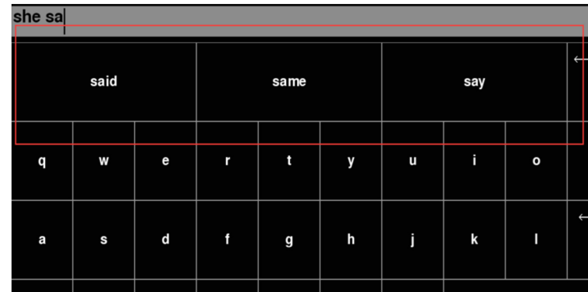

Figure 7: Auto-completion in action

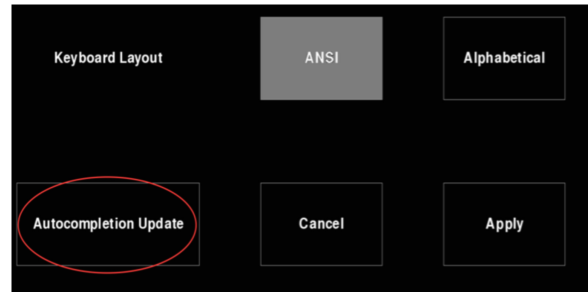

Figure 8: Manual auto-completion component update

3.6 Speech Recognition Feature

The user has the option of selecting the MIC button in order to record a speech and convert it to text displayed in the application. This component is built on a pre-trained HMM-ANN (artificial neural network) hybrid model, which

unlike the auto-completion component, cannot be re-trained to adapt to the user at the moment. However, this improvement is certainly something to consider for future work. The basic procedure during the speech recognition process is as follows:

- Split speech into words based on "quiet" time. For each word:
- Convert recorded waveform into spectrogram
- Perform cepstral analysis to extract features
- Run the MFCC vector through the ANN-HMM model to predict spoken word
- Accumulate the individually recognized words into a sentence and display the word in the TRACE application

3.6.1 *Feature Extraction*

Both individual models (HMM and ANN) were trained independently with the same training data downloaded from Shtooka. The preliminary step in any speech recognition system is to extract features thus selecting particular components associated with an audio signal that can be used to identify the spoken word. The feature extraction process was consistent across both models. The isolated words were converted from waveform to spectrograms and then transformed into Mel Frequency Cepstral Coefficients (MFCC) for feature extraction. Cepstral analysis uses the Mel scale to perform an inverse Fourier transform of the log of the Fourier transform of the signal. The data was consistently sampled at a constant rate of 8kHz. Each word was divided into 80 frames with equal lengths of 10 ms and 25% overlap on each side using the Hamming window function. The cepstral analysis was then applied to each frame:

1. Compute the Fourier transform of each frame
2. Map the resultant energy values to the Mel scale
3. Compute the logs of the powers at each frequency on the Mel scale
4. Apply a cosine transform to the values from the previous step
5. Collect the amplitudes of the resultant spectrum into a vector to represent the feature vector set

3.6.2 *Constituent and Hybrid Models*

A Python program was written to simulate the training and testing of a Hidden Markov Model in recognizing single words from a small dataset. The program preprocessed the training data, converted them into representations that can be used by the model, and "learned" a unique HMM for every word represented by the dataset.

The neural network consists of two hidden layers between its inputs and outputs. The input layer contains 260 nodes corresponding to the number of Mel-frequency Cepstral Coefficients (MFCCs) that make up the feature vector of an input audio signal. The output layer assigns a unique node to each word from the system's entire vocabulary set. The network was trained on the same dataset as the Hidden Markov Model. A Python script was written to remove the background noise from each of the recordings. A function was implemented to calculate the MFCC of each training sample and generate a 260 dimensional column vector. The MFC coefficients were used as input to the neural network which performs the target word classification. The goal of the training process is to take an intended target recording (with the spoken word already known) and to create a multi-dimensional target vector. The procedure is as follows:

- Select a sample at random
- Calculate the output (prediction of what word the recording is associated with)
  - Compute both hidden layers as well as the response using the sigmoid function
  - Map output values to a range of [0, 1] using the sigmoid transfer function
- Compute the error of the output which corresponds to the absolute difference between the actual and the expected outputs
- Update the weights of each hidden layer by using a forward-backward algorithm
- Repeat the process until the maximum number of iterations is reached or until the error is minimized to a satisfactory value

The ANN-HMM hybrid model utilizes a joint probability from the likelihood values received from each individual model. Since both models used the same training dataset and feature extraction process, the combination of their probability values is a valid decision. The hybrid model, thus, does not combine the two statistical models and re-train the data but rather compares and combines the outputs of the two models run in parallel. For example, for a given input W, the ANN model produces probability values $ANN_W{=}x1, x2, ... : 0 \leq x \leq 1$ where each x_i represents the probability of W being a word associated with that index. Similarly, the Hidden Markov Model passes W to each of the "sub-HMMs" that result in a probability vector set $HMM_W{=}m1, m2, ... : 0 \leq m \leq 1$ where m_i is the probability value generated by each HMM that is associated with a particular word. Furthermore, according to the results from running both models on a test dataset, we have two static vectors representing the likelihoods that the model can predict a particular word correctly. This can be represented as:

ANN-SUCCESS-RATE = p1, p2, ...

HMM-SUCCESS-RATE = q1, q2, ...

p_i represents the probability that the ANN model predicts the word associated with i correctly p% of the time.

Similarly, q_i represents the probability that an HMM associated with word i predicts correctly q% of the time. The logic decision of the hybrid model can be summarize as follows:

- Select the maximum probability values from ANN_W and HMM_W
- If the indices corresponding to the two values are equal, both models have predicted the same word as the output. Thus, the hybrid model will output the word associated with this index.
- If the indices mismatch such that the maximum probability value of ANN_W is at index i and the maximum probability value of HMM_W is at index j, compute compounded probabilities of the two models and select the prediction of the higher probability model:
  - P(ANN) = $(x_i p_i + (1 - x_i)(1 - p_i)) * (x_j p_j + (1 - x_j)(1 - p_j))$
  - P(HMM) = $(m_i q_i + (1 - m_i)(1 - q_i)) * (m_j q_j + (1 - m_j)(1 - q_j))$
  - If P(ANN) > P(HMM), then select the word associated with index i
  - If P(HMM) > P(ANN), then select the word associated with index j

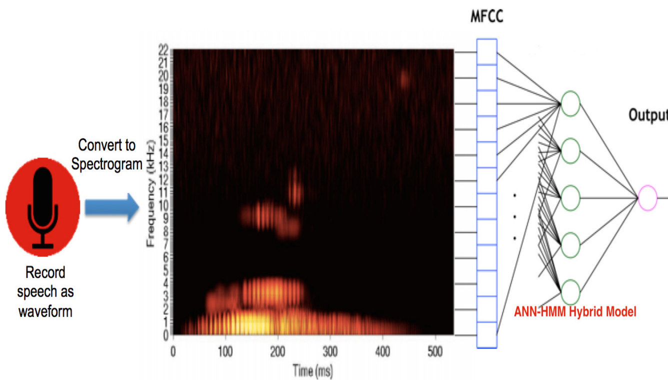

Figure 9: Speech recognition process

3.7 Media Sharing Feature

TRACE is able to communicate with the user's Facebook account in order to post messages on their behalf. After enabling the application to post to Facebook, the user may proceed to eye-type any message in the context of TRACE, gaze at the Facebook button and post the message directly on the Facebook wall.

This component uses the Facebook Graph API in order to get data in Facebook. It is a basic HTTP-based API that can be used by any application to query data, and post comments or photos. Most API calls, in particular posts, require the use of access tokens. A unique user access token is required in order for TRACE to post to an individual's Facebook account. The first time the user starts the application, TRACE requets access and permissions via the Facebook SDK and a login dialog. Once the user authenticates and approves permissions, the user access token is returned to the TRACE app and used for future post calls.

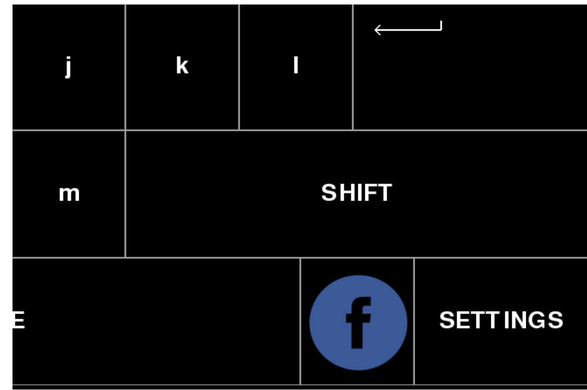

Figure 10: TRACE Facebook button

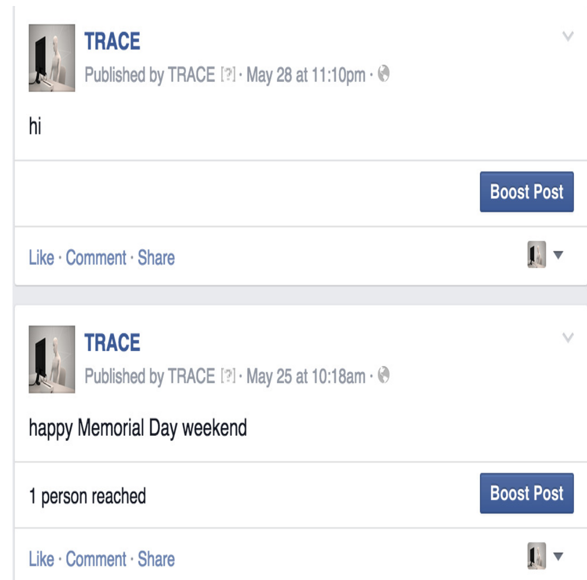

Figure 11: Eye-typed posts on the TRACE Facebook page

3.8 Customizable Settings Feature

With the TRACE software, the user may modify settings and customize the gaze interaction experience with the application. The settings are saved to a configuration file which is loaded in future instances of the application. Currently, the user may adjust and save for future use, the following options from a settings menu:

- Gaze duration : The time required to register a gaze as an input. The default value is set to 1 second with a lower bound of 0.3 second and no upper bound. According to this value, the application determines the frequency of frame data acquisition from the Eye Tribe server. The algorithm to determine whether or not a key has been selected is very simple. Assume that the user has set the gaze duration value to 1 second. The program must then proceed to fetch frame data from the tracker every 1 / 5 = 0.2 seconds since for every frame:
  1. The gaze coordinate is mapped to a button on the

keyboard

2. The value of the button is saved to a queue
3. The program then looks at consecutive sequences of five letters from the queue and if the same letter is repeated five times, it is considered as a key "press".

- Layout of the keyboard : To configure the keyboard to use a different layout, the user can choose the ANSI or Alphabetical buttons from the settings menu by gazing at the intended button. Once selected, the button should be highlighted but the changes will not be displayed until the user selects the Apply button.

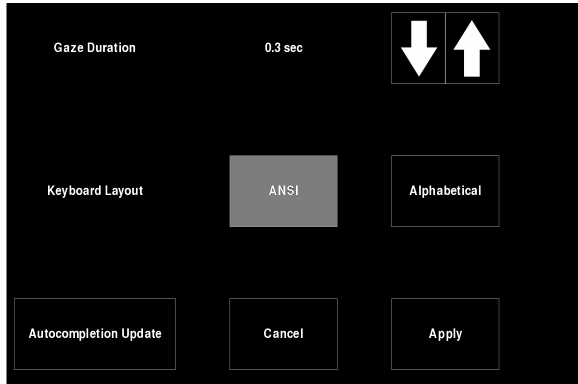

Figure 12: Settings menu

Figure 13: Alphabetical keyboard layout applied

4. END PRODUCT WALK-THROUGH

Every user who starts the TRACE application begins with the same default configuration and user interface. Here, they may proceed to simply gaze at the keys in order to type a message or look at the settings button in order to customize the keyboard layout or the number of seconds it takes to register a fixed gaze. The auto-completion feature (which is updated automatically before the program exits according to typed messages or updated manually from the settings menu) enables the program to adapt uniquely to each user. In the future, the program starts with the user's previously saved settings options and auto-completion data.

At the moment, the only context reached outside of the application is Facebook. Upon gazing at the Facebook button, the user is able to post a typed message from the TRACE textbox to his/her account (after allowing the application to post to Facebook).

The user can speed up the typing process via the speech-to-text or auto-complete features. As the user gazes and enters a character, the top three auto-complete suggestions are displayed as buttons above the screen. At this point, the user may gaze at any word to transfer it to the textbox or proceed to type out extra characters to improve the auto-complete suggestions. By gazing at the speech-to-text button in the lower left corner of the screen, the user may record a clear message, which is then converted to text and displayed on the screen. In order to exit the application, the user must gaze at the ENTER button. At this point, the auto-completion feature is updated according to the displayed message and the message is also added and saved to a local file on disk.

5. FUTURE WORK

Many patients, at times, need to type out the same message on a frequent basis. It would be ideal to be able to implement a way of storing any message typed at some point. Of course, it would be difficult to just scroll through a list of saved messages to select something. Therefore, if the messages can be organized by a category or in an organized fashion, that would be optimal. To store persistent data, I will need to construct a database backend. The database solution must be cost-free and easy to interface with. Therefore a simple solution such as Reddis to store data in key, value format is preferred. Previously written messages can be saved by "category". This feature can also feed saved data into the auto-completion component to furtheradapt to the user.

Future work would primarily focus on extending the functionality of the application to control OS operations, thus replacing the mouse/keyboard entirely. This enables the users to access and use any application using only their eyes (pop-up virtual keyboard).

Future work can also exploit Swype-like technology in order to implement a more accurate keyboard that follows the user's gaze across keys to predict the intended word. The application would feed the first and last keys the user has looked at, as well as the pattern of the eye movements between the two keys, into the algorithm in order to determine the complete word.

Unfortunately, there was not enough time to extensively test the prototype with different users. Understanding how different people use and feel about the product is very important in achieving high user satisfaction. Therefore, a large part of future work consists of talking to testers and watching them interact with the TRACE application.

6. ACKNOWLEDGEMENTS

I owe many thanks to my research mentors, Professor Kenneth Pickar and Professor Adam Wierman, for their guidance and support throughout the course. Thanks also to Professor Matilde Marcolli and Professor Steven Low for their various and invaluable contributions in particular to my research on speech recognition systems.

7. REFERENCES

[1] Eye Tribe: https://theeyetribe.com
[2] PyGame Library: http://www.pygame.org/tags/libraries
[3] Machine Learning Models: http://scikit-learn.org
[4] Tobii DevDocumentation:
http://developer.tobii.com/documentation
[5] Project Gutenberg: http://www.gutenberg.org/wiki/
[6] Shtooka : http://shtooka.net/download.php