VISVESVARAYA TECHNOLOGICAL UNIVERSISTY

"JNANA SANGAMA", BELAGAVI, KARNATAKA-590018

Report of Mini Project on

"Eye Tracking Cursor Control System"

Submitted by

Chandu M	**4MH22CD012**
Rahul D V	**4MH22CD048**
Rudresh Gowda K	**4MH22CD050**

Under the Guidance Of

Prof Chandrashekara M.S
Assistant Professor

Dept of CSE (Data Science)

MIT Mysore

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

MAHARAJA INSTITUTE OF TECHNOLOGY, MYSORE

BELAWADI, NAGUVANAHALLY POST, S.R PATNA TALUK, MANDYA-571 438

2024-25

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATASCIENCE)

MAHARAJA INSTITUTE OF TECHNOLOGY MYSORE MANDYA-571477

CERTIFICATE

This is to certify that the Mini Project report **entitled "Eye Tracking Cursor Control System"** has been successfully carried out by **Chandu M [4MH22CD012]**, **Rahul D V [4MH22CD048], Rudresh Gowda K [4MH22CD050]** bonafide student of **Maharaja Institute of Technology Mysore** in partial fulfilment of Requirement of **Degree of Bachelor of Engineering in Computer Science and Engineering (Data Science) of Visveswaraya Technological University, Belagavi** during the academic year 2024-2025. It is certified that all corrections/suggestions indicated for the internal assessment have been incorporated in the report deposited in the department library.

Signature of Guide

Prof Chandrashekara M.S
Assistant Professor
Dept. of CSE (DS), MITM

Signature of HOD

Dr. Pushpa D
Professor &
Head Dept. of CSE (DS), MITM

ACKNOWLEDEGEMENT

We sincerely owe our gratitude to all the people who helped and guide completing this miniproject work.

We are thankful to **Dr. B.G. Naresh Kumar, Principal, Maharaja Institute of TechnologyMysore**, for having supported us in our academic endeavors.

We are extremely thankful to **Dr. Pushpa D, Professor & Head of Department of Computer Science and Engineering (Data Science),** for her valuable support and her timely inquiriesinto the progress of the work. We are obliged to all **teaching and non-teaching staff members** of the **Department of Computer Science and Engineering (Data Science)** for the valuable information provided by them in their respective fields. We are grateful for their co-operation during the period of our mini project.

Chandu M (4MH22CD012)

Rahul D V (4MH22CD048)

Rudresh Gowda K (4MH22CD050)

Abstract

The Eye Tracking Cursor Control System offers a hands-free solution for computer interaction, tailored especially for individuals with physical disabilities. This system employs cutting-edge computer vision technologies, including Mediapipe, OpenCV, and PyAutoGUI, to deliver precise cursor navigation and click simulation through real-time eye-tracking and gesture detection. By relying solely on a webcam for hardware, the project ensures accessibility, affordability, and ease of implementation.

The system features an intuitive calibration process, allowing users to designate their dominant eye for controlling the cursor and define a dynamic bounding box by focusing on specific screen corners. Through real-time tracking, the system maps the pupil's movements within this bounding box to screen coordinates, enabling smooth and precise cursor control. Click simulations are achieved via blink detection, utilizing eyelid landmark tracking to create a completely touch-free experience.

A web-based interface, implemented using Flask, serves as a responsive platform for user interaction.

This system highlights the potential of hands-free technologies in areas such as accessibility, gaming, and assistive tools. By removing the reliance on conventional input devices, it empowers users with limited mobility to perform everyday computing tasks autonomously, significantly improving their quality of life. Future developments may include enhanced gesture recognition, support for multiple platforms, and robust performance across diverse conditions, broadening the scope of this innovative technology.

TABLE OF CONTENT

1.	**INTRODUCTION**	
1.1	Overview	6
1.2	Problem Staement	7
1.3	Solution	7
1.4	Proposed system	8
2.	**Literature Survey**	
2.1	Survey papers	9
2.2	Survey Findings	13
3.	**SOFTWARE Requirement Specification**	
3.1	What is SRS	14
3.2	Functional& Non Functional Requirements	15
3.3	Software and Hardware Specification	16
4.	**Implementation**	
4.1	System Architecture	17
4.2	Tools and Technologies	17
4.3	Detailed Implementation	18
4.4	System Integration	22
4.5	Testing and Debugging	22
4.6	Challenges and Solution	22
5.	**CODES**	**23**
6.	**SNAPSHOTS**	**36**
7.	**FUTURE SCOPE**	**39**
8.	**CONCLUSION**	**40**
9.	**REFERENCES**	**41**

Chapter 1

INTRODUCTION

1.1 Overview

The Eye Tracking Cursor Control System presents a groundbreaking, hands-free solution for interacting with computers. This system addresses the challenges faced by individuals with physical disabilities by replacing conventional input devices such as mice and keyboards with eye movements and gestures. By utilizing real-time computer vision technologies, the system ensures ease of use and accessibility.

Key technologies such as Mediapipe and OpenCV are employed to detect facial landmarks and eye movements, allowing users to control the cursor with their dominant eye and simulate mouse clicks through deliberate blinks. A straightforward calibration process adjusts to the user's screen configuration, ensuring accurate tracking. Additionally, Python libraries like PyAutoGUI facilitate smooth cursor movement and gesture-based commands.

The primary goal of this system is to empower users with limited mobility, enabling them to independently carry out everyday computer tasks. This approach also showcases the flexibility of hands-free technology, with potential applications in fields such as gaming, assistive devices, and virtual reality. By utilizing readily available hardware like webcams, the project combines affordability with advanced functionality, fostering more inclusive human-computer interaction systems.

1.1.1 Key Features

- **User Setup:** Users designate their dominant eye for cursor control and their non-dominant eye for detecting clicks.
- **Calibration:** The system prompts users to focus on the four corners of the screen to create a tracking bounding box.
- **Eye Tracking:** Tracks the dominant eye's pupil movement to control the cursor position on the screen**.**
- **Click Simulation:** Detects blinks in the non-dominant eye to simulate mouse clicks.
- **Real-Time Adaptation:** Dynamically adjusts the mapping of eye movements to screen coordinates, ensuring smooth and precise control.

This project also integrates Flask for a web-based interface, offering a responsive and accessible platform. Its innovative design makes it ideal for use in accessibility applications, gaming, or experimental human-computer interaction scenarios.

1.2 Problem statement

In today's technology-driven world, traditional input devices like mice and keyboards are not universally accessible, particularly for individuals with mobility challenges. Current hands-free interaction solutions often fall short in terms of precision, user-friendliness, and affordability. Additionally, there is a growing demand for alternative interaction methods, especially in fields such as gaming, accessibility tools, and assistive technologies.

The key challenge is developing a reliable, real-time system that can accurately track eye movements and convert them into cursor controls, while also enabling mouse clicks through blinking. Achieving

this requires precise calibration, continuous tracking, and an intuitive design that ensures the system is usable by a wide range of users.

The proposed solution seeks to overcome these challenges by creating a web-based eye-tracking application that utilizes advanced computer vision techniques to enable hands-free control. The application will incorporate both calibration and real-time interaction features, making technology more inclusive and accessible for individuals with limited mobility.

1.3 Solution

The Eye Tracking Web Application offers a hands-free approach to computer interaction by utilizing real-time eye tracking and blink detection. Users can designate their dominant eye for cursor control and the other eye for detecting clicks. The system includes a guided calibration process that creates a dynamic bounding box for accurate tracking.

The application leverages Mediapipe for facial landmark detection, tracking pupil movements and detecting intentional blinks to simulate mouse clicks. It maps eye movements to screen dimensions for smooth cursor control, ensuring adaptability across various screen sizes. Built using Flask, the web-based interface allows the application to be accessible across multiple platforms without requiring additional installations. This solution not only addresses accessibility needs for individuals with mobility challenges but also introduces innovative interaction methods suitable for gaming, assistive devices, and other applications.

1.4 Proposed System

The proposed system is a web-based Eye Tracking Application designed to provide an innovative hands-free interaction solution with digital devices. This system aims to offer seamless cursor control and click simulation through real-time eye tracking and blink detection. Below are the key components and functionality of the proposed system:

1.4.1 Dominant Eye Selection

The system begins with an intuitive interface that prompts users to select their dominant eye. This enables the application to personalize its tracking and clicking mechanisms based on the user's preference, ensuring greater accuracy and comfort.

1.4.2 Calibration process

The calibration process requires users to focus on the four corners of the screen, allowing the system to establish a personalized bounding box for eye tracking. This process adapts to the user's eye movement range and screen dimensions, ensuring a more accurate and intuitive experience.

1.4.3 Eye Tracking

Using Mediapipe's Face Mesh module, the system detects facial landmarks, including the pupil's position. By continuously tracking these landmarks, the application converts eye movements into precise cursor control on the screen.

1.4.4 Click Simulation

The system detects intentional blinks of the non-dominant eye to simulate mouse clicks. By using Mediapipe's eyelid landmarks, the system differentiates between natural and intentional blinks, ensuring reliable click actions.

1.4.5 Platform Independence

The application is built using Flask, a Python-based web framework, and operates directly in a web browser. This removes the need for specialized hardware or software, making the system more accessible and scalable.

Chapter 2

LITERATURE SURVEY

2.1 Survey papers

2.1.1 H. A. Bhowmick and H. A. Mustafa (2024), "A Framework for Eye-Based Human Machine Interface," IEEE. [1]

Overview:

The paper introduces an eye-based Human Machine Interface (HMI) system designed to facilitate interaction for individuals with motor disabilities, allowing them to control computers and appliances through eye blinks. Traditional HMIs often require physical movement, which can be challenging for motor-impaired users. The proposed framework detects eye blinks and uses predefined blink patterns to trigger various actions, such as controlling a computer application or turning on household devices. This solution harnesses computer vision techniques, making it more accessible and practical for real-time applications.

Advantages:

- Utilizes efficient computer vision algorithms like Haar Cascade and Camshift for real-time tracking.
- Enables hands-free interaction, enhancing accessibility for motor-impaired individuals.
- Offers customizable blink patterns for flexible and user-specific control

Conclusion:

The eye-based Human Machine Interface provides a practical solution for motor-impaired users, enabling seamless interaction with devices through eye blinks. The framework emphasizes the use of accessible technologies and customizable features, ensuring adaptability and real-time usability. Future work should address environmental constraints to enhance system reliability

2.1.2 P. M. Naidu, N. Muthukumaran, S. Chandralekha, K. T. Reddy, and K. S. Vaishnavi (2023), "An Analysis on Virtual Mouse Control using Human Eye," IEEE. [2]

Overview:

This paper presents a virtual mouse control system that replaces traditional input devices with eye and facial gestures, offering a hands-free alternative for users with limited hand mobility. Key features include eye blink detection, mouth movement recognition, and nose-based cursor control, all implemented using computer vision techniques. By leveraging aspect ratios like EAR and MAR, the system achieves precise command execution for mouse functions. Its low-cost setup, requiring only a webcam, ensures accessibility for a broader user base.

Advantages:

- Provides an affordable and practical alternative to traditional mouse input.
- Combines multiple facial features for diverse and accurate mouse control actions.
- Includes calibration for improved detection accuracy tailored to individual users.

Conclusion:

The virtual mouse system demonstrates the potential of facial and eye gesture recognition in enabling hands-free computer interaction. It highlights the importance of accessibility and customization for users with physical disabilities. Future developments should focus on enhancing lighting adaptability and processing efficiency for broader applicability.

2.1.3 V. J. Hariharan, S. Karthiga, and A. M. Abirami (2023), "EyeGaze Control: Enhancing Mouse Cursor Precision Through Eyeball Movements," IEEE. [3]

Overview:

The EyeGaze Control system leverages real-time eye movement tracking to enable precise cursor control for users with motor impairments. Using pupil detection and facial landmarking, it translates eyeball movements into cursor actions, enhancing accessibility to digital interfaces. Calibration ensures customization for individual gaze patterns, while gaze-based fixation enables click functionality without additional physical input. Its focus on precision and intuitiveness addresses the needs of users with limited physical mobility

Advantages:

• Offers fine-grained cursor control through pupil-centered tracking.
• Features customizable calibration to adapt to individual gaze behaviors.
• Promotes hands-free interaction, improving accessibility for users with motor impairments.

Conclusion:

EyeGaze Control demonstrates the potential of eye-tracking technology for enhancing Human-Computer Interaction, particularly for users with disabilities. The system's emphasis on precision and user-centric calibration ensures usability and reliability. Future work should focus on improving robustness against lighting variations and enhancing click detection accuracy for broader applicability

2.1.4 M. Jaiswal, C. Dhakite, N. Dhule, and S. Mungale, (2022),"Smart AI-based eye gesture control system," IEEE. [4]

Overview:

The Smart AI-based Eye Gesture Control System enables hands-free computer interaction by interpreting eye movements as commands. It employs machine learning algorithms to analyze patterns in eye gestures, such as blinks and gaze direction, for precise control of mouse movements and actions. Calibration ensures personalized and accurate performance, making the system particularly beneficial for users with disabilities.

Advantages:

• Leverages AI and machine learning for robust eye gesture recognition.
• Offers personalized calibration to enhance user-specific accuracy.
• Enables handsfree device control, promoting accessibility for individuals with disability

Conclusion:

The Smart AI-based Eye Gesture Control System highlights the potential of integrating AI with eye-tracking for hands-free human-computer interaction. Its personalized approach ensures adaptability for diverse users, though challenges like lighting sensitivity and computational demands remain areas for improvement. The system paves the way for accessible technology, especially for individuals with mobility impairments.

2.1.5 R. N. Phursule, G. Y. Kakade, A. Koul, and S. Bhasin, (2022) "Virtual Mouse and Gesture-Based Keyboard," IEEE. [5]

Overview:

The Virtual Mouse and Gesture-Based Keyboard system offers a touchless interface for human-computer interaction by using hand and finger gestures for cursor control and keyboard inputs. Combining computer vision and machine learning, it enables real-time recognition of gestures for seamless operation. Calibration ensures the system adapts to individual user patterns, making it particularly useful in accessibility-focused and touchless interaction scenarios.

Advantages:

- Facilitates touchless interaction using advanced hand gesture recognition.
- Includes user-specific calibration for improved accuracy.
- Offers accessibility benefits for individuals with physical disabilities or in sterile environments

Conclusion:

The Virtual Mouse and Gesture-Based Keyboard system provides a promising touchless interaction method, leveraging real-time gesture recognition for seamless control. While its effectiveness depends on consistent lighting and adequate computational power, its potential applications in accessibility and healthcare underscore its transformative possibilities.

2.1.6 D. J. Dhanasekar, G. A. K. B., K. A. S., and F. A. Ahamath,(2021) "System Cursor Control Using Human Eyeball Movement," IEEE. [6]

Overview:

The paper describes a system that controls a computer cursor through human eye movements, offering a hands-free alternative for interaction. It uses eye-tracking to determine gaze direction and employs blinks as input for clicking actions. The solution caters to individuals with disabilities and extends its potential applications to gaming and research.

Advantages:

- Enables hands-free computer control, enhancing accessibility.
- Incorporates real-time cursor movement through eye-tracking.
- Uses blinks for click commands, simplifying user input.

- Employs lightweight frameworks like OpenCV and Mediapipe for efficiency

Conclusion:

The proposed eye-controlled cursor system demonstrates significant potential in promoting accessibility and usability. Despite constraints like lighting sensitivity and calibration needs, the integration of machine learning and computer vision ensures robust performance, making it a promising solution for hands-free interaction.

2.2 Survey Findings

- Utilizing MediaPipe for eye landmark detection and tracking to capture precise eye movement data [6][9].
- Implementing a user-friendly calibration process to adjust the bounding box around the eye area, which enhances accuracy in tracking movements [1][9].
- Setting up a bounding box to define the active tracking area and improve control over eye movement detection, creating a clear region for interaction [1][6].
- Enabling mouse click operations based on eye gestures, such as blinks or specific gaze directions, to facilitate hands-free control [3][8].
- Leveraging computer vision techniques, specifically thresholding and contour detection, to isolate and track the pupil within the bounding box for more precise control [9].

Chapter 3

SOFTWARE REQUIREMENT SPECIFICATION

3.1 What is Software Requirement Specification

Software Requirement Specification (SRS) is a comprehensive document that provides a detailed description of the functionalities, features, and performance criteria for the software system. It serves as a foundation for both the development team and stakeholders to ensure that the system meets the users' needs. The SRS is a formal document that outlines the requirements of the software, detailing what the system should do, how it should behave, and the constraints under which it should operate. In essence, it acts as a bridge between the end-users and developers, ensuring that the software's design and implementation align with the user's expectations.

3.2 FUNCTIONAL REQUIREMENTS:

The functional requirements define the core operations and features of the system:

1. **User Setup:**
    - Allows selection of dominant and non-dominant eyes for cursor control and click detection.
2. **Calibration Process:**
    - Guides the user to look at screen corners for dynamic bounding box creation.
3. **Eye Tracking:**
    - Tracks pupil movements to control the cursor's position on the screen.
4. **Click Simulation:**
    - Detects intentional blinks for simulating mouse clicks.
5. **Real-Time Adaptation:**
    - Dynamically maps eye movements to screen coordinates for smooth cursor control.
6. **Platform Independence:**
    - Web-based, accessible through various browsers without extra installations.
7. **Accessibility:**
    - Designed for users with limited mobility to operate the system hands-free.

3.2.1 Non-Functional Requirements:

Non-functional requirements define the quality and constraints of the system:

1. **Performance:**
    - Low latency, smooth cursor control.
2. **Scalability:**
    - Adaptable to different screen sizes and resolutions.
3. **Usability:**
    - Intuitive and easy-to-use interface for all users.
4. **Reliability:**
    - Consistent performance without crashes or errors.
5. **Security:**
    - Ensures privacy and protection of user data (webcam feed).
6. **Maintainability:**
    - Easy to update and debug.
7. **Resource Efficiency:**
    - Low CPU and memory usage, ensuring efficient operation.
8. **Portability:**
    - Compatible with various browsers and devices

3.3 Software and Hardware Specification

3.3.1 Software Specification

1. **Operating System:**
    - Runs on any modern OS (Windows, macOS, Linux) via a web browser.
2. **Web Framework:**
    - **Flask** for the web-based interface.

3. **Libraries:**
    - **Mediapipe** for facial landmark detection and eye tracking.
    - **OpenCV** for image processing.
    - **PyAutoGUI** for simulating mouse actions.
4. **Browser Support:**
    - Works across major browsers (Chrome, Firefox, Safari).

3.3.2 Hardware Requirements

1. **Webcam:**
    - Standard webcam (720p resolution or higher) for capturing eye movements.
2. **Computer/Device:**
    - Any device with a modern browser.
        - Minimum specs: 1.5 GHz CPU, 4 GB RAM, 100 MB free storage.
3. **Internet Connection:**
    - Required for accessing the web app and real-time data processing.
4. **Display:**
    - Supports various screen sizes and resolutions.

Chapter 4

IMPLEMENTATION

The implementation of the Eye Tracking and Gesture-Based Control System involves the integration of several components including real-time eye tracking, gesture recognition, and mouse control. This section details the design and development of each key component of the system.

4.1 System Architecture

The system is designed with a modular architecture to ensure flexibility, scalability, and ease of maintenance. The following major components are integrated into the system:

- **Web Interface (Flask-based):** Handles the user interactions for calibration, tracking, and system control.
- **Eye Tracking and Gesture Recognition (using MediaPipe):** Responsible for tracking eye movements and detecting blinks for mouse cursor control.
- **Mouse Control (using PyAutoGUI):** Moves the mouse cursor based on eye movements and simulates clicks based on blink detection.

4.2 Tools and Technologies

The following tools and libraries were used to develop the Eye Tracking and Gesture-Based Control System:

- **Python 3.12.8**: Programming language used for system development.
- **Flask**: A micro web framework used to develop the web interface for calibration and control.
- **MediaPipe**: A framework for face and hand landmark detection, used for eye tracking and gesture recognition.
- **OpenCV**: For real-time video capture and image processing.
- **PyAutoGUI**: For simulating mouse movements and clicks.

4.3 Detailed Implementation

4.3.1 Web Interface (Flask)

The system uses Flask to provide a web interface where users can interact with the system. The web interface has several pages:

- **Calibration Page**: Guides users to calibrate the system by selecting their dominant eye and defining the calibration points on the screen.
- **Tracking Page**: Displays the live video feed and initiates eye tracking.
- **Real-Time Video Feed**: Streams the webcam feed to the browser to visualize eye tracking.

The Flask routes are structured as follows:

- **/calibration**: For calibrating the user's eye and selecting the dominant eye.
- **/calibration/corners**: For calibration where users define the bounding box area.
- **/tracking**: To start the eye tracking after calibration.

Example of code to render the calibration page:

```
@app.route('/calibration', methods=['GET', 'POST'])
def calibration():
      # Handle user input for dominant eye and calibration points
      return render_template('calibration.html')
```

4.3.2 Eye Tracking and Gesture Recognition (MediaPipe)

The system uses MediaPipe's FaceMesh solution to detect facial landmarks and track eye movements. MediaPipe provides a set of facial landmarks that are used to track the position of the pupil in real-time. The following steps are followed to detect and track the pupil:

- Capture video from the webcam.
- Convert each frame to RGB.
- Process the frame using MediaPipe to detect facial landmarks.
- Extract pupil coordinates and track the movement of the dominant eye.

The pupil positions are used to control the cursor's movement on the screen.

Example of pupil detection code:

```
results = face_mesh.process(rgb_image)
if results.multi_face_landmarks:
      for face_landmarks in results.multi_face_landmarks:
            pupil = face_landmarks.landmark[PUPIL_INDEX]
            pupil_x = int(pupil.x * w)
            pupil_y = int(pupil.y * h)
```

4.3.3 Mouse Control (PyAutoGUI)

The mouse cursor is controlled using PyAutoGUI, which simulates mouse movements and clicks. Based on the detected eye position (pupil coordinates), the cursor is moved. Blinks are detected by monitoring the distance between eye landmarks (upper and lower eyelids), and if the distance drops below a threshold, a click event is simulated.

Example code for moving the mouse based on eye position:

```
pyautogui.moveTo(int(smooth_x), int(smooth_y))
```

And for Simulating a blink:

```
if blink:
      pyautogui.click()
```

4.3.4 Calibration and Bounding Box Calculation

The system requires calibration to map eye movements to screen coordinates. Users are asked to define four corners on the screen (top-left, top-right, bottom-left, bottom-right). These points are used to compute a dynamic bounding box that maps eye movements to the screen.

Bounding box calculation code:

```
x_coords = [pt[0] for pt in corner_points]
y_coords = [pt[1] for pt in corner_points]

# Calculate bounding box dimensions (min and max x, y coordinates)
min_x, max_x = min(x_coords), max(x_coords)
min_y, max_y = min(y_coords), max(y_coords)
box_width = max_x - min_x
box_height = max_y - min_y

# Update the bounding box ratio relative to the screen dimensions
screen_width, screen_height = pyautogui.size()
width_ratio = box_width / screen_width
height_ratio = box_height / screen_height
```

```
# Use the larger of the two ratios for consistency
bounding_box_ratio = max(width_ratio, height_ratio)
if bounding_box_ratio < 0.1:
    bounding_box_ratio = 0.1
if bounding_box_ratio > 0.7:
    bounding_box_ratio = 0.7
```

Mathematical Formula for Bounding Box Calculation:

Given the corner points (x_1, y_1), (x_2, y_2), (x_3, y_3), and (x_4, y_4) where:

- (x_1, y_1) is the top-left corner
- (x_2, y_2) is the top-right corner
- (x_3, y_3) is the bottom-left corner
- (x_4, y_4) is the bottom-right corner

THE BOUNDING BOX WIDTH AND HEIGHT CAN BE CALCULATED AS FOLLOWS:

1. **BOUNDING BOX WIDTH:**

$$\text{box\_width} = \max(x_1, x_2, x_3, x_4) - \min(x_1, x_2, x_3, x_4)$$

2. **BOUNDING BOX HEIGHT:**

$$\text{box\_height} = \max(y_1, y_2, y_3, y_4) - \min(y_1, y_2, y_3, y_4)$$

Next, calculate the bounding box ratios relative to the screen dimensions (denoted as screen_width and screen_height):

3. **WIDTH RATIO:**

$$\text{width\_ratio} = \frac{\text{box\_width}}{\text{screen\_width}}$$

4. **HEIGHT RATIO:**

$$\text{height\_ratio} = \frac{\text{box\_height}}{\text{screen\_height}}$$

5. **Bounding Box Ratio**:

$$\text{bounding\_box\_ratio} = \max(\text{width\_ratio}, \text{height\_ratio})$$

And this ratio is clamped within the limits (0.1 and 0.7):

6. **Clamped Bounding Box Ratio**:

$$\text{bounding\_box\_ratio} = \min(\max(\text{bounding\_box\_ratio}, 0.1), 0.7)$$

4.3.5 Mapping Cursor Movements

The points of the pupil obtained from the mediapipe library is mapped to the screen by the formula.

- (x, y): The current point coordinates relative to the bounding box.
- $\text{min\_x}, \text{max\_x}, \text{min\_y}, \text{max\_y}$: The bounding box's boundaries.
- $\text{screen\_width}, \text{screen\_height}$: Dimensions of the screen.

The mapping formulas are as follows:

Step 1: Normalize Cursor Position

Normalize the cursor's coordinates to the range $[0, 1]$ based on the bounding box:

$$\text{normalized\_x} = \frac{x - \text{min\_x}}{\text{max\_x} - \text{min\_x}}$$

$$\text{normalized\_y} = \frac{y - \text{min\_y}}{\text{max\_y} - \text{min\_y}}$$

Step 2: Map to Screen Dimensions

Scale the normalized coordinates to screen dimensions:

$$\text{screen\_x} = \text{normalized\_x} \times \text{screen\_width}$$

$$\text{screen\_y} = \text{normalized\_y} \times \text{screen\_height}$$

4.4 System Integration

Once all individual components are developed, they are integrated into the Flask web application. The video feed, eye tracking, and mouse control are linked through Flask routes to provide real-time feedback and control.

User Calibration: The user sets their dominant eye and calibration points through the web interface.

Eye Tracking and Mouse Control: After calibration, eye movements control the cursor, and blinks simulate mouse clicks.

4.5 Testing and Debugging

The system was tested using a webcam in a controlled environment. The following test cases were performed:

1. **Eye Tracking Accuracy:** The system correctly maps eye movements to screen movements.
2. **Blink Detection:** Verified that a blink triggers a mouse click.

4.6 Challenges and Solutions

Latency in Eye Tracking: The initial system had a latency in cursor movement. This was reduced by applying smoothing techniques to the cursor position using an averaging method between the current and previous positions.

Calibration Accuracy: Users had difficulty accurately selecting calibration points. The solution was to include visual cues during calibration to guide the user.

FLOW CHART :

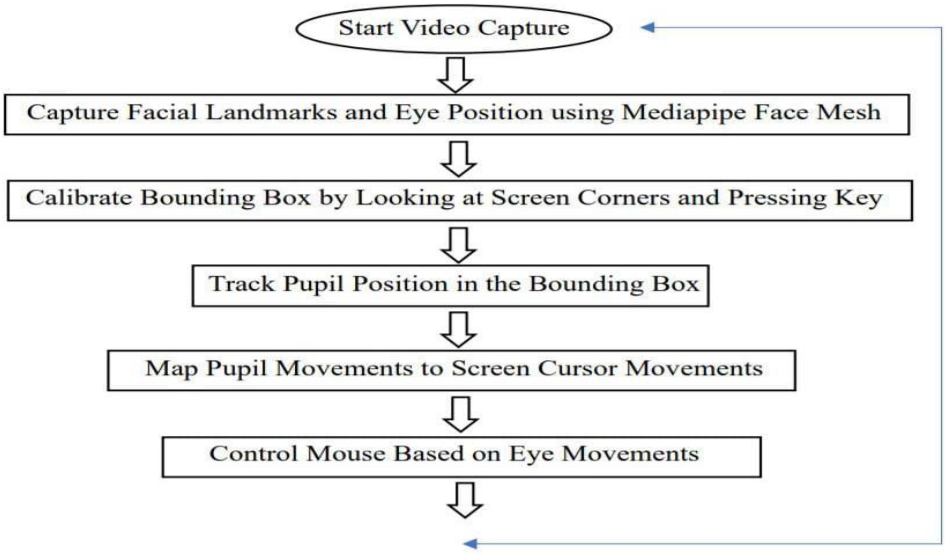

Figure 4.6.1

Chapter 5

CODES

Directory Structure :

```
web-app /
├── app.py
├── templates/
    ├── calibration_corners.html
    ├── calibration.html
    ├── tracking.html
```

app.py:

```
import time
from flask import Flask, render_template, Response, request, redirect, url_for,
jsonify
import cv2
import pyautogui
import mediapipe as mp

mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(static_image_mode=False, max_num_faces=1,
refine_landmarks=True,
                                  min_detection_confidence=0.7)

app = Flask(__name__)
camera = None

tracking_active = False
dominant_eye = None
bounding_box_ratio = 0.1
corner_points = []
corner_labels = ['top-left', 'top-right', 'bottom-left', 'bottom-right']

LEFT_EYE_PUPIL_INDEX = 468
RIGHT_EYE_PUPIL_INDEX = 473
eye_landmarks = {
    'Top Left Eyelid': 159, 'Bottom Left Eyelid': 145,
    'Top Right Eyelid': 386, 'Bottom Right Eyelid': 374,
}
pyautogui.FAILSAFE = False

screen_width, screen_height = pyautogui.size()

prev_x, prev_y = 0, 0
```

```
@app.route('/')
def index():
    return render_template('calibration.html')


@app.route('/calibration', methods=['GET', 'POST'])
def calibration():
    global dominant_eye, corner_points, PUPIL_INDEX

    corner_points = []
    if request.method == 'POST':
        if 'dominant_eye' in request.form:
            dominant_eye = request.form['dominant_eye']
            if str(dominant_eye) == "left":
                print("left eye dominant")
                PUPIL_INDEX = LEFT_EYE_PUPIL_INDEX
            else:
                print("right eye dominant")
                PUPIL_INDEX = RIGHT_EYE_PUPIL_INDEX
            return redirect(url_for('calibration_corners'))
    return render_template('calibration.html')


cap = cv2.VideoCapture(0)


def generate_video_stream():
    global cap
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        _, buffer = cv2.imencode('.jpg', frame)
        frame_bytes = buffer.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n')


@app.route('/video_feed_corner')
def video_feed_corner():
    return Response(generate_video_stream_with_pupil(), mimetype='multipart/x-
mixed-replace; boundary=frame')




def generate_video_stream_with_pupil():
    global cap, face_mesh, PUPIL_INDEX
```

```
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        frame = cv2.flip(frame, 1)

        rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        results = face_mesh.process(rgb_image)

        h, w, _ = frame.shape

        if results.multi_face_landmarks:
            for face_landmarks in results.multi_face_landmarks:
                pupil = face_landmarks.landmark[PUPIL_INDEX]
                pupil_x = int(pupil.x * w)
                pupil_y = int(pupil.y * h)

                cv2.circle(frame, (pupil_x, pupil_y), 5, (0, 255, 0), -1)

        _, buffer = cv2.imencode('.jpg', frame)
        frame_bytes = buffer.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n')


@app.route('/calibration/corners', methods=['GET', 'POST'])
def calibration_corners():
    global corner_points, cap, PUPIL_INDEX
    print(f"Using PUPIL_INDEX: {PUPIL_INDEX}")

    if request.method == 'POST':
        ret, frame = cap.read()

        if not ret:
            return "Error: Could not capture frame", 500

        frame = cv2.flip(frame, 1)
        frame = resize_frame(frame)
        rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        results = face_mesh.process(rgb_image)

        h, w, _ = frame.shape
        print("Landmarks detected:", bool(results.multi_face_landmarks))

        if results.multi_face_landmarks:
            for face_landmarks in results.multi_face_landmarks:
                pupil = face_landmarks.landmark[PUPIL_INDEX]
```

```
                pupil_x = int(pupil.x * w)
                pupil_y = int(pupil.y * h)

                cv2.circle(frame, (pupil_x, pupil_y), 5, (0, 255, 0), -1)

        center_x, center_y = w // 2, h // 2
        corner_points.append((center_x, center_y))

        if len(corner_points) == 4:
            print("Calibration points:", corner_points)
            cap.release()
            return redirect(url_for('tracking'))
    corner_visible = corner_labels[len(corner_points)]
    if len(corner_points) <= 4:
        return render_template('calibration_corners.html',
                               corner_label=corner_labels[len(corner_points)],
                               corner_visible=corner_visible)
    else:
        return redirect(url_for('tracking'))


@app.route('/tracking')
def tracking():
    update_bounding_box_ratio()
    return render_template('tracking.html')


@app.route('/start', methods=['POST'])
def start_tracking():
    global tracking_active, camera
    if camera is None or not camera.isOpened():
        camera = cv2.VideoCapture(0)

    tracking_active = True
    print("Tracking Status : ", tracking_active)

    return "Tracking started", 200


@app.route('/stop', methods=['POST'])
def stop_tracking():
    global tracking_active
    tracking_active = False
    print("Tracking Status : ", tracking_active)
    return "Tracking stopped", 200



@app.route('/video_feed')
def video_feed():
    global tracking_active
```

```
    print("Tracking Status : ", tracking_active)
    if tracking_active:
        return Response(generate_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')
    return "Tracking not active", 200


def map_range(value, from_min, from_max, to_min, to_max):
    return (value - from_min) * (to_max - to_min) / (from_max - from_min) + to_min


def resize_frame(frame, scale=1):
    width = int(frame.shape[1] * scale)
    height = int(frame.shape[0] * scale)
    return cv2.resize(frame, (width, height), interpolation=cv2.INTER_AREA)


def check_blink(landmarks):
    left_eye = abs(landmarks[0][1] - landmarks[1][1])
    right_eye = abs(landmarks[2][1] - landmarks[3][1])

    # Threshold for blinking
    blink_threshold = 15
    if left_eye < blink_threshold and right_eye < blink_threshold:
        return False  # Both eyes closed
    elif left_eye < blink_threshold and PUPIL_INDEX == RIGHT_EYE_PUPIL_INDEX:
        return True  # Right eye controlling cursor, left eye used for clicking
    elif right_eye < blink_threshold and PUPIL_INDEX == LEFT_EYE_PUPIL_INDEX:
        return True  # Left eye controlling cursor, right eye used for clicking
    else:
        return None

def update_bounding_box_ratio():
    global corner_points, bounding_box_ratio

    if len(corner_points) != 4:
        print("Error: Insufficient corner points for bounding box calculation.")
        return

    x_coords = [pt[0] for pt in corner_points]
    y_coords = [pt[1] for pt in corner_points]

    # Calculate bounding box dimensions
    min_x, max_x = min(x_coords), max(x_coords)
    min_y, max_y = min(y_coords), max(y_coords)
    box_width = max_x - min_x
    box_height = max_y - min_y

    # Update the bounding box ratio relative to the screen dimensions
    screen_width, screen_height = pyautogui.size()
    width_ratio = box_width / screen_width
```

```
    height_ratio = box_height / screen_height

    # Use the larger of the two ratios for consistency
    bounding_box_ratio = max(width_ratio, height_ratio)
    if bounding_box_ratio < 0.1:
        bounding_box_ratio = 0.1
    if bounding_box_ratio > 0.7:
        bounding_box_ratio = 0.7
    print(f"Updated bounding_box_ratio: {bounding_box_ratio}")


def generate_frames():
  global tracking_active, prev_x, prev_y, bounding_box_ratio
  print(bounding_box_ratio)

  # Bounding box dimensions
  bounding_box_width = screen_width * bounding_box_ratio
  bounding_box_height = screen_height * bounding_box_ratio

  while tracking_active:
    success, frame = camera.read()
    if not success:
      break

    image = cv2.flip(frame, 1)
    image = resize_frame(image)
    rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    results = face_mesh.process(rgb_image)

    h, w, _ = image.shape

    if results.multi_face_landmarks:
      for face_landmarks in results.multi_face_landmarks:
        pupil = face_landmarks.landmark[PUPIL_INDEX]
        pupil_x = int(pupil.x * w)
        pupil_y = int(pupil.y * h)

        cv2.circle(image, (pupil_x, pupil_y), 5, (0, 255, 0), -1)

        # Blink Detection Part
        landmarks = []
        for eye_part, index in eye_landmarks.items():
          lm = face_landmarks.landmark[index]
          x, y = int(lm.x * w), int(lm.y * h)
          landmarks.append([x, y])

          if PUPIL_INDEX == RIGHT_EYE_PUPIL_INDEX:
            if eye_part == 'Top Left Eyelid' or eye_part == 'Bottom Left Eyelid':
              cv2.circle(image, (x, y), 3, (0, 255, 0), -1)
```

```
          else:
            if eye_part == 'Top Right Eyelid' or eye_part == 'Bottom Right Eyelid':
              cv2.circle(image, (x, y), 3, (0, 255, 0), -1)

        blink = check_blink(landmarks)
        if blink:
          print("Blink Detected!")
          pyautogui.click()
          cv2.putText(image, "Blink Detected! Clicking...", (20, 120),
                      cv2.FONT_HERSHEY_PLAIN, 2, (0, 0, 255), 2)

        # Calculate dynamic bounding box
        box_top_left = (
          int(w / 2 - bounding_box_width / 2),
          int(h / 2 - bounding_box_height / 2)
        )
        box_bottom_right = (
          int(w / 2 + bounding_box_width / 2),
          int(h / 2 + bounding_box_height / 2)
        )

        cv2.rectangle(image, box_top_left, box_bottom_right, (255, 0, 0), 2)

        if (box_top_left[0] <= pupil_x <= box_bottom_right[0] and
            box_top_left[1] <= pupil_y <= box_bottom_right[1]):
          mapped_x = map_range(pupil_x, box_top_left[0],
                               box_bottom_right[0], 0, screen_width)
          mapped_y = map_range(pupil_y, box_top_left[1],
                               box_bottom_right[1], 0, screen_height)

          alpha = 0.5  # Adjust smoothing factor for faster cursor response
          smooth_x = alpha * mapped_x + (1 - alpha) * prev_x
          smooth_y = alpha * mapped_y + (1 - alpha) * prev_y

          try:
            pyautogui.moveTo(int(smooth_x), int(smooth_y))
          except pyautogui.FailSafeException:
            pyautogui.moveTo(0, 0)

          prev_x, prev_y = smooth_x, smooth_y

    _, buffer = cv2.imencode('.jpg', image)
    image = buffer.tobytes()

    yield (b'--frame\r\n'
           b'Content-Type: image/jpeg\r\n\r\n' + image + b'\r\n')


if __name__ == '__main__':
    app.run(debug=True)
```

calibration.html :

```
<!DOCTYPE html>
<html>
<head>
    <title>Calibration - Dominant Eye</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background: #f9f9f9; /* Light background */
            color: #333; /* Dark text for readability */
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }
        h1 {
            font-size: 2rem;
            margin-bottom: 20px;
            text-align: center;
            color: #444; /* Subtle heading color */
        }
        form {
            display: flex;
            justify-content: center;
            gap: 15px;
        }
        button {
            padding: 15px 30px;
            font-size: 1.2rem;
            color: #fff;
            background-color: #007BFF;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            transition: background-color 0.3s ease, transform 0.2s ease;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
        }
        button:hover {
            background-color: #0056b3;
            transform: scale(1.05);
        }


         button:active {
            transform: scale(0.95);
        }
    </style>
</head>
```

```
<body>
    <div>
        <h1>Calibration - Select Your Dominant Eye</h1>
        <form method="POST" action="/calibration">
            <button name="dominant_eye" value="left">Left Eye</button>
            <button name="dominant_eye" value="right">Right Eye</button>
        </form>
    </div>
</body>
</html>
```

calibration_corner.html :

```
<!DOCTYPE html>
<html>
<head>
    <title>Calibration - Corners</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background: #f9f9f9; /* Light background */
            color: #333; /* Dark text for readability */
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            height: 100vh;
            overflow: hidden;
        }
        h1 {
            font-size: 2rem;
            margin-bottom: 20px;
            text-align: center;
            color: #444; /* Subtle heading color */
        }
        #video_feed {
            width: 640px;
            height: 480px;
            border: 3px solid #007BFF;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            margin-bottom: 20px;
        }
        form {
            margin-top: 10px;
        }
        button {
            padding: 10px 20px;
```

```
            font-size: 1rem;
            color: #fff; /* White text for contrast */
            background-color: #007BFF; /* Modern blue color */
            border: none;
            border-radius: 5px;
            cursor: pointer;
            transition: background-color 0.3s ease, transform 0.2s ease;
        }
        button:hover {
            background-color: #0056b3; /* Darker blue on hover */
            transform: scale(1.05);
        }
        button:active {
            transform: scale(0.95);
        }
        .corner {
            position: absolute;
            width: 20px; /* Keeps size */
            height: 20px; /* Keeps size */
            background-color: #FFD700; /* Bright yellow for visibility */
            border-radius: 50%; /* Ensures the dot remains circular */
            opacity: 0; /* Hidden by default */
            transition: opacity 0.3s ease;
        }
        .corner.top-left {
            top: 10px;
            left: 10px;
        }
        .corner.top-right {
            top: 10px;
            right: 10px;
        }
        .corner.bottom-left {
            bottom: 10px;
            left: 10px;
        }
        .corner.bottom-right {
            bottom: 10px;
            right: 10px;
        }

        .visible {
            opacity: 1; /* Makes the corner visible */
        }
    </style>
</head>
<body>
    <h1>Calibration - Look at {{ corner_label }} corner</h1>
    <img id="video_feed" src="/video_feed_corner" alt="Video Feed">
    <form method="POST" action="/calibration/corners">
        <button type="submit">Capture</button>
```

```
    </form>
    <div class="corner top-left {% if corner_visible == 'top-left' %}visible{%
endif %}"></div>
    <div class="corner top-right {% if corner_visible == 'top-right' %}visible{%
endif %}"></div>
    <div class="corner bottom-left {% if corner_visible == 'bottom-left'
%}visible{% endif %}"></div>
    <div class="corner bottom-right {% if corner_visible == 'bottom-right'
%}visible{% endif %}"></div>
</body>
</html>
```

tracking.html :

```
<!DOCTYPE html>
<html>
<head>
    <title>Eye Tracking Cursor Control</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background: #f9f9f9; /* Light background */
            color: #333; /* Dark text for readability */
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            height: 100vh;
            overflow: hidden;
        }
        h1 {
            font-size: 2.5rem;
            margin-bottom: 20px;
            color: #444; /* Subtle heading color */
            text-shadow: 1px 1px 3px rgba(0, 0, 0, 0.1);
        }
        .button-container {
            display: flex;
            gap: 15px; /* Space between buttons */
            margin-bottom: 20px;
        }
        button {
            padding: 10px 20px;
            font-size: 1.2rem;
            color: #fff;
            background-color: #007BFF;
```

```
            border: none;
            border-radius: 8px;
            cursor: pointer;
            transition: background-color 0.3s ease, transform 0.2s ease;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
        }
        button:hover {
            background-color: #0056b3;
            transform: scale(1.05);
        }
        button:active {
            transform: scale(0.95);
        }
        #video {
            width: 60%;
            border: 3px solid #007BFF;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        }
    </style>
    <script>
        let videoFeedAvailable = false;

        function startTracking() {
            fetch('/start', { method: 'POST' })
                .then(response => {
                    if (response.ok) {
                        console.log("Tracking started");
                        videoFeedAvailable = true;
                        updateVideoFeed();
                    } else {
                        console.error("Failed to start tracking");
                    }
                });
        }

        function stopTracking() {
            fetch('/stop', { method: 'POST' })
                .then(response => {
                    if (response.ok) {
                        console.log("Tracking stopped");
                        videoFeedAvailable = false;
                        updateVideoFeed();
                    } else {
                        console.error("Failed to stop tracking");
                    }
                });
        }

        function updateVideoFeed() {
            const videoElement = document.getElementById('video');
```

```
            if (videoFeedAvailable) {
                // Show video element
                if (!videoElement) {
                    const videoTag = document.createElement('img');
                    videoTag.id = 'video';
                    videoTag.src = "/video_feed";
                    videoTag.alt = "Live Video Feed";
                    videoTag.style.width = "60%";
                    videoTag.style.border = "3px solid #007BFF";
                    videoTag.style.borderRadius = "10px";
                    videoTag.style.boxShadow = "0 4px 8px rgba(0, 0, 0, 0.1)";
                    document.body.appendChild(videoTag);
                }
            } else {
                // Hide video element
                if (videoElement) {
                    videoElement.src = ""; // Clear the feed
                    document.body.removeChild(videoElement); // Remove the image
element
                }
            }
        }
    </script>
</head>
<body>
    <h1>Eye Tracking Cursor Control</h1>
    <div class="button-container">
        <button onclick="startTracking()">Start Tracking</button>
        <button onclick="stopTracking()">Stop Tracking</button>
    </div>
</body>
</html>
```

Chapter 6

Snapshots

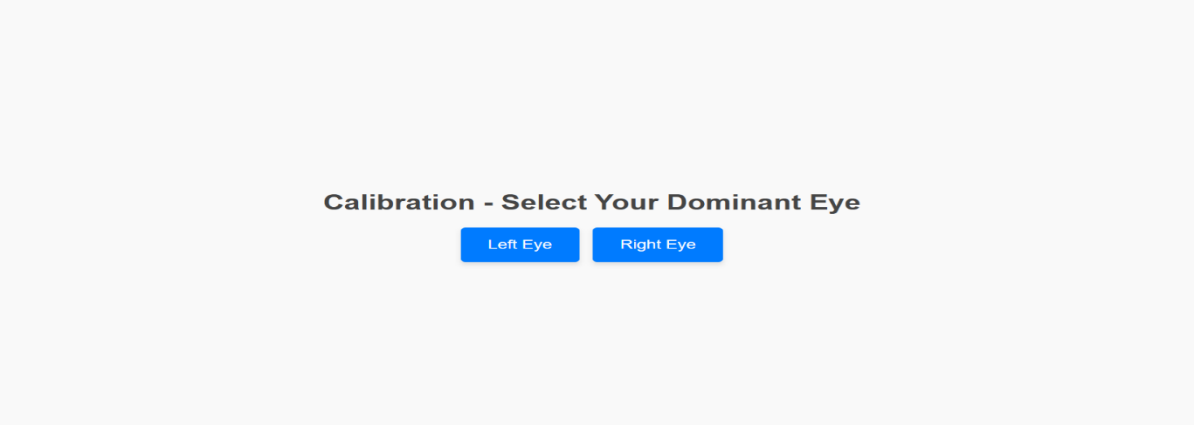

Figure 6.1: The calbiration page

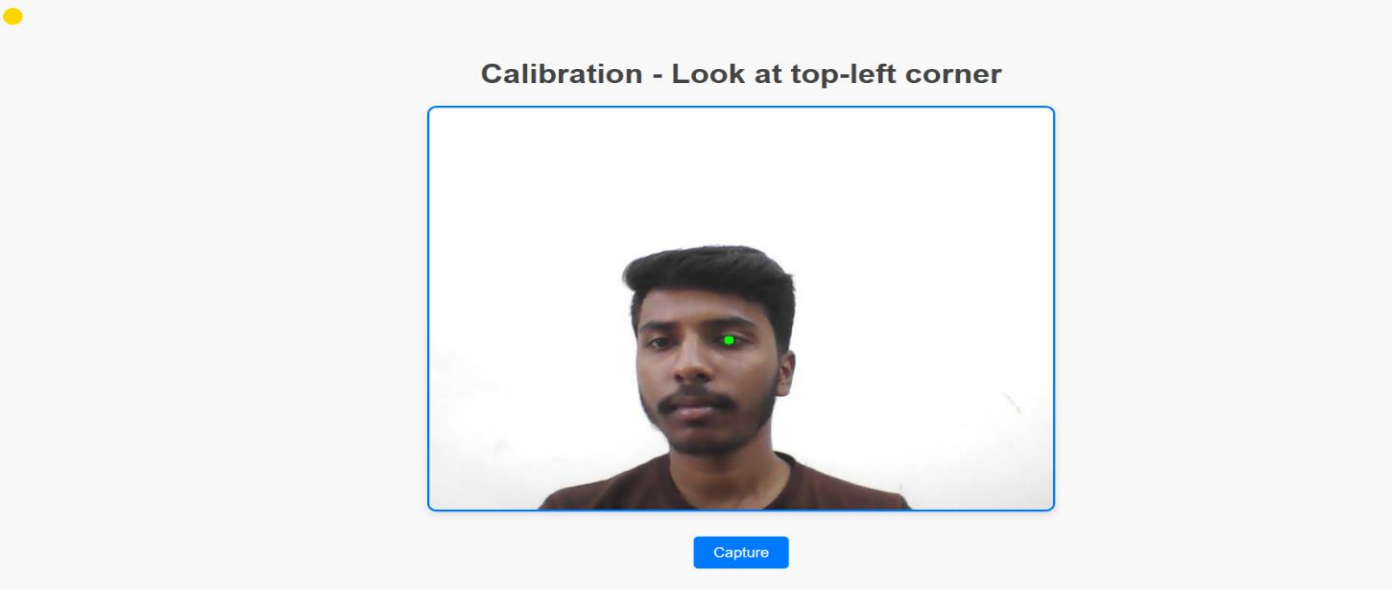

Figure 6.2: The calbiration page Lokking at top-left corner

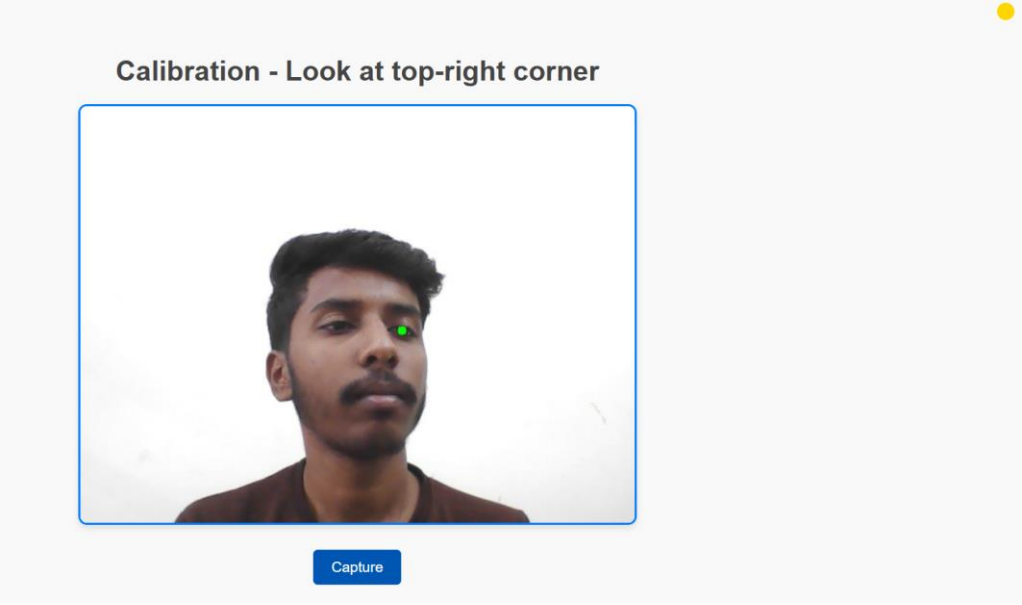

Figure 6.3: The calbiration page Lokking at top-right corner

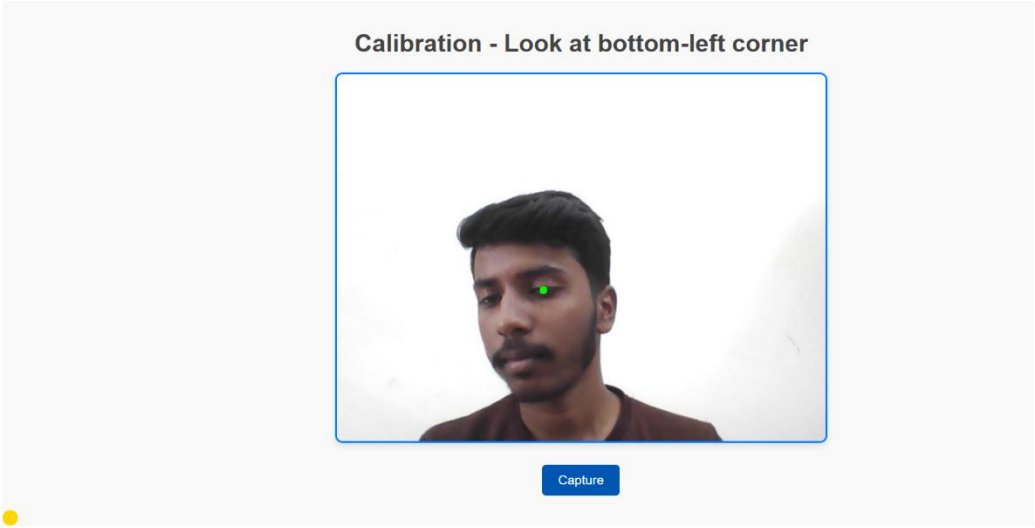

Figure 6.4: The calbiration page Lokking at bottom-left corner

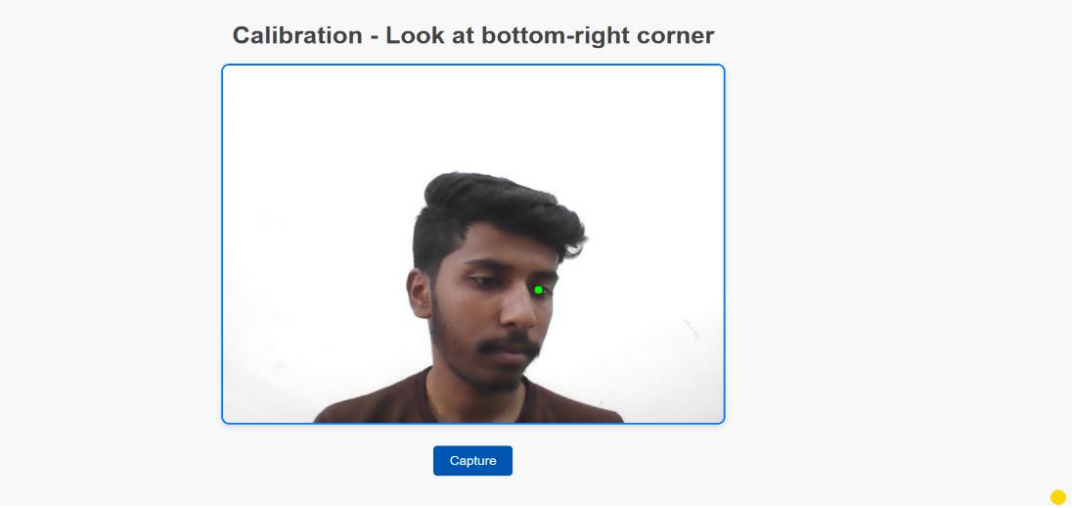

Figure 6.5: The calbiration page Lokking at bottom-right corner

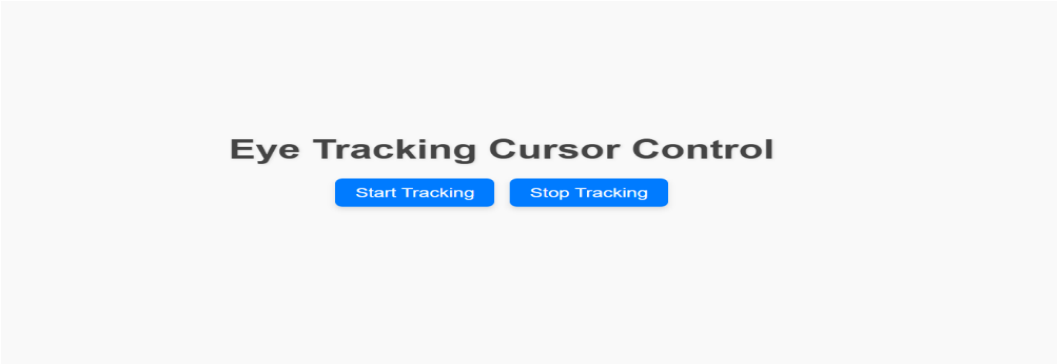

Figure 6.6 The Tracking page:

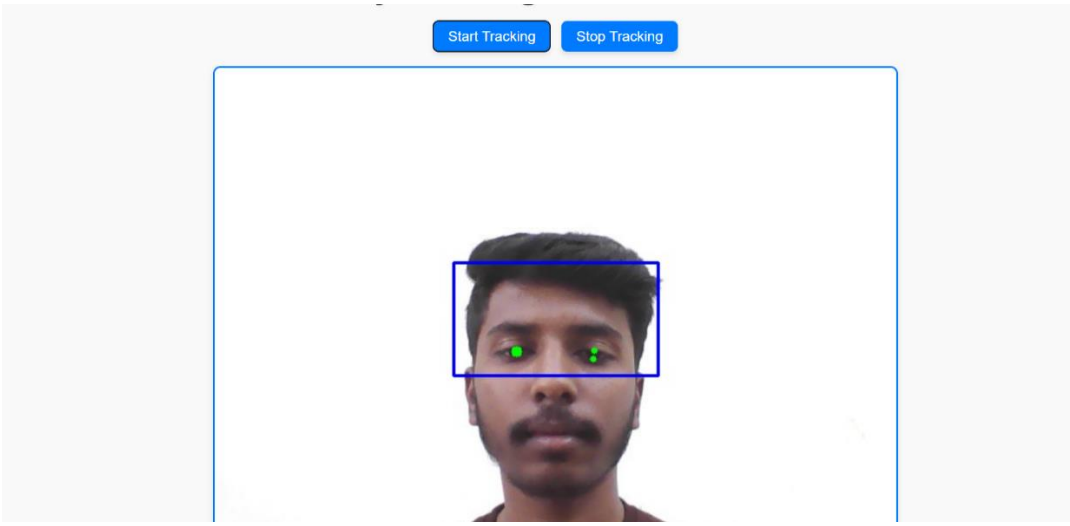

Figure 6.7 : The Tracking page:

FUTURE SCOPE and CONCLUSION

Future Scope

While the current implementation of the project provides an effective solution for hands-free control using eye movements and blinks, there are several areas where the project can be expanded and enhanced in the future. Below are some potential future directions for the development of this system:

VR/AR Integration:

Expanding the system to support Virtual Reality (VR) and Augmented Reality (AR) environments would allow for more immersive and intuitive interactions. This would enable the user to control virtual objects or navigate through 3D spaces using just eye movements and blinks

Advanced Gaze Interactions:

Developing more sophisticated gaze-based interactions, such as selecting items or triggering actions based on gaze duration or intensity, would make the system more versatile and user-friendly. These advanced interactions could improve the user experience in both productivity and entertainment applications.

Customizable Control Schemes:

Offering users the ability to customize control schemes would make the system more adaptable to individual preferences and needs. Users could configure sensitivity settings, blink duration thresholds, and other parameters to optimize the interface for their specific use case.

Enhanced Blink Detection:

Improving blink detection algorithms using deep learning models could make the system more accurate, particularly under various lighting conditions and for users with different eye characteristics. This would lead to more consistent user experiences.

Head Movement Tracking:

Integrating head movement tracking alongside eye tracking would enhance cursor control and interaction precision, particularly in 3D or virtual environments. This integration would provide a more comprehensive control scheme for users in different settings.

Conclusion

- The Eye Tracking and Gesture-Based Control System successfully demonstrates a hands-free, accessible solution for human-computer interaction (HCI).
- By leveraging real-time eye-tracking technology and gesture recognition, this system offers a valuable alternative to traditional input devices such as keyboards and mice, especially for individuals with physical disabilities or limited mobility.
- Throughout the project, we focused on developing a system that is cost-effective, easy to use, and capable of providing an intuitive interface.
- Using readily available hardware, such as standard webcams, along with advanced libraries like MediaPipe and PyAutoGUI, we were able to build a robust platform for controlling a computer by tracking eye movements and detecting blinks for mouse clicks.

Key achievements of this project include:

- **Real-time Eye Tracking:** The system accurately tracks the user's eye movements and translates them into corresponding movements of the mouse cursor on the screen.
- **Blink Detection for Mouse Clicks:** A reliable mechanism was developed to detect blinks and simulate mouse clicks, adding an extra layer of functionality to the system.
- **Calibration Process:** An easy-to-use calibration procedure was integrated, allowing users to define their dominant eye and establish calibration points to enhance tracking accuracy.
- **Web Interface:** A Flask-based interface provides a simple way for users to interact with the system, manage calibration, and start or stop tracking.

In conclusion, this project serves as a foundation for developing more advanced assistive technologies, contributing to making computing more accessible and inclusive. The system showcases the potential of eye tracking and gesture-based interfaces to empower users who may otherwise struggle with traditional input methods, opening up new possibilities for accessibility and innovation in human-computer interaction.

Chapter 8

References

1. **P. Miah, M. R. Gulshan, and N. Jahan, "Mouse cursor movement and control using eye gaze- A human computer interaction," IEEE, 2022.**
2. **D. Balakrishnan, U. Mariappan, V. Niteesh, Y. A. Reddy, V. M. Reddy, and V. V. Reddy, "Real-time eye-tracking mouse control using Recurrent Neural Network," in Proc. Int. Conf. Integrated Intelligence and Communication Systems (ICIICS), 2023.**
3. **R. N. Phursule, G. Y. Kakade, A. Koul, and S. Bhasin, "Virtual mouse and gesture-based keyboard," in Proc. Int. Conf. Computing, Communication, Control and Automation (ICCUBEA), 2023.**
4. **N. L. Reddy, R. Murugeswari, M. Imran, N. Subhash, N. V. K. Reddy, and N. B. Adarsh, "Virtual mouse using hand and eye gestures," in Proc. Int. Conf. Integrated Intelligence and Communication Systems (ICIICS), 2023.**
5. **[P. M. Naidu, N. Muthukumaran, S. Chandralekha, and K. T. Reddy, "An analysis on virtual mouse control using human eye," in Proc. 5th Int. Conf. Image Processing and Capsule Networks (ICIPCN), 2024.**
6. **Mediapipe Face Mesh Documentation – https://mediapipe.readthedocs.io/en/latest/solutions/face_mesh.html**
7. **OpenCV Documentation – https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html**
8. **Flask Documentation – https://flask.palletsprojects.com/**
9. **W3Schools – https://www.w3schools.com/**
10. **GeeksForGeeks – https://www.geeksforgeeks.org/**
11. **Stack Overflow – https://stackoverflow.com/**