Hands-free Typing and Text-to-Speech Using Eye-Tracking

Haoxiang Li, Simon Cadavid, George

I. Background, Motivation, and Review of Existing Solutions

1.1 Background

The project addresses the challenge of enabling communication for individuals with motor disabilities, specifically those affecting hand and/or facial movements. It focuses on hands-free typing and Text-to-Speech (TTS) capabilities using gaze tracking technology. The system utilizes affordable hardware components, such as the Raspberry Pi 4, Pi Camera, and a mini-speaker, to provide a cost-effective and accessible solution.

The core technologies include image processing, computer vision, and machine learning applied to capture and process eye gaze data for cursor control and keyboard interactions. The goal is to create a user-friendly, low-cost system that ensures functionality for individuals with limited mobility.

1.2 Motivation

Motor Disabilities: Individuals with conditions like Parkinson's, ALS, or other movement disorders struggle with traditional typing mechanisms. Some individuals also face speech impairments due to facial muscle disorders, requiring alternatives like TTS systems.

Accessibility Gap: Existing solutions, like the Tobii Eye Tracker, are state-of-the-art but expensive (~$300) and robust to head movement [1]. Many other solutions rely on headsets or glasses [2], which are less accessible and cost-prohibitive [3].

Project Goals: (1) Enable typing using only eye gaze, making it accessible for those with severe motor impairments. (2) Include a TTS component for individuals unable to speak, bridging the gap between typing and verbal communication. (3) Build a system costing under $100, ensuring affordability as well as reliability.

1.3 Review of Existing Solutions

Existing eye-tracking solutions, such as the Tobii Eye Tracker 5 [1], are highly advanced and widely regarded due to their non-contact design and robustness to head movement. However, these systems are prohibitively expensive, with a price point of approximately $300, making them inaccessible to a significant portion of the target audience. Other available systems often

rely on head-mounted devices like headsets or glasses [2], which can be cumbersome, uncomfortable, and even more costly [3]. These solutions struggle to strike a balance between affordability, ease of use, and robustness. In contrast, our system is designed to be functional, accessible, and cost-effective, with a price point under $100. By leveraging gaze tracking combined with linear filters using mediapipe for noise reduction, the system provides a practical alternative. While it is less robust than premium commercial options like Tobii, it effectively addresses the needs of users who require an affordable and user-friendly solution.

II. High Level Description

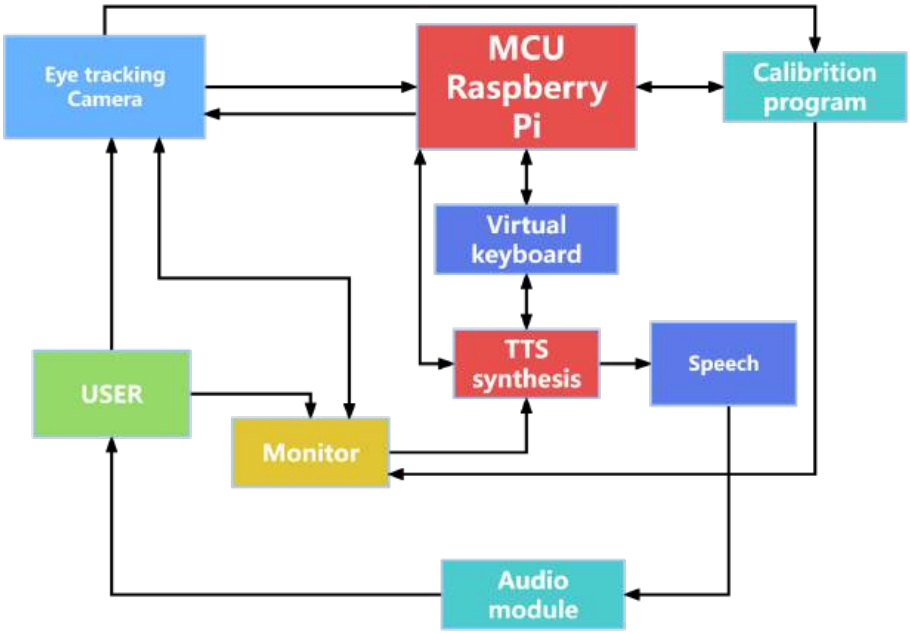

Figure 1: High-level System Diagram

Figure 2: Final System Setup

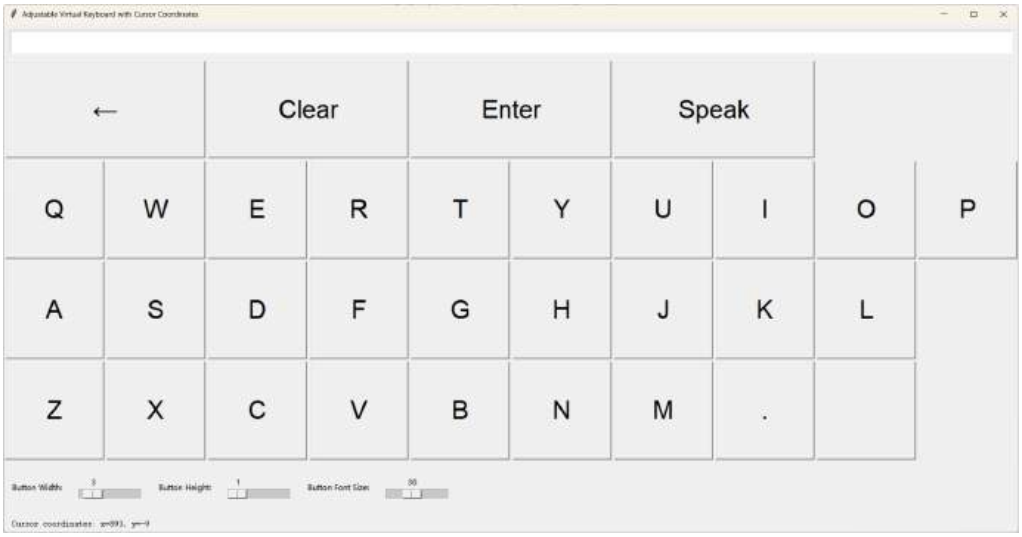

Figure 3: Adjustable Virtual Keyboard

Overview

We developed a hands-free typing and Text-to-Speech (TTS) system using eye-tracking technology. This innovative system enables individuals with motor impairments to type and communicate efficiently without relying on traditional input devices like keyboards or mice. By leveraging affordable components such as a Raspberry Pi 4, Pi Camera, and a mini-speaker, the system balances functionality, accessibility, and cost-effectiveness.

The project involves several critical signal processing steps that enable accurate gaze tracking, screen mapping, and data smoothing for cursor stability. Below are the primary methods and associated algorithms.

2.1 Eye Detection and Pupil Tracking (from gaze_tracking.py)

This step identifies the user's face, locates the eyes, and extracts pupil positions to estimate gaze direction. Facial landmarks are detected using a pre-trained mediapipe model (we used dlib originally, but it was slower and not as robust), and we identify the pupil positions from two of those landmarks.These pupil positions will be used for transformation (from pupil positions to target position on the screen).

2.2 Outlier Removal (from target_tracking.py)

Pupil data often contains noise and outliers due to environmental factors or blink artifacts. The system uses a clustering-based approach to detect and remove outliers, improving data reliability. Gaussian modeling identifies inliers and assigns noisy data a label of -1, effectively excluding it from further processing. This preprocessing step helps maintain the consistency and precision of the gaze tracking pipeline by ensuring that only valid data points are used for cursor positioning and transformations.

2.3 Calibration and Transformation (from transformation.py)

To map gaze points to screen coordinates accurately, the system performs calibration and applies affine or perspective transformations. Affine transformation maps pupil coordinates to screen coordinates using a matrix transformation, calculated through linear algebra. We found that affine transformation is insufficient and perspective transformation is used to handle this. Calibration involves directing the user to gaze at specific screen points, capturing real-world and detected coordinates, and storing transformation matrices for real-time use. This step ensures high accuracy across the screen space.

2.4 Exponential Moving Average (EMA) Filtering (from target_tracking.py)

Gaze data can be jittery, especially during prolonged use, making cursor movement unstable. EMA filtering smooths incoming data by applying weighted averaging, balancing responsiveness and stability. The filtered output is determined by a smoothing factor, which controls the influence of new data compared to past values. This step ensures smooth and natural cursor transitions, improving the overall user experience for hands-free typing.

2.5 Real-Time Gaze Estimation

Once gaze direction ratios are calculated and smoothed, they are mapped to screen coordinates in real-time. The system processes video frames continuously, applies calibrated transformations, and filters data to output stabilized cursor positions. These coordinates are then used to interact with the virtual keyboard, providing seamless and responsive control for

the user. The pipeline's efficiency allows for accurate and near-instantaneous gaze tracking across various screen sizes and configurations.

III. Important Technical Issues We Faced

3.1 Gaze Detection Accuracy

Issue: Accurately detecting gaze direction is challenging due to the small size of eye features, lighting variations, and movement noise. Misalignments during gaze estimation can lead to incorrect cursor placement on the screen, affecting usability.

Solution: We employed the mediapipe face_mesh model to extract facial landmarks and identify the eye region. An Exponential Moving Average (EMA) filter was applied to smooth the data. Additionally, calibration processes ensured accurate screen mapping, adapting to each user's unique gaze patterns.

3.2 Screen Mapping and Calibration

Issue: Mapping gaze points to specific screen coordinates requires precise calibration. Without accurate mapping, gaze positions may not correspond to the intended screen locations, leading to usability issues.

Solution: Calibration points were introduced to align detected gaze ratios with screen coordinates. Affine and perspective transformations were used. This approach provided reliable mapping accuracy, covering the majority of the screen area with less than a 13.3% error margin.

3.3 Noise and Outlier Removal

Issue: Noisy data from environmental factors, such as sudden lighting changes or blinking, introduced inaccuracies in gaze tracking. Outliers in pupil tracking caused erratic cursor behavior.

Solution: A clustering-based approach was implemented to identify and remove outliers. Gaussian modeling determined valid pupil data, excluding outliers from further processing. This preprocessing step ensured a stable and consistent data pipeline, significantly improving cursor performance and usability.

3.4 Cursor Stability

Issue: Gaze-based cursor movement is naturally unstable due to micro-saccades and other involuntary eye movements. This instability makes precise control difficult.

Solution: The EMA filter was integrated into the pipeline to stabilize cursor movement by smoothing gaze data. The filter dynamically balanced responsiveness and noise suppression

using an adjustable smoothing factor. This resulted in a smooth and natural cursor experience, enabling users to interact seamlessly with the virtual keyboard.

3.5 Text-to-Speech Integration

Issue: Incorporating a robust Text-to-Speech (TTS) system required handling text input dynamically while maintaining performance and user feedback.

Solution: The pyttsx3 Python library was used to convert typed text into speech. Adjustable parameters for volume, speech rate, and voice were introduced to allow user customization. TTS was integrated as a feature of the virtual keyboard, activated through a "Speak" button, enabling smooth communication for users with speech impairments.

IV. Investigation into Test Results

4.1 Gaze Target Error

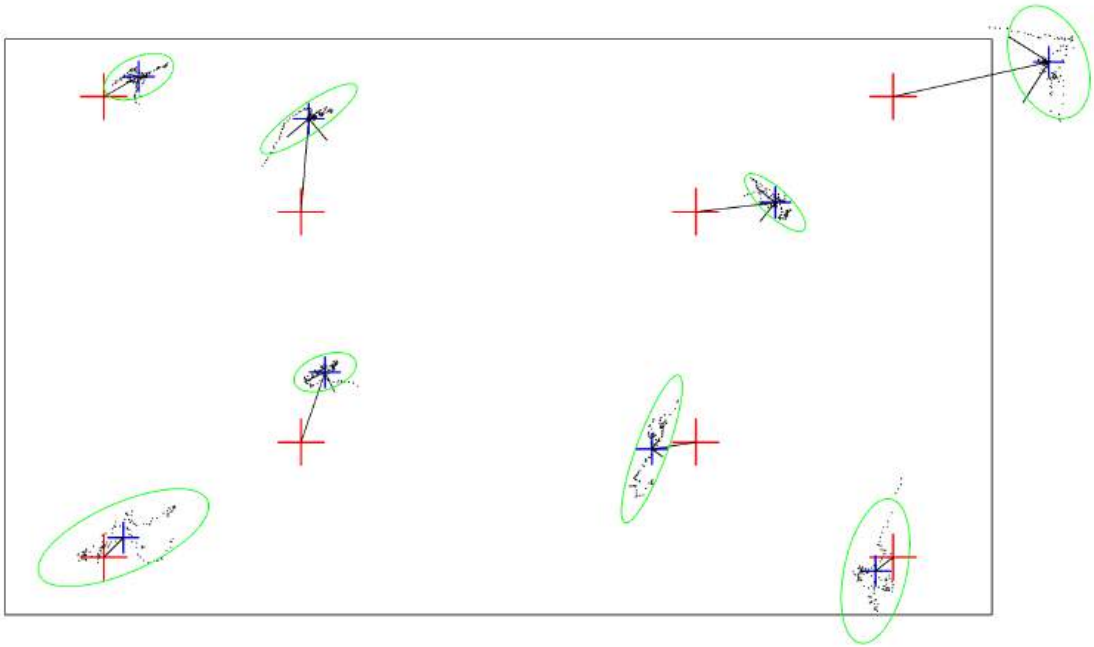

Figure 4: Reference points vs. actual program output

Red crosses are true targets and blue crosses are mean of estimated targets. Blue dots are the sampled estimated targets. Green eclipse shows the possibility boundary of the distribution of samples (assume a gaussian distribution). Arrow lines are the biases.

We think the bias is due to the nonlinearity and head movement.

Mean of error distance = 186.33156

Std of error distance = 109.26938

Point	Bias [y, x]	Standard Deviation [y, x]
Bottom Left (Outer)	[-50.19165 51.408325]	[44.713776 78.29341]
Top Left (Outer)	[-51.324997 91.66666]	[21.506649 31.997742]
Top Right (Outer)	[-89.59167 404.56665]	[52.01001 37.529484]
Bottom Right (Outer)	[36.641724 -45.416748]	[66.649666 31.242485]
Bottom Left (Inner)	[-181.25836 62.36664]	[18.002079 28.591352]
Top Left (Inner)	[-241.55833 20.608337]	[32.19105 44.19733]
Top Right (Inner)	[-23.174988 206.25]	[26.593441 28.12124]
Bottom Right (Inner)	[18.199951 -113.43335]	[67.965515 27.911985]

4.2 GazeTracking Function and Method Latencies

Number of refreshes: 69 (we average over all of these)
Average values for each parameter:
get camframe: 31.1042 ms
get landmarks with mediapipe: 75.2757 ms
- process facemesh: 67.6530 ms (includes running a pre-trained neural network)

apply pupil transform: 1.3547 ms
EMA filter: 0.0000 ms
display: 0.0021 ms
refresh total (includes all the above and other methods): 115.3061 ms

From the above values, we see that the bulk of time spent in doing the GazeTracking processing is in (a) getting the camera frame from Picamera (which runs at approximately 30 frames per second), and (b) getting the facial landmarks with mediapipe, which together take up most of the time (on average). Facial landmarks take ~65% of the time and getting the camera frame takes ~25% of the time.

V. Possible Extensions and Future Work

5.1 Camera

An area we could explore is a camera more specially-suited for our gaze-tracking task, as we used a general-purpose plug-and-play camera for the raspberry pi. Near-infrared cameras are more sensitive to key eye characteristics under varying light conditions and are used by Tobii for their state-of-the-art eye tracking devices [4], and there are cameras with near-infrared sensitivity made by adafruit such as [5] available for purchase online. Also, a camera with a higher resolution and a higher frame rate could improve performance. The Picamera v2 supports 1080p30, 720p60 and 640x480p60/90 video [6], but a higher resolution and framerate would allow us to capture more minute changes in pupil angle and hopefully also reduce the noise in the pupil position measurements. We see from section IV in the GazeTracking latencies section that getting the camera frame is also a time bottleneck in terms of the performance of our program.

5.2 Nonlinear Mappings

Another area to explore is nonlinear mappings to get from the pupil positions to the gaze target output. Our current method is an affine transformation, and although it is efficient to implement and fast, it is not the most accurate model for the task. For a nonlinear example, [7] uses support vector regression (SVR) to both minimize calibration and allow for natural head movement when doing gaze tracking, whereas our method requires a calibration step and for the user to maintain their head straight.

5.3 End-to-End Machine Learning

In a similar vein, end-to-end machine learning methods (i.e., from image frame to target location) could prove useful as well, although the challenge there is to find enough (good quality) training data to fit a model to our task.

VI. References

[1] (https://gaming.tobii.com/product/eye-tracker-5/)
[2] (https://www.bitbrain.com/blog/eye-tracking-devices)
[3] (https://imotions.com/blog/learning/product-news/eye-tracker-prices/)
[4] (https://www.tobii.com/resource-center/learn-articles/how-do-eye-trackers-work)
[5] (https://www.raspberrypi.com/products/pi-noir-camera-v2/)
[6] (https://www.farnell.com/datasheets/2056179.pdf)
[7] (https://sites.ecse.rpi.edu/~cvrl/Publication/pdf/Zhu2006b.pdf)