A

MINI PROJECT REPORT

ON

Gesture Controlled Cursor

Submitted in partial fulfillment of the requirements
For the award of Degree of

BACHELOR OF ENGINEERING

IN

CSE (Artificial Intelligence & Machine Learning)

Submitted By

Sigalampatla Premsai	**245322733182**
Murtaza Mohammed Rafiqi	**245322733167**
Pattan Abdul Khayum	**245322733170**

Under the Supervision

Of

Dr. R. Srilakshmi

ASSISTANT PROFESSOR

Department of CSE(AIML)

NEIL GOGTE INSTITUTE OF TECHNOLOGY

Kachavanisingaram Village, Hyderabad, Telangana 500058. **February 2025**

NEIL GOGTE INSTITUTE OF TECHNOLOGY

A Unit of Keshav Memorial Technical Education (KMTES) Approved by
AICTE, New Delhi & Affiliated to **2012-2016** Osmania University, Hyderabad

CERTIFICATE

This is to certify that the project work (PW533CSM) entitled "**GESTURE CONTROLLED CURSOR"** *is a bonafide work carried out by* **Sigalampatla Premasai (245322733182), Murtaza Mohammed Rafiqi (245322733167), Pattan Abdul Khayum (245322733170)** of III-year V semester **Bachelor of Engineering** *in* **CSE (Artificial Intelligence & Machine Learning)** *by Osmania University, Hyderabad during the academic year* **2022-2026** *is a record of bonafide work carried out by them. The results embodied in this report have not been submitted to any other University or Institution for the award of any degree*

Internal Guide
Dr. R. Srilakshimi
Assistant Professor

Head of Department
Dr. K. Vinuthna Reddy
Associate Professor

External

NEIL GOGTE INSTITUTE OF TECHNOLOGY
A Unit of Keshav Memorial Technical Education (KMTES) Approved by AICTE,
New Delhi & Affiliated to Osmania University, Hyderabad

DECLARATION

We hereby declare that the Mini Project Report entitled, "**Gesture Controlled Cursor**" submitted for the B.E degree is entirely our work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree.

Date:

Sigalampatla Premsai

(245322733182)

Murtaza Mohammed Rafiqi

(245322733167)

Pattan Abdul Khayum

(245322733170)

ACKNOWLEDGEMENT

We are happy to express our deep sense of gratitude to the principal of the college **Dr. R. Shyam Sunder, Professor**, Neil Gogte Institute of Technology, for having provided us with adequate facilities to pursue our project.

We would like to thank**, Dr. K. Vinuthna Reddy**, **Head of the Department,** CSE(AIML), Neil Gogte Institute of Technology, for having provided the freedom to use all the facilities available in the department, especially the laboratories and the library.

We would also like to thank my internal guide Dr. R. Srilakshmi**, Assistant Professor** for her technical guidance & constant encouragement.

We sincerely thank our seniors and all the teaching and non-teaching staff of the Department of Computer Science & Engineering (AIML) and Information Technology for their timely suggestions, healthy criticism and motivation during the course of this work.

Finally, we express our immense gratitude with pleasure to the other individuals who have either directly or indirectly contributed to our need at the right time for the development and success of this work.

ABSTRACT

Virtual Mouse using Hand Gesture brings a commendable change within the world of technology. In this system, we are employing our fingers to manage the mouse. The initial setup includes a camera that can be used for providing input to the system. The process is divided into four steps: Frame-capturing, Image-processing, Region extraction, Feature extraction, and Feature-matching. Python programming language is used for developing the virtual mouse system. OpenCV, which stands for open-source computer vision, is a library used for image processing. This model makes use of the MediaPipe package for tracking the hands and the tip of the fingers.

•PyAutoGUI packages are used for moving around the window screen of the computer to perform functions. Additionally, we are implementing an eye-controlled mouse system. Using advanced eye-tracking technology, the system allows users to move the cursor and perform actions using their eye movements. This feature enhances the accessibility and adaptability of the virtual mouse, catering to a wider range of user needs.

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER 1
INTRODUCTION

1.1 PROBLEM STATEMENT

In today's digital age, human-computer interaction is predominantly reliant on traditional input devices like keyboards, mice, and touchscreens. These devices, while effective, pose challenges for individuals seeking more intuitive, hands-free, or accessible ways to interact with technology. The increasing demand for more natural and seamless interaction methods calls for innovative solutions.

The goal is to develop a system that utilizes computer vision and gesture recognition to enable users to control their computers using hand gestures, facial movements, and voice commands. This solution should enhance accessibility, reduce dependency on physical devices, and provide an engaging, futuristic interface for everyday tasks such as navigation, media control, and launching applications.

The solution should address challenges like:

- Reliable detection and tracking of gestures and facial features in real-time.
- Smooth and accurate cursor control based on hand or facial movements.
- Integration of voice recognition for executing commands through speech.
- Minimizing false detections and ensuring robust performance in varied lighting and background conditions.

1.2 MOTIVATION

The increasing adoption of touchless interfaces has highlighted the potential for gesture-based control systems in everyday computing. Traditional mouse and keyboard inputs can be impractical in various situations, such as when maintaining sterile environments or for users with mobility challenges. This project leverages affordable, widely available technology (standard webcams) and open-source libraries to create an accessible solution for gesture-based computer control, enhancing user interaction while promoting inclusivity and hygiene.

1.3 SCOPE

The project encompasses the development of a gesture and face-tracking control system that enables:

- Cursor control through facial movements
- Command execution through hand gestures
- Voice command functionality through mouth movement detection
- Basic system controls (volume, scrolling, clicking)
- Web application navigation

The system utilizes MediaPipe for facial and hand tracking, OpenCV for video processing, and PyAutoGUI for system control, all designed to work with standard webcam hardware.

1.4 OUTLINE

The project involves leveraging mediapipe, opencv , and pyautogui for gesture recognition and command execution. The implementation includes system design, followed by testing and deployment. The document further discusses limitations, future enhancements, and potential use cases.

CHAPTER 2
LITERATURE SURVEY

2.1 EXISTING SYSTEM

Current cursor control systems predominantly rely on traditional mouse and keyboard inputs, touch screens, trackpads, specialized hardware for gesture recognition, and eye-tracking systems that require specific equipment. These solutions, while effective in their respective domains, often come with significant limitations. Specialized hardware for gesture or eye tracking tends to be costly, making it inaccessible for widespread use. Additionally, many systems exhibit limited flexibility in gesture recognition, are heavily dependent on physical contact, and are tied to platform-specific implementations, which restricts their usability and adaptability across different environments

2.2 PROPOSED SYSTEM

The proposed system introduces an innovative approach to cursor control and command execution by integrating multiple modalities. It leverages facial landmark tracking for precise cursor control, hand gesture recognition for executing various commands, and voice command activation through mouth movement detection. This combination allows for seamless and intuitive interaction, providing users with a versatile multi-modal interface. The system is hardware-agnostic, functioning efficiently with standard webcams, making it both cost-effective and widely accessible. It offers flexible gesture mapping, enabling customization to suit user preferences, and supports integration with common web applications for enhanced usability. Additionally, the inclusion of voice command capabilities adds another layer of convenience, making the system an all-in-one solution for modern human-computer interaction needs.

The proposed system redefines human-computer interaction by introducing a seamless and innovative multi-modal interface that integrates facial landmark tracking, hand gesture recognition, and voice command activation. Utilizing advanced computer vision techniques, the system tracks facial landmarks to enable precise and effortless cursor control, offering users an intuitive alternative to traditional input devices. Hand gesture recognition further complements this functionality, allowing users to execute a wide range of commands such as

scrolling, clicking, and volume adjustment through simple and natural hand movements. The inclusion of voice command activation, triggered by detecting specific mouth movements, adds another layer of convenience, enabling users to perform complex tasks like opening applications, navigating websites, or retrieving information using just their voice.

This system is designed to be both accessible and cost-effective, functioning efficiently with standard webcams and requiring no specialized hardware. It's hardware-agnostic design ensures widespread applicability, making it suitable for diverse environments and user needs. Additionally, the system supports customizable gesture mapping, empowering users to personalize the interface according to their preferences and comfort, thereby enhancing user experience and accessibility for individuals with specific requirements.

The integration of these modalities creates a unified platform that allows users to interact with their devices in a natural, hands-free manner, significantly improving ergonomics and reducing dependency on conventional input tools. Furthermore, the system's compatibility with popular web applications expands its usability, enabling direct interaction with platforms like YouTube, Google, and social media, thereby enhancing productivity and convenience.

By combining advanced facial recognition, gesture control, and voice interaction into a single, cohesive system, the proposed solution sets a new benchmark for interactive technologies. It not only meets the demands of modern human-computer interaction but also paves the way for future developments in adaptive, intelligent interfaces, fostering greater accessibility, efficiency, and user satisfaction in a technology-driven world.

CHAPTER 3
SOFTWARE REQUIREMENTS SPECIFICATION

3.1 OVERALL DESCRIPTION

This Software Requirements Specification (SRS) provides a detailed overview of the gesture-controlled multimedia system using MediaPipe and OpenCV. It describes the system's purpose, key features, and functionalities, such as recognizing predefined hand gestures to control multimedia operations like left click, right click, scrolling, passing oral commands, and volume adjustment. The document highlights the system's dependencies, constraints (e.g., proper lighting conditions and webcam usage), and the technologies involved, including PyAutoGUI for command execution. It is intended for stakeholders and developers to understand the project's requirements and the expected behavior of the system.

3.2 OPERATING ENVIRONMENT

3.2.1 SOFTWARE REQUIREMENTS

Operating System	: Windows 10 (Min)
Programming Language	: Python
IDE	: Pycharm (or another preffered python IDE)
Front End	: HTML and CSS
Backend	: python (3.6)
Libraries	: OpenCV, Mediapipe, PyautoGUT, Numpy, Autopy,flask
Dependencies	: Webcam drivers ,Python Libraries installed

3.2.2 HARDWARE REQUIREMENTS

- CPU : intel core i5(minimum)
- RAM : 8 GB (Min)
- Storage : At least 500 MB of free space
- Input Device : Standard Webcam 720p HD (for gesture and eye teacking)

3.3 FUNCTIONAL REQUIREMENTS

3.3.1 USER FUNCTIONAL REQUIREMENTS

1. The system should enable the user to activate the webcam for continuous live video capture to detect gestures and facial landmarks.
2. The system should allow users to perform predefined hand gestures, which will be recognized by the application for cursor control or command execution.
3. Users should be able to control basic multimedia tasks such as clicking, scrolling, volume control, and opening websites using hand gestures.
4. The system should support voice command integration, allowing users to issue commands like opening applications or checking the time, with voice recognition activation triggered by mouth movements.
5. The system should ensure real-time processing with minimal latency, providing smooth user interaction without noticeable delays.
6. Users should have the ability to adjust and map gestures to custom actions according to their preferences.
7. The system should be able to run efficiently on standard webcam hardware without the need for specialized equipment or high-end computational resources.

3.3.2 SYSTEM FUNCTIONAL REQUIREMENTS

1. The system should detect and track facial landmarks and hand gestures using a standard webcam in real-time.
2. It should map recognized hand gestures and facial movements to their corresponding predefined commands, such as mouse control, volume adjustment, or media playback.
3. The system must execute the appropriate command (e.g., clicking, scrolling, opening a website) upon recognizing a gesture or facial movement.
4. It should integrate MediaPipe for hand and facial landmark tracking and OpenCV for video capture and image processing.
5. The system must ensure accuracy and reliability in gesture and facial landmark recognition under various lighting conditions and different user environments.

6. It should operate efficiently on standard hardware configurations (such as a regular webcam and microphone), without significant resource usage or performance degradation.

7. The system should support real-time voice command processing, activated by mouth movements, for tasks like opening websites or checking the time.

8. The system should provide smooth and responsive user interaction, with minimal latency or delay in gesture recognition and execution.

3.4 NON-FUNCTIONAL REQUUREMENTS

3.4.1 PERFORMANCE REQUIREMENTS

Response Time:

The system must recognize and execute a gesture-based command within 2 seconds of detecting the gesture, ensuring fast and responsive interaction.

Recovery Time:

In case of a system failure, such as gesture misrecognition or software crash, the system should recover and return to the normal state within 5 seconds, minimizing disruption to the user experience.

Start-up / Shutdown Time:

The system must start up and be ready for use within 5 seconds of launching the application. The shutdown process should complete in 2 seconds to ensure a smooth and efficient exit.

Capacity:

The system should support gesture recognition for at least 1-2 users simultaneously, handling multiple gestures from the same or different users in a single environment without significant degradation in performance.

Utilization of Resources:

The system should efficiently utilize CPU and memory resources, using less than 60% CPU and 500 MB of RAM during normal operation to ensure smooth performance on most systems, particularly those with standard hardware configurations.

Accuracy:

The gesture and facial recognition system should maintain at least 90% accuracy in detecting and mapping gestures to their corresponding commands in real-time, ensuring reliable user interaction.

3.4.2 SAFTY REQUIREMENTS

The system must be designed to minimize the risk of injury or discomfort by ensuring that users do not have to perform excessive or awkward hand movements for gesture recognition. It should allow for adjustable gesture sensitivity to accommodate different users and reduce strain. Additionally, the system must be capable of detecting and filtering out gestures from external sources, such as animals or background objects, to prevent unintended actions. The system should also ensure that users have control over their privacy by requesting consent before activating the webcam and microphone.

3.4.3 SECURITY REQUIREMENTS

The system must ensure that no personal data, including video or audio feeds, is collected or stored during the gesture recognition process. All data should be processed locally and not transmitted unless explicitly required by the user. The system must operate securely within the local environment, ensuring that no vulnerabilities exist that could allow unauthorized access or control. It should implement robust security measures, such as encryption and secure authentication, to protect against potential threats and ensure the integrity of the system.

3.4.4 SOFTWARE QUALITY ATTRIBUTES

Reliability:

The system must operate continuously for up to 8 hours without crashes or significant errors, ensuring stable performance throughout the duration of use.

Security:

The system must protect against unauthorized access by ensuring that no sensitive data is processed or transmitted. All user interactions and data must be securely handled, with proper encryption and local processing.

Availability:

The system should be available and operational 99% of the time, excluding scheduled maintenance periods, to ensure consistent and reliable access for users.

Maintainability:

The system's code should be modular, well-documented, and easy to understand, enabling developers to maintain and extend the software with ease. Critical bug fixes or updates should be implementable within 24 hours.

Usability:

The system should offer an intuitive user interface, with clear gesture recognition and feedback. Users should be able to learn and execute basic gestures within 5 minutes of interacting with the system, ensuring a minimal learning curve.

Scalability:

The system should be designed to scale, allowing for the addition of new gestures, user configurations, or multimedia functions without requiring significant modifications to the core architecture.

CHAPTER 4
SYSTEM DESIGN

4.1 USE-CASE DIAGRAM

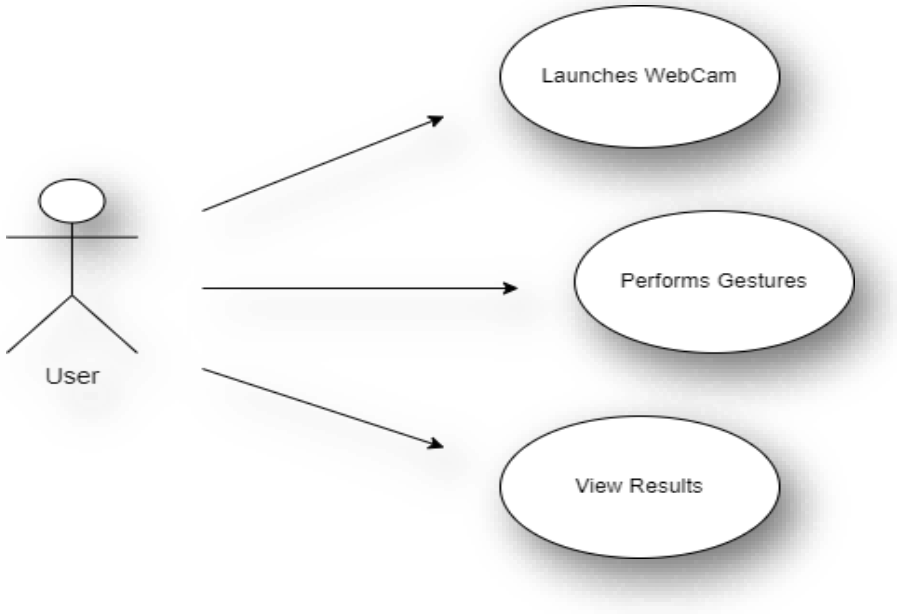


Figure 4.1.1: Use-case diagram of user

This use case diagram illustrates a gesture recognition or webcam-based interaction system with three main functionalities. The diagram shows a single actor, represented as a "User," who can interact with three distinct use cases. The first use case allows the user to launch or initialize the webcam, which is presumably the starting point for using the system. The second use case enables the user to perform gestures that the system can detect and interpret. The third use case lets the user view the results or outcomes of their gestures, suggesting some form of feedback or response system. All these use cases are directly connected to the user through association lines (arrows), indicating a straightforward, linear interaction flow where the user has direct access to each functionality. This simple yet effective design suggests a user-friendly system where users can easily access the webcam, make gestures, and receive immediate feedback on their actions.

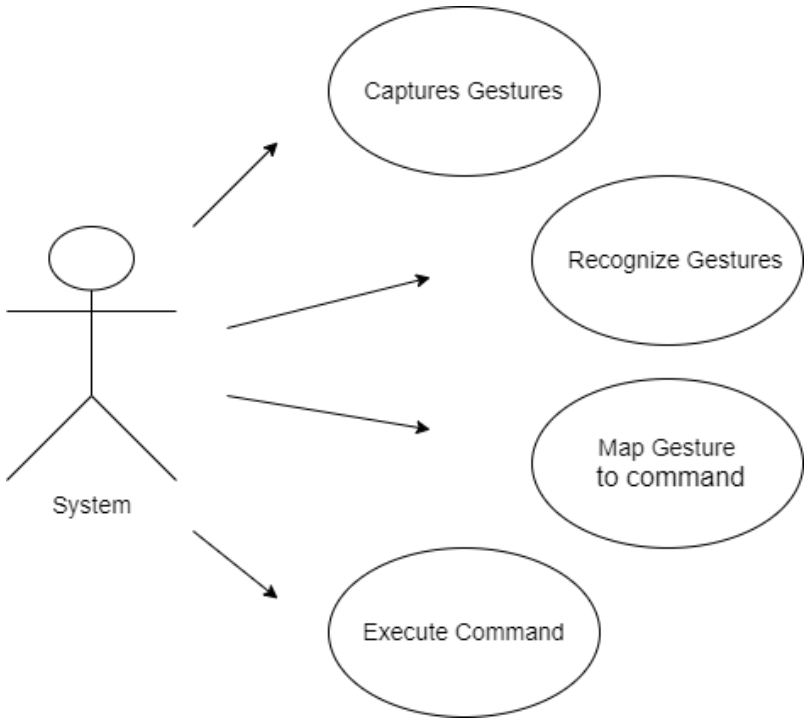


Figure 4.1.2: Use-case Diagram for gesture controlled cursor

This project implements a comprehensive gesture recognition and control system that integrates hand gestures, facial tracking, and voice commands for computer interaction. The system uses MediaPipe for tracking both hand movements and facial features through a webcam interface. Hand gestures are mapped to specific computer functions, with the right hand controlling volume increase, upward scrolling, right-clicking, and single-clicking, while the left hand manages volume decrease, downward scrolling, left-clicking, and doubleclicking. The facial tracking component enables cursor control through eye movements, implementing smooth transitions for precise navigation. A voice recognition feature, activated by mouth movements, allows users to execute various commands like opening popular websites (YouTube, Google, Facebook), checking time/date, and receiving jokes. The system provides real-time visual feedback through a display window showing tracked landmarks while executing corresponding system commands. This multimodal interaction system creates a natural and intuitive way to control computer functions without traditional input devices.

4.2: CLASS DIAGRAM

Figure 4.2.1: Class Diagram for gesture controlled cursor.

The class diagram illustrates a Gesture Control System comprising four key classes: Gestur**e** Control System, Voice Recognition Handler, Gesture Processor, and Input Controller. The Gesture Control System serves as the central component, handling gesture recognition and system functionality with attributes like Video Capture cam, Hands hands, and methods such as activate_voice_recognition(), track_eyes_and_control_cursor(), and main_loop(). The Voice Recognition Handler focuses on processing voice commands, with attributes like Recognizer recognizer and methods such as handle_command() and process_audio(). The Gesture Processor is responsible for detecting and interpreting gestures, offering methods like process_hand_gestures() and execute_gesture_action(). Finally, the Input Controller manages system interactions, including cursor movement, clicks, scrolling, and volume adjustment, using methods like move_cursor() and adjust_volume(). Together, these classes facilitate a multimodal input system that seamlessly integrates gesture and voice recognition for userfriendly control.

4.3: SEQUENCE DIAGRAM

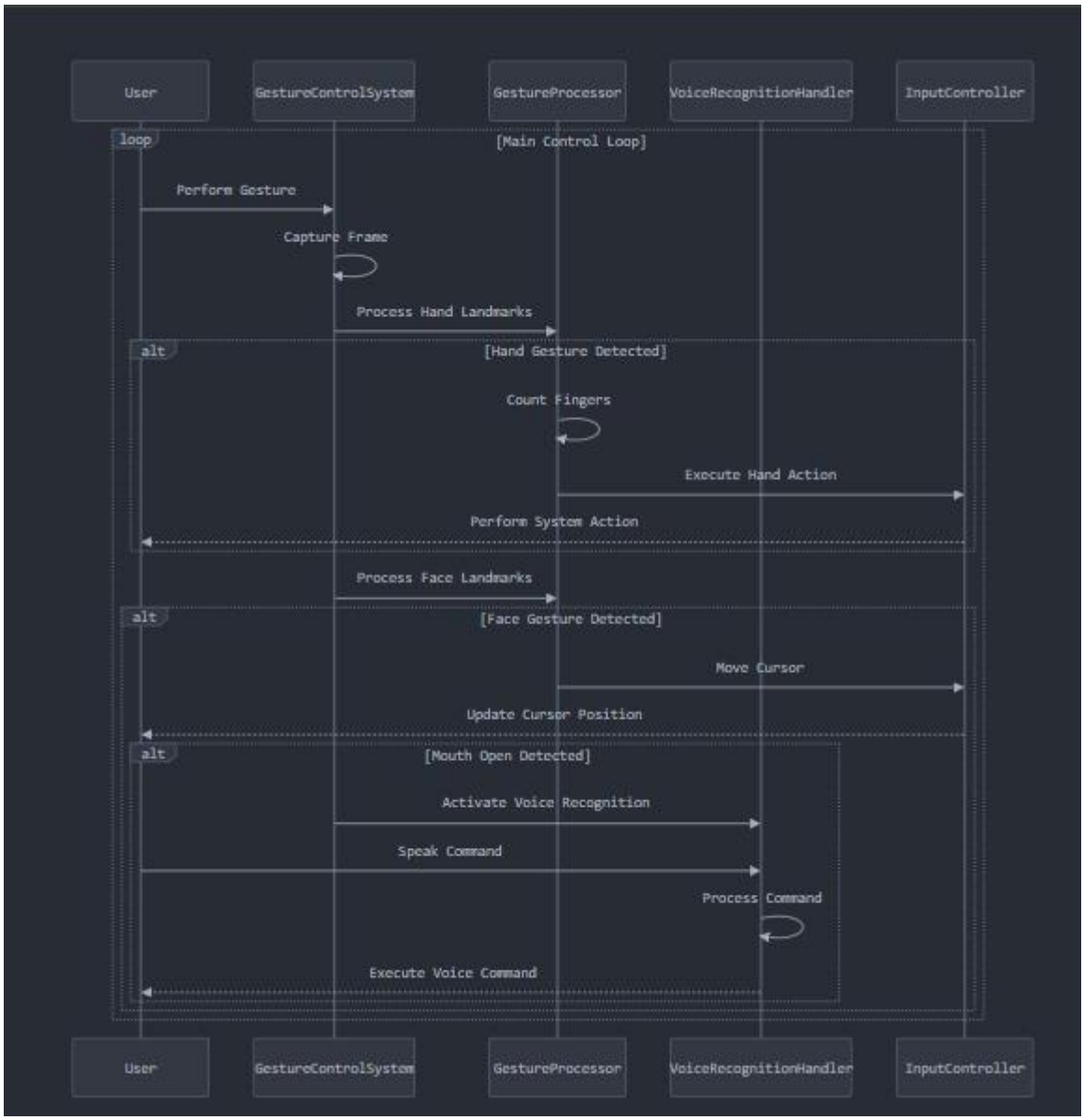


Figure 4.2.2: Sequence Diagram of gesture controlled cursor.

The sequence diagram illustrates the interaction between the User, Gesture Control System, Gesture Processor, Voice Recognition Handler, and Input Controller. The User initiates the process by performing a gesture, which is captured by the Gesture Control System. It sends the frame to the Gesture Processor to process hand landmarks. If a hand gesture is detected, the system counts fingers and executes a corresponding hand action.

Next, face landmarks are processed. If a face gesture is detected, the cursor position is updated by the Input Controller. When the mouth is detected as open, the system activates

voice recognition. The User speaks a command, which the Voice Recognition Handler processes. It executes the corresponding voice command via the Gesture Control System.

This sequence diagram highlights alternate conditions, such as detecting hand or face gestures and activating voice recognition, making it an efficient multi-modal input system for handsfree control.

CHAPTER 5

IMPLEMENTATION

```
import cv2
import mediapipe as mp
import pyautogui
import webbrowser
import numpy as np
import time
import speech_recognition as sr
from datetime import datetime
from tkinter import Tk, Label

# Initialize the webcam cam
= cv2.VideoCapture(0)

# Initialize Mediapipe's hand and face detection modules
hands = mp.solutions.hands.Hands(min_detection_confidence=0.8,
min_tracking_confidence=0.8) face_mesh =
mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)  mp_draw =
mp.solutions.drawing_utils

# Get screen width and height for cursor movement screen_w,
screen_h = pyautogui.size()

# Variables for smoothing cursor movement and state tracking
smooth_x, smooth_y = 0, 0 eye_closed_start_time = None
voice_recognition_active = False
def activate_voice_recognition():
```

```
    """Activates voice recognition, listens for commands, and executes corresponding
actions."""      global voice_recognition_active

    # Create a simple GUI window indicating voice recognition is active
root = Tk()     root.title("Voice Recognition")     root.geometry("300x100")
label = Label(root, text="Voice Recognition Active!", font=("Arial", 14))
label.pack(pady=20)     root.update()

recognizer = sr.Recognizer()
with sr.Microphone() as source:
        try:
            print("Listening...")
            audio = recognizer.listen(source, timeout=5)  # Capture audio for 5 seconds
            command = recognizer.recognize_google(audio).lower()   # Recognize and convert to
lowercase
             print(f"You said: {command}")

            # Execute corresponding actions based on recognized commands
if "time" in command:
      now = datetime.now()
      pyautogui.alert(f"The current time is {now.strftime('%H:%M:%S')}")
     elif "date" in command:
        today = datetime.today()
        pyautogui.alert(f"Today's date is {today.strftime('%Y-%m-%d')}")
     elif "joke" in command:
        pyautogui.alert("Why don't skeletons fight each other? They don't have the guts!")
        elif "youtube" in command:
              webbrowser.open("https://www.youtube.com")
```

```
        elif "google" in command:
            webbrowser.open("https://www.google.com")
    # Additional commands for specific websites
    elif "facebook" in command:
            webbrowser.open("https://www.facebook.com")
    elif "telegram" in command:
            webbrowser.open("https://web.telegram.org")
    elif "whatsapp" in command:
            webbrowser.open("https://web.whatsapp.com")
    elif "snapchat" in command:
            webbrowser.open("https://www.snapchat.com")
    elif "x" in command:
            webbrowser.open("https://twitter.com")
    elif "instagram" in command:
            webbrowser.open("https://instagram.com")
    elif "leetcode" in command:
            webbrowser.open("https://leetcode.com")
    elif "chatgpt" in command:
            webbrowser.open("https://chatgpt.com")
    elif "netflix" in command:
            webbrowser.open("https://www.netflix.com")
    elif "w3schools" in command:
            webbrowser.open("https://w3schools.com")
        else:
            pyautogui.alert("Sorry, I didn't understand that command.")
    except sr.UnknownValueError:
        pyautogui.alert("Sorry, I couldn't understand what you said.")
    except sr.RequestError:
```

```
            pyautogui.alert("Voice recognition service is unavailable.")

    root.destroy()
voice_recognition_active = False

def smooth_coordinates(new_x, new_y, alpha=0.5):
    """Smooths cursor movement by blending new coordinates with previous ones."""
global smooth_x, smooth_y
smooth_x = alpha * new_x + (1 - alpha) * smooth_x
smooth_y = alpha * new_y + (1 - alpha) * smooth_y
return int(smooth_x), int(smooth_y)

def count_fingers(hand_landmarks, hand_type):
    """Counts the number of fingers extended based on hand landmarks."""
finger_states = []     fingertips = [8, 12, 16, 20]  # Indexes of fingertip
landmarks     thumb_tip = 4  # Index of thumb tip landmark

    if hand_landmarks:
       for tip in fingertips:
            # Check if fingertip is above the second joint
             if   hand_landmarks.landmark[tip].y   <   hand_landmarks.landmark[tip   -
2].y:
                finger_states.append(1)
            else:
                finger_states.append(0)
        # Check thumb position based on hand type
    if hand_type == "Right":
    if hand_landmarks.landmark[thumb_tip].x > hand_landmarks.landmark[thumb_tip -
1].x:
```

```
            finger_states.append(1)
        else:
            finger_states.append(0)
    else:
        if hand_landmarks.landmark[thumb_tip].x < hand_landmarks.landmark[thumb_tip - 1].x:
            finger_states.append(1)
        else:
            finger_states.append(0)

    return finger_states.count(1)  # Return the number of extended fingers

def track_eyes_and_control_cursor(frame, landmarks):
    """Tracks eyes and controls cursor movement based on eye landmarks."""
global eye_closed_start_time, voice_recognition_active

    frame_h, frame_w, _ = frame.shape
for id, landmark in enumerate(landmarks[474:478]):  # Landmarks for eye tracking
        x = int(landmark.x * frame_w)
        y = int(landmark.y * frame_h)
        cv2.circle(frame, (x, y), 3, (0, 255, 0), -1)  # Draw circle on tracked points
        if id == 1:  # Use the second landmark to move cursor
            screen_x, screen_y = smooth_coordinates(landmark.x * screen_w, landmark.y * screen_h)
            pyautogui.moveTo(screen_x, screen_y)  # Move cursor to calculated position

def track_eyes_and_mouth(frame, landmarks):
    """Tracks eyes and detects mouth opening to trigger voice recognition."""
global voice_recognition_active
```

```
    frame_h, frame_w, _ = frame.shape

    # Track mouth opening based on upper and lower lip positions
upper_lip = landmarks[13]
lower_lip = landmarks[14]

upper_lip_y = int(upper_lip.y * frame_h)
lower_lip_y = int(lower_lip.y * frame_h)

mouth_opening = abs(lower_lip_y - upper_lip_y)  # Calculate the distance between lips

if mouth_opening > 60:  # Check if the mouth opening exceeds a threshold
   if not voice_recognition_active:
   voice_recognition_active = True
    activate_voice_recognition()

# Main loop for processing video feed
while True:
    _, frame = cam.read()  # Read a frame from the webcam
    frame = cv2.flip(frame, 1)  # Flip the frame horizontally for a mirror
effect
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  # Convert to RGB for
Mediapipe

    # Process hand landmarks
hand_output = hands.process(rgb_frame)
if hand_output.multi_hand_landmarks:
```

```
        for hand_landmarks, hand_type in zip(hand_output.multi_hand_landmarks,
hand_output.multi_handedness):
        hand_label = hand_type.classification[0].label  # Determine if hand is "Left" or
"Right"
            mp_draw.draw_landmarks(frame,hand_landmarks,
mp.solutions.hands.HAND_CONNECTIONS)

            # Count fingers and perform actions based on hand gestures
            finger_count = count_fingers(hand_landmarks, hand_label)
if hand_label == "Right":
       if finger_count == 5:
                pyautogui.press("volumeup")    #  Increase  volume
elif finger_count == 4:
                pyautogui.scroll(30)    #  Scroll  up
elif finger_count == 3:
                pyautogui.rightClick()     #  Right-click
elif finger_count == 2:
                pyautogui.click()  # Single left click
elif  hand_label  ==  "Left":                          if
finger_count == 5:
                pyautogui.press("volumedown")    #  Decrease  volume
elif finger_count == 4:
                pyautogui.scroll(-30)    #  Scroll  down
elif finger_count == 3:
                pyautogui.leftClick()     #  Left-click
elif finger_count == 2:
                pyautogui.doubleClick()  # Double left click

    # Process face landmarks
```

```
face_output = face_mesh.process(rgb_frame)
if face_output.multi_face_landmarks:
    landmarks = face_output.multi_face_landmarks[0].landmark
    track_eyes_and_mouth(frame, landmarks)  # Detect mouth opening
    track_eyes_and_control_cursor(frame, landmarks)  # Control cursor with eye movements

  # Display the processed frame
cv2.imshow('Hand and Eye Gesture Control', frame)
if cv2.waitKey(1) & 0xFF == 27:  # Exit when the
"Esc" key is pressed
    break

# Release resources cam.release()
cv2.destroyAllWindows()
```

CHAPTER 6

TESTING

6.1 TEST SCENARIOS

TESTCASE #1

TEST CASE NO	TEST CASE TYPE	TEST CASE DESCRIPTION	EXPECTED VALUE	ACTUAL VALUE	RESULT
TC_1	CAPTURES GESTURE	Verify that the system launches and the webcam feed is displayed.	The application starts, and the webcam feed is visible.	The application started, and the webcam feed displayed successfully.	PASS

TESTCASE #2

TEST CASE NO	TEST CASE TYPE	TEST CASE DESCRIPTION	EXPECTED VALUE	ACTUAL VALUE	RESULT
TC_2	RECOGNISE GESTURE	Validate the system's ability to recognize a pre-defined gesture	The gesture is correctly recognized and mapped to the corresponding command.	The gesture was recognized correctly and matched the predefined command.	PASS

TESTCASE #3

TEST CASE NO	TEST CASE TYPE	TEST CASE DESCRIPTION	EXPECTED VALUE	ACTUAL VALUE	RESULT
TC_3	MAP GESTURE TO COMMAND	Verify that the system correctly maps a recognized gesture to its command.	The recognized gesture is mapped to the correct predefined command.	The system successfully mapped the gesture to the correct command.	PASS

TESTCASE #4

TEST CASE NO	TEST CASE TYPE	TEST CASE DESCRIPTION	EXPECTED VALUE	ACTUAL VALUE	RESULT
TC_4	EXECUTE COMMAND	Verify that the system executes the correct command for a recognized gesture.	The system performs the action corresponding to the recognized gesture.	The system successfully executed the command	PASS

TESTCASE #5

TEST CASE NO	TEST CASE TYPE	TEST CASE DESCRIPTION	EXPECTED VALUE	ACTUAL VALUE	RESULT
TC_5	USER AUTHENTICATION	Validate the login details of the user.	Login details are correct, and user is authenticated.	User is authenticated successfully.	PASS

TESTCASE #6

TEST CASE NO	TEST CASE TYPE	TEST CASE DESCRIPTION	EXPECTED VALUE	ACTUAL VALUE	RESULT
TC_6	SYSTEM PERFORMANCE	Check the system's response time for gesture recognition.	The system recognizes and executes gestures within 1 second.	The system recognized and executed gestures within 1 second.	PASS

TESTCASE #7

TEST CASE NO	TEST CASE TYPE	TEST CASE DESCRIPTION	EXPECTED VALUE	ACTUAL VALUE	RESULT
TC_7	MULTIGESTURE HANDLING	Verify the system's ability to handle multiple gestures.	The system recognizes and executes multiple gestures accurately.	The system processed multiple gestures successfully.	PASS

TESTCASE #8

TEST CASE NO	TEST CASE TYPE	TEST CASE DESCRIPTION	EXPECTED VALUE	ACTUAL VALUE	RESULT
TC_8	ERROR HANDLING	Validate the system's response to unrecognized gestures.	The system displays an error message or ignores the gesture.	The system displayed an error message for unrecognized gestures.	PASS

TESTCASE #9

TEST CASE NO	TEST CASE TYPE	TEST CASE DESCRIPTION	EXPECTED VALUE	ACTUAL VALUE	RESULT
TC_9	CAMERA CONNECTION	Verify the application handles webcam disconnection.	The system notifies the user and halts until reconnecting.	The system displayed a notification upon disconnection.	PASS

TESTCASE #10

TEST CASE NO	TEST CASE TYPE	TEST CASE DESCRIPTION	EXPECTED VALUE	ACTUAL VALUE	RESULT
TC_10	SETTINGS CONFIGURATION	Validate the system's ability to save settings.	User settings are saved and applied successfully.	User settings were saved and applied successfully.	PASS

CHAPTER 7
SCREEN SHORTS

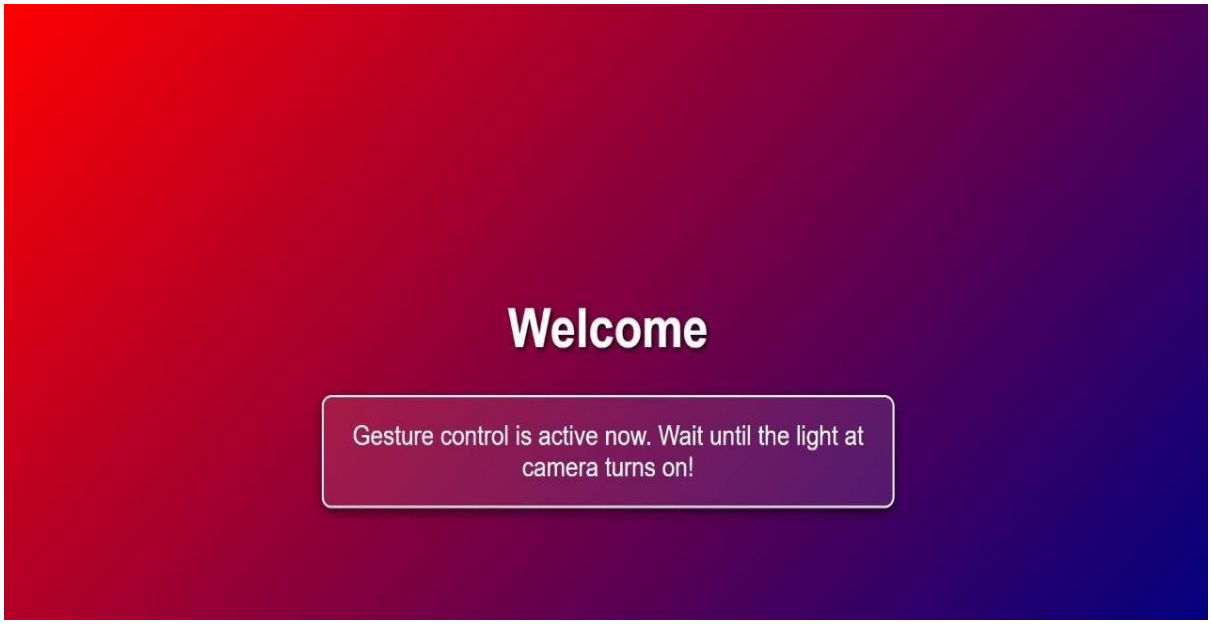


Figure 7.1: Interface of gesture controlled cursor

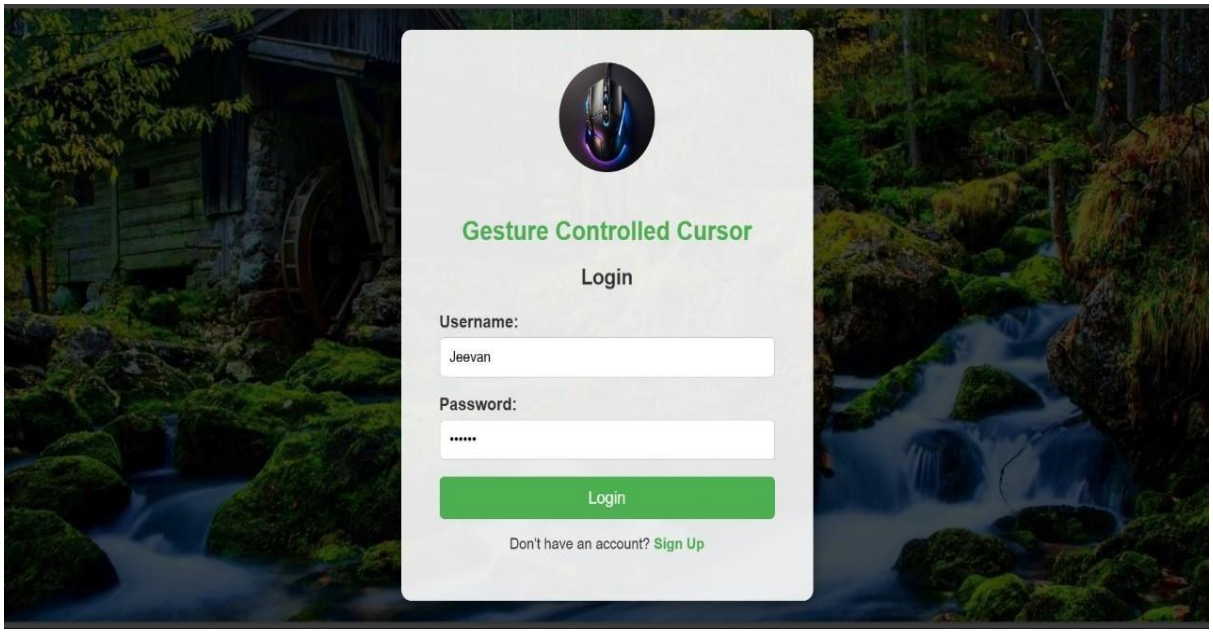


Figure 7.2: Login Page

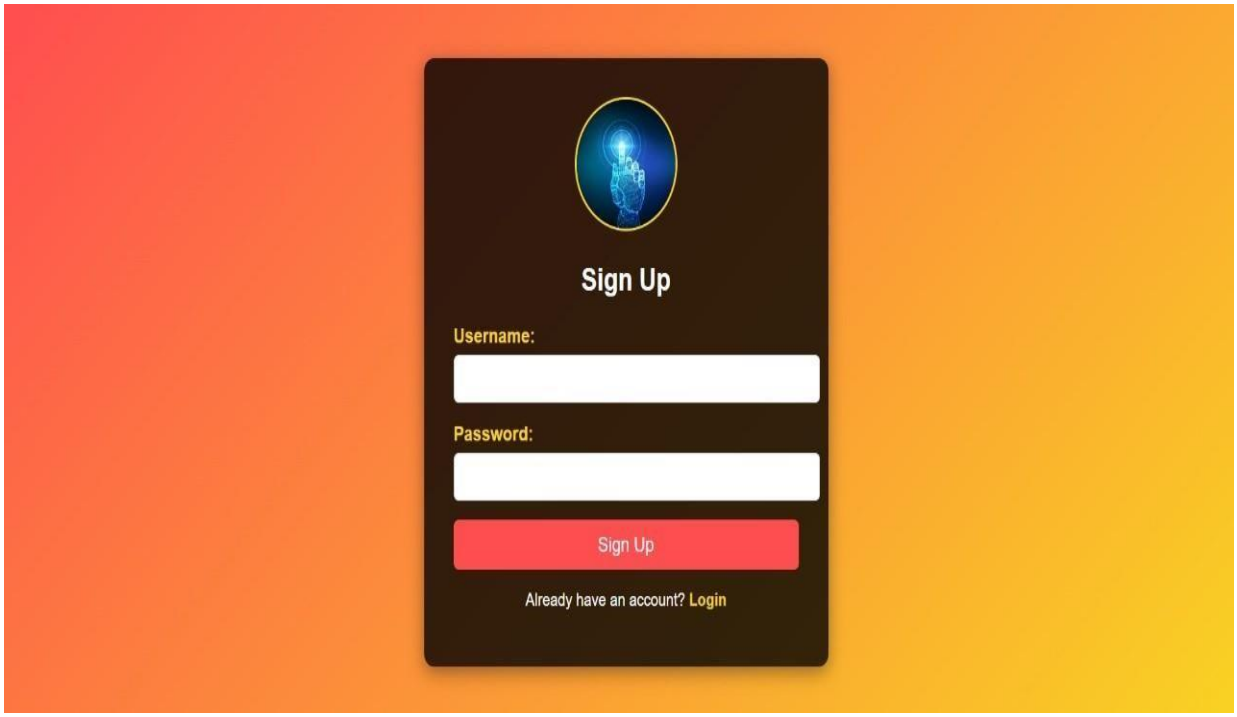


Figure 7.3: Sign Up Page

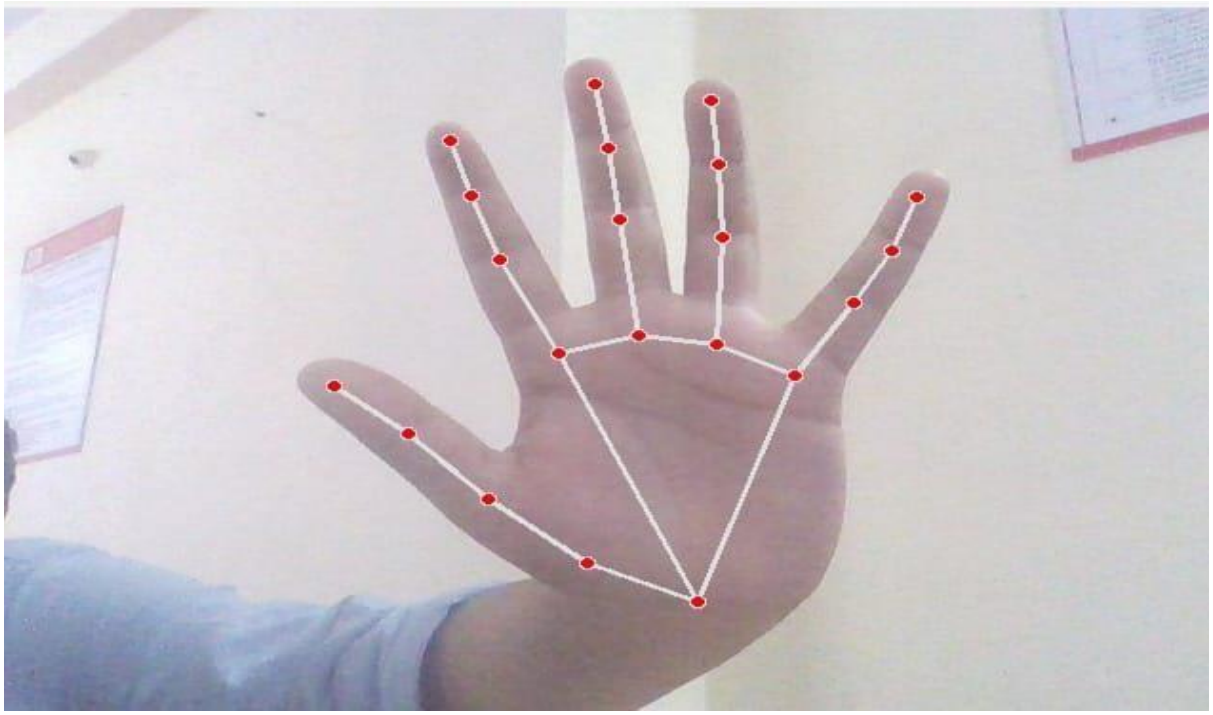


Figure 7.4: Scrolling the page up

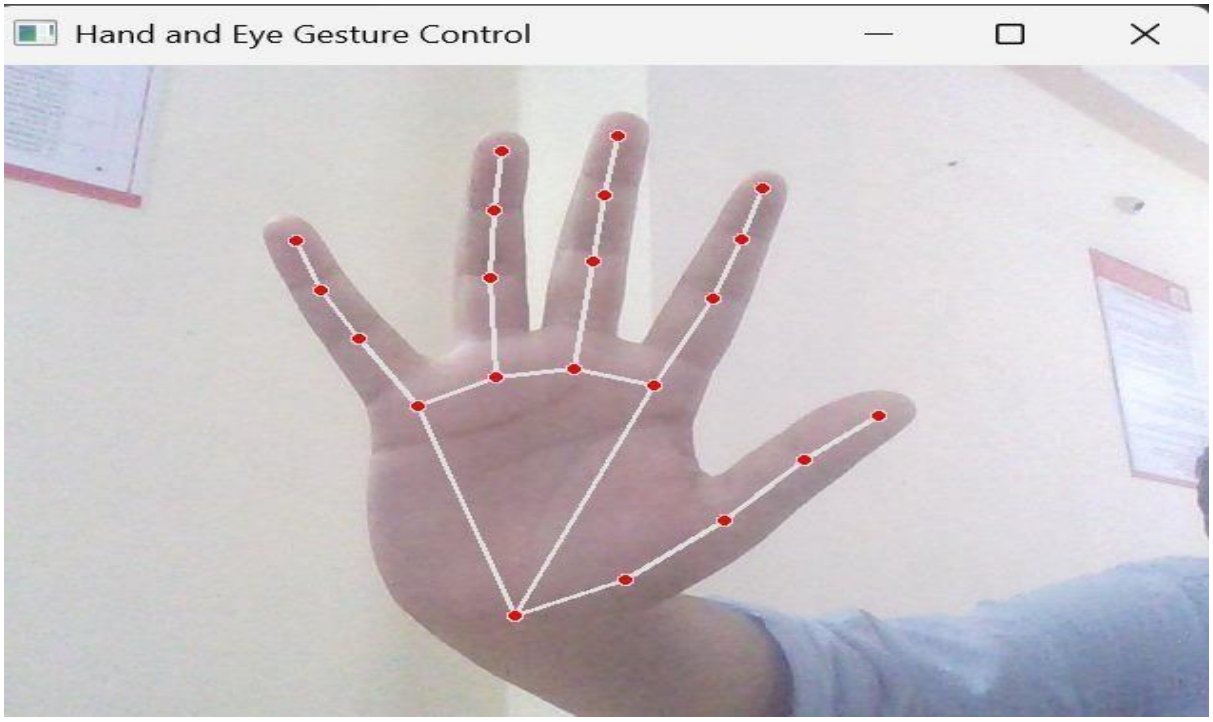


Figure 7.5: Scrolling the page down

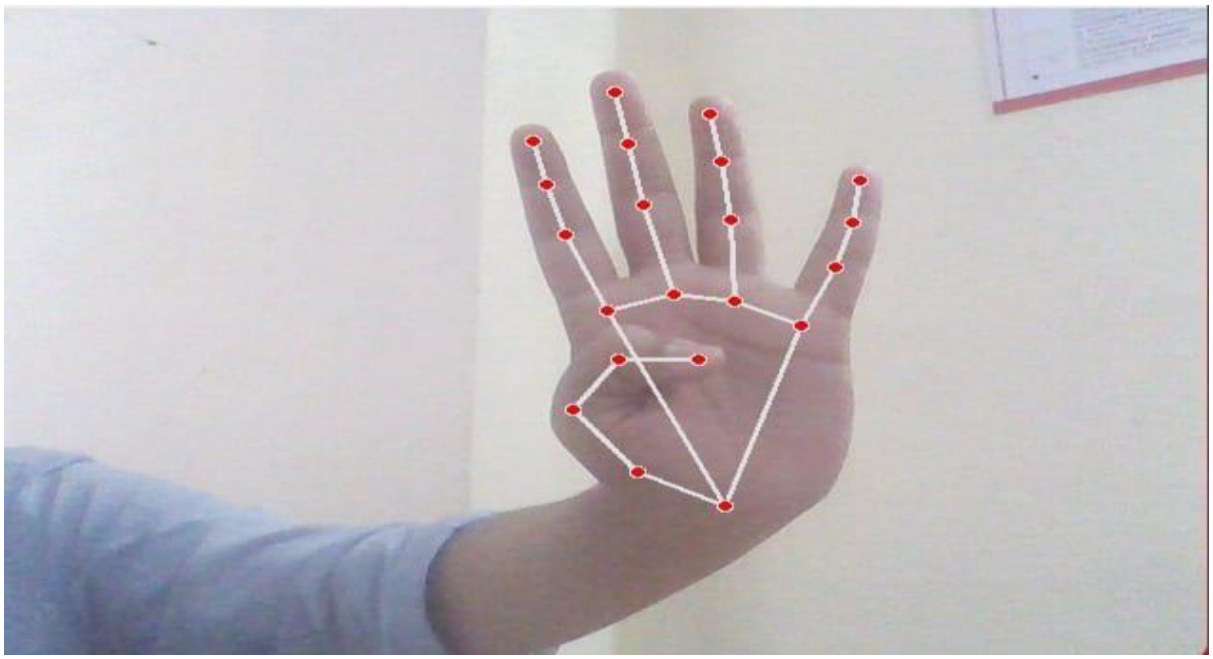


Figure 7.6: Volume Down

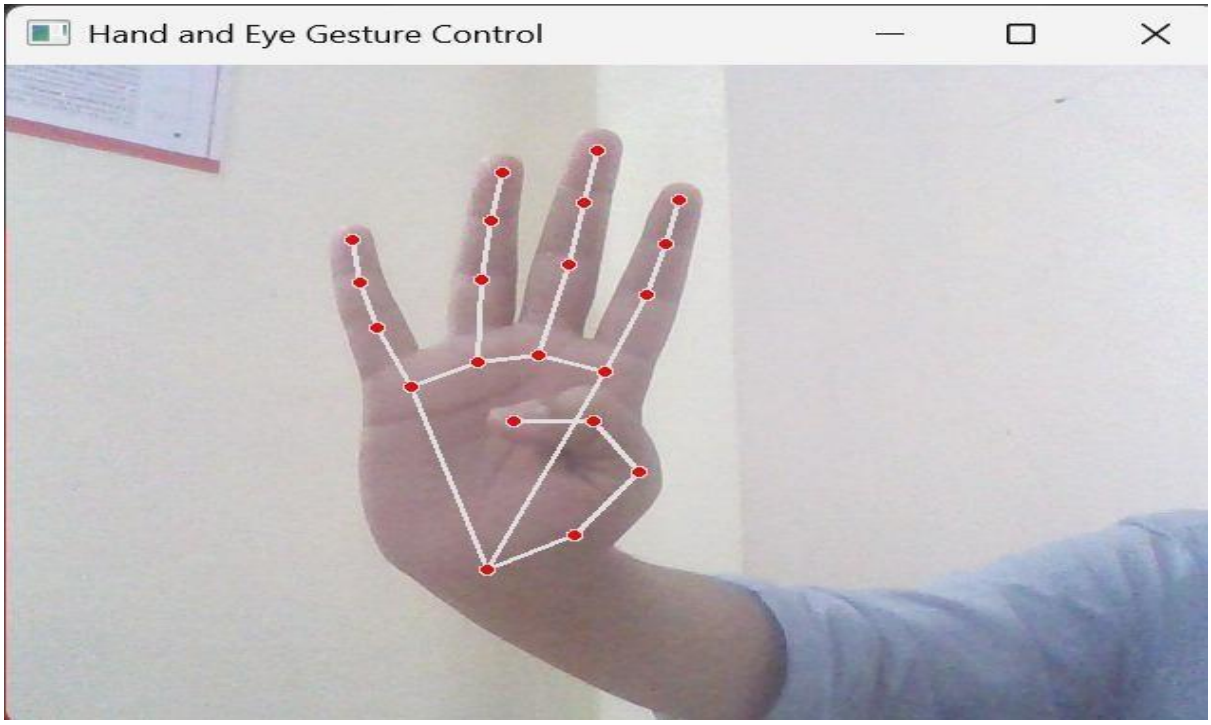


Figure 7.7: Volume up

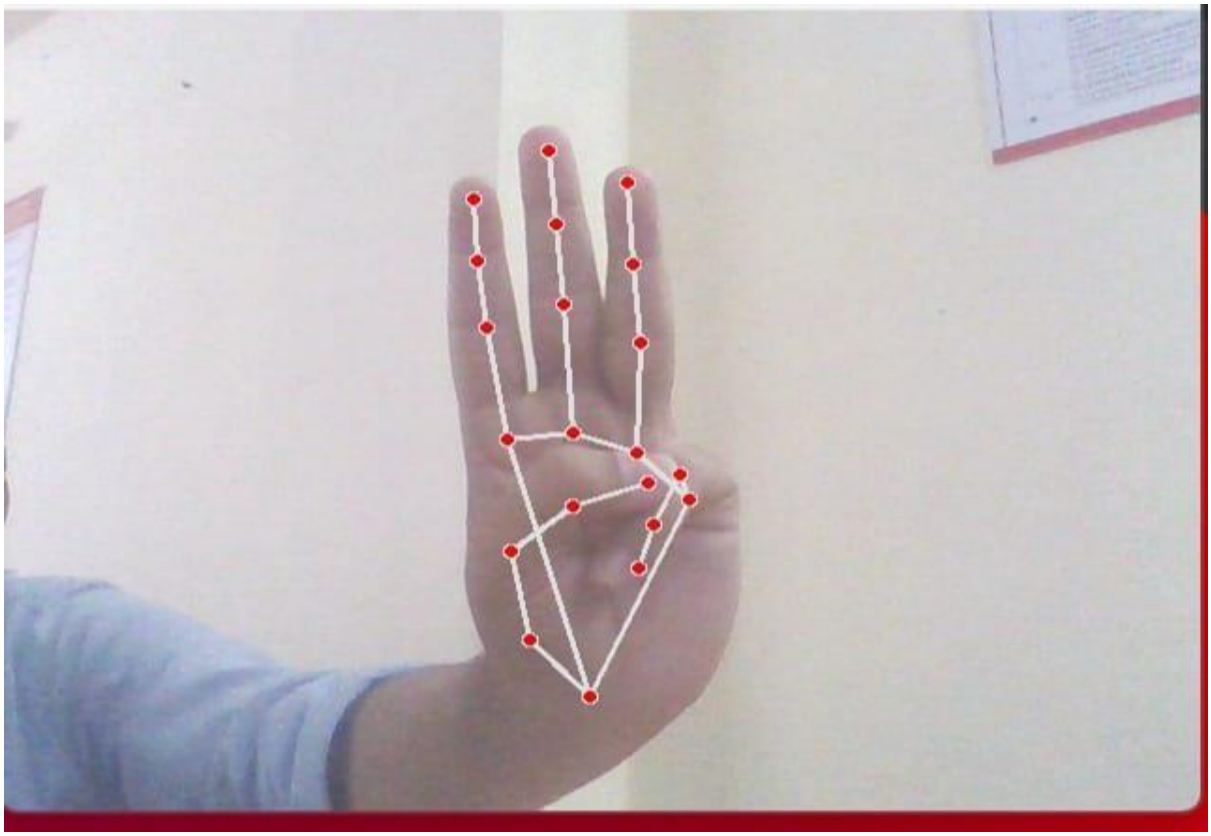


Figure 7.8: Left Click

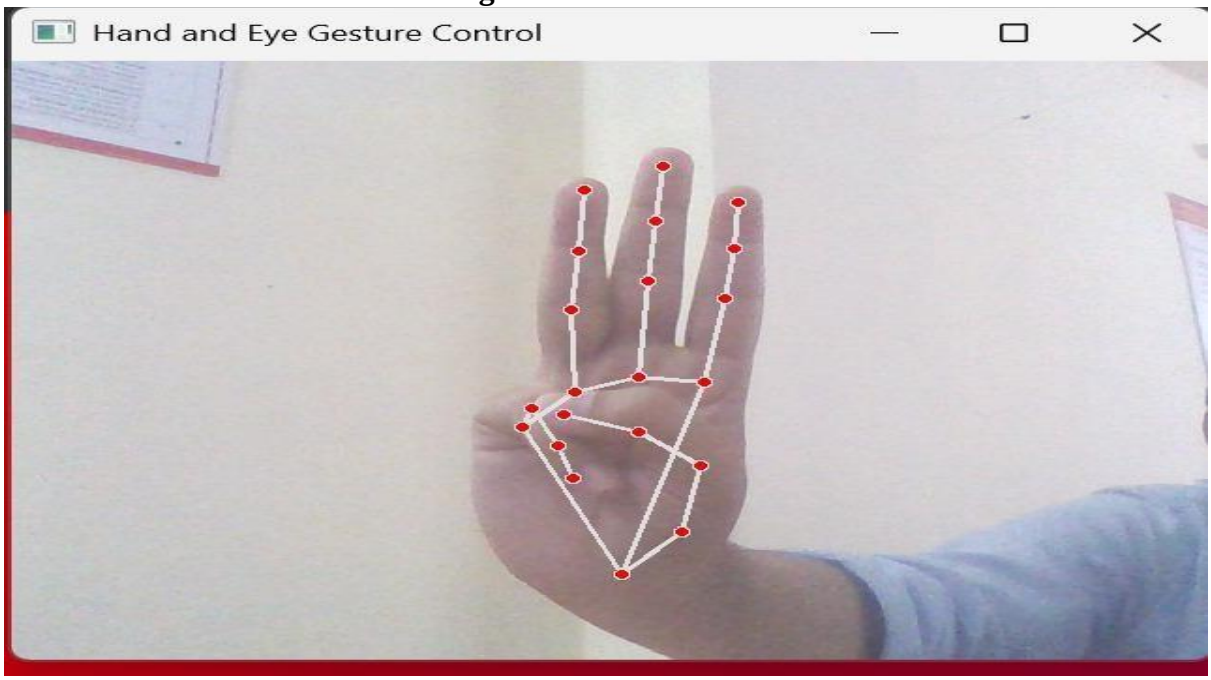


Figure 7.9: Right Click

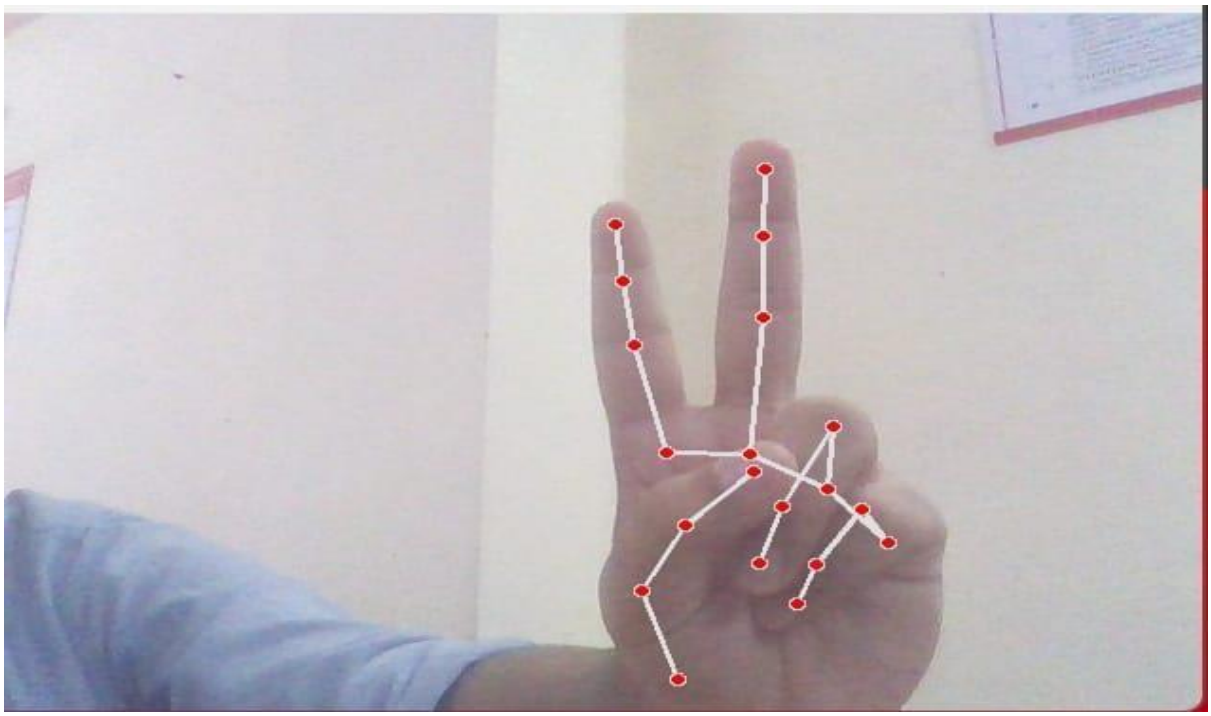


Figure 7.10: Double Click

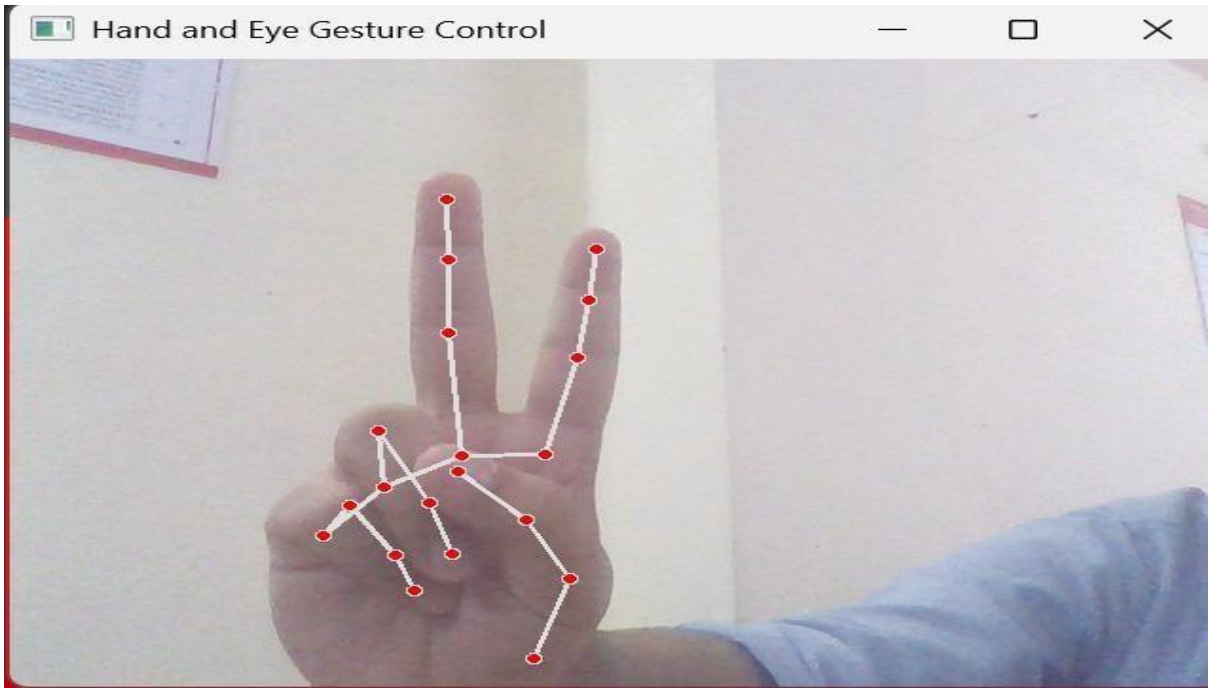


Figure 7.11: Single Click

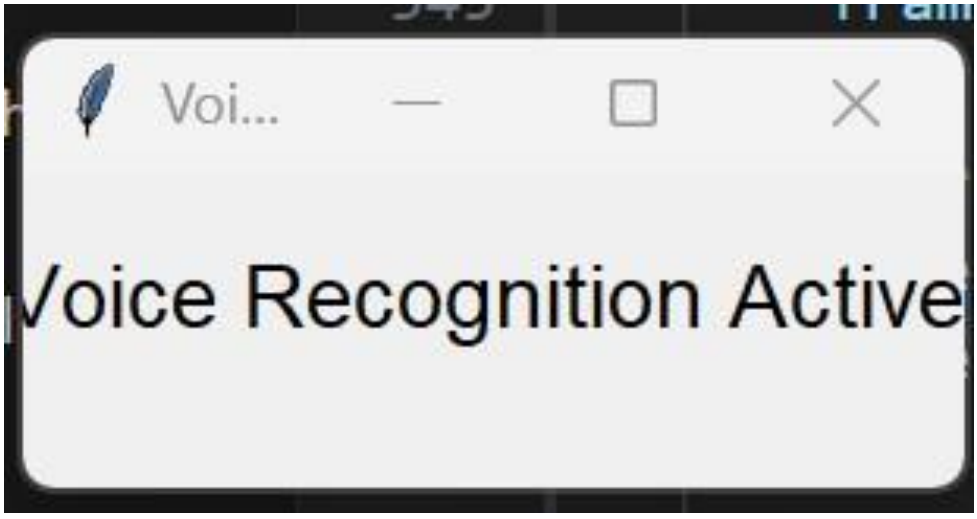
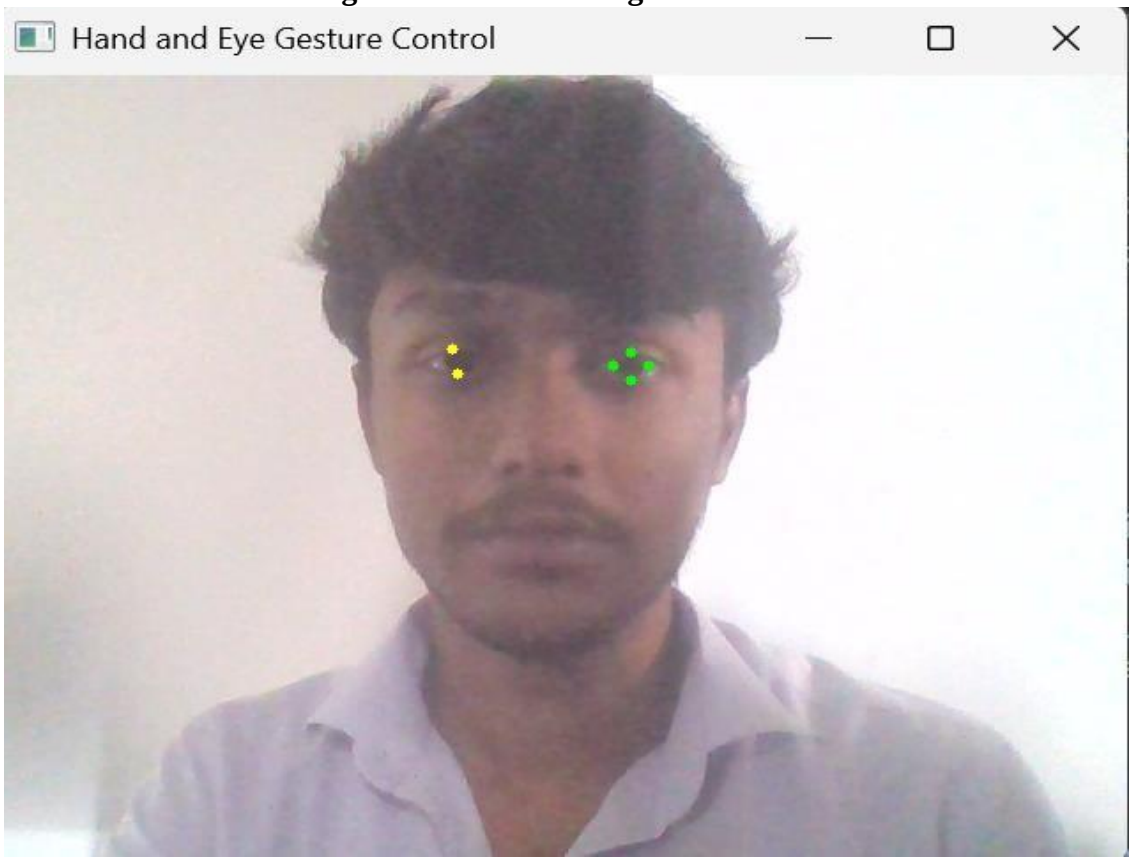


Figure 7.12:Voice Recognition is activated



7.13 Cursor Movement Using Eye

CHAPTER 8

CONCLUSION & FUTURE SCOPE

The gesture-based control system offers an innovative, intuitive alternative to traditional input methods by leveraging tools like MediaPipe and OpenCV for seamless gesture recognition. It emphasizes accessibility and efficiency, with potential applications in healthcare, gaming, and smart homes. Future advancements in gesture accuracy, adaptability, and integration with NLP could enable gesture-based typing, multilingual support, and dynamic gestures, revolutionizing human-computer interaction and creating a more inclusive, hands-free communication platform.

The future scope of the gesture-based control system also includes advancements in virtual and augmented reality (VR/AR), allowing users to navigate immersive environments without traditional controllers. This technology could transform gaming, virtual meetings, and remote collaboration. Integration with wearable devices, such as smart gloves or AR glasses, could offer enhanced precision and intuitive control over IoT devices, robotics, and industrial machinery. Gesture-based systems might also advance accessibility in autonomous vehicles, where hand movements could manage navigation, entertainment, or safety systems. Furthermore, the incorporation of biometrics into gesture recognition could enhance security, creating personalized and secure interaction systems for banking, healthcare, and enterprise applications. As AI and machine learning evolve, gesture control systems could adapt in real time, learning user behavior for increasingly seamless and efficient experiences.

BIBLIOGRAPHY

The official documentation for the technologies used in this project (e.g., Flask, React, MongoDB, PyAutoGUI, Mediapipe) are:

1. Flask Documentation: https://flask.palletsprojects.com/
2. React Documentation: https://reactjs.org/docs/getting-started.html
3. MongoDB Documentation: https://www.mongodb.com/docs/
4. Mediapipe: https://google.github.io/mediapipe/
5. PyAutoGUI: https://pyautogui.readthedocs.io/

The Git Repository referenced in this project is: https://github.com/askitlouder/Media-PlayerControlling-by-HandGestures-using-OpenCV-and-Python

APPENDIX

TOOLS AND TECHNOLOGY

1. MediaPipe

MediaPipe is an open-source framework by Google designed for building cross-platform machine learning solutions. It provides pre-built models and tools for real-time hand tracking and gesture recognition. One of its standout features is its ability to track 21 precise hand landmarks, enabling detailed gesture recognition. In this project, MediaPipe detects and tracks hand movements, extracts key landmarks, and identifies gestures to map them to system commands. Its lightweight and efficient design ensures smooth performance, even on standard hardware.

2. OpenCV

OpenCV (Open Source Computer Vision Library) is a powerful library used for image and video processing. In this project, it complements MediaPipe by capturing video input from the webcam and preprocessing frames for gesture analysis. Its real-time capabilities and robust set of functions make it ideal for detecting hand movements and assisting with gesture recognition.

3. PyAutoGUI

PyAutoGUI is a Python library that automates graphical user interface (GUI) interactions. It allows programmatically simulating mouse movements, clicks, and keyboard actions. In this project, PyAutoGUI executes the commands triggered by recognized gestures, such as controlling media playback, adjusting volume, or navigating files.

4.Flask

Flask is used as the backend framework for this project, providing a lightweight and flexible environment to handle API requests and implement core logic. It manages user interactions,

processes gesture recognition, and executes commands based on detected gestures. Flask also integrates with MongoDB for data storage and facilitates communication with the React frontend through RESTful APIs.

5. Tkinter

Tkinter is the standard Python library used for creating graphical user interfaces (GUIs). It is a wrapper around the Tk GUI toolkit, which is a widely used and cross-platform graphical user interface library. Tkinter allows developers to easily create windows, buttons, labels, text boxes, menus, and other widgets to build interactive desktop applications.

6. Speech Recognition

The **Speech Recognition** module in Python converts speech to text using APIs like Google Web Speech. It captures audio via Microphone, processes it with Recognizer, and supports error handling, enabling applications like voice assistants and transcription tools.

7. SQLite3

SQLite3 is a lightweight, self-contained, and serverless relational database management system designed for simplicity and efficiency. It operates directly from disk files, eliminating the need for a separate server process. SQLite3 is embedded within the application, making it ideal for small to medium-sized projects where minimal configuration and maintenance are required. It is ACID-compliant, ensuring reliable transactions and data integrity

8. Python Programming Language

Python serves as the backbone of this project, integrating all the tools and libraries. Its simplicity, versatility, and extensive library ecosystem make it ideal for developing gesturebased applications. Python's support for libraries like MediaPipe, OpenCV, and PyAutoGUI ensures seamless functionality and easy implementation of complex features.

9. Webcam

The webcam is a critical hardware component for this project, capturing real-time video input of the user's gestures. The video feed serves as the data source for MediaPipe and OpenCV to process and recognize hand gestures. A standard webcam suffices, ensuring accessibility for most users.

10. Webbrowser

The **webbrowser** module in Python provides a convenient interface to allow you to display web-based content in a web browser. It can be used to open URLs in the default web browser or even in a specific browser, making it easy to automate the process of viewing websites or linking to online resources directly from a Python script.

11. Integrated Development Environment (IDE)

IDEs like PyCharm or Visual Studio Code were used to write, test, and debug the application. These tools provide features like syntax highlighting, code suggestions, and debugging capabilities, which streamline the development process. They enable efficient coding and troubleshooting, ensuring smooth project execution**.**