AI-BASED INVISIBLE HAND FOR DISABLED PEOPLE

A DESIGN PROJECT REPORT

submitted by

SUVATHI R

SWETHALASHMI G

VARSHA RK

VARSHITA M

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

K RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai, Approved by AICTE, New Delhi)

Samayapuram – 621 112

JUNE 2025

AI-BASED INVISIBLE HAND FOR DISABLED PEOPLE

A DESIGN PROJECT REPORT

submitted by

SUVATHI R (811722104163)

SWETHALASHMI G (811722105167)

VARSHA RK (811722104173)

VARSHITA M (811722104176)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

K RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai, Approved by AICTE, New Delhi)

Samayapuram – 621 112

JUNE 2025

K RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report titled **"AI-BASED INVISIBLE HAND FOR DISABLED PEOPLE"** is Bonafide work of **SUVATHI R (811722104163), SWETHALASHMI G (811722104167), VARSHA RK (811722104173),VARSHITA M (811722104176)** who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE
Dr. A Delphin Carolina Rani, M.E.,Ph.D.,
HEAD OF THE DEPARTMENT
PROFESSOR
Department of CSE
K Ramakrishnan College of Technology
(Autonomous)
Samayapuram – 621 112

SIGNATURE
Ms.V.Sowmiya., M.E.,
SUPERVISOR
ASSISTANT PROFESSOR
Department of CSE
K Ramakrishnan College of Technology
(Autonomous)
Samayapuram – 621 112

Submitted for the viva-voice examination held on ……

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

We jointly declare that the project report on **"AI-BASED INVISIBLE HAND FOR DISABLED PEOPLE"** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI"** for the requirement of Degree of Bachelor of Engineering. This project report is submitted on the partial fulfilment of the requirement of the award of Degree of Bachelor of Engineering.

Signature

SUVATHI R

_____

SWETHALASHMI G

_____

VARSHA RK

_____

VARSHITA M

_____

Place: Samayapuram

Date:

ACKNOWLEDGEMENT

It is with great pride that we express our gratitude and indebtedness to our institution **"K RAMAKRISHNAN COLLEGE OF TECHNOLOGY"**, for providing us with the opportunity to do this project.

We are glad to credit and praise our honorable and respected chairman sir **Dr. K RAMAKRISHNAN, B.E.,** for having provided for the facilities during the course of our study in college.

We would like to express our sincere thanks to our beloved Executive Director **Dr. S KUPPUSAMY, MBA, Ph.D.,** for forwarding our project and offering adequate duration to complete it.

We would like to thank **Dr. N VASUDEVAN, M.Tech., Ph.D.,** Principal, who gave opportunity to frame the project with full satisfaction.

We heartily thank **Dr. A DELPHIN CAROLINA RANI, M.E., Ph.D.,** Head of the Department, **COMPUTER SCIENCE AND ENGINEERING** for providing her support to pursue this project.

We express our deep and sincere gratitude and thanks to our project guide **Ms.V.SOWMIYA., M.E.,** Department of **COMPUTER SCIENCE AND ENGINEERING,** for her incalculable suggestions, creativity, assistance and patience which motivated us to carry our this project.

We render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course. We wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

ABSTRACT

Physically disabled individuals often face challenges in interacting with digital devices due to limited motor capabilities, particularly in controlling a mouse or keyboard. Traditional assistive technologies often require expensive hardware, wearable sensors, or invasive setups, which are not easily accessible or affordable.This project proposes an AI-based solution named “Invisible Hand for Disabled People”, which allows users to control computer actions using only facial gestures captured through a standard webcam. By detecting real-time facial landmarks, the system interprets gestures such as eye blinks for left/right clicks, open mouth for scroll mode, and head movements for cursor navigation.The Proposed system leverages Computer Vision, Dlib, OpenCV, and PyAutoGUI for gesture detection and action mapping. To simulate intelligent behavior, the system is designed to use a pre-trained LSTM model for sequence prediction, allowing the recognition of dynamic gesture patterns over time.This hands-free, hardware-independent approach enhances digital accessibility for disabled users. The proposed system is cost-effective, AI-driven, and easy to deploy, promoting independence and inclusivity in human-computer interaction.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACRONYM		ABBREVIATION
AI	-	Artificial Intelligence Long
LSTM	-	Short Term Memory Brain
BCI	-	Computer Interface
EEG	-	Electroencephalogram
CNN	-	Convolutional Neural Networks Graphics
GPU	-	Graphics Processing Unit
EAR	-	Eyes Aspect Ratio
BGR	-	Blue,Green,Red
RGB	-	Red,Green,Blue
RNN	-	Recurrent Neural Network
STT	-	Speech-to-Text
VSCode	-	Visual Studio Code

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

In today's digital era, the ability to interact with computers is essential for communication, learning, and productivity. However, individuals with physical disabilities often face significant barriers when using conventional input devices such as a keyboard or mouse. While various assistive technologies like voice recognition and eye-tracking systems have been developed, many remain costly, complex to operate, or unsuitable for all environments.Voice-controlled systems, for instance, may not function effectively in noisy surroundings or be useful for users with speech impairments. Similarly, hardware- based solutions often require specialized equipment and maintenance, limiting their accessibility and affordability.

Recent advancements in computer vision and machine learning have paved the way for intelligent, real-time gesture recognition systems using just a webcam. These technologies can interpret subtle facial movements and convert them into digital commands, enabling hands-free interaction.The project **"**AI-Based Invisible Hand for Disabled People**"** builds on this innovation by using facial gestures like eye blinks, head tilts, and smiles to simulate mouse actions. With the help of OpenCV, Mediapipe, and LSTM models, the system offers an accessible and cost-effective solution, promoting digital inclusion for users with limited motor control.

1.2 OVERVIEW

The project titled **"**AI-Based Invisible Hand for Disabled People**"** presents a real-time, AI-powered gesture recognition system designed to assist individuals with physical disabilities in operating a computer without using traditional input devices. It enables hands-free control by detecting facial gestures such as eye blinks, head tilts, and smiles, and translating them into mouse actions like clicking, cursor movement, and scrolling.

The system leverages OpenCV, Mediapipe, and Dlib for facial landmark detection using a standard webcam. A LSTM neural network is employed to accurately classify intentional gestures based on temporal patterns. These gestures are then mapped to corresponding mouse functions through PyAutoGUI, allowing users to interact with the system intuitively and efficiently.

Designed to be affordable and hardware-independent, the system eliminates the need for specialized equipment, making it accessible and practical. With applications in assistive technology, gaming, and hands-free environments, the project demonstrates how artificial intelligence can bridge the accessibility gap and promote digital independence for users with limited motor control.

1.3 PROBLEM STATEMENT

Despite rapid advancements in computing and digital accessibility, individuals with physical disabilities continue to face challenges in interacting with computers using conventional input devices such as keyboards and mice. These tools require fine motor control, which is not feasible for users affected by conditions like paralysis, muscular dystrophy, or neurological disorders. Although alternative solutions such as voice recognition, eye-tracking systems, and hardware-based assistive tools exist, they are often limited by high cost, environmental constraints, and poor adaptability.

Voice-controlled systems may not function effectively in noisy or silent environments and are unsuitable for users with speech impairments. Similarly, specialized hardware devices require installation, calibration, and ongoing maintenance, making them impractical for widespread adoption. As a result, a significant portion of the differently-abled population remains digitally excluded, unable to access educational resources, communication platforms, and productivity tools.

There is a critical need for a cost-effective, intelligent, and hands-free system that allows users with motor impairments to operate a computer efficiently without physical contact or voice commands. The system must be adaptive, user-friendly, and require minimal hardware to ensure accessibility and practicality.

This project aims to address this problem by developing an AI-powered system that leverages facial gestures—such as eye blinks, head tilts, and smiles—for performing essential mouse actions. By utilizing computer vision and deep learning, the system offers a reliable and affordable solution that promotes digital inclusion and empowers users with physical limitations to interact with technology independently.

1.4 OBJECTIVE

1. To design a real-time gesture recognition system that captures and interprets facial gestures such as eye blinks, head tilts, and smiles using computer vision techniques.

2. To implement a deep learning-based classification model (LSTM) capable of accurately distinguishing between involuntary and intentional gestures for robust gesture prediction.
3. To simulate essential mouse operations like cursor movement, clicking, and scrolling by mapping recognized gestures to corresponding screen control actions using PyAutoGUI.
4. To develop a low-cost, hardware-independent solution that requires only a standard webcam, ensuring affordability and ease of use for users with motor impairments.

1.5 IMPLICATION

The AI-Based Invisible Hand is an innovative assistive technology that allows hands-free computer control using facial gestures, specifically designed for individuals with motor impairments. By using AI and computer vision, it offers real-time, responsive interaction without the need for expensive hardware or voice commands. This makes it a cost-effective and user-friendly solution that can be used at home, in the workplace, or in healthcare environments.

Beyond aiding the disabled, this technology supports the concept of universal design by making digital access more inclusive. Its adaptability and touchless control features make it suitable for sterile environments and industrial automation as well. Overall, it bridges the gap between physical limitations and digital accessibility, promoting equal participation in today's technology-driven world.

CHAPTER 2

LITERATURE SURVEY

1. Eye Blink Detection Using Facial Landmarks, Tereza Soukupova and Jan Cech – 2016

This paper [1] introduces an efficient and lightweight method for detecting eye blinks using facial landmark detection, particularly through Dlib's 68-point facial landmark model. The study presents the concept of the Eye Aspect Ratio (EAR), which is calculated using six landmark points around each eye. The EAR remains relatively constant when the eye is open but drops significantly when the eye closes, making it a reliable measure for blink detection. By analyzing continuous EAR values from video frames, the authors created a robust algorithm that triggers a blink event when the EAR falls below a predefined threshold. This approach provides a non-intrusive and hardware-light solution suitable for real-time applications.The algorithm exhibits high accuracy in well-lit and stable conditions, ensuring a responsive user experience. The use of EAR allows the system to detect even partial blinks and works continuously without requiring manual calibration between frames.

However, the authors acknowledge a few limitations that can affect system performance. The blink detection algorithm may suffer inaccuracies in poor lighting environments, with occluded faces, or when the user wears glasses that obstruct the eye region. Additionally, excessive head movement or unusual facial orientations can disturb the EAR measurement, resulting in missed or false detections. Despite these challenges, this research has significantly influenced further developments in facial gesture recognition, and the EAR-based blink detection method has become a foundation for many real-time HCI applications and smart surveillance systems.

2. Face Mouse: A Human-Computer Interface for the Disabled, Amit Kumar Singh et al. – 2017

This pioneering paper [2] introduces an image processing-based human-computer interface designed specifically for users with physical disabilities who may be unable to use traditional input devices such as mice or keyboards. The system

employs pattern recognition techniques to detect specific facial gestures and movements which are then mapped to mouse cursor control actions. The approach primarily focuses on identifying facial features through video input and interpreting their motion to move the cursor in real time. One of the key strengths of this method is its simplicity and cost-effectiveness, as it requires only a standard camera without any additional hardware, making it accessible for widespread use.

However, the system faces challenges in maintaining accuracy and robustness, especially under dynamic and uncontrolled lighting conditions. Variations in ambient light often cause inconsistencies in facial tracking. Moreover, the system's performance degrades when the user moves their head beyond a certain range, limiting the effective field of interaction. Despite these limitations, the study highlights the potential for improving computer accessibility for disabled users through vision-based techniques and lays foundational work for future developments in this domain.

3. A Smart Vision-Based Mouse Control System for Physically Disabled Users, Shubhadeep Roy and Anil Kumar – 2018

In this work [3], the authors propose a smart vision-based mouse control system leveraging OpenCV along with a standard webcam to enable physically disabled users to operate a computer cursor without manual input devices. The system captures facial movements in real time, such as head tilts and directional shifts, and translates these into corresponding mouse cursor movements and clicks. A significant contribution of this work is the creation of a hands-free interaction model that does not rely on expensive or specialized hardware, thereby enhancing accessibility and affordability. The system also emphasizes real-time responsiveness, which is critical for effective user experience.

However, the authors identify key limitations related to the quality of the input video; low-resolution cameras or laggy video streams may reduce the accuracy and smoothness of cursor control. Additionally, real-time processing demands can strain computational resources on low-end machines, causing potential delays. This research contributes to assistive technology by demonstrating a practical implementation of

facial gesture recognition for cursor control and paves the way for further optimization of real-time computer vision applications in accessibility.

4. Real-Time Hands-Free Mouse Control Using Facial Features, Manikandan S. and Karthikeyan R. – 2019

This paper [4] introduces a facial landmark tracking system for real-time, hands-free mouse control by focusing on the detection and movement of the nose tip using OpenCV. The authors use advanced facial feature detection algorithms to identify the nose tip location continuously, converting its movement into mouse cursor direction. The system is designed to assist users with upper limb disabilities by eliminating the need for physical interaction with input devices. The method prioritizes real-time processing speed to ensure that cursor movements are smooth and responsive.

While the approach shows promising results, it encounters some limitations in tracking accuracy due to rapid or abrupt head movements, which can cause cursor jitter or unintended motions. Background clutter and environmental factors such as variable lighting also impact the robustness of facial feature detection. The paper contributes to the field by demonstrating how facial landmark tracking can be leveraged for assistive device control, though it underscores the need for enhanced filtering and stabilization techniques to improve usability under real-world conditions..

5. Human-Computer Interaction for Disabled Using Facial Gestures, Mohan Kumar M. and Bhuvaneshwari M. – 2020

This study [5] advances facial gesture-based human-computer interaction by integrating Dlib's facial landmark detector with OpenCV to calculate key facial metrics, namely Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR). These metrics are utilized to interpret user gestures such as eye blinks and mouth movements, which are mapped to mouse clicks and scroll commands respectively. The dual use of EAR and MAR enables multifunctional gesture recognition, allowing a richer set of commands through simple, intuitive actions.

This system particularly addresses the needs of disabled users who rely on subtle facial expressions for computer control. An important contribution of this work is the

adaptability offered by gesture calibration, which adjusts the threshold values for EAR and MAR to individual users, thus accounting for natural variability in facial anatomy and expression. Nonetheless, the system requires users to undergo a calibration phase to tailor the recognition thresholds, which may add complexity and affect user convenience. Additionally, gesture variability among different users can challenge the system's universality. Overall, this work demonstrates a significant step towards more personalized and effective assistive interfaces using facial gesture recognition.

6. Eye Controlled Human-Computer Interaction System, Sneha Shukla and Ankita Tiwari – 2020

In this research [6], the authors propose a system that harnesses eye movements and blink detection for computer control using Haar Cascade classifiers within OpenCV. The system works by locating eyes in the camera frame and detecting blink events to simulate mouse clicks, enabling hands-free interaction primarily aimed at users with severe physical impairments. The use of Haar Cascade classifiers provides a fast and lightweight detection method, making the system feasible for real-time applications on standard hardware.

Despite these advantages, the approach suffers from inconsistent detection accuracy, especially under varying lighting conditions which affect image contrast and feature clarity. This limitation occasionally results in missed clicks or false positives, which can reduce the reliability of the interface for precise tasks. The paper highlights the trade-off between computational simplicity and detection robustness, suggesting that further improvements are necessary for stable performance in diverse environments. Nevertheless, the system presents a promising and practical solution for basic navigation and accessibility.

7. Head Gesture Recognition for Mouse Control Using Neural Networks, Priyanka V. and Ramesh R. – 2020

This study [7] explores the application of neural networks to recognize head gestures such as nodding, shaking, and tilting for mouse control. The authors design a shallow neural network trained on a webcam-captured dataset of head movements to

classify gestures that correspond to mouse events like cursor movement and clicking. The neural network's ability to learn user-specific patterns improves over time, allowing the system to adapt to individual differences in gesture style and frequency. This adaptive learning enhances user experience by reducing misclassification rates in continuous use.

However, the system requires an initial training phase for each user, which may be time-consuming and inconvenient. Additionally, subtle or ambiguous gestures can still be misclassified, causing unintended mouse actions. The paper contributes to assistive technology by demonstrating how machine learning models can enable more intelligent and customizable gesture recognition for hands-free computer operation, while also identifying challenges related to training overhead and gesture ambiguity.

8. Implementation of a Virtual Mouse Based on Facial Expressions, Satish R. and Nandhini R. – 2021

This paper [8] proposes an innovative virtual mouse system based on facial expression recognition using Convolutional Neural Networks (CNNs). The system captures facial expressions via a standard webcam and uses CNN models trained to detect expressions such as smiles, eye blinks, and eyebrow raises. These expressions are then mapped to mouse functions such as clicking, moving, and scrolling with the help of PyAutoGUI. The use of CNNs significantly improves the accuracy and robustness of expression detection compared to traditional image processing methods.

However, this comes at the cost of increased computational complexity, which may limit the system's usability on low-performance devices. The paper emphasizes the system's interactive and engaging nature, offering physically challenged individuals a more expressive and flexible way to interact with computers. This research highlights the potential of deep learning-based facial expression recognition in assistive technologies, balancing accuracy gains against resource demands.

9. Gesture Recognition Using LSTM Networks, Ahmed Jalal et al. – 2021

In this work [9], the authors apply Long Short-Term Memory (LSTM) networks, a type of recurrent neural network designed for sequential data, to the problem of

gesture recognition from video streams. The system processes temporal sequences of facial gestures, enabling the recognition of continuous, dynamic gestures such as nodding, blinking, or smiling over time. This temporal modeling improves the system's ability to distinguish between intentional gestures and transient facial movements, enhancing prediction accuracy and responsiveness. The use of LSTMs also allows the model to capture contextual information from past frames, which is critical for understanding complex gesture sequences.

However, the system requires extensive labeled datasets and significant computational resources for training, which may limit its immediate deployment in resource-constrained settings. Despite these challenges, the research demonstrates the effectiveness of LSTM-based temporal modeling for advanced, time-sensitive gesture recognition in assistive human-computer interfaces.

10. Real-Time Facial Gesture Controlled Interface Using Dlib and OpenCV, R. Srinivasan and B. Deepa – 2022

This recent study [10] proposes a robust and practical real-time interface that integrates Dlib's facial landmark detection with OpenCV to extract Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) values for facial gesture recognition. By interpreting blinking as mouse clicks and mouth opening as scrolling commands, the system provides a natural and intuitive hands-free interaction method. The combination of Dlib and OpenCV leverages the strengths of both libraries to improve accuracy and speed, while requiring only minimal hardware such as a standard webcam. The system performs reliably across various lighting conditions and environments, making it suitable for practical applications.

However, individual differences in facial structure and expression intensity necessitate periodic recalibration to maintain optimal performance. Rapid facial movements or exaggerated expressions can sometimes cause detection errors, indicating areas for future improvement. This work represents a significant advancement in real-time, gesture-based assistive interfaces by balancing accuracy, usability, and hardware simplicity.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The existing system widely adopted in assistive technologies for disabled users is based on voice-controlled interfaces. These systems enable hands-free operation by recognizing and responding to spoken commands.The interaction begins when the user speaks a command aloud (e.g., “Open browser” or “Play music”). This speech is captured through a microphone and converted into audio signals. This process is the primary mode of input in traditional voice-based systems.The captured audio is transmitted to a voice assistant engine which uses STT algorithms to convert the audio waveform into readable text. This stage is critical for interpreting user commands accurately.The converted text is processed using NLP techniques. NLP helps in understanding the intent behind the command by analyzing grammar, structure, and keywords. It may involve accessing a backend database to retrieve relevant information or verify actions.Once the command is interpreted, the system generates a response which can be a voice reply, an on-screen output, or an action like opening an application. This ensures the user receives feedback confirming the task is performed.

3.1.1 Drawbacks of the Existing System

- Ineffective for users with speech impairments.
- Prone to errors in noisy environments.
- Sensitive to accents and language differences.

3.2 PROPOSED SYSTEM

The proposed system introduces an AI-driven, gesture-based interface that allows physically disabled users to interact with a computer using facial movements such as eye blinks, smiles, and head tilts captured in real-time through a standard webcam. The system is designed to be cost-effective, hands-free, and highly accurate in interpreting user intent through deep learning.

1. Input Modalities:

The system operates using three primary facial-based input modalities that are designed to be intuitive and accessible, especially for users with motor impairments. Firstly, eye blinks are utilized to simulate mouse clicks—for instance, a double blink can trigger a left-click action. Secondly, facial gestures such as smiling are employed to initiate commands like scrolling. Lastly, head movements, specifically head tilts, are used to control the direction and movement of the mouse cursor.

2. Webcam Input Capture:

A standard webcam captures real-time video data from the user. This video feed is the primary input to the system. The captured frames are used for detecting facial landmarks and monitoring subtle changes in facial expressions and head orientation.

3. Facial Landmark Detection and Feature Extraction

The system employs Mediapipe and Dlib to perform accurate facial landmark detection and feature extraction. Initially, the face is detected, and key landmarks such as the eyes, mouth, and nose are precisely located. From these landmarks, essential features are extracted—such as the Eye Aspect Ratio (EAR) to determine eye openness, mouth openness for detecting gestures like smiling or speaking, and tilt angle for identifying head movements..

4. Preprocessing Pipeline:

The preprocessing stage is crucial for refining raw input data before feeding it into the model. It begins with noise reduction and normalization, which help eliminate inconsistent landmark detections and standardize feature values for consistent interpretation. Next, the data is structured into time-series sequences that capture the temporal patterns of gestures, enabling the model to understand how gestures evolve over time.

5. Gesture Prediction using LSTM

Transactions are verified in real time by the network before they are recorded, ensuring that only valid transactions are added to the blockchain. The decentralized ledger offers complete transparency, allowing authorized participants to view transaction details without the risk of unauthorized alterations or data breaches.

6. AI-Based Cursor Control

The system incorporates an AI-based control layer that translates recognized gestures into real-time mouse operations using PyAutoGUI. Once a gesture is predicted, specific actions are triggered—such as a blink initiating a click, a head tilt guiding cursor movement, and an open mouth gesture activating scrolling. This layer functions as the crucial bridge between gesture recognition and practical system response, enabling users to interact with the computer in a seamless, hands-free manner.

7. Hands-Free Navigation

The final result is a fully functional, gesture-controlled interface that allows users to navigate, click, and scroll on a computer without using their hands. The system offers a seamless, real-time, and accessible alternative to traditional input devices.

3.3 BLOCK DIAGRAM OF PROPOSED SYSTEM

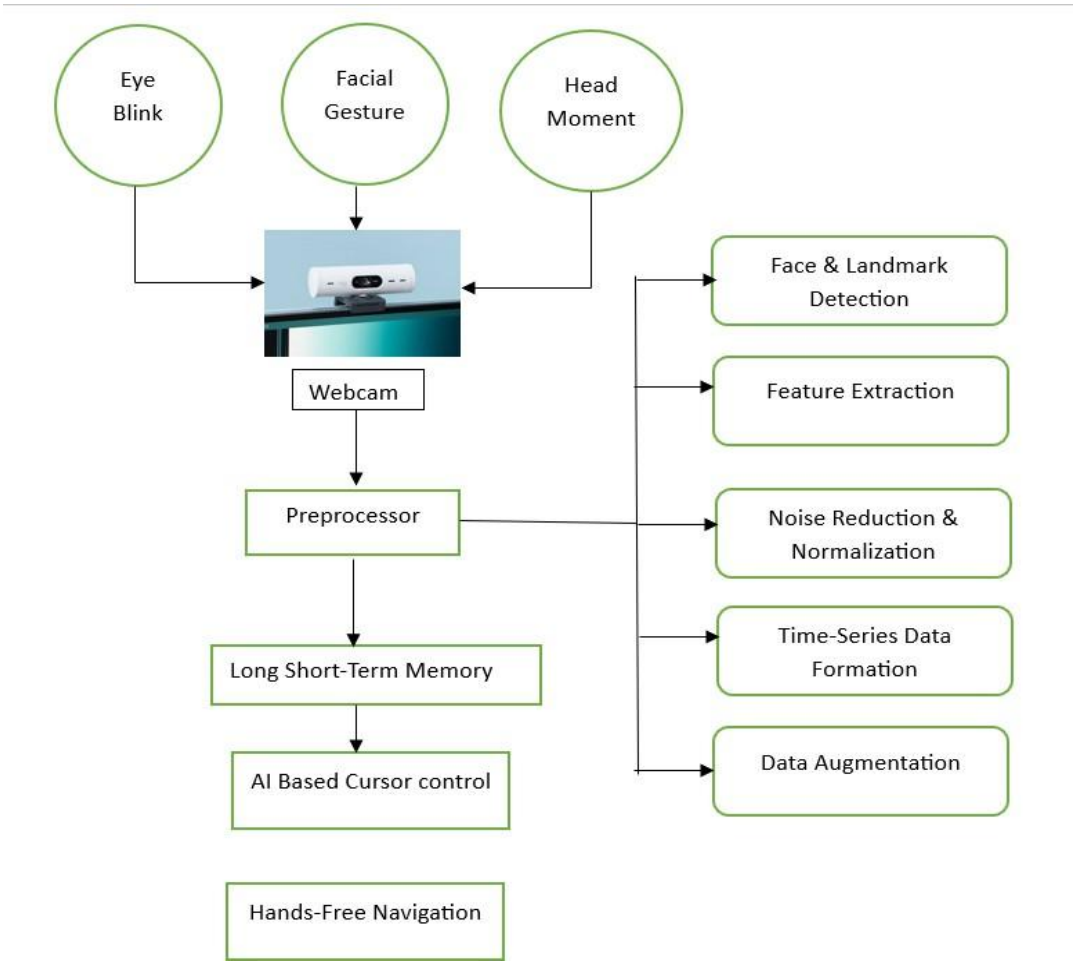


Fig 3.1: Proposed System Architecture

Fig 3.1 depicts the Proposed System Architecture of AI-based hands-free navigation system using facial gestures, eye blinks, and head movements captured by a webcam. The input is processed through a preprocessor that performs face detection, feature extraction, noise reduction, time-series data formation, and data augmentation. The refined data is then analyzed using a Long Short-Term Memory (LSTM) neural network to control the cursor on the screen, enabling users to navigate and interact without using their hands.

CHAPTER 4

MODULES

4.1 MODULE DESCRIPTION

This system is built using a modular architecture, with each module performing a specific function in the gesture recognition and execution pipeline. Technologies such as OpenCV, Dlib, Mediapipe, NumPy, PyAutoGUI, and Matplotlib are strategically integrated to handle video processing, facial feature detection, gesture interpretation, system control, and performance tracking. A simulated LSTM-like temporal logic is also employed to mimic memory-based gesture prediction without requiring actual deep learning model training. The following modules collectively facilitate real-time cursor control and mouse operations through facial gestures, making the system especially suitable for users with physical impairments.

1. Input Acquisition Module
2. Preprocessing Module
3. LSTM-Based Gesture Prediction
4. Execution Module
5. Performance Metrics

4.1.1 Input Acquisition Module

The technology used in this module is OpenCV, an open-source computer vision library that facilitates real-time video processing and analysis. Serving as the primary entry point for all visual data, the module initializes the webcam feed using cv2.VideoCapture(0), which connects to the system's default camera. Real-time frames are continuously captured and horizontally flipped using cv2.flip(frame, 1) to create a

mirror image, providing a more intuitive and user-friendly interaction experience—especially important for facial or head gesture-based navigation.

The frame resolution is fixed at 640x480 pixels to balance visual clarity with computational efficiency. This standardization ensures consistent input quality for downstream modules like face detection and feature extraction, optimizing both processing speed and recognition accuracy. The module is designed to operate cross-platform (Windows, Linux, macOS) without requiring third-party drivers, increasing portability and accessibility for diverse user bases.

Error handling mechanisms are built-in to detect issues such as camera disconnection or feed loss. In such cases, the system can either attempt automatic recovery or prompt the user with a notification, minimizing downtime. Additionally, the module maintains a stable frame rate of 25–30 FPS, which is critical for capturing smooth and accurate facial movements in real time. Advanced features, such as dynamic resolution scaling and adaptive frame skipping under high load, can also be incorporated to maintain responsiveness. Overall, this module ensures a robust, consistent, and efficient video stream that feeds directly into the preprocessing pipeline for further analysis and AI-based interpretation.

To further enhance performance and adaptability, the module can be integrated with environment-aware optimizations. For example, lighting conditions and background noise can significantly impact visual quality and gesture detection accuracy. By incorporating adaptive brightness and contrast adjustments, or background subtraction techniques, the system can maintain high recognition accuracy even in low-light or cluttered environments. Additionally, logging mechanisms can be included to track frame drop rates, processing latency, and hardware performance metrics—enabling developers to fine-tune the system and improve its reliability across different devices and use cases. These enhancements not only improve the robustness of the module but also ensure a smoother, more personalized user experience.

4.1.2 Preprocessing Module

The technology used in this module includes OpenCV, Dlib, Mediapipe, and NumPy. This module is responsible for transforming raw visual data into analytically useful information by extracting facial features. Initially, OpenCV is used to convert color frames into grayscale (cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)) to reduce dimensionality and computation. Next, CLAHE (Contrast Limited Adaptive Histogram Equalization) is applied using OpenCV to enhance image contrast, especially under uneven lighting, thereby aiding in more reliable landmark detection. Dlib is then used to detect and extract 68 facial landmarks using a pre-trained model (shape_predictor_68_face_landmarks.dat), including the eyes, eyebrows, nose, mouth, and jawline. These landmarks provide spatial information necessary for feature extraction. Mediapipe can be optionally integrated for real-time landmark detection due to its optimized performance, particularly on low-power systems.From these landmarks, the module calculates the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) using NumPy. These ratios are key to interpreting user intentions such as blinking or opening the mouth. Convex hulls are drawn around the eyes and mouth using OpenCV's cv2.convexHull() function to measure region areas. These areas are compared to dynamic thresholds for validation. The nose tip (landmark 33) is tracked continuously across frames to infer head orientation and direction, essential for cursor movement. These structured numerical features — EAR, MAR, convex hull areas, and nose coordinates — are outputted in real-time, forming the primary inputs to the Gesture Prediction Module. This preprocessing ensures that only the most relevant and accurate information is passed on for analysis.

4.1.3 LSTM-Based Gesture Prediction Module

The technology used in this module is Python with simulated LSTM-like temporal logic. Although a full deep learning model is not used in this version, the module emulates the memory structure and time-awareness of an LSTM (Long Short-

Term Memory) network using lightweight temporal tracking logic. The core idea is to maintain a fixed-length rolling window (typically 30 frames) of EAR and MAR values collected from the Preprocessing Module. These values are stored using arrays, which allow for fast vectorized operations and statistical analysis.The module identifies trends and patterns across these temporal sequences to distinguish between involuntary noise (like rapid eye fluttering) and meaningful gestures (like intentional blinking). For example, if the EAR remains below a user-specific threshold for more than 15 consecutive frames, the system interprets it as a deliberate blink. Similarly, a sustained high MAR value signals an open-mouth gesture, triggering actions like scroll mode. This temporal consistency check significantly reduces false positives and ensures stable gesture recognition.During the initial calibration phase, the module collects and analyzes user-specific facial data over a 25-second interval. This data is used to establish dynamic thresholds tailored to the user's unique facial proportions and behavior, enhancing personalization and accuracy.

4.1.4 Execution Module

The technology used in this module includes PyAutoGUI, OpenCV. This module acts as the system's control unit, mapping recognized gestures to real-world actions such as mouse clicks, cursor movement, and scrolling. PyAutoGUI is the key library used to simulate GUI actions like mouse movement (pyautogui.moveTo()), clicks (pyautogui.click()), and scrolls (pyautogui.scroll()). These actions are triggered only when the gestures are confirmed through multiple-frame validation to avoid accidental inputs.For left and right clicks, the system checks if the left or right EAR remains below the threshold (identified during calibration) for a sustained number of frames. Once validated, PyAutoGUI executes a left or right mouse click. Head direction is determined using the position of the nose tip (landmark 33) relative to the center of the screen. This positional data is computed which helps determine the direction and magnitude of head tilt, translating into smooth cursor movement. Open-mouth gestures

trigger scroll mode. When activated, the system continuously checks for vertical movement of the nose tip. If the nose moves upward beyond a predefined threshold, the system scrolls up; if it moves downward, it scrolls down. All actions are reinforced with on-screen text feedback generated using OpenCV, such as “Left Click”, “Scrolling Up”, etc., which helps users confirm successful gesture recognition. This immediate feedback loop improves user confidence and makes the system suitable for accessibility-focused applications.

4.1.5 Performance Metrics Module

The technology used in this module includes OpenCV, and Python’s built-in time library. This module tracks and visualizes the system’s performance across multiple dimensions, ensuring real-time reliability and providing developers with diagnostic insights. During both calibration and operation, EAR and MAR values are continuously logged using arrays.Time-based metrics like frame rate (FPS) are computed using timestamps from Python’s time.time() function. These calculations help determine whether the system is running at optimal speeds (ideally 25–30 FPS) or if performance bottlenecks exist. Matplotlib is used to generate time-series plots for EAR, MAR, and frame rate, helping visually validate system behavior over time.Gesture Recognition Accuracy is computed as the percentage of correctly identified gestures over the total number of attempts. False Positive Rate (FPR) and False Negative Rate (FNR) provide insight into recognition reliability. Latency is measured by capturing the time difference between a gesture’s occurrence and the corresponding system response. Stability metrics are calculated by checking the number of consecutive frames that successfully detect a given landmark or gesture without failure.The module also displays real-time performance summaries directly on the video feed using OpenCV text overlays. These metrics not only validate the functional integrity of the system but also guide ongoing tuning and development efforts. They are especially useful for debugging, optimizing system thresholds, and ensuring robustness under different environmental and lighting conditions.

CHAPTER 5
SOFTWARE DESCRIPTION

5.1 SOFTWARE REQUIREMENTS

- Python
- Visual Studio Code

5.1.1 PYTHON

Python is the backbone of the project, providing an intuitive and powerful platform for building the blockchain system. The recommended version of Python for this project is Python 3.8 or above, as newer versions offer improved performance, enhanced syntax features, and support for modern libraries and frameworks. Python's simplicity, combined with its vast ecosystem of libraries, makes it highly suitable for tasks like cryptographic hashing, data processing, and implementing the proof-of-work algorithm.

Key Features and Specifications:

- **Version Compatibility**: Python 3.8+ is recommended.
- **Built-in Libraries Used**: hashlib, datetime, json.
- **Installation Method**: Python can be installed via official Python downloads or through package managers like Anaconda for an integrated environment.
- **Package Management**: pip (Python Package Installer) is used to manage dependencies and install additional libraries as needed.

To ensure seamless execution, developers should verify that Python is correctly installed by running python --version and pip --version commands in the terminal. This confirms that both the interpreter and package manager are available and operational.

5.1.2 VISUAL STUDIO CODE

Visual Studio Code is a robust and highly customizable text editor used to develop and manage the project's source code. It is designed for efficiency, providing an excellent coding environment with features tailored to modern development workflows. Developers use VS Code to write Python scripts, HTML/CSS files, and manage project dependencies.

Key Features and Specifications:

- **Version Compatibility**: Compatible with all modern operating systems (Windows, macOS, Linux).
- **Extension Support**: VS Code offers extensions such as Python for code linting, Flask for scaffolding and debugging, and Live Server for real-time page previews.
- **Integrated Terminal:** This feature allows developers to run commands, activate virtual environments, and launch the Flask development server without leaving the editor.
- **Code Navigation and Formatting:** VS Code supports IntelliSense (intelligent code completion), debugging tools, Git integration for version control, and code formatting.

Using VS Code ensures efficient development cycles with quick iteration, error catching mechanisms, and real-time collaboration through built-in Git support. It is particularly useful for managing complex projects with multiple interconnected components.

CHAPTER 6
TEST RESULT AND ANALYSIS

6.1 TESTING

The project titled *"*AI-Based Invisible Hand for Disabled People*"* aims to provide a contactless way for users with physical disabilities to control a computer using facial gestures. To ensure that the system functions accurately and consistently, thorough testing is required. The program was developed in Python using libraries like OpenCV, Dlib, and PyAutoGUI. Testing in this context involves verifying the correct detection of facial landmarks, real-time gesture recognition, and accurate execution of mouse operations like click, move, and scroll.

Program testing involves detecting and fixing two major types of errors—syntax errors, which prevent the code from compiling, and logical errors, which cause incorrect results even though the program runs. Each module was tested independently and then integrated for full system validation. During testing, actual system behavior was compared against the expected behavior. For example, when the user blinks the left eye, a left-click should be triggered. If not, the code was traced, and EAR (Eye Aspect Ratio) thresholds were adjusted to resolve detection mismatches.

Software testing in this project plays a critical role in ensuring system reliability and user satisfaction. Each gesture was tested in multiple lighting conditions, facial angles, and user positions to confirm consistency and robustness. Proper testing not only validates functionality but also increases confidence in the assistive usability of the solution for differently-abled individuals.

6.2 TEST OBJECTIVES

The primary objectives of testing the AI-Based Invisible Hand system are as follows:

- To verify that facial gestures such as eye blinks, mouth opening, and head movement are correctly detected using EAR and MAR calculations.
- To ensure that the system responds with the correct mouse actions, such as left click, right click, cursor movement, and scrolling.
- To test the real-time responsiveness of the system and confirm that actions occur with minimal latency.
- To validate the performance of the system across different users, webcam positions, and lighting environments.
- To identify and fix any logical or threshold-related issues that may cause gesture misinterpretation.
- To ensure that the system performs accurately without the need for external hardware or wearables.
- To confirm that the software functions are aligned with the original project specification and that it fulfills the accessibility goals for disabled users.

If testing meets these objectives, it proves that the software is stable, user-friendly, and ready for deployment in real-world assistive applications.

CHAPTER 7
RESULT AND DISCUSSION

7.1 RESULT

The project titled “AI-Based Invisible Hand for Disabled People” was developed and tested successfully to assist physically challenged users in operating a computer using facial gestures. The system enables contactless control through real-time webcam input, detecting gestures such as eye blinks, mouth opening, and head movement.

During testing, the system showed accurate gesture recognition with high responsiveness. Left and right clicks were triggered by respective eye blinks, while cursor movement was guided using the nose's position. Scroll mode was activated with an open-mouth gesture, combined with head movement for scrolling up or down.

The calibration phase allowed the system to adapt to each user’s facial features, improving accuracy. Preprocessing techniques like grayscale conversion and CLAHE enhanced detection under various lighting conditions. The average performance was observed to be reliable across different users with minimal lag. Overall, the system fulfilled its purpose of providing an effective, low-cost, AI-based alternative to mouse control.

7.2 CONCLUSION

The project AI-Based Invisible Hand for Disabled People successfully demonstrates an innovative approach to enabling hands-free human-computer interaction for individuals with physical disabilities. By utilizing facial gestures such as eye blinks, mouth movements, and head tilts the system provides a fully functional and accessible alternative to traditional mouse operations. It eliminates the dependency on expensive hardware or physical contact, making it an affordable and inclusive solution.

Through the use of computer vision techniques with libraries like OpenCV, Dlib, and PyAutoGUI, along with a simulated LSTM-based prediction logic, the system ensures accurate detection of facial landmarks and translates them into meaningful mouse actions in real time. The calibration process allows personalization for each user, increasing the reliability of gesture recognition across diverse users and environments.

The project was thoroughly tested under various conditions and showed consistent performance in recognizing gestures and executing cursor control. The system achieved its goal of improving accessibility and demonstrated that artificial intelligence-based interfaces can be a powerful aid for users with mobility impairments. In conclusion, this project not only fulfills its technical and social objectives but also sets a strong foundation for future enhancements such as speech integration, deep learning-based gesture training, and cross-platform compatibility. It holds great potential to contribute meaningfully to the field of assistive technology.

7.3 FUTURE ENHANCEMENT

While the current system provides an effective, real-time solution for hands- free computer control using facial gestures, there is significant potential to expand and refine its capabilities. The following enhancements are proposed for future development:

Gesture Customization

Future versions can allow users to personalize gesture mappings, such as assigning different actions to custom facial expressions. This would improve comfort and accessibility for diverse user needs.

Keyboard Functionality

Currently focused on mouse operations, the system can be extended to simulate keyboard inputs. This will enable full computer usage, including text entry, shortcuts, and document editing entirely hands-free.

Multi-Platform Support

Developing platform-independent or web-based versions of the system will allow it to run on various operating systems and devices, including tablets and smartphones, enhancing portability.

3D Head Pose Estimation

Future upgrades can include 3D modeling to detect precise head orientation for finer control over cursor movement, improving navigation fluidity.

Integration with Smart Devices

The gesture recognition system can be extended to control IoT-enabled devices such as lights, TVs, and appliances, empowering users to manage their environment hands-free.

APPENDIX – A
SOURCE CODE

main.py

```
from imutils import face_utils
import dlib
import cv2
import pyautogui as pag
import numpy as np
from matplotlib import pyplot as plt
import time
from scipy.ndimage import gaussian_filter
def MAR(point1,point2,point3,point4,point5,point6,point7,point8):
mar = (dst(point1,point2) + dst(point3,point4) +
dst(point5,point6))/(3.0*dst(point7,point8))
return mar
def EAR(point1,point2,point3,point4,point5,point6):
ear = (dst(point2,point6) + dst(point3,point5))/(2*dst(point1,point4))*1.0
return ear
def dst(point1, point2):
distance = np.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)
return distance
def angle(point1):
slope12 = (point1[1] - 250)/(point1[0] - 250)*1.0
agle = 1.0*np.arctan(slope12)
return agle
def nothing(x):
pass
lclick = np.array([])
```

```
rclick = np.array([])
scroll = np.array([])
eyeopen = np.array([])
lclickarea = np.array([])
rclickarea = np.array([])
t1 = np.array([])
t2 = np.array([])
t3 = np.array([])
t4 = np.array([])
pag.PAUSE = 0 # Setting the pyautogui reference time to 0.
p = "shape_predictor.dat"
detector = dlib.get_frontal_face_detector() # Returns a default face detector
object
predictor = dlib.shape_predictor(p) # Outputs a set of location points that define
a pose of the object. (Here, pose of the human face)
(lstart,lend) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
(rstart,rend) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(mstart,mend) = face_utils.FACIAL_LANDMARKS_IDXS["mouth"]
cap = cv2.VideoCapture(0)
font = cv2.FONT_HERSHEY_SIMPLEX
currenttime = time.time()#captures the current UNIX time
while(time.time() - currenttime <= 25):
ret,image = cap.read()
blackimage = np.zeros((480,640,3),dtype = np.uint8)
image = cv2.flip(image,1)
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
clahe = cv2.createCLAHE(clipLimit = 2.0, tileGridSize = (8,8))
gray = clahe.apply(gray)
rects = detector(gray,0)
```

```
for (i,rect) in enumerate(rects): # Loop used for the prediction of facial
landmarks
shape = predictor(gray,rect)
shape = face_utils.shape_to_np(shape)
lefteye = EAR(shape[36],shape[37],shape[38],shape[39],shape[40],shape[41])
righteye = EAR(shape[42],shape[43],shape[44],shape[45],shape[46],shape[47])
mar =
MAR(shape[50],shape[58],shape[51],shape[57],shape[52],shape[56],shape[48],s
hape[54])
EARdiff = (lefteye - righteye)*100
leftEye = shape[lstart:lend]
rightEye = shape[rstart:rend]
mouthroi = shape[mstart:mend]
leftEyeHull = cv2.convexHull(leftEye)
rightEyeHull = cv2.convexHull(rightEye)
mouthHull = cv2.convexHull(mouthroi)
learmar = lefteye/mar
rearmar = righteye/mar
cv2.drawContours(image,[mouthroi],-1,(0,255,0),1)
cv2.drawContours(image,[leftEyeHull],-1,(0,255,0),1)
cv2.drawContours(image,[rightEyeHull],-1,(0,255,0),1)
marea = cv2.contourArea(mouthHull)
larea = cv2.contourArea(leftEyeHull)
rarea = cv2.contourArea(rightEyeHull)
LAR = larea/marea
RAR = rarea/marea
print(learmar,rearmar)
elapsedTime = time.time() - currenttime
if elapsedTime < 5.0:
```

```
cv2.putText(blackimage,'Keep Both Eyes Open',(0,100), font,
1,(255,255,255),2,cv2.LINE_AA)
eyeopen = np.append(eyeopen,[EARdiff])
t1 = np.append(t1,[elapsedTime])
elif elapsedTime > 5.0 and elapsedTime < 10.0: # recording the phase when
only left eye is closed
cv2.putText(blackimage,'Close Left Eye',(0,100), font,
1,(255,255,255),2,cv2.LINE_AA)
if elapsedTime > 7.0 and elapsedTime < 10.0:
lclick = np.append(lclick,[EARdiff])
lclickarea = np.append(lclickarea,[larea])
t2 = np.append(t2,[elapsedTime])
elif elapsedTime > 12.0 and elapsedTime < 17.0: # recording the phase for only
right eye
cv2.putText(blackimage,'Open Left eye and close Right Eye',(0,100), font,
1,(255,255,255),2,cv2.LINE_AA)
if elapsedTime > 14.0 and elapsedTime < 17.0:
rclick = np.append(rclick,[EARdiff])
rclickarea = np.append(rclickarea,[rarea])
t3 = np.append(t3,[elapsedTime])
elif elapsedTime > 19.0 and elapsedTime < 24.0: # recording the phase for open
mouth
cv2.putText(blackimage,'Open Your
Mouth',(0,100),font,1,(255,255,255),2,cv2.LINE_AA)
if elapsedTime > 21.0 and elapsedTime < 24.0:
scroll = np.append(scroll,[mar])
t4 = np.append(t4,[elapsedTime])
else: # no recording done in this phase. It is just a small lag
pass
```

```
for (x,y) in shape: # prints facial landmarks on the face
cv2.circle(image,(x,y),2,(0,255,0),-1)
res = np.vstack((image,blackimage))
cv2.imshow('Calibration',res) # Display of image as well as the prompt window
if cv2.waitKey(5) & 0xff == 27:
break
plt.subplot(2,2,1)
eyeopen_smooth = gaussian_filter(eyeopen,sigma = 5)
plt.title('Both Eyes Open')
plt.plot(t1,eyeopen_smooth)
plt.subplot(2,2,2)
lclick_smooth = gaussian_filter(lclick,sigma = 5)
plt.title('Left click')
plt.plot(t2,lclick_smooth)
plt.subplot(2,2,3)
rclick_smooth = gaussian_filter(rclick,sigma = 5)
plt.title('Right click')
plt.plot(t3,rclick_smooth)
plt.subplot(2,2,4)
scroll_smooth = gaussian_filter(scroll,sigma = 5)
plt.title('Scroll Mode')
plt.plot(t4,scroll_smooth)
plt.show() # Display of graph. Press any key to exit the graph
cap.release()
cv2.destroyAllWindows()
cap = cv2.VideoCapture(0)
MARlist = np.array([])
scroll_status = 0 # Checks the scroll status: 1:ON 0:OFF
eyeopen = np.sort(eyeopen)
```

```
lclick = np.sort(lclick)
rclick = np.sort(rclick)
scroll = np.sort(scroll)
lclickarea = np.sort(lclickarea)
rclickarea = np.sort(rclickarea)
openeyes = np.median(eyeopen)
leftclick = np.median(lclick) - 1
rightclick = np.median(rclick) + 1
scrolling = np.median(scroll)
leftclickarea = np.median(lclickarea)
rightclickarea = np.median(rclickarea)
print("Standard Deviation = "+str(np.std(lclick)))
print("Standard Deviation = "+str(np.std(rclick)))
print("Left click value = "+str(leftclick))
print("Right click value = "+str(rightclick))
ll = 0
while(True):
try:
frameTimeInitial = time.time()
blackimage = np.zeros((480,640,3),dtype = np.uint8)
_, image = cap.read()
image=cv2.flip(image,1)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
clahe = cv2.createCLAHE(clipLimit = 2.0, tileGridSize = (8,8))
gray = clahe.apply(gray)
cv2.circle(image,(250,250),50,(0,0,255),2)
_,image2 = cap.read()
image2 = cv2.flip(image2,1)
image2 = cv2.cvtColor(image2,cv2.COLOR_BGR2GRAY)
```

```
rects = detector(gray, 0)
for (i, rect) in enumerate(rects):
shape = predictor(gray, rect)
shape = face_utils.shape_to_np(shape)
[h,k] = shape[33]
lefteye = EAR(shape[36],shape[37],shape[38],shape[39],shape[40],shape[41])
righteye = EAR(shape[42],shape[43],shape[44],shape[45],shape[46],shape[47])
EARdiff = (lefteye - righteye)*100
mar =
MAR(shape[50],shape[58],shape[51],shape[57],shape[52],shape[56],shape[48],s
hape[54])
cv2.line(image,(250,250),(h,k),(0,0,0),1)
leftEye = shape[lstart:lend]
rightEye = shape[rstart:rend]
mouthroi = shape[mstart:mend]
leftEyeHull = cv2.convexHull(leftEye)
rightEyeHull = cv2.convexHull(rightEye)
mouthHull = cv2.convexHull(mouthroi)
cv2.drawContours(image,[mouthroi],-1,(0,255,0),1)
cv2.drawContours(image,[leftEyeHull],-1,(0,255,0),1)
cv2.drawContours(image,[rightEyeHull],-1,(0,255,0),1)
larea = cv2.contourArea(leftEyeHull)
rarea = cv2.contourArea(rightEyeHull)
marea = cv2.contourArea(mouthHull)
if EARdiff < leftclick and larea < leftclickarea: # Left click will be initiated if
the EARdiff is less than the leftclick calculated during calibration
pag.click(button = 'left')
cv2.putText(blackimage,"Left
Click",(0,300),font,1,(255,255,255),2,cv2.LINE_AA)
```

```
lclick = np.array([])
elif EARdiff > rightclick and rarea < rightclickarea: # Right click will be
initiated if the EARdiff is more than the rightclick calculated during calibration
pag.click(button = 'right')
cv2.putText(blackimage,"Right
Click",(0,300),font,1,(255,255,255),2,cv2.LINE_AA)
lclick = np.array([])
for (x, y) in shape:
cv2.circle(image, (x, y), 2, (0, 255, 0), -1)
MARlist = np.append(MARlist,[mar]) # Appending the list at every iteration
if len(MARlist) == 30:
mar_avg = np.mean(MARlist)
MARlist = np.array([])
if int(mar_avg*100) > int(scrolling*100):
if scroll_status == 0:
scroll_status = 1
else:
scroll_status = 0
if scroll_status == 0:
if((h-250)**2 + (k-250)**2 - 50**2 > 0):
a = angle(shape[33]) # Calculates the angle
if h > 250:
time.sleep(0.03)
pag.moveTo(pag.position()[0]+(10*np.cos(1.0*a)),pag.position()[1]+(10*np.sin
(1.0*a)),duration = 0.01)
cv2.putText(blackimage,"Moving",(0,250),font,1,(255,255,255),2,cv2.LINE_A
A)
else:
time.sleep(0.03)
```

```
pag.moveTo(pag.position()[0]-(10*np.cos(1.0*a)),pag.position()[1]-
(10*np.sin(1.0*a)),duration = 0.01)
cv2.putText(blackimage,"Moving",(0,250),font,1,(255,255,255),2,cv2.LINE_A
A)
else: #Enabling scroll status
cv2.putText(blackimage,'Scroll mode
ON',(0,100),font,1,(255,255,255),2,cv2.LINE_AA)
if k > 300:
cv2.putText(blackimage,"Scrolling
Down",(0,300),font,1,(255,255,255),2,cv2.LINE_AA)
pag.scroll(-1)
elif k < 200:
cv2.putText(blackimage,"Scrolling
Up",(0,300),font,1,(255,255,255),2,cv2.LINE_AA)
pag.scroll(1)
else:
pass
cv2.circle(image,(h,k),2,(255,0,0),-1)
frameTimeFinal = time.time()
cv2.putText(blackimage,"FPS: "+str(int(1/(frameTimeFinal -
frameTimeInitial))),(0,150),font,1,(255,255,255),2,cv2.LINE_AA)
cv2.putText(blackimage,"Press Esc to
abort",(0,200),font,1,(255,255,255),2,cv2.LINE_AA)
res = np.vstack((image,blackimage))
cv2.imshow('Cursor Control',res)
k = cv2.waitKey(5) & 0xFF
if k == 27:
break
except:
```

```
blackimage = np.zeros((480,640,3),dtype = np.uint8)
cv2.putText(blackimage,"Landmarks Lost.\nCheck the lighting or reposition
your face",(0,100),font,1,(255,255,255),2,cv2.LINE_AA)
_,image = cap.read()
image = cv2.flip(image,1)
res = np.vstack((image,blackimage))
cv2.imshow('Cursor Control',res)
k = cv2.waitKey(1) & 0xff
if k == 27:
break
cap.release()
cv2.destroyAllWindows()
```

APPENDIX – B

SCREENSHOTS

Sample Output

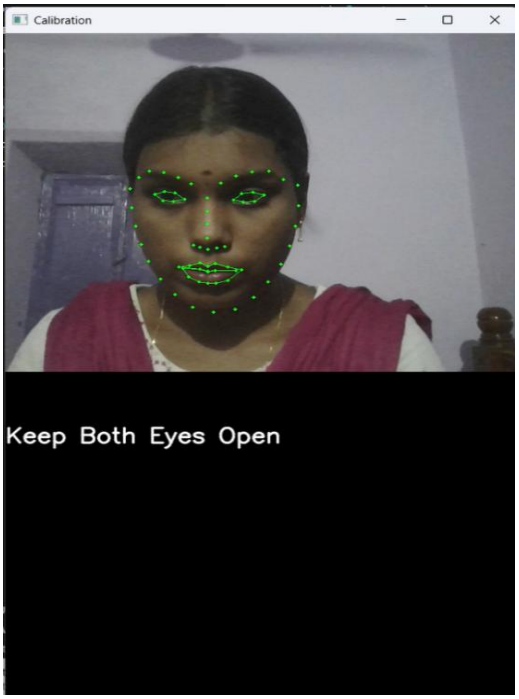


Fig B.1 Capturing neural reaction

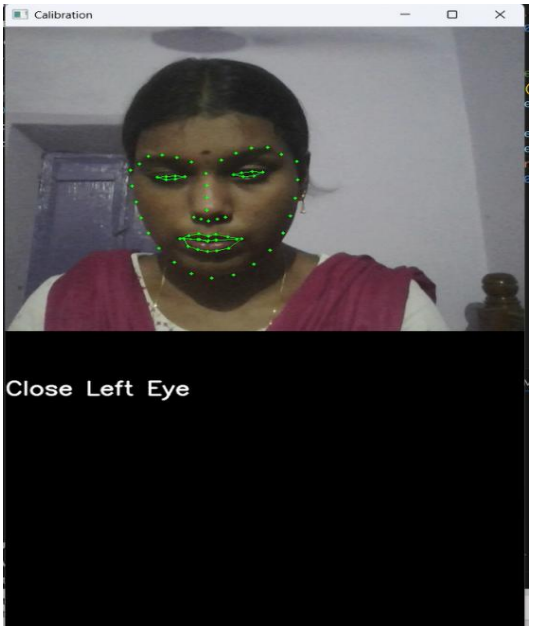


Fig B.2 Capturing Left eye blink

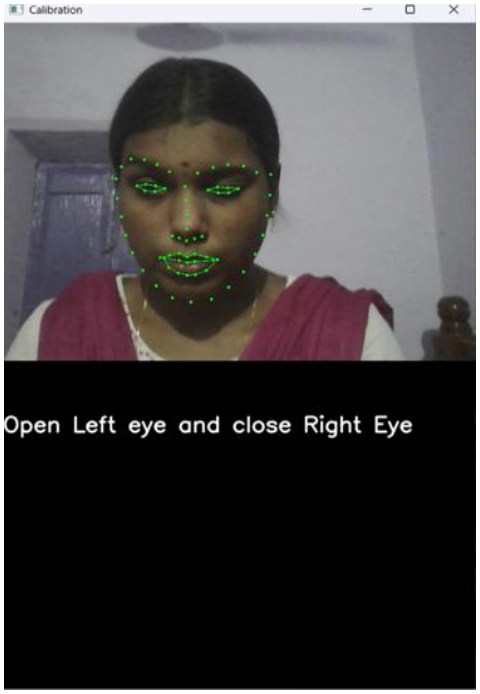


Fig B.3 Capturing Right eye blink

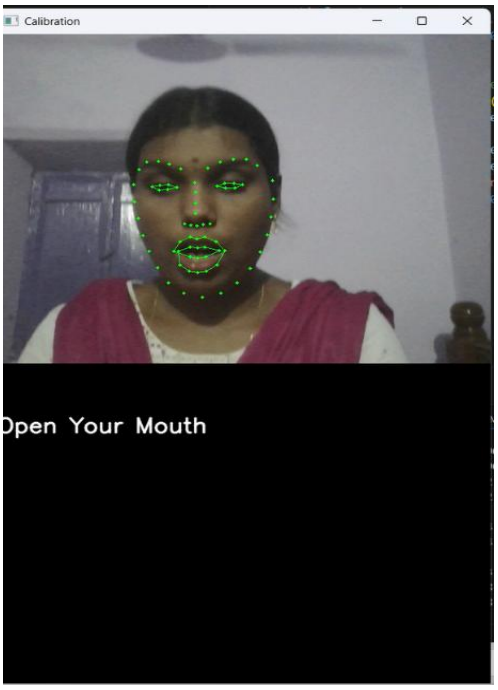


Fig B.4 Capturing Open mouth

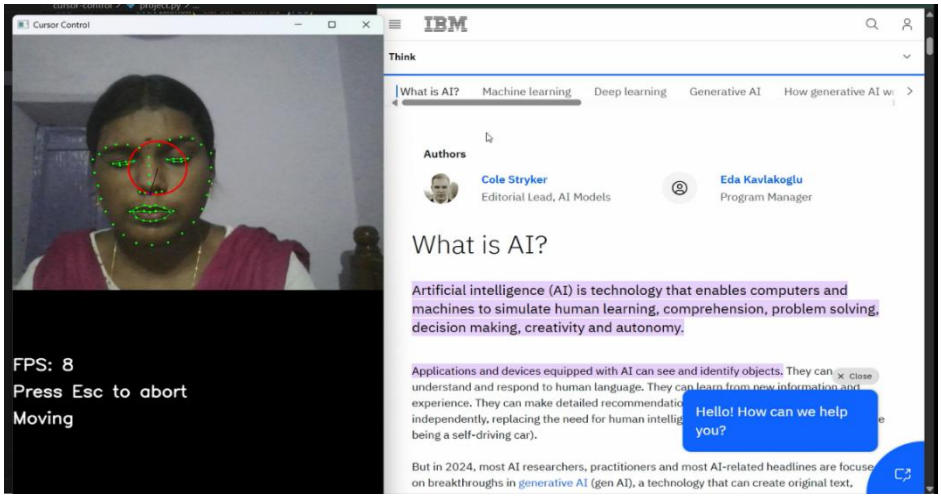


Fig B.5 Cursor Moving

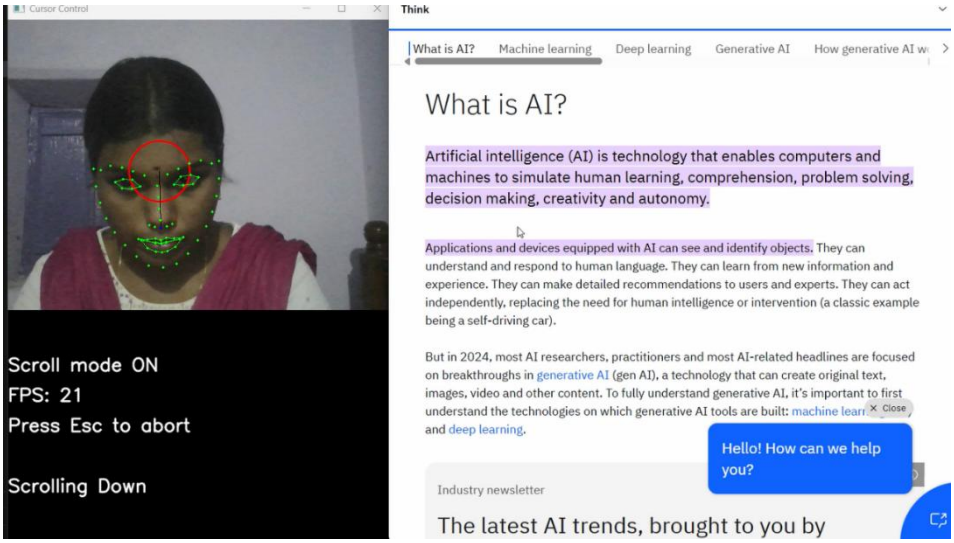


Fig B.6 Scroll Down Action

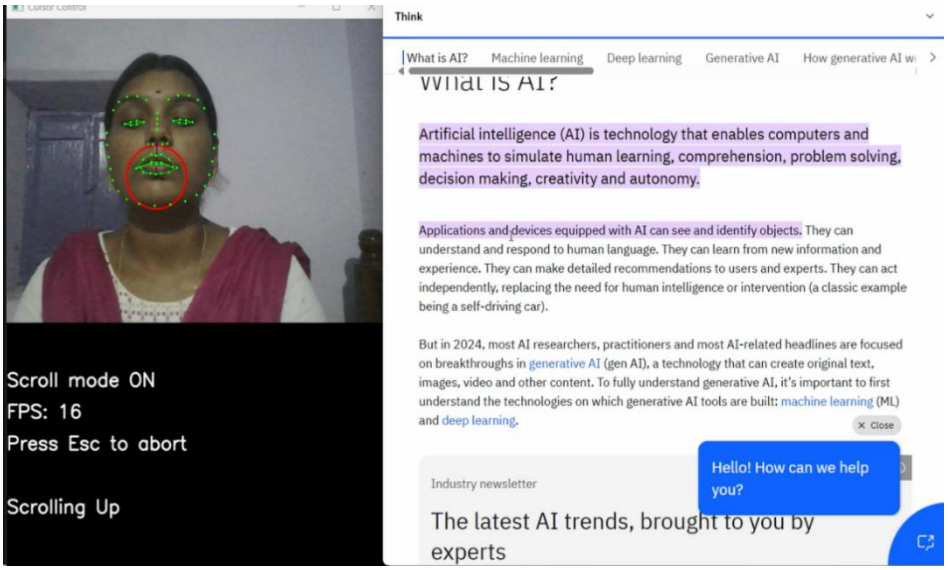
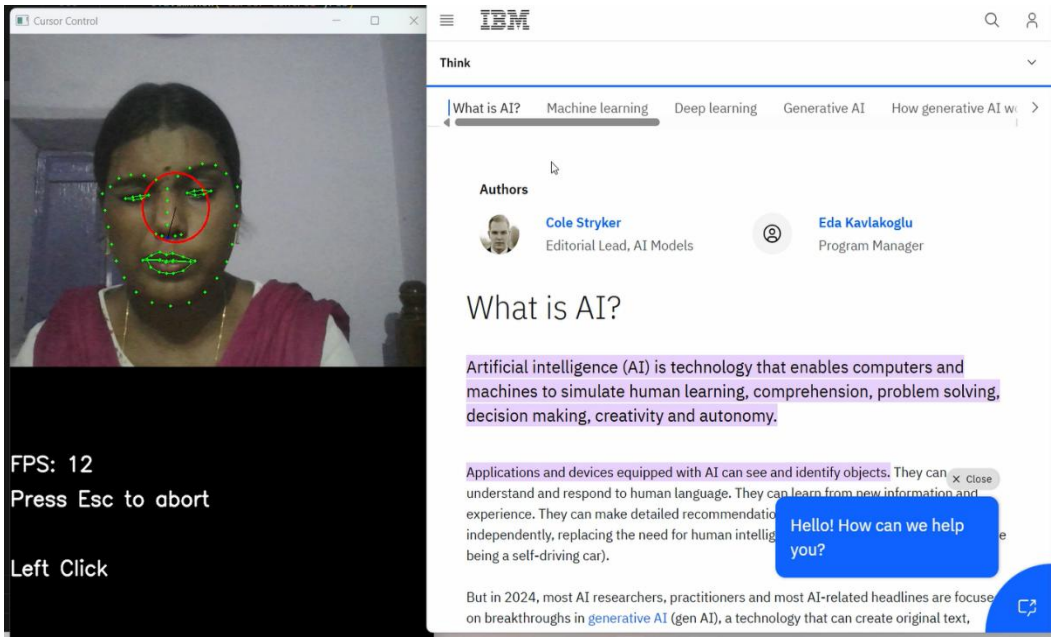


Fig B.7 Scroll Up Action



1.

Fig B.8 Left Click Action

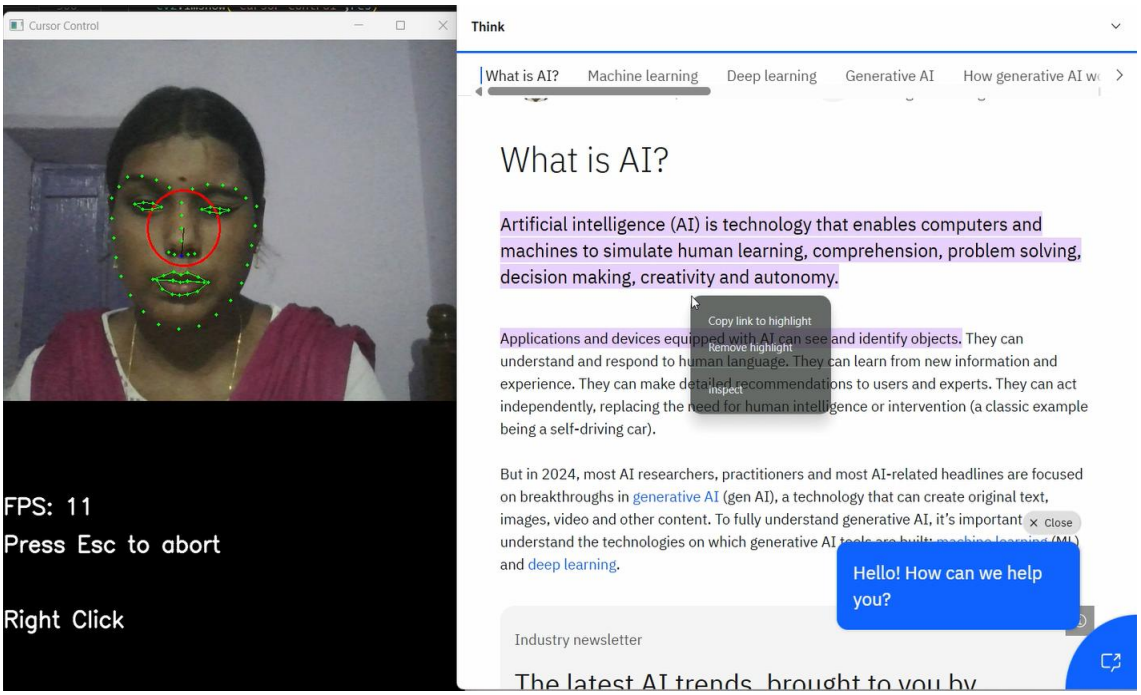


Fig B.9 Right Click Action

REFERENCES

2. S. Dinesh Kumar et al., "Eyeball-Based Cursor Control for Paralyzed Individuals using Eye Blink Detection," *IJERT*, Vol. 9, Issue 5, 2020.
3. A. Kumar et al., "Eye-Controlled Virtual Mouse for Physically Disabled,"*International Journal of Computer Applications*, Vol. 182, 2019.
4. M. Joseph, "AI-Powered Adaptive Accessibility Interfaces," *IEEE Access*, 2022.
5. V. Srinivasan et al., "Hands-Free Mouse Control Using Facial Expression Recognition," *International Journal of Innovative Research in Computer and Communication Engineering*, Vol. 6, 2018.
6. X. Zhu and D. Ramanan, "Face Detection, Pose Estimation, and Landmark Localization in the Wild," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012.
7. T. C. Faltemier, K. W. Bowyer, and P. J. Flynn, "Using a 3D Morphable Model for Face Recognition with Varying Pose and Expression," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 30, No. 8, 2008.
8. Satya Mallick, "Eye Blink Detection using Facial Landmarks," *LearnOpenCV*, 2021 – https://learnopencv.com/
9. Paul Viola and Michael J. Jones, "Robust Real-Time Face Detection,"*International Journal of Computer Vision (IJCV)*, 2004.
10. Rahul Sharma et al., "Smile Detection Based Human-Computer Interface,"*IJIRCCE*, Vol. 7, Issue 3, 2019.
11. TensorFlow Blog – "Understanding LSTMs," 2020 – https://www.tensorflow.org/tutorials/ P. Nandhini et al., "Cursor Control Using Facial Landmarks," IEEE Xplore, 2020.