

VIRTUAL BLINK TRANSCRIBER

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

S SRINANDH (Reg. No - 40111266)

SUJAI M (Reg. No - 40111281)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Category-1 University by UGC**

**Accredited with Grade "A++" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHISALAI,
CHENNAI – 600119**

APRIL - 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category - I University by UGC

Accredited "A++" by NAAC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Srinandh (40111266)** and **Sujai M (40111281)** who carried out the project work entitled “**VIRTUAL BLINK TRANSCRIBER**” under my supervision from November 2023 to April 2024.

Internal Guide

Dr. R. Sethuraman, B.E., M.E., Ph.D.,

HEAD OF DEPARTMENT

Dr. L. Lakshmanan, M.E., Ph.D.,

Submitted for Viva-voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **S. Srinandh (Reg. No - 40111266)**, hereby declare that the Project Report entitled "**VIRTUAL BLINK TRANSCRIBER**" done by me under the guidance of **Dr. R. Sethuraman, B.E., M.E., Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management of Sathyabama Institute of Science and Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, and **Dr. L. Lakshmanan, M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me with necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. R. Sethuraman, B.E., M.E., Ph.D.**, for his valuable guidance, suggestions and constant encouragement that paved the way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Paralyzed people lack the ability to control muscle function in one or more muscle groups. The condition can be caused by stroke, ALS, multiple sclerosis, and many other diseases. Locked-in Syndrome (LIS) is a form of paralysis where patients have lost control of nearly all voluntary muscles. These people are unable to control any part of their body, besides eye movement and blinking. Due to their condition, these people are unable to talk, text, and communicate in general. The software platform converts eye blinks to text. Every feature of the software can be controlled by eye movement. Thus, the software can be independently operated by paralyzed people. Using the software, patients can record messages, recite those messages aloud, and send the messages to others. Key features of the "VIRTUAL BLINK TRANSCRIBER" application include real-time eye-tracking, a user-friendly interface, customizable blink patterns, predictive text input, and integration with various communication channels, such as text messaging, email, and social media. Users can also personalize their blink patterns and customize the application to suit their individual needs and preferences.

The "VIRTUAL BLINK TRANSCRIBER" application represents a significant advancement over current systems for paralyzed individuals by offering several key improvements. Firstly, its real-time eye-tracking feature ensures precise detection of blink patterns, enhancing the accuracy and reliability of text transcription. This improvement minimizes errors and ensures more seamless communication for users. Furthermore, the customizable blink patterns feature allows users to personalize their input methods according to their specific needs and preferences, thereby increasing comfort and usability. By providing a user-friendly interface and customizable options it aims to optimize the user experience and maximize accessibility for paralyzed individuals.

Overall, through its innovative features and commitment to improving accuracy and usability, application strives to significantly enhance communication and quality of life for paralyzed individuals, empowering them to express themselves more effectively and independently in their daily lives. It is also helpful in making such paralysed patients independent by making them communicate freely.

TABLE OF CONTENTS

Chapter No	TITLE	Page No.
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF ABBREVIATIONS	ix
1	INTRODUCTION	1
2	LITERATURE SURVEY	3
	2.1 Overview of Existing research and Literature	3
	2.2 Inferences from Literature Survey	6
	2.3 Open problems in Existing System	8
3	REQUIREMENTS ANALYSIS	10
	3.1 Feasibility Studies/Risk Analysis of the Project	10
	3.2 Software Requirements Specification Document	12
	3.2.1 Functional Requirements	12
	3.2.2 Non Functional Requirements	13
	3.2.3 Software Requirements Specification	13
4	DESCRIPTION OF SYSTEM	14
	4.1 Selected Methodology or process model	14
	4.2 Architecture / Overall Design of Proposed System	14
	4.2.1 Hardware Components	14
	4.2.2 Software Components	14
	4.3 Description of Software for Implementation and Testing plan of the Proposed Model/System	17

5	IMPLEMENTATION DETAILS	20
5.1	Development and deployment setup	20
5.2	Algorithms	21
5.2.1	HaarCascade algorithm	21
5.2.2	OpenCV	21
5.2.3	Hough Transform	22
5.2.4	Twilio	22
5.3	Testing	23
6	RESULTS AND DISCUSSIONS	24
6.1	Presentation of findings	24
6.2	Eye Localization	24
6.3	Blink Detection	24
6.4	Texting and SMS	25
7	CONCLUSIONS	28
7.1	Conclusion	28
7.2	Future work	28
7.3	Research issues	29
7.4	Implementation issues	29
	REFERENCES	31
	APPENDIX	33
	A. SOURCE CODE	33
	B. SCREENSHOTS	53
	C. RESEARCH PAPER	57
	D. CONFERENCE ACCEPTANCE AND PAYMENT SCREENSHOTS	61

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
4.1	Flow chart	15
4.2	System architecture	16
6.1	Face detection and eye localization	24
6.2	Iris detection to detect blinks using Hough transform	25
6.3	Virtual Blink Transcriber GUI	25
6.4	Word recommendation	26
6.5	SMS sent via Twilio	27
B.1	Highlighting rows	53
B.2	Highlighting columns	53
B.3	Training the dataset	54
B.4	Blink detection	54
B.5	GUI for the Mainwindow	55
B.6	Recite word and send SMS	55
B.7	Word Prediction	56
B.8	Integrating Twilio to the software	56

LIST OF ABBREVIATIONS

ABBREVIATIONS	EXPANSION
ALS	Amyotrophic Lateral Sclerosis
CNN	Convolutional Neural Networks
EAR	Eye Aspect Ratio
EEG	Electroencephalogram
GUI	Graphical User Interface
LIS	Locked-In Syndrome
SVM	Support Vector Machine

CHAPTER - 1

INTRODUCTION

"VIRTUAL BLINK TRANSCRIBER" is a technology or system that enables individuals to convert their eye blinks or specific eye movements into text or other forms of communication. This innovation is primarily designed to assist people with severe disabilities, such as locked-in syndrome or advanced stages of motor neuron diseases, who may have limited or no ability to speak or use traditional communication devices. The way "Virtual blink transcriber" works typically involves the use of specialized eye-tracking devices, cameras, or sensors that can detect and interpret eye movements and blinks. These movements are then translated into text on a screen or synthesized speech, allowing the user to communicate with others, operate computers, or interact with their environment.

This technology has the potential to significantly improve the quality of life for those with limited mobility and communication abilities, granting them a means to express their thoughts, needs, and emotions effectively. "Virtual Blink Transcriber" systems have evolved over time, with advances in machine learning and artificial intelligence improving their accuracy and speed. This technology can be a powerful tool for enhancing the independence and social interaction of individuals facing severe physical and communication challenges. The Virtual Blink Transcriber offers customizable features and personalized settings, catering to the unique needs and preferences of individual users. Its seamless integration with external communication channels, such as Twilio for message sending, further expands its utility and versatility.

It is a testament to the potential of innovation in assistive technologies to break down barriers and provide a voice to those who would otherwise be voiceless. Research studies stated that most ALS/PLS patients have a good acceptance rate of using technologies based on eye tracking to initiate communications with their surrounding world and tracking the patient's eyes is much simpler than tracking or detecting other signals, such as EEG/ECG. Eye tracking technologies are utilized in many sciences such as cognition, psychology, computer, and medicine to digitize how people interact with their living environment. Tracking human eyes,

which implies recording their different movements, is essential for many pervasive applications such as eye-based communications and computer environment interactions.

In the past there has been much interest in improving the use of computer interfaces like mouse, keyboard or intelligent interfaces that make it easy for users to interact with computers effectively and naturally. People with certain disabilities cannot operate with these interfaces. Therefore, computers are required to understand patient's natural indications such as facial expressions, gestures and ocular region movements. Hence, we propose a communication interface based on eye blinks so that a patient capable of blinking voluntarily can operate with it. Therefore, the user interacts with the proposed system where the eye gaze is classified into long and short blink. This is inspired by the fact happened in a 1966 telecasted interview, during which an official was being interviewed while held captive and made use of Morse Code by blinking his eyes to spell SOS words.

Computer vision techniques enable to establish a real-time communication by using different machine learning models such as Support Vector Machines (SVM) or Convolutional Neural Networks (CNN). A prediction mechanism helps select the starting lexicon of a word and the user can choose the word from many predicted words. Through a frequency table analysis, commonly used words are chosen at random. We highlight that there is no prior manual initialization, special lighting, or previous face detection as our system can calibrate it on the condition that the user is in the camera region and close. This report aims to provide a comprehensive overview of the Virtual Blink Transcriber, exploring its functionality, evolution, and impact on the lives of its users. Through an examination of its underlying technology, including specialized eye-tracking devices, cameras, and machine learning algorithms, we seek to elucidate the mechanisms that enable this innovative system to translate eye movements into meaningful communication.

The Virtual Blink Transcriber (VBT) stands as a prototype heralding a transformative revolution in communication for individuals facing physical challenges. It has many applications in the medical industry for its practicality. Also, the extensivity of the potential usages and the versatility of its adaptations is huge.

CHAPTER - 2

LITERATURE SURVEY

2.1 OVERVIEW OF EXISTING RESEARCH AND LITERATURE

In B.Raghavan's [1] paper titled "Real Time Blink Detection For Burst Capture" the author presents an adaptive template-based blink detection algorithm aimed at identifying frames containing human blinks for automatic removal from devices. While achieving an initial F1-score of 94.39% and demonstrating real-time performance on mobile platforms, the authors recognize the need for a new evaluation approach before integrating the algorithm effectively into a mobile application.

In the research work of Elle Luo et al [2], the paper introduces Eslucent, a wearable capacitive sensing device utilizing eyelid stickers to detect blinks. Through direct contact with the eyelid crease, Eslucent employs a falling edge detection algorithm to detect intentional and involuntary blinking activities. In a study involving 14 participants, Eslucent achieved an average precision of 82% and recall of 70%, with over 90% precision and recall in intentional blinking. This innovative integration of interactive technology into a common beauty product offers a unique wearable solution for blink detection and human-computer interaction.

In K. Toennis et al.'s [3] research, the authors propose a rapid iris feature localization method in frontal face images using the Hough transform. Their approach comprises an initial iris detection step and a tracking step leveraging the detected features for faster computation. Demonstrating feasibility for tracking at 200 frames per second or higher, their prototype implementation on a 266 MHz Pentium II PC achieves initial detection in about 6 seconds and tracking in 0.05 seconds per step.

In Lam L et al.'s [4] paper, the authors propose a method for extracting eye boundaries in head-and-shoulders images. They utilize approximate eye positions based on anthropometric measures and introduce a corner detection scheme to

set initial parameters for eye templates. By incorporating corner information, they achieve a 40% reduction in processing time and improved eye boundary representation.

In Mengxi Liu et al.'s [5] paper, titled "Non-contact, real-time eye blink detection with capacitive sensing" the authors proposed in their work a robust eye detection solution based on capacitive sensing technique. The solution was demonstrated with a series of experiments considering both static and dynamic body states. With the capacitive sensing unit deployed on a pair of standard glass, they achieved 92% detection precision and 94% recall.

In this work by Mohit Agarwal et al. [6], the authors consider the problem of eye-blink detection that can then be employed to reliably remove eye-blinks from EEG signals. The paper titled "Blink: A fully automated unsupervised algorithm for eye-blink detection in eeg signals" proposes a fully automated and unsupervised eye blink detection algorithm that self-learns user-specific brainwave profiles for eye-blinks. Hence, the algorithm does away with any user training or manual inspection requirements.

In the works of Paulo Augusto de Lima Medeiros et al.[7], the authors have attempted to detect the face of the user. Then, landmark points are detected from the current face. These points are then used for face alignment and for extracting coordinates of both eye-patch and eye. Since the eye-patch extraction may fail by retrieving a patch without a real eye, a CNN model is used to classify these extractions into eye images or non-eye images.

The last step is eye-state classification and two models were developed for this classification task. The first one is based on a CNN and will classify the correct eye-patches into open or closed eyes, whereas the second one uses an SVM to sort the extracted eye coordinates.

In the studies of Sarmad Al-gawwam et al. [8], the authors introduce a novel technique for detecting eye blinks by employing automatic tracking of facial landmarks to localize the eyes and eyelid contours. Their facial landmarks detectors, trained on an in-the-wild dataset, exhibit remarkable robustness to

diverse conditions. The proposed method estimates landmark positions, extracts vertical eyelid distances, and utilizes a Savitzky–Golay filter for signal smoothing while retaining peak information for blink detection. A finite state machine discerns false and true blinks based on duration, showcasing superior efficiency compared to state-of-the-art methods across three standard datasets

In Scott MacKenzie et al.'s [9] paper, a new text entry system is proposed, implemented, and evaluated. *BlinkWrite* provides a communication gateway for cognitively able motor-impaired individuals who cannot use a traditional eye-tracking system. In a user study, 12 participants entered text using the system with three settings for scanning interval: 1,000, 850, and 700 ms. An average text entry rate of 4.8 wpm was observed with accuracy >97%. The highest average text entry rate was achieved with the scanning interval of 850 ms. The lack of messaging integration limits its functionality compared to systems with broader communication capabilities, potentially impacting user satisfaction and adoption. The absence of word prediction in BlinkWrite could hinder user efficiency and speed, as it relies solely on blink input without anticipatory text suggestions.

In T. Soukupova et al.'s [10] research, the authors have implemented a linear SVM classifier (called EAR SVM) trained from manually annotated sequences. Positive examples are collected as ground-truth blinks, while the negatives are those that are sampled from parts of the videos where no blink occurs, with 5 frames spacing and 7 frames margin from the ground-truth blinks. While testing, a classifier is executed in a scanning-window fashion. A 13-dimensional feature is computed and classified by EAR SVM for each frame except the beginning and ending of a video sequence.

The proposed real-time algorithm leverages recent landmark detectors trained on diverse datasets, ensuring robustness against varying conditions like head orientation, illumination, and facial expressions. By precisely estimating landmark positions, it extracts the eye aspect ratio (EAR) as a scalar quantity to characterize eye opening. Through an SVM classifier, it detects blink patterns in short temporal windows. While effective, it faces challenges in scenarios with high variability in facial expressions or lighting conditions, potentially impacting detection accuracy.

2.2 INFERENCES FROM LITERATURE SURVEY

This work described a novel non-contact, wearable, real-time eye blink detection solution based on capacitive sensing technology. A low-cost and low-power consumption capacitive sensing prototype was developed and deployed on a pair of standard glasses with a copper electrode attached to the glass frame. The eye blink action will cause the capacitance variation between the electrode and the eyelid.

Thus by monitoring the capacitance variation caused oscillating frequency shift signal, the eye blink can be abstracted by a simple comparison of the raw frequency signal with a customized threshold. The feasibility and robustness of the proposed solution were demonstrated in five scenarios performed by eight volunteers with an average precision of 92% and recall of 94%.

Existing eye blink detection approaches are mainly grouped into vision-based and sensor-based solutions. The vision solutions capture the landmarks of the face and abstract eye blink features with a deep convolutional model. Such a system provides high accuracy of eye blink detection and real-time inferences benefiting from the development of hardware resources and the feature abstracting deep models. However, such systems are constrained in practical scenarios. Firstly, environmental conditions like light could cause a sharp drop in accuracy. Secondly, there are a lot of cases in everyday life where a camera does not exist when eye blink detection is needed. Thirdly, the vision solution is not a privacy-respect approach, thus, resulting in a low acceptance rate among users. On the other side, the sensor-based solutions do not face privacy issues since it captures only the eye-blink messages from the user.

Jialin proposed "BlinkListener" to sense the subtle eye blink motion using acoustic signals in a contact-free manner and demonstrated the robustness on the Bela platform as well as a smartphone with a median detection accuracy of 95%.

Previous research has introduced the Eslucent, an eyelid interface for detecting eye blinking, which is built upon the fact that the eyelid moves during the blink. By

deploying an on-skin electrode on the eye-lid, the capacitance value formed by the electrode and the eyelid will variate. The experiments show that the Eslucent achieves an average precision of 82.6% and a recall of 70.4%.

Some other sensor-based approaches were also proposed for eye blink detection, like the doppler sensor EEG signals, mmWaves, etc. However, the proposed sensor-based solutions lack either practicability, meaning that the long-term and in-the-wild eye blink detection is not possible, or robustness, meaning that the dynamic environmental conditions result in a limited number of usage scenarios.

For example, the angular sensing range of the "BlinkListener" is limited. MmWave and Doppler sensors cannot supply eye blink detection when people are out of the sensing field. EEG and the said "Eslucent" need special head/face preparation work before the detection.

To address the existed sensing limitations, we proposed in this work a robust eye detection solution based on capacitive sensing technique. The solution was demonstrated with a series of experiments considering both static and dynamic body states.

With the capacitive sensing unit deployed on a pair of standard glass, we achieved 92% detection precision and 94% recall.

Overall, we have the following two contributions from this work:

- (1) A general concept of using capacitive sensing for eye blink detection in a non-contact way.
- (2) A wearable and low-power eye blink detection system with demonstrated robustness and accuracy.

Research studies stated that most ALS/PLS patients have a good acceptance rate of using technologies based on eye tracking to initiate communications with their surrounding world and tracking the patient's eyes is much simpler than tracking or detecting other signals, such as EEG/ECG.

2.3 OPEN PROBLEMS IN EXISTING SYSTEM

The existing technology implements the concept of converting eye blinks or movements into text or meaningful input. This can be relevant for individuals with disabilities, especially those who have limited or no ability to speak or type. While there has been some progress in this field, several challenges and open problems remain:

1. **Accuracy and Reliability** : One of the primary concerns is achieving high accuracy and reliability in converting blinks to text. Blink detection can be influenced by various factors, including lighting conditions, head movements, and individual differences. Developing more accurate and robust detection algorithms is crucial. Accuracy is not optimal in the existing technology.
2. **Vocabulary and Language Support** : Extending blink-to-text systems to support a wide vocabulary and multiple languages is a challenge. The system needs to recognize and translate various words, phrases, and languages accurately. The existing system cannot recommend words to the user.
3. **Adaptation to Individual Users** : People have different blink patterns and habits. A robust system should be adaptable to individual users and learn their specific blink patterns to improve accuracy.
4. **Real-Time Processing** : Blink-to-text systems should operate in real-time, allowing users to communicate at a reasonable pace. Reducing latency and response time is essential.
5. **Accessibility and Inclusivity** : Blink-to-text technology should be accessible to a wide range of users, including those with severe disabilities. This requires making the technology affordable and adaptable to various disability types.
6. **Integration with Existing Technologies** : Integrating blink-to-text systems with existing assistive technologies and communication devices can be challenging. Seamless integration can enhance the overall user experience. Since the existing

systems cannot help the user communicate with others remotely the technology is not at its most effective. It requires integration.

7. Training Data and Diverse User Groups : Many machine learning-based systems require extensive training data. Ensuring that data is collected from a diverse group of users, including those with rare or complex disabilities, can be a challenge. Newer studies have achieved much more accuracy.

8. Long-Term User Engagement: Maintaining long-term user engagement and adoption of blink-to-text systems is a challenge. Designing strategies to encourage consistent usage and address user feedback and preferences is essential for the success of these systems.

9. Cross-Platform Compatibility: Ensuring compatibility across different devices and platforms is important for widespread adoption of blink-to-text systems. Developing solutions that work seamlessly on various hardware and software configurations can improve accessibility and usability for users.

10. Evaluation and Benchmarking: Establishing standardized evaluation metrics and benchmarks for assessing the performance of blink-to-text systems is necessary for comparing different approaches and tracking progress in the field. Consistent evaluation methodologies can help identify strengths and weaknesses in existing systems.

11. Multimodal Integration: Exploring integration with other modalities, such as speech recognition or gesture input, can enhance the versatility and usability of blink-to-text systems.

12. Ethical and Societal Implications: Considering the ethical, legal, and societal implications of blink-to-text systems, including issues related to privacy, consent, and accessibility, is crucial. Ensuring that these systems are designed and deployed in an ethical and responsible manner is essential for their acceptance and adoption.

CHAPTER - 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES/RISK ANALYSIS OF THE PROJECT

Feasibility Studies - Evaluating the Viability:

Feasibility studies are essential early in the project lifecycle to determine the practicality of implementing a communication system based on user blinks. A feasibility study for a Blink to Text conversion technology project involves assessing the practicality, viability, and potential success of implementing such a system

Technical Feasibility: Assess the technological requirements and capabilities needed to develop and implement the Virtual Blink Transcriber, including eye-tracking devices, cameras, and machine learning algorithms. Evaluate the feasibility of integrating these technologies into a cohesive system that accurately translates eye blinks and movements into text or synthesized speech. Consider the availability of necessary resources, expertise, and infrastructure to support the development and deployment of the Virtual Blink Transcriber.

Financial Feasibility: Conduct a cost-benefit analysis to determine the financial viability of the project, considering factors such as research and development expenses, equipment costs, and potential revenue streams. Identify funding sources and explore financing options to support the development and implementation of the Virtual Blink Transcriber. Assess the potential return on investment and long-term sustainability of the project, taking into account market demand and competitive landscape.

Operational Feasibility: Evaluate the practicality and effectiveness of integrating the

Virtual Blink Transcriber into existing communication and assistive technology systems. Consider the ease of use, reliability, and scalability of the system, ensuring that it meets the needs of users with severe disabilities in various

environments. Assess the level of support and training required for users, caregivers, and healthcare professionals to effectively utilize the Virtual Blink Transcriber.

Risk Analysis:

Technological Risks: Potential technical challenges such as compatibility issues, hardware limitations, and software bugs could delay or disrupt the development and implementation of the Virtual Blink Transcriber. Changes in eye-tracking technology or advancements in machine learning algorithms may require ongoing updates and maintenance to ensure the system remains accurate and reliable.

Regulatory and Compliance Risks: Compliance with regulatory standards and guidelines governing medical devices and assistive technologies is essential to ensure the safety and effectiveness of the Virtual Blink Transcriber. Failure to meet regulatory requirements could result in legal and financial penalties, as well as damage to the reputation of the project and its stakeholders.

Accessibility and Usability Risks: The effectiveness of the Virtual Blink Transcriber may be limited by factors such as user comfort, ergonomics, and accessibility for individuals with diverse needs and abilities. Usability testing and ongoing feedback from users and caregivers are essential to identify and address potential barriers to the adoption and usage of the system.

Market and Competitive Risks: Market dynamics, including changing consumer preferences, technological advancements, and competitive pressures, could impact the demand for the Virtual Blink Transcriber and its long-term viability. Effective marketing and strategic partnerships may be necessary to differentiate the Virtual Blink Transcriber from competing products and ensure its success in the marketplace. Additionally, technological obsolescence looms as rapid advancements could render existing blink-to-text solutions outdated, prompting the need for continual innovation to stay relevant. Additionally, regulatory changes,

economic downturns, and supply chain disruptions further compound market risks. To navigate these challenges, companies must conduct thorough market research, adapt to changing consumer preferences, and invest in innovation and resilience.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

The Software Requirements Specification (SRS) document outlines the requirements

for developing the Virtual Blink Transcriber software. This system aims to assist individuals with severe disabilities by converting their eye blinks and movements into text or synthesized speech, enabling effective communication and interaction with their environment. This document serves as a guideline for developers, stakeholders, and project teams involved in the creation and implementation of the Virtual Blink Transcriber.

3.2.1 Functional Requirements

Eye Tracking: The system shall integrate with eye-tracking devices, cameras, or sensors to accurately detect and interpret eye blinks and movements. Eye tracking algorithms shall capture and analyse user eye movements in real-time.

Text and Speech Generation: The system shall translate detected eye blinks and movements into text or synthesized speech. Text prediction models shall be implemented to facilitate efficient word selection based on user input.

User Interface: The system shall feature a user-friendly interface that allows individuals with severe disabilities to navigate and interact with ease. The interface shall display predicted words and phrases, enabling users to select desired communication options.

Accessibility and Usability Risks: The effectiveness of the Virtual Blink Transcriber may be limited by factors such as user comfort, ergonomics, and accessibility for individuals with diverse needs and abilities. Usability testing and

ongoing feedback from users and caregivers are essential to identify and address potential barriers to the adoption and usage of the system.

Market and Competitive Risks: Market dynamics, including changing consumer preferences, technological advancements, and competitive pressures, could impact the demand for the Virtual Blink Transcriber and its long-term viability. Effective marketing and strategic partnerships may be necessary to differentiate the Virtual Blink Transcriber from competing products and ensure its success in the marketplace.

3.2.2 Non-functional Requirements

Performance: The system shall exhibit high performance and responsiveness, with minimal latency in detecting and translating eye blinks and movements. Response times for text and speech generation shall be within acceptable limits to facilitate seamless communication.

Reliability: The system shall be reliable and robust, capable of operating effectively in various environments and conditions. Measures shall be implemented to prevent system crashes, errors, or data loss during operation.

Security: The system shall implement appropriate security measures to protect user data and ensure confidentiality.

3.2.3 Software Requirements Specification

Hardware specifications:

- Webcam with high resolution
- Higher RAM, of about 4GB or above
- Processor of frequency 1.5GHz or above

Software specifications:

- Python 3.6 or higher
- Visual Studio Code

CHAPTER - 4

DESCRIPTION OF PROPOSED SYSTEM

4.1 SELECTED METHODOLOGY OR PROCESS MODEL

To implement this model, execution of program is done through Visual Studio Code. Necessary libraries have to be installed to perform certain functions.

4.2 ARCHITECTURE / OVERALL DESIGN OF PROPOSED SYSTEM

Designing the architecture and overall system for the software involves a combination of hardware and software components that enable users to convert their eye blinks into text for communication. Here's an overview of the architecture and design considerations for the proposed system:

4.2.1. Hardware Components

Eye-Tracking Device: The system includes an eye-tracking device equipped with cameras and sensors to monitor eye movements and blinks accurately.

Processing Unit: A dedicated processing unit, which can be a powerful microcontroller or a small computer, should be responsible for real-time data processing, including blink detection and interpretation.

Display Interface: The system may require a display interface, such as a screen or a head-mounted display, to present the generated text to the user.

4.2.2. Software Components

Blink Detection Algorithm: Develop a robust blink detection algorithm that can accurately differentiate between voluntary and involuntary blinks, interpret their duration and frequency, and distinguish between different types of blinks (e.g., long and short blinks).

Text Generation Engine: Implement a text generation engine

that converts the interpreted blink patterns into readable text. This engine may incorporate machine learning techniques to improve accuracy over time.

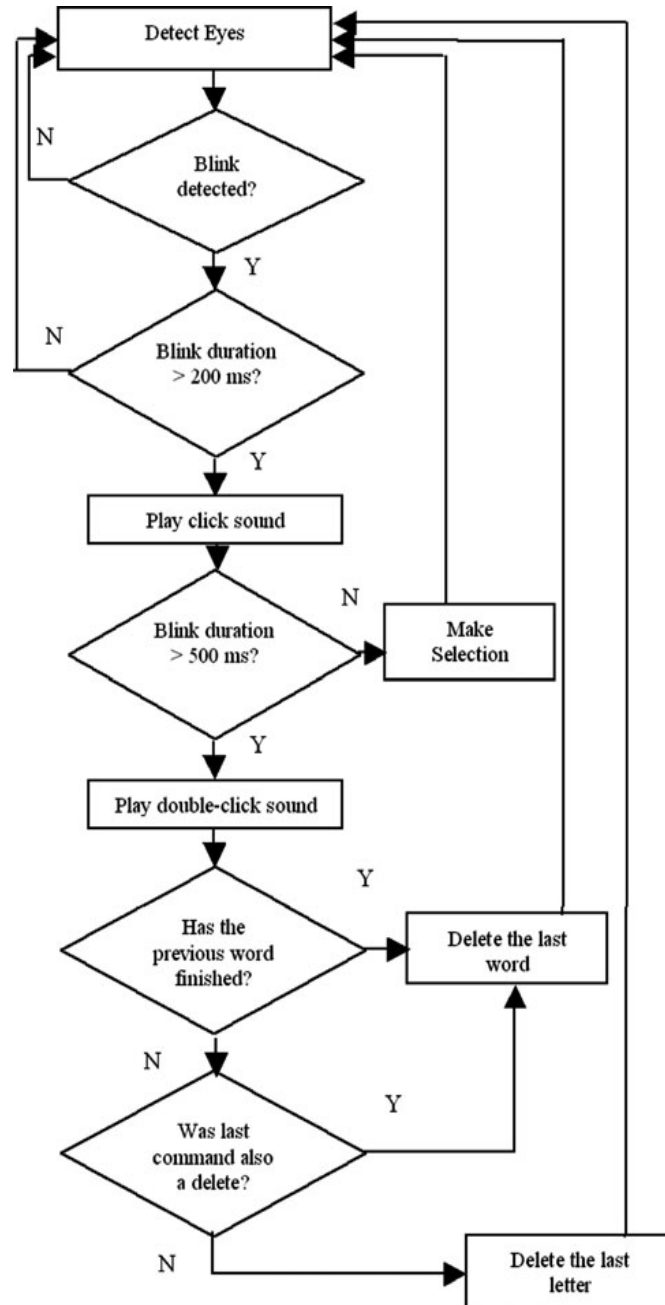


Fig 4.1: Flowchart

User Interface (UI): Design an intuitive and user-friendly UI for user calibration, system setup, and interaction. Ensure that the UI is accessible to individuals with varying degrees of disabilities.

Data Management: Implement a data storage and management system to store user preferences, blink patterns, and system settings securely.

Real-Time Processing: The software should process eye blinks in real-time, with low latency, and convert them into text as quickly as possible to support natural conversations.

Adaptation and Personalization: Incorporate algorithms that adapt to individual users' blink patterns, improving accuracy over time as the system learns from user interactions.

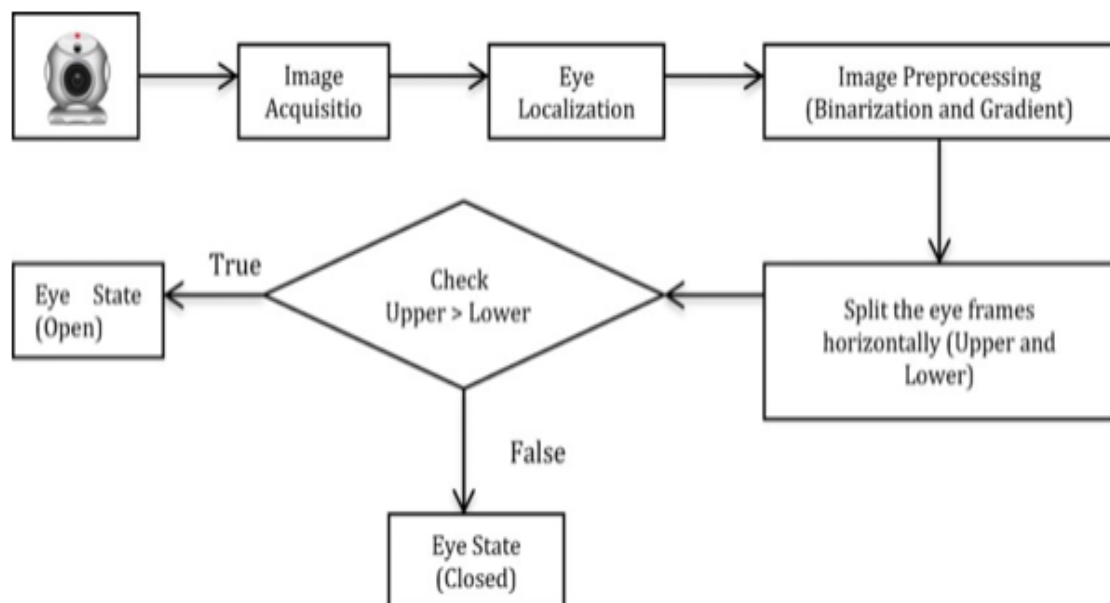


Fig 4.2: System Architecture

Frontend Interface: The frontend interface serves as the user-facing component of the system, providing a graphical user interface (GUI) for users to interact with. Users input commands through the interface, which are then processed and translated into text or synthesized speech.

Eye-Tracking Module: The eye-tracking module consists of hardware components such as eye-tracking devices or cameras, which capture and track the user's eye movements in real-time. Eye-tracking algorithms analyze the captured data to detect eye blinks and movements, which are then translated into commands for the system.

Text-to-Speech (TTS) Module: The TTS module converts the text generated by the system into synthesized speech. It utilizes text-to-speech algorithms and libraries to generate natural-sounding speech output from the translated text.

SMS Integration: The SMS integration feature enables the system to send text messages (SMS) to designated recipients. Users can compose messages using the frontend interface, which are then converted into text and sent via SMS through integration with a mobile messaging service or API.

4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL/SYSTEM

PYTHON

Python is the primary programming language used in this project to create a GUI and implement machine learning algorithms and train the model to achieve the proposed solution. Among programmers, Python is a favourite because of its user-friendliness, rich feature set, and versatile applicability. Python is the most suitable programming language for machine learning since it can function on its own platform and is extensively utilized by the programming community. Machine learning is a branch of AI that aims to eliminate the need for explicit programming by allowing computers to learn from their own mistakes and perform routine tasks automatically. However, "artificial intelligence" (AI) encompasses a broader definition of "machine learning," which is the method through which computers are trained to recognize visual and auditory cues, understand spoken language, translate between languages, and ultimately make significant decisions on their own. The desire for intelligent solutions to real-world problems has necessitated the need to develop AI further in order to automate tasks that are arduous to

programme without AI. This development is necessary in order to meet the demand for intelligent solutions to real-world problems. Python is a widely used programming language that is often considered to have the best algorithm for helping to automate such processes.

The following methodology and testing plan is used to train the model and create a program that runs in real time to detect the user's blinks and type the corresponding text onto the textbox. It also has additional features such as reciting the typed message with a voice and sending the text messages via SMS.

VisualStudioCode

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It supports various programming languages and offers features such as syntax highlighting, code completion, debugging, version control integration, and extensions support. VSCode is known for its lightweight and fast performance, making it popular among developers across different platforms, including Windows, macOS, and Linux. Its customizable interface allows users to tailor their coding environment according to their preferences, while its extensive library of extensions enables the integration of additional functionalities and tools, making it a versatile choice for software development projects of all sizes and complexities.

Blink Detection

The program will search the eye region of interest for pupils. Before, analyzing the region, the program will ensure that there is a high contrast between objects in the image. It is done using HaarCascade. By analyzing changes in eye features over time, such as closure or movement, blink events can be detected and interpreted as input signals for various applications, including assistive technologies and human-computer interaction systems.

Therefore, the program converts the region to gray-scale and runs a Gaussian blur. The image filtering is conducted only on this region instead of the whole image for efficiency purposes.

Blink detection algorithms often rely on sophisticated techniques, such as landmark detection and eye aspect ratio (EAR) computation, to accurately identify blink patterns in video sequences. These algorithms leverage recent advancements in computer vision and machine learning, enabling real-time detection of blinks across diverse conditions like varying head orientations, illumination levels, and facial expressions.

Iris Detection

If the software can detect a circle roughly in the center of the region and that circle takes up roughly the correct amount of area, the software assumes that the circle is the pupil. If the pupil is detected, then the software can assume that the eye is open. Iris detection involves the precise localization of iris features within facial images, typically using techniques like the Hough transform. Recent advancements in iris detection algorithms focus on enhancing speed and accuracy, enabling efficient processing even on standard hardware.

Intentional Blink Detection

It is important that the application can differentiate between intentional and unintentional blinks. Therefore, the software does not recognize a blink unless it lasts longer than 1.25 seconds. This ensures that any involuntary blinks do not affect the software.

Algorithm, Image Processing and UI

OpenCV, a computer vision library, and PyQt, a GUI toolkit for Python, synergize to enable comprehensive application development. Utilizing Haarcascade, an object detection algorithm, within OpenCV facilitates the recognition of predefined features like faces within images and video streams. PyQt complements this functionality by providing a user-friendly interface through which users can interact with the application. Together, these technologies empower developers to create sophisticated applications with seamless integration of computer vision capabilities and intuitive user interfaces. The combination of OpenCV, PyQt, and Haarcascade offers endless possibilities for innovative solutions across various domains.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

The development and deployment setup for the Virtual Blink Transcriber (VBT) involves configuring the software environment, integrating with Twilio for messaging, designing a user-friendly interface, and ensuring robust testing and maintenance procedures to enable efficient and accessible communication for individuals with paralysis. The following steps are utilized for development and deployment of the proposed system. The following are the steps to be considered.

- **Development Environment Setup** - To initiate the development of the "VIRTUAL BLINK TRANSCRIBER" application, first, install Python and PyQt for GUI development. Configure a virtual environment for dependency management, and include OpenCV for computer vision tasks.
- **Designing the GUI** - Utilize PyQt to craft an intuitive and user-friendly graphical user interface (GUI) for seamless interaction, ensuring accessibility for paralyzed individuals.
- **Implementing Eye Tracking** - Leverage OpenCV to implement eye-tracking functionality within the application, enabling accurate detection of eye movements and blinks.
- **Blink Detection** - Integrate blink detection algorithms into the application, leveraging techniques such as Haarcascade for preliminary eye detection and the Hough Transform for precise blink identification. This ensures accurate detection of blinks for effective communication.
- **Adding Functionalities** - Integrate various functionalities into the application, including blink detection algorithms, predictive text input, and customizable blink patterns, to enhance usability and customization options.

5.2 ALGORITHMS

5.2.1 HaarCascade algorithm

Haarcascade, short for Haar Cascade Classifier, is a machine learning-based object detection algorithm used to identify objects in images or video streams. It is particularly well-known for its ability to detect faces and facial features, although it can be trained to recognize other objects as well. Haarcascade works by utilizing a set of pre-trained classifiers that describe the features of the object being detected. These classifiers are based on Haar-like features, which are small rectangular regions with specific properties, such as contrast between adjacent regions.

The algorithm scans the image or video frame with a sliding window technique, applying the classifiers to different regions of the image. At each step, it evaluates whether the features described by the classifier are present in the current region. If the features match, the algorithm identifies that region as containing the object of interest.

Haarcascade is widely used in various applications, including face detection in digital cameras, video surveillance systems, and facial recognition software. It is favored for its relatively fast processing speed and accuracy, making it suitable for real-time applications. Additionally, Haarcascade classifiers are available pre-trained for common objects, simplifying the implementation process for developers.

5.2.2 OpenCV

OpenCV, short for Open Source Computer Vision Library, is a versatile open-source software library widely used in various fields such as robotics, augmented reality, and medical imaging. These include basic image processing operations like filtering and transformation, as well as advanced functionalities such as object detection, recognition, and tracking. One of OpenCV's notable features is its support for object detection and recognition, facilitated by algorithms like the Haar

Cascade Classifier, which is commonly employed for tasks like face detection. Moreover, it provides capabilities for video analysis, enabling tasks like motion detection, optical flow estimation, and background subtraction.

Written primarily in C++, OpenCV supports multiple programming languages such as Python, Java, and MATLAB through bindings. With its extensive documentation, active community, and continuous development, OpenCV remains a go-to tool for researchers, engineers, and developers seeking robust solutions in the realm of computer vision. Its flexibility, scalability, and comprehensive feature set make it indispensable for a wide range of applications across different domains.

5.2.3 Hough Transform

The Hough Transform is a technique used in image processing and computer vision to detect geometric shapes, particularly lines and curves, within an image. Developed by Paul Hough in the early 1960s, it remains a fundamental tool in many computer vision applications. The primary objective of the Hough Transform is to identify shapes that can be described by mathematical equations, such as lines or circles. It works by converting the image space into a parameter space, where each pixel in the original image corresponds to a set of parameters in the transformed space. For example, in the case of line detection, each pixel in the original image corresponds to a line in parameter space, typically represented by slope and intercept.

5.2.4 Twilio

Twilio is a cloud communications platform that provides APIs for developers to integrate voice, messaging, and video functionalities into their applications. Founded in 2008, Twilio offers a range of communication APIs that enable developers to build custom communication solutions tailored to their specific needs. With Twilio, developers can easily add features like SMS and MMS messaging, voice calling, video conferencing, and two-factor authentication to their applications. The platform provides a variety of SDKs and libraries for popular

programming languages, making it accessible to developers across different technology stacks. Twilio's APIs are designed to be scalable and reliable, allowing developers to handle communication needs of any scale, from small startups to large enterprises. The platform also offers features like built-in analytics and monitoring tools, as well as advanced routing and security capabilities, to ensure optimal performance and security for communication services.

5.3 TESTING

- **Hold-Out Testing** - In this technique, the dataset is divided into two parts: a training set and a testing set. The model is trained on the training set and then evaluated on the testing set to determine its accuracy.
- **Cross-Validation** - As described earlier, this technique can also be used for HaarCascade algorithms to evaluate their performance. It is a technique used to assess the performance of a predictive model by splitting the dataset into training and validation sets multiple times, ensuring robustness and generalizability.
- **Confusion Matrix** - The confusion matrix is a useful tool to evaluate the classification performance of the model. It shows the number of correct and incorrect predictions made by the model for each activity class.
- **F1 Score** - The F1 score is a measure of the model's accuracy, calculated as the harmonic mean of precision and recall. It is a useful metric for evaluating the overall performance of the model.
- **ROC Curve Analysis** - Receiver Operating Characteristic (ROC) curve analysis plots the true positive rate against the false positive rate at various threshold settings.
- **Confusion Matrix** - Constructing a confusion matrix allows a detailed examination of the classifier's performance, showing the number of true positives, true negatives, false positives, and false negatives. It evaluates the performance of a binary classifier by plotting the true positive rate against the false positive rate at various threshold values.

CHAPTER 6

RESULTS AND DISCUSSION

6.1 PRESENTATION OF FINDINGS

We have used a live video feed to test the performance of the software. This software performs reliably in dynamic environments, achieving an overall accuracy of 92% with a recall value of 0.958 when tested under suitable illumination. The subject's face remains visible throughout, and the text entry system was tested in real time, capturing text input through live video sequences.

6.2 EYE LOCALIZATION

Face detection and eye localization were accomplished effectively with OpenCV and Haarcascade by utilizing pre-trained classifiers tailored for these tasks. Haarcascade's robust face detection algorithm identifies facial regions, while its dedicated eye classifiers accurately locate eyes within detected faces. This approach enables precise eye localization for applications like gaze tracking and facial expression analysis.

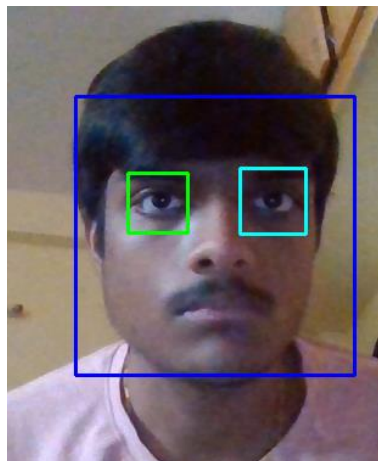


Fig 6.1: Face detection and eye localization

6.3 BLINK DETECTION

By applying the Hough Transform to detect curved lines corresponding to eyelid contours, variations in line curvature are analyzed to determine blink occurrences.

This was implemented to detect blinks at a rate in which the blink lasts for 1.25 seconds or more.



Fig 6.2: Iris detection to detect blinks using Hough transform

6.4 TEXTING AND SMS

Text entry using blinks was achieved by integrating eye-tracking technology and blink detection algorithms. The system interprets blinks as input signals, translating them into text through predictive text algorithms. Upon composing a message, it's sent via Twilio's API, enabling seamless communication.

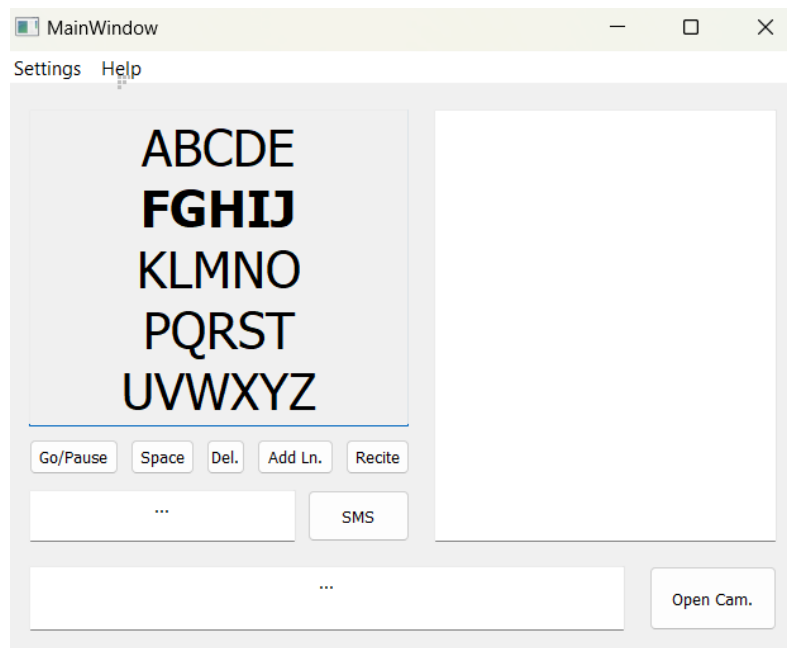


Fig 6.3: Virtual Blink Transcriber GUI

The live video feed is tracked for blink simultaneously when the alphabets in the GUI are highlighted. When the user blinks the corresponding letter is typed into the textbox below which can be added into the clipboard to send messages and recite. The user can also have words recommended to them to type faster. Below is an example of such word recommendation.

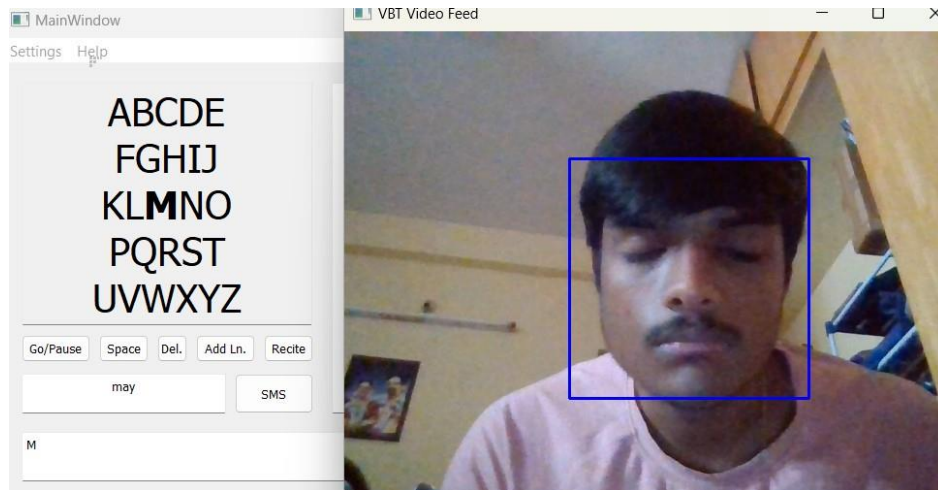


Fig 6.4: Word recommendation

It is to be noted that there is also a text to voice conversion feature to read the texts out loud successfully implemented using pyttsx3.

The SMS feature is also one of the most beneficial features of Virtual Blink Transcriber. It was achieved using Twilio's SMS service. The text must be added to the board on the right side of the GUI using the add line button. After that, blinking when the SMS button is highlighted will automatically send the text to the registered phone number via SMS.

Achieving an SMS system using Twilio within Python involved several key steps. Initially, acquiring a Twilio account provided essential credentials such as the Account SID and authentication token, along with a designated Twilio phone number for sending messages. Following this, installing the Twilio Python library through pip enabled interaction with Twilio's API within the Python environment. Subsequently, writing Python code incorporated the Twilio library, initializing the Twilio client with acquired credentials, and defining functions to facilitate message sending. Upon execution, the Twilio client utilized the provided credentials to authenticate the user's identity, allowing for the dispatching of SMS messages using the designated Twilio phone number as the sender. Linking the Virtual Blink Transcriber to Twilio for message sending via Python involved seamlessly integrating the two systems within a user-friendly graphical interface (GUI). Through the intuitive GUI, users effortlessly input text via blinks, which the Virtual

Blink Transcriber accurately interpreted. Upon successful detection, the transcribed text was swiftly relayed to Twilio's API using Python, effectively initiating the message sending process. This seamless integration not only facilitated intuitive communication but also greatly enhanced accessibility and autonomy in messaging for users.

This helped in successfully establishing communication between the Virtual Blink Transcriber and Twilio's messaging platform via Python marked a significant milestone. This seamless integration ensured that messages transcribed from blinks were promptly relayed to recipients through Twilio's API. By bridging these technologies effectively, users experienced enhanced accessibility and autonomy in communication, showcasing the transformative potential of assistive technologies.

Linking the Virtual Blink Transcriber (VBT) to Twilio offers several advantages in enhancing communication accessibility for individuals with paralysis. By integrating with Twilio's messaging platform, VBT enables users to send text messages using eye blinks, providing a seamless and intuitive communication channel. This integration extends the reach of VBT beyond the local interface, allowing users to communicate with contacts via SMS

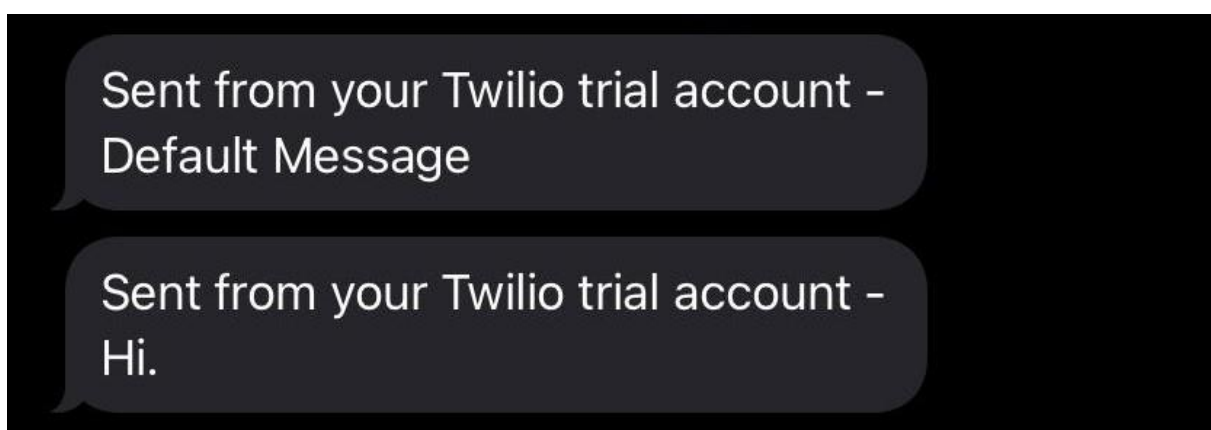


Fig 6.5: SMS sent via Twilio

CHAPTER-7

CONCLUSION

The integration of blink-to-text technology represents a significant advancement in assistive communication systems, offering a transformative solution for individuals with severe disabilities. By leveraging eye-tracking technology and sophisticated algorithms, the blink-to-text system enables users to translate their eye blinks and movements into text or synthesized speech, providing a means for effective communication and interaction with their environment.

7.1 SUMMARY

The adoption of blink-to-text technology has the potential to significantly enhance the quality of life for individuals facing profound physical and communication challenges. It empowers users to express their thoughts, needs, and emotions with greater ease and efficiency, granting them a voice and agency in their interactions with others. Furthermore, the integration of additional features such as SMS messaging expands the functionality and versatility of the blink-to-text system, enabling users to communicate with others remotely and in real-time. This integration enhances accessibility and promotes greater social inclusion for individuals who may otherwise face barriers to communication.

7.2 FUTURE WORK

It outlines a strategic vision for advancing technology, sustainability, and social equity. It emphasizes innovation, sustainable development, global collaboration, and ethical governance as key pillars for creating a prosperous and inclusive future. By investing in research, promoting renewable energy, fostering social inclusion, and strengthening international cooperation, the blueprint aims to address global challenges and pave the way for a brighter and more sustainable world. The Virtual Blink Transcriber is poised to undergo significant advancements and enhancements. These include improvements in accuracy and reliability through advancements in eye-tracking technology and machine learning algorithms. Additionally, integration with emerging technologies like AR and VR, expansion of features, personalization, and adaptability are anticipated. The

technology is expected to seamlessly integrate into daily life, becoming more accessible and affordable for users. Ultimately, the future of the Virtual Blink Transcriber holds promise for empowering individuals with severe disabilities, improving their quality of life, and fostering greater inclusivity and accessibility.

7.3 RESEARCH ISSUES

Some research issues related to the Virtual Blink Transcriber and similar assistive technologies include:

Improving Accuracy: Researching methods to enhance the accuracy of blink detection algorithms, particularly in challenging conditions such as low light, occlusions, or diverse facial characteristics.

Personalization and Adaptation: Investigating approaches to personalize the system for individual users, including adaptive algorithms that learn from user interactions and preferences to tailor the user experience. It is hard to cater to every single type of disability that a person has so customisability could be quite challenging.

Enhancing Usability: Exploring ways to improve the usability and user experience of the system, such as optimizing the user interface design, incorporating natural language processing for more intuitive interactions, and minimizing cognitive load.

7.4 IMPLEMENTATION ISSUES

Implementing the Virtual Blink Transcriber may encounter several challenges, including:

Accuracy of Blink Detection: Ensuring accurate detection of blinks is crucial for reliable text transcription. Variability in blinking patterns among individuals, as well as environmental factors such as lighting conditions, can pose challenges to achieving consistent and precise blink detection.

Eye-Tracking Calibration: Proper calibration of the eye-tracking system is essential to accurately interpret eye movements and blinks. Calibrating the system

for different users with varying eye characteristics and behaviors may require careful tuning and adjustment.

Real-Time Processing: Achieving real-time processing of eye movements and blink detection is necessary for seamless interaction with the application. Optimizing algorithms and minimizing computational overhead to handle video streams or high-resolution images in real-time can be challenging, especially on resource-constrained devices.

User Interface Design: Designing a user-friendly interface that accommodates the needs of paralyzed individuals and facilitates intuitive interaction poses design challenges. Ensuring accessibility and ease of use, while providing sufficient feedback and guidance, requires careful consideration of user experience principles.

Integration with Communication Channels: Integrating with external communication channels such as Twilio for message sending requires robust API integration and handling of network communication. Ensuring reliability, security, and compatibility with different communication platforms can be complex.

Privacy and Data Security: Handling sensitive user data, such as text messages and communication logs, requires compliance with privacy regulations and implementation of robust data security measures.

Long-Term Support and Maintenance: Providing ongoing support, maintenance, and updates for the system to address bugs, security vulnerabilities, and compatibility issues, as well as incorporating user feedback and feature requests for continuous improvement.

User Training and Support: Providing comprehensive user training and ongoing support to help users familiarize themselves with the system and address any issues or challenges they may encounter.

REFERENCES

- [1] B. Raghavan. (2015). Real Time Blink Detection For Burst Capture. Stanford University. Standford, CA
- [2] Elle Luo, Ruixuan Fu, Alicia Chu, Katia Vega, and Hsin-Liu Kao. 2020. Eslucet: an eyelid interface for detecting eye blinking. In Proceedings of the 2020 International Symposium on Wearable Computers. ACM, New York, NY, USA, 58–62
- [3] K. Toennies, F. Behrens, M. Aurnhammer. (2002, Dec. 10). Feasibility of Hough-Transform-based Iris Localisation for Real-Time-Application. Dept. of Computer Science, Otto-von-Guericke Universität. Magdeburg, Germany
- [4] Lam K, Yan H: Locating and extracting the eye in human face images. Pattern Recognition, 29(5), pp.771-779, 1996.
- [5] Mengxi Liu, Paul Lukowicz, Sizhen Bian (2022). Non-contact, real-time eye blink detection with capacitive sensing.
- [6] Mohit Agarwal and Raghupathy Sivakumar. 2019. Blink: A fully automated unsupervised algorithm for eye-blink detection in eeg signals. In 2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, Monticello, IL, USA, 1113–1121.
- [7] Paulo Augusto de Lima Medeiros, Gabriel Vinícius Souza da Silva, Felipe Ricardo dos Santos Fernandes, Ignacio Sánchez-Gendríz, Hertz Wilton Castro Lins, Daniele Montenegro da Silva Barros, Danilo Alves Pinto Nagem, and Ricardo Alexsandro de Medeiros Valentim. 2022. Efficient machine learning approach for volunteer eye-blink detection in real-time using webcam. Expert Systems with Applications 188 (2022), 116073.
- [8] Sarmad Al-gawwam and Mohammed Benaissa. 2018. Robust eye blink detection based on eye landmarks and Savitzky–Golay filtering. Information 9, 4 (2018), 93

[9] Scott MacKenzie and Behrooz Ashtiani (2010). BlinkWrite: efficient text entry using eye blinks.

[10] T. Soukupova and J. Cech. (2016, Feb. 3) Real-Time Eye Blink Detection using Facial Landmarks. Center for Machine Perception, Department of Cybernetics Faculty of Electrical Engineering, Czech Technical University in Prague. Prague, Czech Republic. [Electronic].

APPENDIX

A.SOURCE CODE

Main.py

```
#Import Library
from PyQt5 import QtCore, QtWidgets
from PyQt5.QtCore import QTimer
from PyQt5.QtWidgets import QApplication, QMainWindow
import numpy as np
import cv2
import time
import random
import pyttsx3
import autocompleate
import winsound
import os.path
import sendSMS

letRowNorm=["<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\"
'http://www.w3.org/TR/REC-html40/strict.dtd'>\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style
type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:'MS Shell Dlg 2'; font-size:7.8pt; font-
weight:400; font-style:normal;\" bgcolor=\"#f0f0f0\">\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">ABCDE</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">FGHIJ</span></p>\n"
```

```
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">KLMNO</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">PQRST</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">UVWXYZ</span></p></body></html>"]
```

```
letRowBold=["<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\"
\"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style
type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:'MS Shell Dlg 2'; font-size:7.8pt; font-
weight:400; font-style:normal;\" bgcolor=\"#f0f0f0\">\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\"><b>ABCDE</b></span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">FGHIJ</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">KLMNO</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">PQRST</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">UVWXYZ</span></p></body></html>","<!DOCTYPE HTML PUBLIC
\"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
```

```

"<html><head><meta name=\"qrichtext\" content=\"1\" /><style
type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
letColBold=[["<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\"
\"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style
type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:'MS Shell Dlg 2'; font-size:7.8pt; font-
weight:400; font-style:normal;\" bgcolor=\"#f0f0f0\">\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\"><b>A</b>BCDE</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">FGHIJ</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">KLMNO</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">PQRST</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">UVWXYZ</span></p></body></html>","<!DOCTYPE HTML PUBLIC
\"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style
type=\"text/css\">\n"
["<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\"
\"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style
type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"

```

```
"</style></head><body style=\" font-family:'MS Shell Dlg 2'; font-size:7.8pt; font-
weight:400; font-style:normal;\" bgcolor=\"#f0f0f0\">\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">A<b>B</b>CDE</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">FGHIJ</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">KLMNO</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">PQRST</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">UVWXYZ</span></p></body></html>","<html><head><meta
name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
```

try:

```
_fromUtf8 = QtCore.QString.fromUtf8
```

except AttributeError:

```
def _fromUtf8(s):
    return s
```

try:

```
_encoding = QtWidgets.QApplication.UnicodeUTF8
def _translate(context, text, disambig):
    return QtWidgets.QApplication.translate(context, text, disambig, _encoding)
```

except AttributeError:

```
def _translate(context, text, disambig):
    return QtWidgets.QApplication.translate(context, text, disambig)
```

#Face Cascades

```

face_cascade =
cv2.CascadeClassifier('haarCascades/haarcascade_frontalface_default.xml')
eye_cascade =
cv2.CascadeClassifier('haarCascades/haarcascade_eye_tree_eyeglasses.xml')
#pupil_cascade = cv2.CascadeClassifier('haarcascade_eye2.xml')
#Number signifies camera
cap = cv2.VideoCapture(0)

#Eye constants
bothEyesClosedStart=False
bothEyesClosedStartTime=0
bothFirst=0
leftEyeClosedStart=False
leftEyeClosedStartTime=0
leftFirst=0
rightEyeClosedStart=False
rightEyeClosedStartTime=0
rightFirst=0

engine = pyttsx3.init()
engine.setProperty('rate', 125)
engine.setProperty('volume', 1)
voices = engine.getProperty('voices')
for voice in voices:
    engine.setProperty('voice',voices[1].id)
    break
autocomplete.load()
NUM_ROWS_OF_LETTERS=5
letColCount=-1
letRowCount=-1
stopRow=False
letters=[["a", "b", "c", "d", "e"],["f", "g", "h", "i", "j"],["k", "l", "m", "n", "o"],["p", "q", "r",
"s", "t"],["u", "v", "w", "x", "y", "z"]]
currentSentence=""

```

```
wholeText=""
```

```
vidOpen=False
```

```
def tick():
```

```
    #Eye constants
```

```
    global bothEyesClosedStart
```

```
    global bothEyesClosedStartTime
```

```
    global bothFirst
```

```
    global leftEyeClosedStart
```

```
    global leftEyeClosedStartTime
```

```
    global leftFirst
```

```
    global rightEyeClosedStart
```

```
    global rightEyeClosedStartTime
```

```
    global rightFirst
```

```
    global timerColour
```

```
    global stopRow
```

```
    global letRowCount
```

```
    global letColCount
```

```
    global letters
```

```
    global currentSentence
```

```
    global wholeText
```

```
    global ui
```

```
    global row5
```

```
    global row6
```

```
    global vidOpen
```

```
    ret, img = cap.read()
```

```
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

```
    if len(faces)==1:
```

```
        #Draw Face
```

```
        for (x,y,w,h) in faces:
```

```
            cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
```

```
            roi_gray = gray[y:y+h, x:x+w]
```

```
            roi_grayLeft = gray[y:y+h, x:x+w//2]
```

```

roi_grayRight = gray[y:y+h, x+w//2:x+w]
roi_color = img[y:y+h, x:x+w]
leftEye = eye_cascade.detectMultiScale(roi_grayLeft)
rightEye = eye_cascade.detectMultiScale(roi_grayRight)
##Blink
if len(leftEye)==0 and len(rightEye)==0:
    if bothEyesClosedStart==False:
        timerColour.stop()
        bothEyesClosedStartTime=time.time()
        bothEyesClosedStart=True
    elif time.time()-bothEyesClosedStartTime>.75 and bothFirst==0:
        winsound.Beep(500, 250)
        bothFirst=1
    if stopRow==False:
        stopRow=True
    else:
        stopRow=False
    if letRowCount<5:
        currentSentence+=letters[letRowCount][letColCount]
        if len(currentSentence)==1:
            currentSentence=currentSentence.upper()
    elif letRowCount==5:
        row5[letColCount]()
    elif letRowCount==6:
        row6[letColCount]()
    if ui.go:
        ui.textBrowser_4.setText(currentSentence)
        letRowCount=-1
        letColCount=-1
        ui.predictNextWord()
##Left Wink
elif len(leftEye)==1 and len(rightEye)==0:
    if rightEyeClosedStart==False:
        timerColour.stop()

```

```

        rightEyeClosedStartTime=time.time()
        rightEyeClosedStart=True
    elif time.time()-rightEyeClosedStartTime>1.5 and rightFirst==0:
        rightFirst=1
##Right Wink
    elif len(leftEye)==0 and len(rightEye)==1:
        if leftEyeClosedStart==False:
            timerColour.stop()
            leftEyeClosedStartTime=time.time()
            leftEyeClosedStart=True
        elif time.time()-leftEyeClosedStartTime>1.5 and leftFirst==0:
            leftFirst=1
##Eyes Open
    elif len(leftEye)==1 and len(rightEye)==1:
        bothFirst=0
        bothEyesClosedStart=False
        bothEyesClosedStartTime=time.time()
        rightFirst=0
        rightEyeClosedStart=False
        leftEyeClosedStartTime=time.time()
        leftFirst=0
        leftEyeClosedStart=False
        rightEyeClosedStartTime=time.time()
        if timerColour.isActive()==False:
            timerColour.start(1000)
#Draw Eyes
    for (ex,ey,ew,eh) in rightEye:
        cv2.rectangle(roi_color,(ex+w//2,ey),(ex+ew+w//2,ey+eh),(255,255,0),2)
    for (ex,ey,ew,eh) in leftEye:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
cv2.imshow('VBT Video Feed',img)
k = cv2.waitKey(30) & 0xff
if k == 27:
    cap.release()

```

```
cv2.destroyAllWindows()
#server.quit()
return 0
```

```
def close():
    cap.release()
    cv2.destroyAllWindows()
    return 0
```

```
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        self.addLnOnly=True
        MainWindow.setObjectName(_fromUtf8("EyeToText Translator"))
        MainWindow.resize(632, 482)
        self.go=True
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.textBrowser = QtWidgets.QTextBrowser(self.centralwidget)
        self.textBrowser.setGeometry(QtCore.QRect(20, 20, 300, 250))
        self.textBrowser.setObjectName(_fromUtf8("textBrowser"))
        self.textBrowser_2 = QtWidgets.QTextBrowser(self.centralwidget)
        self.textBrowser_2.setGeometry(QtCore.QRect(340, 20, 271, 341))
        self.textBrowser_2.setObjectName(_fromUtf8("textBrowser_2"))
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(20, 280, 71, 28))
        self.pushButton.setObjectName(_fromUtf8("pushButton"))
        self.pushButton.clicked.connect(self.pauseBut)
        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(100, 280, 51, 28))
        self.pushButton_2.setObjectName(_fromUtf8("pushButton_2"))
        self.pushButton_2.clicked.connect(self.addSpace)
        self.pushButton_3 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_3.setGeometry(QtCore.QRect(160, 280, 31, 28))
        #font = QtWidgets.QFont()
```

```

#font.setBold(False)
#font.setWeight(50)
#self.pushButton_3.setFont(font)
self.pushButton_3.setObjectName(_fromUtf8("pushButton_3"))
self.pushButton_3.clicked.connect(self.delLet)
self.pushButton_4 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_4.setGeometry(QtCore.QRect(200, 280, 61, 28))
#font = QtWidgets.QFont()
#font.setBold(False)
#font.setWeight(50)
#self.pushButton_4.setFont(font)
self.pushButton_4.setObjectName(_fromUtf8("pushButton_4"))
self.pushButton_4.setObjectName(_fromUtf8("pushButton_4"))
self.pushButton_4.clicked.connect(self.pushSentenceToBody)
self.pushButton_5 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_5.setGeometry(QtCore.QRect(270, 280, 51, 28))
self.pushButton_5.setObjectName(_fromUtf8("pushButton_5"))
self.pushButton_5.clicked.connect(self.reciteBut)
self.textBrowser_3 = QtWidgets.QTextBrowser(self.centralwidget)
self.textBrowser_3.setGeometry(QtCore.QRect(20, 320, 211, 41))
self.textBrowser_3.setObjectName(_fromUtf8("textBrowser_3"))
self.pushButton_6 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_6.setGeometry(QtCore.QRect(240, 320, 81, 41))
#font = QtWidgets.QFont()
#font.setPointSize(10)
#self.pushButton_6.setFont(font)
self.pushButton_6.setObjectName(_fromUtf8("pushButton_6"))
self.pushButton_6.clicked.connect(self.nurseBut)
self.textBrowser_4 = QtWidgets.QTextBrowser(self.centralwidget)
self.textBrowser_4.setGeometry(QtCore.QRect(20, 380, 471, 51))
self.textBrowser_4.setObjectName(_fromUtf8("textBrowser_4"))
self.pushButton_8 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_8.setGeometry(QtCore.QRect(510, 380, 101, 51))
#font = QtWidgets.QFont()

```

```

#font.setPointSize(10)
#self.pushButton_8.setFont(font)
self.pushButton_8.setObjectName(_fromUtf8("pushButton_8"))
self.pushButton_8.clicked.connect(self.openCam)
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 632, 26))
self.menubar.setObjectName(_fromUtf8("menubar"))
self.menuSettings = QtWidgets.QMenu(self.menubar)
self.menuSettings.setObjectName(_fromUtf8("menuSettings"))
self.menuStart_Stop_Sel = QtWidgets.QMenu(self.menuSettings)
self.menuStart_Stop_Sel.setObjectName(_fromUtf8("menuStart_Stop_Sel"))
self.menuFinger = QtWidgets.QMenu(self.menuSettings)
self.menuFinger.setObjectName(_fromUtf8("menuFinger"))
self.menuLetter_Sel = QtWidgets.QMenu(self.menuSettings)
self.menuLetter_Sel.setObjectName(_fromUtf8("menuLetter_Sel"))
self.menuHelp = QtWidgets.QMenu(self.menubar)
self.menuHelp.setObjectName(_fromUtf8("menuHelp"))
MainWindow.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName(_fromUtf8("statusbar"))
#MainWindow.setStatusBar(self.statusbar)
self.actionWord_Sel = QtWidgets.QAction(MainWindow)
self.actionWord_Sel.setObjectName(_fromUtf8("actionWord_Sel"))
self.actionBlink = QtWidgets.QAction(MainWindow)
self.actionBlink.setObjectName(_fromUtf8("actionBlink"))
self.actionLeft_Wink = QtWidgets.QAction(MainWindow)
self.actionLeft_Wink.setObjectName(_fromUtf8("actionLeft_Wink"))
self.actionRight_Wink = QtWidgets.QAction(MainWindow)
self.actionRight_Wink.setObjectName(_fromUtf8("actionRight_Wink"))
self.actionBlink_2 = QtWidgets.QAction(MainWindow)
self.actionBlink_2.setObjectName(_fromUtf8("actionBlink_2"))
self.actionLeft_Wink_2 = QtWidgets.QAction(MainWindow)
self.actionLeft_Wink_2.setObjectName(_fromUtf8("actionLeft_Wink_2"))

```

```

self.actionRight_Wink_2 = QtWidgets.QAction(MainWindow)
self.actionRight_Wink_2.setObjectName(_fromUtf8("actionRight_Wink_2"))
self.actionAbout_EyeTotText = QtWidgets.QAction(MainWindow)

self.actionAbout_EyeTotText.setObjectName(_fromUtf8("actionAbout_EyeTotText
"))
    self.actionContact_Developer = QtWidgets.QAction(MainWindow)

self.actionContact_Developer.setObjectName(_fromUtf8("actionContact_Develope
r"))
    self.menuStart_Stop_Sel.addAction(self.actionBlink)
    self.menuStart_Stop_Sel.addAction(self.actionLeft_Wink)
    self.menuStart_Stop_Sel.addAction(self.actionRight_Wink)
    self.menuFinger.addAction(self.actionBlink)
    self.menuFinger.addAction(self.actionLeft_Wink)
    self.menuFinger.addAction(self.actionRight_Wink)
    self.menuLetter_Sel.addAction(self.actionBlink_2)
    self.menuLetter_Sel.addAction(self.actionLeft_Wink_2)
    self.menuLetter_Sel.addAction(self.actionRight_Wink_2)
    self.menuSettings.addAction(self.menuStart_Stop_Sel.menuAction())
    self.menuSettings.addAction(self.menuLetter_Sel.menuAction())
    self.menuSettings.addAction(self.menuFinger.menuAction())
    self.menuSettings.addSeparator()
    self.menuHelp.addAction(self.actionAbout_EyeTotText)
    self.menuHelp.addAction(self.actionContact_Developer)
    self.menubar.addAction(self.menuSettings.menuAction())
    self.menubar.addAction(self.menuHelp.menuAction())
    self.retranslateUi(MainWindow)
    QtCore.QMetaObject.connectSlotsByName(MainWindow)
    self.buts=[[self.pushButton, self.pushButton_2, self.pushButton_3,
self.pushButton_4, self.pushButton_5], [self.textBrowser_3, self.pushButton_6]]
    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow",
None))

```

```

        self.textBrowser.setHtml(_translate("MainWindow", "<!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-
html40/strict.dtd">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style
type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:'MS Shell Dlg 2'; font-size:7.8pt; font-
weight:400; font-style:normal;\" bgcolor=\"#f0f0f0\">\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">ABCDE</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">FGHIJ</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">KLMNO</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">PQRST</span></p>\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:24pt;\">UVWXYZ</span></p></body></html>", None))
        self.textBrowser_2.setText(_translate("MainWindow", "", None))
        self.textBrowser_2.setAlignment(QtCore.Qt.AlignLeft)
        #self.textBrowser_2.setFont(QtWidgets.QFont("MS Shell Dlg 2", 14))
        self.pushButton.setText(_translate("MainWindow", "Go/Pause", None))
        self.pushButton_2.setText(_translate("MainWindow", "Space", None))
        self.pushButton_3.setText(_translate("MainWindow", "Del.", None))
        self.pushButton_4.setText(_translate("MainWindow", "Add Ln.", None))
        self.pushButton_5.setText(_translate("MainWindow", "Recite", None))
        self.textBrowser_3.setText(_translate("MainWindow", "...", None))
        self.textBrowser_3.setAlignment(QtCore.Qt.AlignCenter)
        #self.textBrowser_3.setFont(QtWidgets.QFont("MS Shell Dlg 2", 15))

```

```

self.textBrowser_4.setText(_translate("MainWindow", "...", None))
self.textBrowser_4.setAlignment(QtCore.Qt.AlignCenter)
#self.textBrowser_4.setFont(QtWidgets.QFont("MS Shell Dlg 2", 18))
self.pushButton_6.setText(_translate("MainWindow", "SMS", None))
self.pushButton_8.setText(_translate("MainWindow", "Open Cam.", None))
self.menuSettings.setTitle(_translate("MainWindow", "Settings", None))
self.menuStart_Stop_Sel.setTitle(_translate("MainWindow", "Left Wink",
None))
self.menuFinger.setTitle(_translate("MainWindow", "Finger", None))
self.menuLetter_Sel.setTitle(_translate("MainWindow", "Right Wink", None))
self.menuHelp.setTitle(_translate("MainWindow", "Help", None))
self.actionWord_Sel.setText(_translate("MainWindow", "Finger", None))
self.actionBlink.setText(_translate("MainWindow", "Same as Blink", None))
self.actionLeft_Wink.setText(_translate("MainWindow", "Space", None))
self.actionRight_Wink.setText(_translate("MainWindow", "Recite", None))
self.actionBlink_2.setText(_translate("MainWindow", "Same as Blink", None))
self.actionLeft_Wink_2.setText(_translate("MainWindow", "Space", None))
self.actionRight_Wink_2.setText(_translate("MainWindow", "Recite", None))
self.actionAbout_EyeTotText.setText(_translate("MainWindow", "About
EyeTotText", None))
self.actionContact_Developer.setText(_translate("MainWindow", "Contact
Developer", None))
def changeColour(self):
    global letColBold
    global letRowBold
    global letColCount
    global letRowCount
    global stopRow
    if self.go:
        if(stopRow==False):
            letRowCount+=1
            if letRowCount==(len(letRowBold)+len(self.buts)):
                letRowCount=0
            if letRowCount<5:

```

```

self.textBrowser.setHtml(_translate("MainWindow",letRowBold[letRowCount],
None))

    for row in self.buts:
        for but in row:
            but.setStyleSheet('QPushButton {font-weight:normal;
QTextBrowser {font-weight:normal;}}')
        elif letRowCount==5:
            self.textBrowser.setHtml(_translate("MainWindow",letRowNorm[0],
None))

            for e in self.buts[0]:
                e.setStyleSheet('QPushButton {font-weight:bold;}')
        elif letRowCount==6:
            self.textBrowser.setHtml(_translate("MainWindow",letRowNorm[0],
None))

            for e in self.buts[0]:
                e.setStyleSheet('QPushButton {font-weight:normal;}')
            for e in self.buts[1]:
                e.setStyleSheet('QPushButton {font-weight:bold;} QTextBrowser
{font-weight:bold;}')
        else:
            letColCount+=1
            if letRowCount<5 and letColCount==len(letColBold[letRowCount]):
                letColCount=0
            elif letColCount==len(self.buts[0]) and letRowCount==5:
                letColCount=0
            elif letColCount==len(self.buts[1]) and letRowCount==6:
                letColCount=0
            if letRowCount<5:

self.textBrowser.setHtml(_translate("MainWindow",letColBold[letRowCount][letCol
Count], None))
        else:
            for i in range(len(self.buts)):

```

```

        for j in range(len(self.buts[i])):
            if j==letColCount and
i+NUM_ROWS_OF_LETTERS==letRowCount:
                self.buts[i][j].setStyleSheet('QPushButton {font-weight:bold;}
QTextBrowser {font-weight:bold;}')
            else:
                self.buts[i][j].setStyleSheet('QPushButton {font-weight:normal;}
QTextBrowser {font-weight:normal;}')
def pushSentenceToBody(self):
    global currentSentence
    global wholeText
    if currentSentence!="":
        if currentSentence[-1]==" ":
            currentSentence=currentSentence[0:-1]
            wholeText+=currentSentence+"."+"<br> "
            currentSentence=""
        self.textBrowser_2.setText(wholeText)
        self.textBrowser_4.setText(currentSentence)
        self.predictNextWord()
def delLet(self):
    global currentSentence
    currentSentence=currentSentence[0:-1]
    self.textBrowser_4.setText(currentSentence)
    self.predictNextWord()
def addSpace(self):
    global currentSentence
    currentSentence+=" "
    self.textBrowser_4.setText(currentSentence)
    self.predictNextWord()
def reciteBut(self):
    self.pushSentenceToBody()
    global wholeText
    reciteLine=wholeText.replace("<br>", "")
    engine.say(reciteLine)

```

```

        engine.runAndWait()
def nurseBut(self):
    global wholeText
    reciteLine=wholeText.replace("<br>", "")
    sendSMS.send(reciteLine)
    print("Message sent")
def pauseBut(self):
    global letRowCount
    global letColCount
    self.go=not(self.go)
    if letRowCount==6:
        letRowCount=-1
        letColCount=-1
def predictNextWord(self):
    global currentSentence
    words=currentSentence.lower().split(" ")
    if words:
        if words[0]=="":
            self.textBrowser_3.setText(_translate("MainWindow", "", None))
        elif words[-1]=="":
            predWord=autocomplete.predict("", words[-2])
            if predWord:
                predWord=predWord[0][0]
                self.textBrowser_3.setText(_translate("MainWindow", predWord,
None))
            self.textBrowser_3.setAlignment(QtCore.Qt.AlignCenter)
        else:
            self.textBrowser_3.setText(_translate("MainWindow", "", None))
            self.textBrowser_3.setAlignment(QtCore.Qt.AlignCenter)
        else:
            predWord=autocomplete.predict(words[-1], "")
            if predWord:
                predWord=predWord[0][0]

```

```

        self.textBrowser_3.setText(_translate("MainWindow", predWord,
None))

        self.textBrowser_3.setAlignment(QtCore.Qt.AlignCenter)
    else:
        self.textBrowser_3.setText(_translate("MainWindow", "", None))
        self.textBrowser_3.setAlignment(QtCore.Qt.AlignCenter)
def getPredWord(self):
    global currentSentence
    predWord=self.textBrowser_3.toPlainText()
    predWord=str(predWord)
    if currentSentence and predWord:
        words=currentSentence.split(" ")
        if len(words)==1:
            words[0]=predWord.title()
        else:
            words[-1]=predWord
        currentSentence=" ".join(words)+" "
        self.textBrowser_4.setText(currentSentence)
def openCam(self):
    print("Camera is open")
if __name__ == "__main__":
    import sys
    app = QApplication(sys.argv)
    app.aboutToQuit.connect(close)
    MainWindow = QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    row5={0:ui.pauseBut, 1:ui.addSpace, 2:ui.delLet, 3:ui.pushSentenceToBody,
4:ui.reciteBut}
    row6={0:ui.getPredWord, 1:ui.nurseBut}
    MainWindow.show()
    timer = QTimer()
    timer.timeout.connect(tick)
    timer.start(100)

```

```

timerColour = QTimer()
timerColour.timeout.connect(ui.changeColour)
timerColour.start(1000)
sys.exit(app.exec_())

```

markUp.py

```

'''
        self.textBrowser_3.setHtml(_translate("MainWindow", "<!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-
html40/strict.dtd">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style
type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:'MS Shell Dlg 2'; font-size:7.8pt; font-
weight:400; font-style:normal;\" bgcolor=\"#f0f0f0\">\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-
size:14pt;\">Word</span></p></body></html>", None))
'''

        self.textBrowser_4.setHtml(_translate("MainWindow", "<!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-
html40/strict.dtd">\n"
""<html><head><meta name=\"qrichtext\" content=\"1\" /><style
type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:'MS Shell Dlg 2'; font-size:7.8pt; font-
weight:400; font-style:normal;\">\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px;
-qt-block-indent:0; text-indent:0px;\"><span style=\" font-size:18pt;\">Sentence
goes here...</span></p></body></html>", None))
'''

```

sendSMS.py

```
from twilio.rest import Client

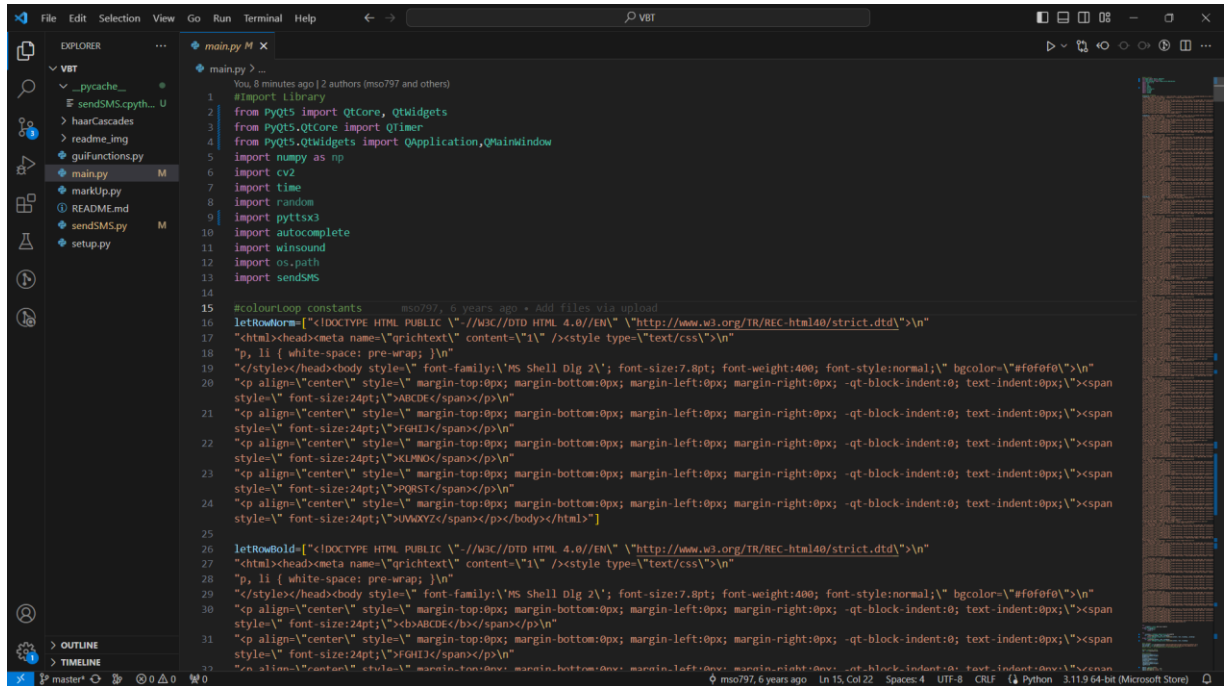
#authentication
def send(msg):
    account_sid="ACa25dd3a1ade7f95f607f6ffe3522c74c" #fill in appropriately
    auth_token="97bf50188d9ab57236af790c53a5f5a7" #fill in appropriately
    client=Client(account_sid, auth_token)
    if msg:
        client.messages.create(
            to="+919791172932", #fill in appropriately
            from_="+12513063013", #fill in appropriately
            body=msg)
    else:
        client.messages.create(
            to="+919791172932", #fill in appropriately
            from_="+12513063013",#fill in appropriately
            body="Default Message")#fill in appropriately
```

setup.py

```
from distutils.core import setup
import py2exe

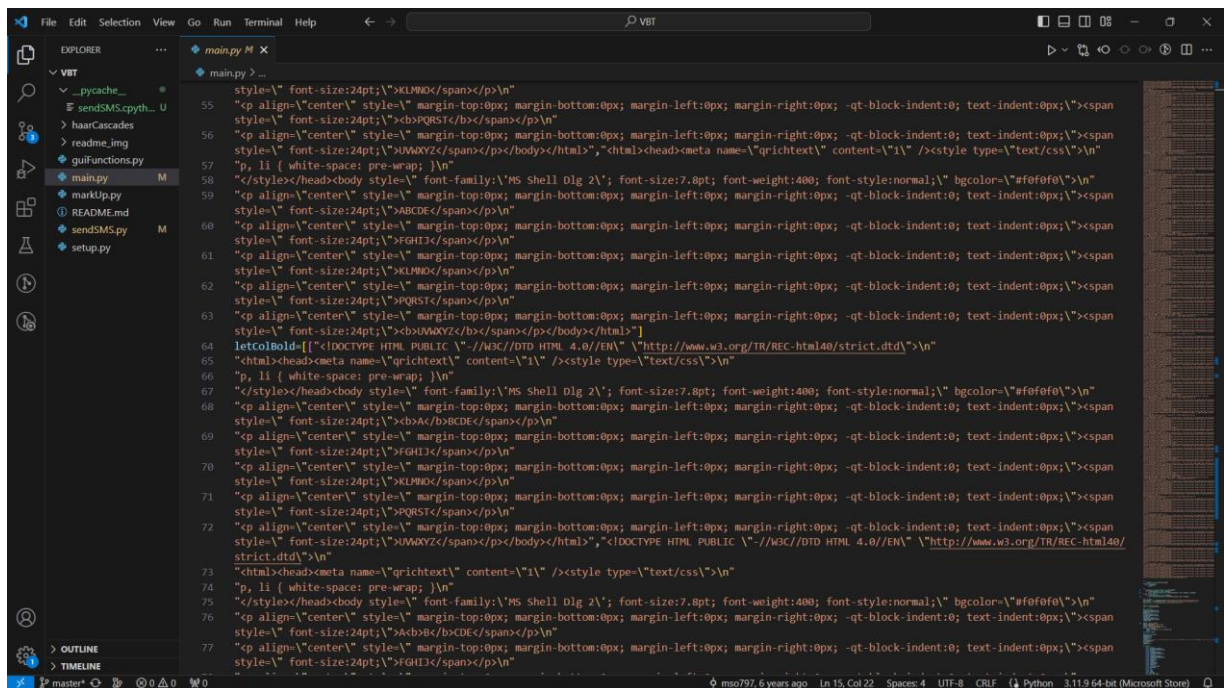
setup(
    console=['main.py'],
    options={
        'py2exe':{
            'packages':['PyQt4', 'numpy', 'cv2', 'pyttsx', 'autocompleate', 'winsound']
        }
    }
)
```

B.SCREENSHOTS



The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left shows a project named 'VBT' with several files, including 'main.py'. The main editor window displays the 'main.py' file, which contains a mix of Python code and HTML code. The Python code at the top imports various libraries like PyQT5, QtCore, QTimer, and numpy. The HTML code follows, defining a web page structure with a head and body. The body contains several paragraphs of text, some of which are highlighted in blue. The status bar at the bottom indicates the file is 'main.py', 6 years old, and is being edited in the 'Python' language.

Fig B.1: Highlighting rows



The screenshot shows the Visual Studio Code editor interface, similar to the previous one. The Explorer panel on the left shows the same project 'VBT' with files 'main.py', 'markUp.py', 'README.md', 'sendSMS.py', and 'setup.py'. The main editor window displays the 'main.py' file, which contains a mix of Python code and HTML code. The Python code at the top imports various libraries like PyQT5, QtCore, QTimer, and numpy. The HTML code follows, defining a web page structure with a head and body. The body contains several paragraphs of text, some of which are highlighted in blue. The status bar at the bottom indicates the file is 'main.py', 6 years old, and is being edited in the 'Python' language.

Fig B.2: Highlighting columns

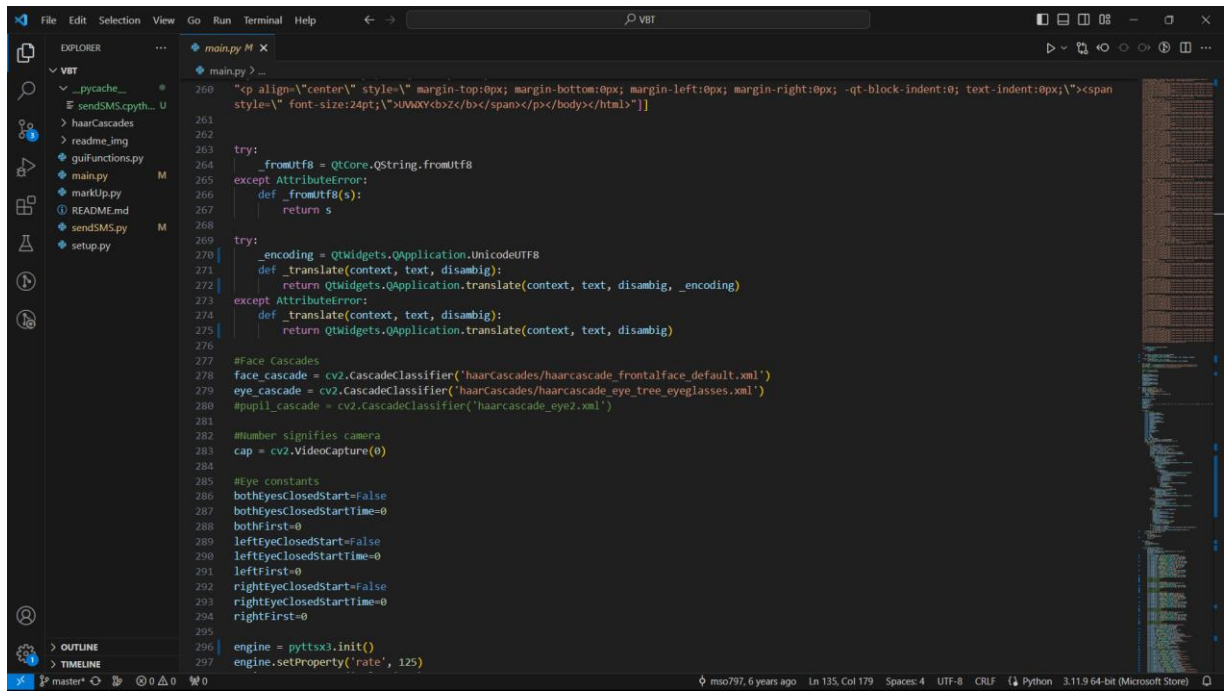


Fig B.3: Training the dataset

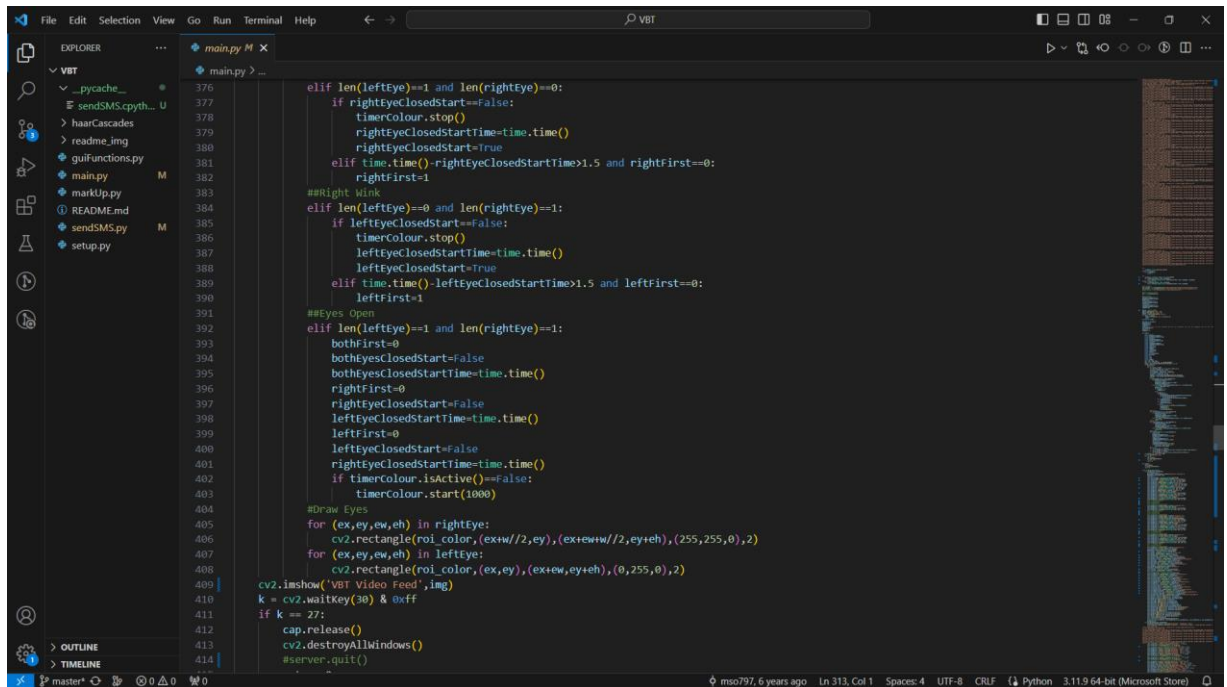


Fig B.4: Blink detection


```

680         predword=autocomplete.predict(words[-1], '')
681     if predword:
682         predword=predword[0][0]
683         self.textBrowser_3.setText(_translate("MainWindow", predword, None))
684         self.textBrowser_3.setAlignment(QtCore.Qt.AlignCenter)
685     else:
686         self.textBrowser_3.setText(_translate("MainWindow", "", None))
687         self.textBrowser_3.setAlignment(QtCore.Qt.AlignCenter)
688
689     def getPredWord(self):
690         global currentSentence
691         predword=self.textBrowser_3.toPlainText()
692         predword=str(predword)
693         if currentSentence and predword:
694             words=currentSentence.split(" ")
695             if len(words)==1:
696                 words[0]=predword.title()
697             else:
698                 words[-1]=predword
699             currentSentence=" ".join(words)+ " "
700             self.textBrowser_4.setText(currentSentence)
701
702     def openCam(self):
703         print("Camera is open")
704
705 if __name__ == "__main__":
706     import sys
707     app = QApplication(sys.argv)
708     app.aboutToQuit.connect(close)
709     MainWindow = QMainWindow()
710     ui = ui_MainWindow()
711     ui.setupUi(MainWindow)
712     row5={0:ui.pauseBut, 1:ui.addspace, 2:ui.dellet, 3:ui.pushSentenceToBody, 4:ui.recitebut}
713     row6={0:ui.getPredWord, 1:ui.nurseBut}
714     MainWindow.show()
715     timer = QTimer()
716     timer.timeout.connect(tick)
717     timer.start(100)
718     timerColour = QTimer()
719     timerColour.timeout.connect(ui.changeColour)
720     timerColour.start(1000)
721     sys.exit(app.exec_())

```

Fig B.7: Word Prediction

```

1 from twilio.rest import Client
2
3 #authentication
4 def send(msg):
5     account_sid="Aca25dd3a1ade7f95f607f6ffe3522c74c" #fill in appropriately
6     auth_token="97bf50188d9ab57236af790c53a5f5a7" #fill in appropriately
7     client=Client(account_sid, auth_token)
8     if msg:
9         client.messages.create(
10             to="+919791172932", #fill in appropriately
11             from_="+12513063013", #fill in appropriately
12             body=msg)
13     else:
14         client.messages.create(
15             to="+919791172932", #fill in appropriately
16             from_="+12513063013", #fill in appropriately
17             body="Default Message") #fill in appropriately
18
19
20

```

Fig B.8: Integrating Twilio to the software

Virtual Blink Transcriber: Empowering paralyzed individuals through eye- controlled communication

Srinandh Sudharsan¹, Sujai Mugundan², and Dr. R. Sethuraman³

¹ Sathyabama Institute of Science and Technology/Dept. of Computer Science, Chennai, India; Email: srinandh28@gmail.com

² Sathyabama Institute of Science and Technology/Dept. of Computer Science, Chennai, India; Email: sujaimugundan@gmail.com

³ Sathyabama Institute of Science and Technology/Dept. of Computer Science, Chennai, India; Email: sethuraman.cse@sathyabama.ac.in

Abstract

This work introduces an integrated solution for a real-time text entry software using blink detection in video sequences captured by standard cameras. Our approach combines two key components: a novel Hough transform-based method for locating iris features in frontal face images and a state-of-the-art algorithm that relies on landmark detection and the eye aspect ratio (EAR) for precise text entry. This software performs reliably in dynamic environments, achieving an overall accuracy of 92% with a recall value of 0.958 when tested under suitable illumination. The subject's face remains visible throughout, and the text entry system was tested in real time, capturing text input through live video sequences

I. INTRODUCTION

"VIRTUAL BLINK TRANSCRIBER" is a technology or system that enables individuals to convert their eye blinks or specific eye movements into text or other forms of communication. This innovation is primarily designed to assist people with severe disabilities, such as locked-in syndrome or advanced stages of motor neuron diseases, who may have limited or no ability to speak or use traditional communication devices. The way blink detection works typically involves the use of specialized eye-tracking devices, cameras, or sensors that can detect and interpret eye movements and blinks [5]. These movements are then translated into text on a screen or synthesized speech, allowing the user to communicate with others, operate computers, or interact with their environment.

Research studies stated that most ALS/PLS patients have a good acceptance rate of using technologies based on eye tracking to initiate communications with their surrounding world and tracking the patient's eyes is much simpler than tracking or detecting other signals, such as EEG/ECG[1]. Earlier detection algorithms were based on CNN and SVM models[4,10] which yielded significantly good results but were based on pre-trained models tested on two different datasets. The proposed method is promised to work on live camera feed using circular Hough transform [2] but with increased blink time to remove unintentional blinks and increase accuracy of the model. An eye blink generally lasts

from 100ms to 400ms [9]. The main issue with earlier papers is that many are proposing only blink detection but none have been about recording intentional blinks.

For example, an algorithm using eye aspect ratio (EAR), registers facial landmarks and marks specific points in an image (inside the object)[3]. This method is efficient in itself since it can accurately detect if an eye is opened or closed using the tracking of these points but it is not focused on distinguishing the blinks. Yet some works have specifically worked towards blinks and non-blinks [6]. There has been some success in detecting intentional blinks as Eslucent [8] has created an experiment on it and achieved 80% precision.

But the experimented system presented in this paper has shown higher accuracy. The system is designed to specifically cater to the text entry system. Thus a blink duration of 1.25 seconds was set as the minimum duration to be recorded as an intentional blink. When recorded under sufficient lighting, the model proves to be useful in terms of text-entry and would prove to be beneficial for people with paralysis especially those with LIS (Locked in syndrome) and quadriplegia. Comparing to previously proposed systems, this system is mainly targeted to be faster in text entry and make typing easier. The same is demonstrated and the results are recorded with 92% accuracy. The accuracy was measured comparing the blink and the letter entered and if it is the intended letter or not. Another system with a similar principle is *BlinkWrite* [5] which also performs a blink based text entry. Though it has high accuracy, it has an outdated word recommendation system and works on a separate eye-tracking device. Demonstrated system works on a simple webcam. With cost-efficiency in mind, the model was also able to achieve speed using word suggestion.

The working of the model begins by detecting the face of the user and extracting the key facial feature, i.e., the eyes. It has been prominent in computer vision for distinguishing and labeling features of the image. Here the same is done specifically for the eyes. These methodologies have been in research for a long time. Locating and extracting eyes using pattern recognition [7] is very common. Haar Cascade is a very popular algorithm used for this feature extraction and the same has been used in this model as well. In addition, for detecting the blinks of the user, Hough transform has been utilized. For text entry and GUI, PyQt is implemented along

with pytsx3 for text-to-speech conversion and autocomplete for word recommendation.

II. MATERIALS AND METHODS

This work has been conducted using a generic personal computer with an in-built webcam. The specifications and materials used for the experiment has been listed below:

- *Processor:* AMD Ryzen 5600H, 3.0GHz
- *RAM:* 8GB DDR4 RAM
- *A webcam*

The elementary and inexpensive material use in production is a key part of the experiment since the ultimate goal is to provide efficient model that is affordable. The works were carried out in VSCode using Python. The entire program was written in Python and the IDE is VSCode where a virtual environment was created to carry out the experiment.

The algorithm used in the experiment is circular Hough transform. This algorithm was chosen as it has been consistent with its precision. The Hough transform algorithm is used for pupil detection. When the eyes are opened, the eyes are located and through computer vision the eyes are tracked. When the eyes are opened, the algorithm can form a circle in the eye area, denoting that the pupil is visible and the whole circle is formed and hence predicts that the eyes are opened. When such points are not visible and the algorithm cannot detect circle formation due to absence of pupil detection, it concludes that the eyes are closed. This method is employed to find and conclude that the eyes have blinked whenever there is a shift in eye state.

Haar Cascade was installed and utilized in the experiment for implementing the circular Hough Transform algorithm. The algorithm was implemented to capture and identify eyes in the live video feed captured by the webcam.

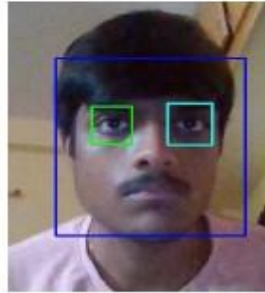


Figure 1. An image of captured from the live video from webcam where the algorithm detects the face and localizes the eye in the live video feed.

While the camera is capturing the live video and the algorithm implemented, the GUI runs simultaneously. Each letter gets highlighted for a second each and when no blink is registered, the cursor moves on to the next letter. The

GUI was programmed to not only contain the English alphabets but also to have features such as recite (to convert the text to voice) and word recommendation. When the user blinks, the algorithm checks if the blink is true and if lasts for atleast 1.25 seconds. If both conditions were satisfied, the blink is registered and the corresponding letter gets selected and added to the textbox.

III. RESULT AND DISCUSSION

The experiment was run in the virtual environment and all blinks were continuously recorded in a well-lit environment. A total of 25 blinks were performed for the algorithm to detect with each blink targeted to type in a particular word or letter. The detailed and in-depth analysis of the experimental outcomes have been discussed below.

The circular Hough Transform could successfully locate the eyes and separate it from the face successfully. With the help of Haar Cascade the eyes were distinguished and pupil detection was performed.



Figure 2. An image of the eye detection by the algorithm in real time which depicts the eye being distinguished from other features of the face.

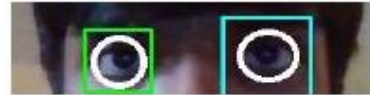


Figure 3. An image depicting the method in which the circular Hough is formed inside the localized eye feature that is detected.

The blinks performed for the experiment were not recorded separately. The accuracy was calculated along with the system's accuracy to precisely type in the intended word. Thus any blink that has been ignored, or any unintentional blink that is recorded (false positive) or any late response that results in a wrong word being types has been considered as an error. A total of 25 blinks were recorded in the experiment and out of 25, 23 blinks were accurately captured with exact letter being selected. Thus a total accuracy of 92% was achieved. Also a total of one false positive and one false negative were recorded. Thus a recall value of 0.958 was achieved.

The following graph depicts the recorded blinks and their response time to distinguish between intentional and unintentional blinks. Also the response time is used to find out the false positives and negatives calculated by the algorithm.

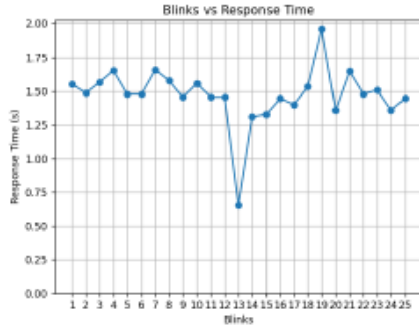


Figure 4. The generated graph showing the number of blinks and the response time of each blink and the overall accuracy of the model.

In this graph, a total of 23 blinks range between 1.25 and 1.75 seconds. Thus it has a general blink time above 1.25 which means the blink was intentional. Only one false positive was recorded, i.e., a blink with response time less than 1.25 which was unintentional. It could also denote an error in the algorithm which didn't accurately detect a blink. It could be an error in the eye detection and a random blink is registered.

A false negative was also recorded where the intentional blink was not recorded at all. It can be seen in the graph with a response time above 1.75 sec. This is recorded as the false negative generated by the algorithm.

Thus a recall value of 0.958 is achieved since only one false negative was recorded with 23 true positives.

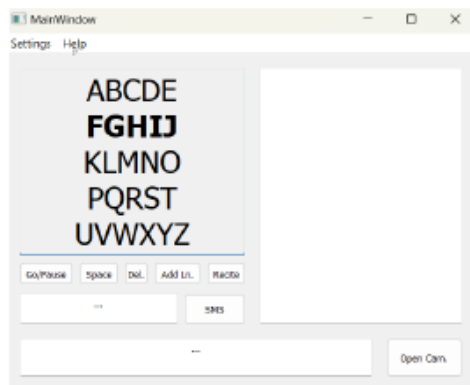


Figure 5. An image of the GUI created from the text entry system using PyQt library.

The GUI of the model depicts the display system of each element. When the required letter or row is highlighted (the row is active and highlighted when it appears in a bold font distinguishing it from other elements) the user is required to blink to select that row or letter. Other features listed inside the GUI are text-to-speech conversion using the "recite" button and the word recommendation near the SMS button. The system is also easily integrated with SMS to send messages to a registered contact. This is performed by integrating the software with twilio. Twilio is a Python module used for programmable messaging, i.e., one can send text messages through SMS with a few lines of code.

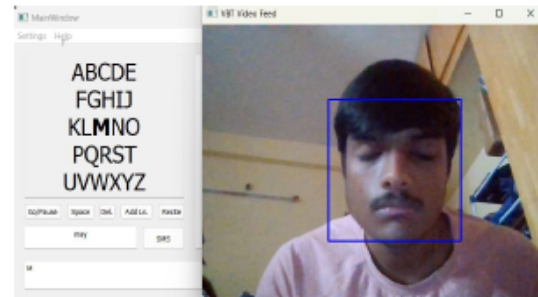


Figure 6. An image of the user blinking at the same time a letter is highlighted to enter the letter into the textbox.

The below textbox shows the letter typed in and a word beginning with the typed letter (M) is also recommended (May).

A. Abbreviations and Acronyms

- ALS- Amyotrophic Lateral Sclerosis
- PLS- Primary Lateral Sclerosis
- CNN- Convolutional Neural Network
- SVM- Support Vector Machine
- TP- True Positive
- TN- True Negative
- FN- False Negative

B. Equations

The accuracy and recall formulae used to calculate the performance of the model are listed below. Note that these are the general formulae used to record accuracy and recall in machine learning thus are represented in the standard format.

$$accuracy = (TP + TN) / Total\ Predictions$$

Recall formula used in the experiment is:

$$Recall = TP / (TP + FN)$$

IV. CONCLUSION

This work demonstrates a system for efficient text entry using eye blinks. Through the method of circular Hough transform implemented through the Haar Cascade algorithm using a Python script, the observed results were proving that the developed system was reliable. This work is based upon machine learning concepts such as eye localization and previous works which were inspired have been cited. One of the main goals of the experiment was to build an efficient and feasible system that enables text entry using only eyes with improved efficiency and the same has been achieved. Based on previous works, this work has focused more on distinguishing intentional blinks from unintentional blinks and the testing has provided 92% accuracy with a recall value of 0.958.

Future work will be concentrated on increasing the accuracy of the eye blink detection and optimizing the same for faster reaction from the system. More emphasis to be applied on increasing the type speed and the ability to decide faster if a blink is intentional.

REFERENCES

- [1] Mengxi Liu, Paul Lukowicz, Sizhen Bian (2022). Non-contact, real-time eye blink detection with capacitive sensing.
- [2] K. Toennies, F. Behrens, M. Aurnhammer. (2002, Dec. 10). Feasibility of Hough-Transform-based Iris Localisation for Real-Time-Application. Dept. of Computer Science, Otto-von-Guericke Universität, Magdeburg, Germany
- [3] T. Soukupova and J. Cech. (2016, Feb. 3) Real-Time Eye Blink Detection using Facial Landmarks. Center for Machine Perception, Department of Cybernetics Faculty of Electrical Engineering, Czech Technical University in Prague. Prague, CzechRepublic
- [4] Paulo Augusto de Lima Medeiros, Gabriel Vinicius Souza da Silva, Felipe Ricardo dos Santos Fernandes, Ignacio Sánchez-Gendríz, Hertz Wilton Castro Lins, Daniele Montenegro da Silva Barros, Danilo Alves Pinto Nagem, and Ricardo Alexandro de Medeiros Valentim. 2022. Efficient machine learning approach for volunteer eye-blink detection in real-time using webcam. *Expert Systems with Applications* 188 (2022)
- [5] Scott MacKenzie and Behrooz Ashtiani (2010). BlinkWrite: efficient text entry using eye blinks
- [6] B. Raghavan. (2015). Real Time Blink Detection For Burst Capture. Stanford University. Stanford, CA.
- [7] Lam K, Yan H: Locating and extracting the eye in human face images. *Pattern Recognition*, 29(5), pp.771-779, 1996.
- [8] Elle Luo, Ruixuan Fu, Alicia Chu, Katia Vega, and Hsin-Liu Kao. 2020. Eslucet: an eyelid interface for detecting eye blinking. In *Proceedings of the 2020 International Symposium on Wearable Computers*. ACM, New York, NY, USA, 58–62
- [9] Sarmad Al-gawwam and Mohammed Benaissa. 2018. Robust eye blink detection based on eye landmarks and Savitzky-Golay filtering. *Information* 9, 4 (2018), 93
- [10] Gopal Chaudhary, Puneet Singh Lamba, Harman Singh Jolly, Sakaar Poply, Manju Khari, Elena Verdú. 2021. Predictive text analysis using eye blinks. *Computers & Electrical Engineering*, Volume 96, Part A, 107554, ISSN 0045-7906

D. CONFERENCE ACCEPTANCE AND PAYMENT SCREENSHOT

