GESTURE CONTROLLED VIRTUAL MOUSE

A DESIGN PROJECT REPORT -III

Submitted by

KALAIYAMUTHAN T (811721104042)

MAHENDRAN N (811721104062)

PADMANABAN U (811721104303)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY

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

SAMAYAPURAM – 621 112

NOVEMBER, 2024

K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report titled **“GESTURE CONTROLLED VIRTUAL MOUSE”** is the bonafide work of **KALAIYAMUTHAN T(811721104042), MAHENDRAN N(811721104062), PADMANABAN U (811721104303)** who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

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

Submitted for the viva-voce examination held on …………….

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

We jointly declare that the project report on **“GESTURE CONTROLLED VIRTUAL MOUSE”** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **“ANNA UNIVERSITY CHENNAI”** for the requirement of Degree of **BACHELOR OF ENGINEERING** in the department of computer science and engineering. This project report is submitted on the partial fulfillment of the requirement of the award of Degree of **BACHELOR OF ENGINEERING**.

Signature

____

KALAIYAMUTHAN T

____

MAHENDRAN N

____

PADMANABAN U

Place: Samayapuram

Date:

ACKNOWLEDGEMENT

It is with great pride that we express our gratitude and in-debt to our institution "**K.Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

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

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

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

We whole heartily thanks to **Dr. A. DELPHIN CAROLINA RANI, M.E., Ph.D.,** Head of the department, **COMPUTER SCIENCE AND ENGINEERING** for providing her encourage pursuing this project.

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

We render our sincere thanks to Course Coordinator and other staff membersfor providing valuable information during the course.

We wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

ABSTRACT

Traditional input devices like keyboards and mice are essential for human-computer interaction but come with several limitations. These devices require continuous physical interaction, making them inaccessible to individuals with physical impairments and unsuitable for sterile environments such as hospitals and laboratories where hygiene is critical. This project addresses the limitations of traditional input devices, such as physical constraints, hygiene concerns, and lack of accessibility for individuals with physical impairments. These issues make traditional devices unsuitable for modern applications requiring intuitive, touchless interaction, especially in environments like healthcare and laboratories where sterility is critical. To solve this problem, a gesture-controlled virtual mouse system was developed using computer vision techniques and algorithms powered by the MediaPipe library for hand tracking and gesture recognition. The system leverages standard webcams and open-source libraries to provide real-time functionality for cursor movement, clicking, and scrolling.

The results demonstrated a gesture recognition accuracy of over 90% in well-lit conditions, with minimal latency, ensuring smooth and responsive performance. While occasional challenges were noted in low-light settings, the system proved to be cost-effective, user-friendly, and scalable for diverse applications. These findings highlight its potential to redefine human-computer interaction by offering a practical, touchless, and ergonomic alternative to traditional input devices, with promising applications in fields like AR, VR, gaming, and smart home systems.

TABLE OF CONTENTS

CHAPTER **TITLE** **PAGE NO.**

LIST OF FIGURES

LIST OF ABBREVIATIONS

G- CVM	-	GESTURE CONTROLLED VIRTUAL MOUSE
HCI	-	HUMAN COMPUTER INTERACTION
SVM	-	SUPPORT VECTOR MACHINE
AR	-	AUGMENTED REALITY
VR	-	VIRTUAL REALITY
PCA	-	PRINCIPAL COMPONENT ANALYSIS
HOG	-	HISTOGRAM OF ORIENTED GRADIENTS
CNN	-	CONVOLUTIONAL NEURAL NETWORK
HCL	-	HARDWARE COMPATIBILITY LIST

CHAPTER 1

INTRODUCTION

In the modern world, human-computer interaction (HCI) plays a pivotal role in the way individuals engage with technology. Traditional input devices such as keyboards and mice have long been the cornerstone of this interaction, offering users a reliable means of controlling digital systems. However, these devices come with inherent limitations, including dependency on physical hardware, susceptibility to wear and tear, and challenges in accessibility for individuals with physical impairments. Moreover, the demand for contactless technology has grown in recent years, driven by concerns around hygiene, particularly in shared or public spaces, and the rising need for more intuitive and innovative interaction methods.

This project introduces a gesture-controlled virtual mouse, an advanced solution that combines the fields of computer vision, artificial intelligence, and natural language processing to provide a hands-free alternative to traditional input devices. The system utilizes a standard webcam to track and analyze hand gestures, translating them into corresponding mouse actions such as cursor movement, clicking, and scrolling. By eliminating the need for physical interaction, the virtual mouse offers a more hygienic, accessible, and user-friendly way to interact with computers.

To further enhance its capabilities, the system integrates voice commands, enabling users to execute tasks through natural language input. This multimodal approach combining gesture recognition and voice control—ensures greater flexibility and inclusivity, making the system suitable for a wide range of users and environments. Built using popular programming tools such as Python, OpenCV, and MediaPipe, the project leverages real-time hand-tracking algorithms, speech recognition libraries, and GUI automation techniques to deliver a seamless and intuitive experience.

1.1 OVERVIEW

The Gesture-Controlled Virtual Mouse project is a novel solution that replaces traditional input devices with an intuitive, touchless interface. By leveraging computer vision techniques, this system detects and interprets hand gestures to perform standard mouse operations such as cursor movement, clicking, and scrolling. This project aims to provide an innovative, cost-effective, and hygienic alternative to conventional interaction methods, with applications in accessibility tools, gaming, and smart environments.

1.1.1 FEATURES OF PROPOSED WORK

- **Real-Time Gesture Recognition:** The system processes live video feed from a webcam, identifying hand gestures and translating them into mouse commands instantly.
- **Advanced Image Processing:**Built on Python and OpenCV, the project utilizes image processing techniques like contour analysis, thresholding, and tracking algorithms for robust gesture detection under varying conditions.
- **Integration with Anaconda Navigator:** The development environment is managed using Anaconda Navigator, ensuring streamlined installation and execution of dependencies. The PowerShell Bash Prompt within Anaconda serves as the interface for running and testing the program.
- **User-Friendly Interaction:** Users can perform various mouse actions effortlessly using simple hand movements, providing an accessible solution for individuals with mobility challenges or in environments where touchless control is preferred.
- **Low Hardware Requirements:** The system requires only a standard webcam and does not rely on specialized hardware, making it widely deployable and cost-effective.

This project highlights the potential of combining computer vision, Python, and accessible technologies to innovate everyday computing tasks.

1.2 PROBLEM STATEMENT

Traditional input devices like keyboards and mice are essential for human-computer interaction but come with several limitations. These devices require continuous physical interaction, making them inaccessible to individuals with physical impairments and unsuitable for sterile environments such as hospitals and laboratories where hygiene is critical. Additionally, they are not compatible with emerging applications in augmented reality (AR), virtual reality (VR), and smart systems, which demand more intuitive and immersive interaction methods. The dependency on physical devices also leads to ergonomic challenges, such as repetitive strain injuries and user fatigue during prolonged use. Existing gesture-based systems that address these issues often rely on expensive hardware, limiting their affordability and scalability. This project aims to develop a cost-effective, accessible, and touchless gesture-controlled virtual mouse system to overcome these challenges and provide a modern solution for seamless human-computer interaction.

1.3 OBJECTIVES

- To create an intuitive and user-friendly interface that operates in real-time.
- To reduce dependency on physical peripherals for accessibility and hygiene in shared environments.
- To demonstrate the integration of computer vision techniques with practical applications.

1.4 GESTURE CONTROLLED VIRTUAL MOUSE

The **Gesture-Controlled Virtual Mouse** is a touchless interface system that allows users to control their computer's cursor and perform standard mouse operations using hand gestures instead of a physical mouse. It utilizes a webcam to capture real-time video and processes the input using Python and OpenCV to recognize gestures and translate them into actions like cursor movement, clicking, scrolling, and drag-and-drop.

This system is designed to enhance usability, accessibility, and hygiene, especially in environments where physical contact with devices needs to be minimized. It is implemented with cost-effective tools, requiring only a standard webcam and open-source software. This innovative approach has applications in accessibility tools, gaming, interactive kiosks, and smart environments.

1.4.1 NEED OF GESTURE CONTROLLED VIRTUAL MOUSE

1. Touchless Interaction
2. Enhanced Accessibility
3. Reduced Hardware Dependence
4. Cost-Effective Solution
5. Faster Input Mechanism
6. Hygienic and Safe
7. Scalable for Multiple Applications
8. Innovative User Experience
9. Paperless and Eco-Friendly
10. Cross-Platform Usability

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

In existing system, the human-computer interaction was mainly depends on physical devices like mice, keyboards, and touchpads, which leads to challenges in terms of accessibility, hygiene, and intuitiveness. These existing devices were difficult for individuals with physical disabilities to use effectively and are prone to contamination in shared environments, especially during health crises. While some gesture recognition systems exist, they often depends on expensive hardware like infrared sensors or specialized gloves, making them less accessible and cost-effective. Here, the traditional image processing techniques in gesture systems struggle with real-time performance, dynamic lighting conditions, and environmental dependencies. These limitations highlight the need for a more affordable, efficient, and user-friendly solution that leverages readily available resources to provide a touchless, hygienic, and intuitive interface for human-computer interaction.

2.1.1 DISADVANTAGES

- **Physical Dependency:** Requires users to physically interact with the device, which can be limiting for people with mobility challenges.
- **Hygiene Concerns:** Shared use in public spaces can lead to hygiene issues, especially in healthcare or communal environments.
- **Wear and Tear:** Devices are prone to damage over time due to continuous use.
- **Limited Accessibility:** Not ideal for users with disabilities or in situations where physical interaction is inconvenient.
- **Restricted Functionality:** Traditional devices cannot be adapted for touchless or advanced interactive applications like VR or gesture-based controls.

2.2 PROPOSED SYSTEM

In proposed system, **Gesture-Controlled Virtual Mouse** has been introduced to provide touchless, gesture-based interface for controlling the cursor and performing mouse operations. This work eliminates the need for physical input devices by leveraging a webcam and real-time image processing using Python and OpenCV.

The **Gesture-Controlled Virtual Mouse** introduces a transformative approach to human-computer interaction by eliminating the need for physical input devices. Instead, it relies on hand gestures captured through a standard webcam, processed in real time using Python and OpenCV. This work is particularly advantageous in environments where touchless control is preferred, such as during a pandemic or in healthcare facilities. The proposed work is accessible, allowing individuals with physical disabilities or mobility impairments to interact seamlessly with computers using simple hand movements.

2.2.1 ADVANTAGES

- **Touchless Interaction**: Promotes hygiene by eliminating the need for physical contact with a device.
- **Accessibility**: Ideal for individuals with disabilities or mobility challenges.
- **Cost-Effective**: Requires only a standard webcam, eliminating the need for additional hardware.
- **Customizable**: Gesture recognition can be tailored to user preferences, allowing for personalized controls.
- **Scalable**: Can be extended to other applications like gaming, virtual reality, and smart devices.
- **Innovative and Modern**: Offers a futuristic and engaging way to interact with computers.

2.3 GOALS

- **Ease of Operation**: The system should be intuitive and user-friendly, ensuring that users can easily operate it without requiring extensive training or technical expertise.
- **Streamlined Workflow**: The proposed system will ensure that tasks are well planned and organized, leading to an efficient and structured workflow.
- **Enhanced Accuracy**: By leveraging advanced algorithms for hand detection and gesture recognition, the proposed system will achieve a higher level of accuracy compared to traditional methods.
- **High Reliability**: The system will offer consistent performance and ensure proper storage of relevant data configurations, enhancing its reliability and reducing chances of error.
- **Quick and Efficient Response**: The system will provide real-time processing and execution of gestures, enabling quick and efficient retrieval of actions and information for a seamless user experience.

CHAPTER 3

LITERATURE SURVEY

3.1 Title: Hand Gesture Recognition for Human-Computer Interaction: A Review

Authors: Rautaray, S. S., & Agrawal, A.

Year: 2015

Rautaray, S.S et al.[1] provides a comprehensive review of hand gesture recognition techniques used for human- computer interaction. It categorizes gesture recognition methods into vision-based and sensor-based approaches. The authors emphasize the growing importance of vision-based techniques due to their cost-effectiveness and ease of deployment, leveraging standard cameras and computer vision algorithms. Challenges such as real-time processing, lighting variations, and occlusions are discussed in detail. The review highlights how advancements in machine learning and deep learning have significantly improved the accuracy and robustness of gesture recognition systems. This work serves as a foundational guide for researchers developing gesture-controlled applications.

3.2 Title: "A Survey on Gesture Recognition Systems: Methods and Applications"

Author: J. Zhang, Y. Yao, T. Xiang

Year: 2017

In this paper, the research has been carried out on field of hand gesture recognition, reviewing different methods and applications of gesture control. J.Zhang et al. [3] discussed about both traditional image processing techniques (such as those used in OpenCV) and modern machine learning-based approaches (such as neural networks). The paper provides a comprehensive comparison of these methods, with a focus on their respective strengths and weaknesses in gesture recognition tasks. J.Zhang et al.[3] also cover various domains where gesture recognition systems have been applied, including virtual environments, smart home control, and multimedia systems.

3.3 Title: "Hand Gesture Recognition for Human-Computer Interaction"

Author: S. L. Saha, K. K. M. Kumar

Year: 2018

S. L. Saha et al.[5] explores the various methods and techniques used in hand gesture recognition for improving human-computer interaction (HCI). It highlights the role of gesture-based systems in enhancing the user experience, particularly in applications such as virtual mice. S. L. Saha et al.[5] emphasize the use of vision-based approaches for gesture recognition, with a focus on contour-based detection methods. These techniques are essential for developing efficient and intuitive gesture-controlled interfaces, which can replace traditional input devices like a mouse and keyboard. **S. L. Saha** et al.[1] discussed the potential of gesture recognition in diverse HCI applications, from entertainment to accessibility tools.

3.4 Title: "Hand Gesture Recognition Using Machine Learning Algorithms"

Author: T. K. Sharma, M. P. Singh

Year: 2019

T. K. Sharma et al.[8] investigates the application of machine learning algorithms in hand gesture recognition systems. It compares supervised learning methods, such as Support Vector Machines (SVM) and Decision Trees, for their efficiency in gesture classification. The study focuses on using feature extraction techniques like Histogram of

Oriented Gradients (HOG) and Principal Component Analysis (PCA) to improve the accuracy and speed of gesture recognition systems. T. K. Sharma et al.[8] highlights the importance of training datasets for better model performance and explore the system's potential in touchless applications, such as virtual mouse control.

3.5 Title: "Real-Time Hand Gesture Recognition for Virtual Mouse Control Using OpenCV"

Author: P. Bhattacharya, S. Pal, R. Sharma

Year: 2020

P. Bhattacharya et al.[9] introduced an real-time hand gesture recognition system using Python and OpenCV. This work allows users to control a computer's mouse cursor with hand gestures, providing a touchless alternative to traditional input devices. P. Bhattacharya et al.[9] focuses on real-time processing and highlight the use of color filtering and contour detection techniques within OpenCV to track hand movements. The system has been optimized for performance in various lighting conditions, ensuring reliable gesture recognition in diverse environments. Here, P. Bhattacharya et al.[9] demonstrates how OpenCV’s robust image processing capabilities can be leveraged for practical, real-time gesture-controlled systems.

3.6 Title: "Vision-Based Hand Gesture Recognition in Real-Time Human-Computer Interaction"

Author: A. Patel, L. Rao

Year: 2020

A. Patel, L. Rao et al.[10] introduced a vision-based approach to hand gesture recognition, focusing on real-time applications in human-computer interaction. The authors use image segmentation, edge detection, and skin color-based filtering techniques for identifying and tracking hand gestures. The system aims to reduce computational complexity while maintaining high responsiveness. It also addresses challenges like variable lighting conditions and occlusion, proposing adaptive algorithms to handle these issues. A. Patel, L. Rao et al.[10] demonstrates the system's use in controlling virtual environments and as an input for touchless user interfaces.

3.7 Title: Vision-Based Hand Gesture Recognition Systems: A Survey of Recent Developments

Authors: Khan, R. A., & Mohandes, M.

Year: 2020

Khan, R. A et al.[11] examines recent advancements in vision-based hand gesture recognition systems, focusing on their application in touchless interfaces. The study evaluates various algorithms, including those based on MediaPipe, Convolutional Neural Networks (CNNs), and Support Vector Machines (SVMs). The authors note that lightweight frameworks like MediaPipe enable real-time gesture tracking on standard devices without requiring high computational power. They also address the limitations of these systems, such as dependency on environmental factors like lighting and background complexity. Khan, R. A et al.[11] concludes by suggesting hybrid approaches that combine traditional algorithms with deep learning models to achieve higher accuracy and adaptability in gesture-controlled systems.

3.8 Title: "Computer Vision-Based Gesture Recognition for Touchless Mouse Control"

Author: R. D. Doshi, V. P. Patil, S. Patel

Year: 2021

R. D. Doshi et al.[14] proposed a touchless mouse control system based on computer vision techniques. The system tracks hand movements to perform mouse actions such as cursor movement and clicking, enabling a touch-free interface. R. D. Doshi et al.[14] explore various gesture recognition methods, including motion tracking, image segmentation, and edge detection, which are integral to the success of the virtual mouse system. The system's primary goal is to provide a hygienic, touchless alternative to traditional input devices, making it particularly useful in public spaces and for users with accessibility needs.

CHAPTER 4

SYSTEM SPECIFICATION

4.1 HARDWARE SPECIFICATION

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems.

4.2 HARDWARE REQUIREMENTS

Processor	:	Intel core 13
RAM	:	8GB
Hard disk	:	ITB (Minimum 80GB)

4.3 SOFTWARE SPECIFICATION

Software Requirements deal with defining software resource requirements and pre-requisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or pre-requisites are generally not included.

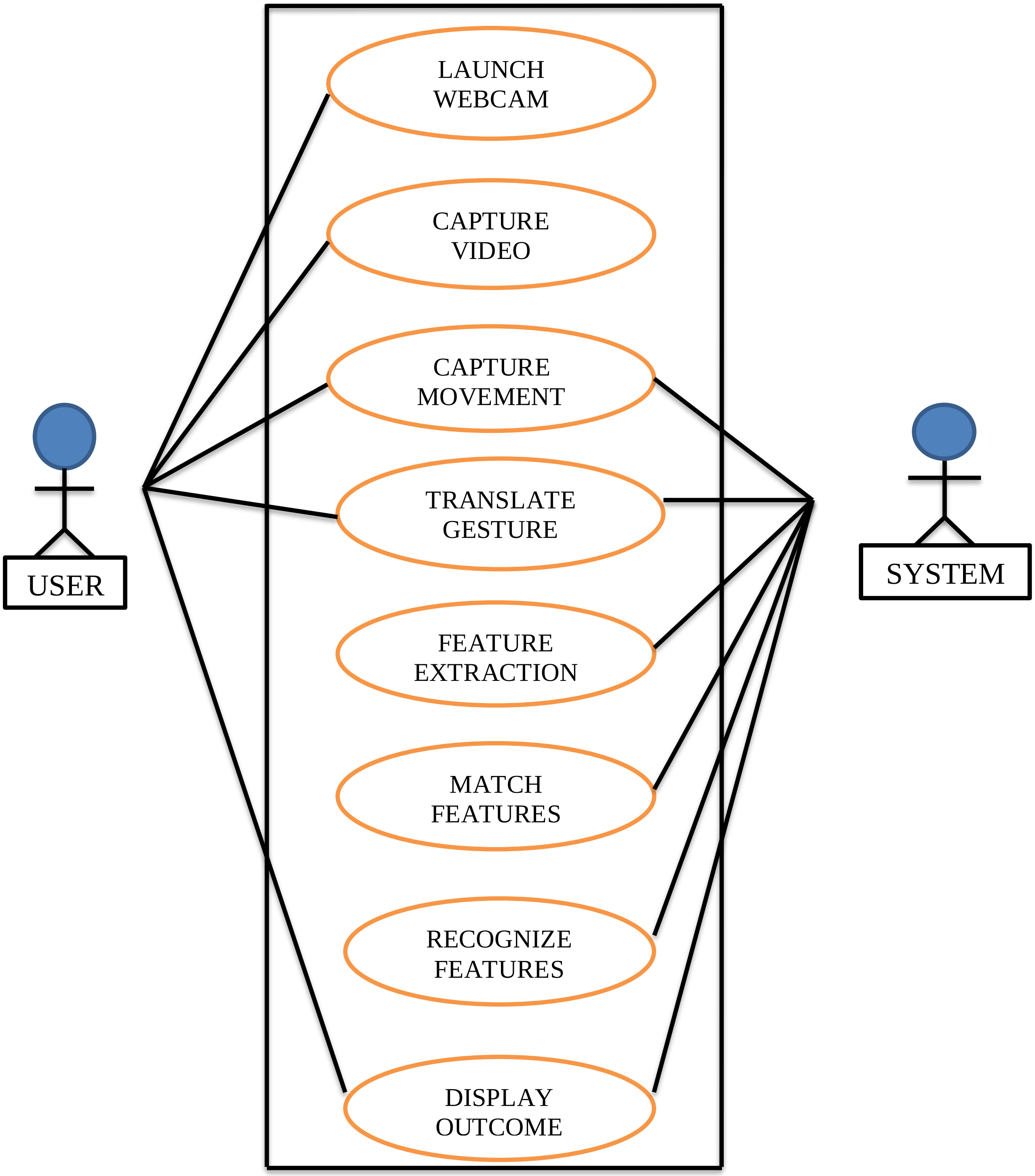
4.4 SOFTWARE REQUIREMENTS

Operating system	:	Windows 7 - 10
Python	:	3.6 – 3.8.5
Anaconda Prompt	:	Anaconda 2024.10-1
OpenCV	:	4.x or higher

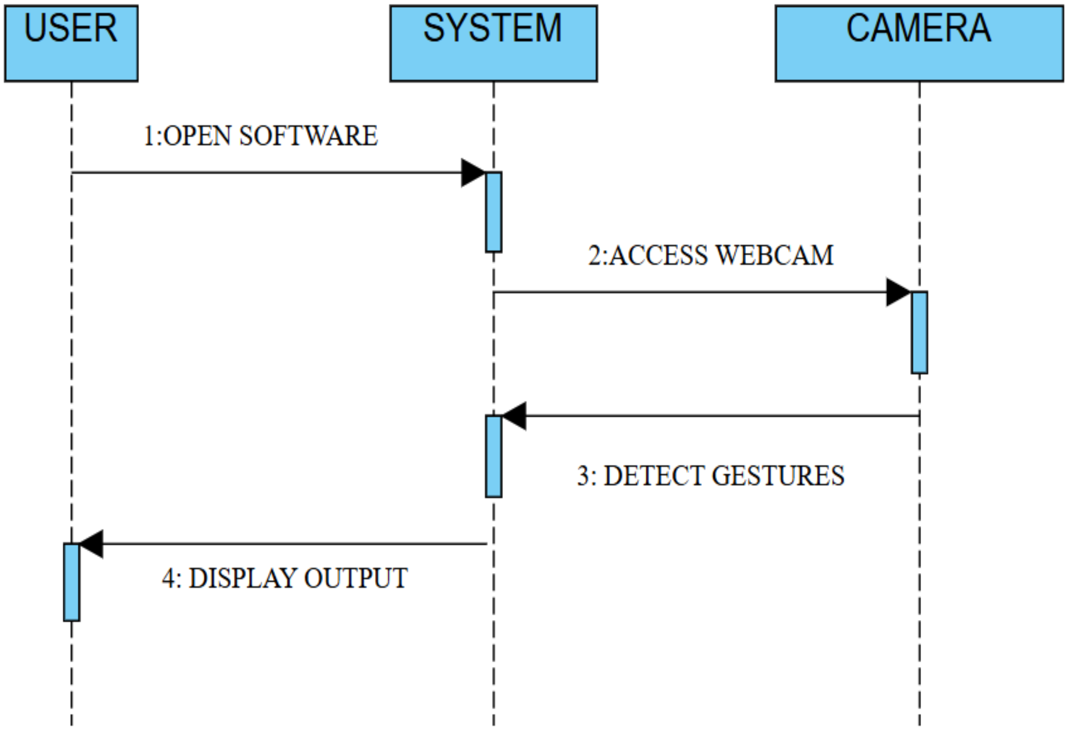
CHAPTER 5

SYSTEM DESIGN

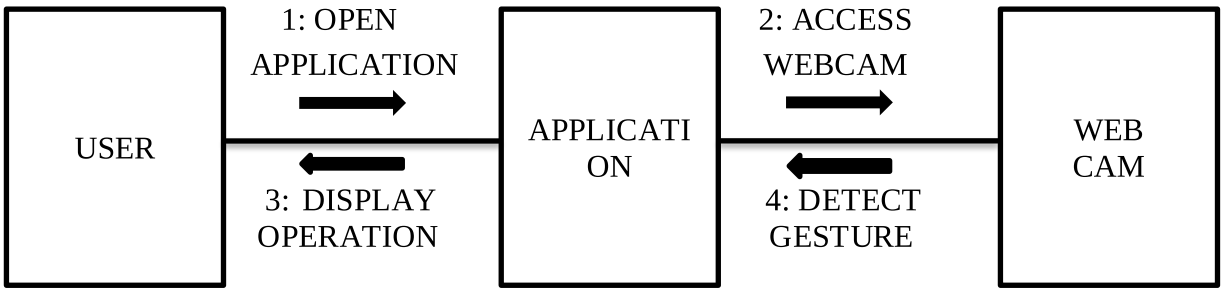
5.1 USE CASE DIAGRAM



5.2 SEQUENCE DIAGRAM



5.3 COLLABORATION DIAGRAM



CHAPTER 6

MODULE DESCRIPTION

The Gesture-Controlled Virtual Mouse project is structured into five key modules to facilitate its functionality and ease of use:

- Hand detection
- Cursor control
- Mouse actions
- System integration
- Gesture mapping

6.1 HAND DETECTION

- **Hand Recognition**: Detects the presence of a hand in the video feed using computer vision techniques such as contour detection or MediaPipe.
- **Gesture Classification**: Identifies specific gestures (e.g., open palm, closed fist) for mapping to corresponding mouse actions.
- **Calibration:** Adjusts detection parameters to ensure accuracy under varying lighting and hand sizes.

6.2 CURSOR CONTROL

- **Cursor Movement:** Tracks hand movement and translates it into cursor motion on the screen.
- **Sensitivity Adjustment:** Allows tuning of movement speed and precision.

6.3 MOUSE ACTIONS

- **Left Click:** Executes a single click when a predefined gesture is detected (e.g., index finger tap).
- **Right Click:** Executes a right-click action with a specific gesture (e.g., two-finger tap).
- **Scrolling:** Performs vertical or horizontal scrolling when a gesture like dragging fingers up or down is recognized.

6.4 SYSTEM INTEGRATION

- **Startup and Initialization:** Configures the webcam, loads required libraries, and initializes the system.
- **Error Handling:** Detects and resolves errors during gesture recognition or action mapping.
- **Environment Setup:** Ensures compatibility with Anaconda Navigator and PowerShell Bash Prompt.

6.5 HAND TRACKING & MOTION ANALYSIS

- **Hand Position Tracking:** Continuously tracks the hand’s position relative to the webcam feed to ensure accurate cursor movement.
- **Gesture Recognition Accuracy:** Uses advanced algorithms like Optical Flow or CNN-based hand gesture detection for higher precision.
- **Motion Smoothing:** Applies algorithms to smooth out jagged or erratic hand movements, providing a more stable cursor experience.

CHAPTER 7

METHODOLOGY

7.1 INTRODUCTION

The methodology outlines the systematic approach followed to design, develop, and implement the gesture-controlled virtual mouse. The project leverages a combination of computer vision techniques, hand-tracking algorithms, and Python libraries to enable seamless interaction between users and their devices using hand gestures. The process begins by defining the objectives and requirements of the system, followed by selecting the appropriate tools and technologies. The methodology emphasizes the integration of real-time video processing for gesture recognition with precise system control mechanisms. Each step, from capturing hand gestures to translating them into meaningful system actions, is structured to ensure accuracy.

This structured approach ensures a robust system capable of handling dynamic inputs in various environments. The methodology also considers scalability, allowing for additional features like voice control and system settings adjustments. By adhering to this methodology, the project achieves its goal of creating an intuitive and versatile input system that bridges the gap between users and technology.

7.2 PROPOSED SYSTEM ARCHITECTURE

The architecture of the gesture-controlled virtual mouse is designed to enable seamless and real-time interaction between the user and their computer through hand gestures. The system starts with the Input Layer, where a webcam captures live video. This video is pre-processed to ensure consistency, including operations like flipping frames horizontally and converting them into a format compatible with the following modules.Next is the Gesture Recognition Module, which uses computer vision techniques to process the video feed. OpenCV is utilized for image processing, while MediaPipe detects and tracks hand landmarks. This module identifies specific gestures by analyzing the positions of key hand points, such as fingertips and joints, enabling precise recognition of actions like pointing or pinching.

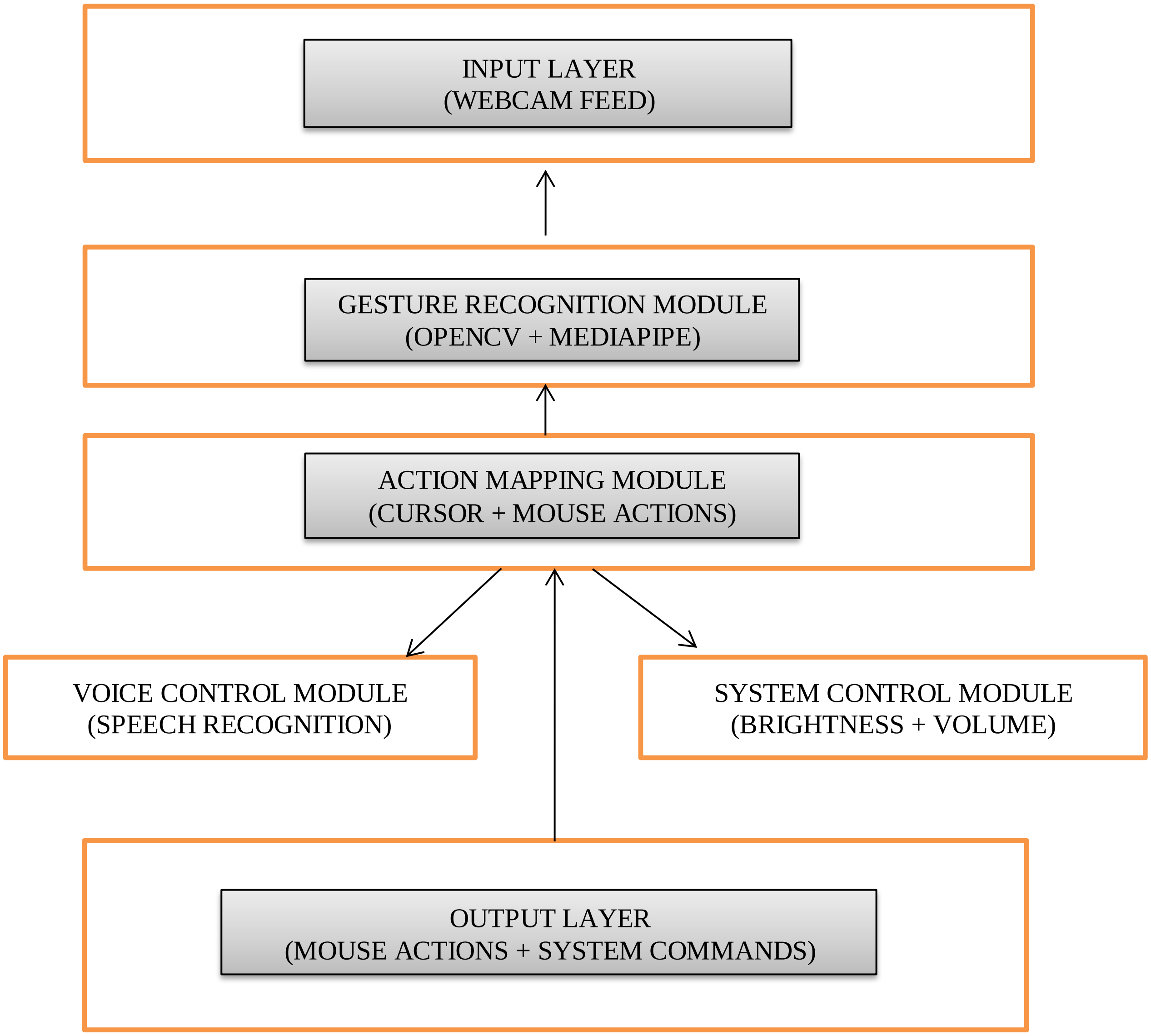


FIGURE 7.2.1 ARCHITECTURE DIAGRAM

The Action Mapping Module translates the recognized gestures into computer actions. Hand movements are mapped to cursor motion, while gestures such as pinching or forming a fist simulate mouse clicks, scrolling, or dragging. This module ensures accurate and smooth interactions using tools like pyautogui.

The System Control Module extends the functionality by allowing users to adjust system settings, such as screen brightness and audio volume. It ensures that users can interact not only with applications but also with system-level features, providing a versatile control mechanism.

Finally, the Output Layer executes the mapped actions, delivering real-time cursor movement, mouse operations, and system control feedback. This layered approach ensures a modular, scalable, and user-friendly system, offering a robust alternative to traditional input devices.

7.3 MEDIA PIPE

MediaPipe is an open-source framework by Google designed for building real-time machine learning solutions, especially in computer vision. It offers pre-built tools for tasks such as hand tracking, pose estimation, and facial landmark detection. Its cross-platform support and GPU acceleration ensure efficient, real-time performance on various devices, including Android, iOS, and desktops. MediaPipe's Hands solution, in particular, provides accurate tracking of 21 3D landmarks, making it ideal for gesture-based applications.

In gesture-controlled virtual mouse project, MediaPipe enables seamless integration of hand gestures with its Python API. Its graph-based architecture simplifies the processing of video feeds, detecting and tracking hand movements to facilitate smooth interaction. This makes MediaPipe a powerful and user-friendly tool for creating innovative applications.

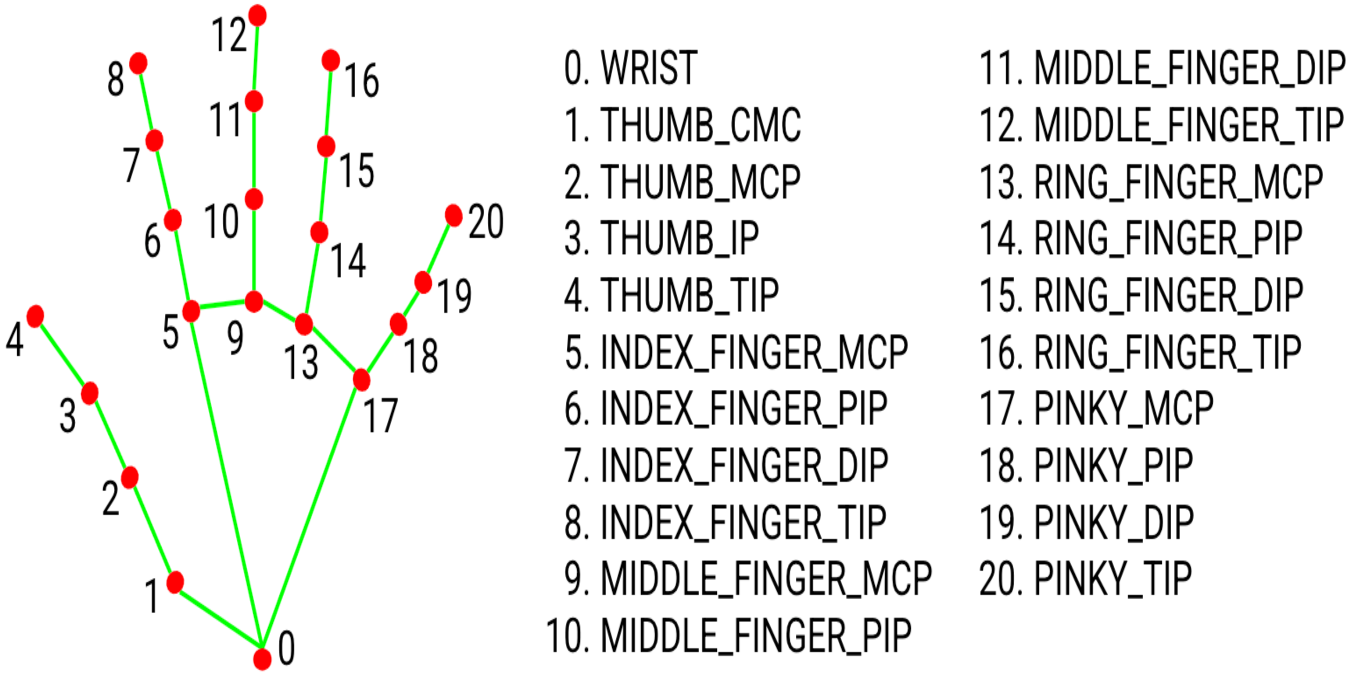


FIGURE 7.3.1 MODEL GRAPH OF MEDIA PIPE

7.4 FUNCTIONAL TESTING

1. Input Handling

- The system must capture real-time video input from a webcam.
- The input must be processed for gesture detection, including resizing and format conversion.
- The system should handle varying lighting conditions to ensure accurate gesture recognition.

2. Gesture Recognition

The system must detect and track hand gestures accurately using MediaPipe.

It should identify specific gestures such as:

- Single Finger Point: For cursor movement.
- Pinch Gesture: For left-click or drag-and-drop actions.
- Fist Gesture: For right-click.
- Vertical Hand Movement: For scrolling up and down.
- The recognition process must work in real time with minimal latency.

3. Cursor Control

- The system must map hand movements to cursor motion on the screen.
- The cursor should respond smoothly and naturally to gestures.

4. Mouse Actions

Recognized gestures should trigger standard mouse actions:

- Left-click: Pinch and release gesture.
- Right-click: Fist gesture.
- Scrolling: Vertical movement of the hand while a gesture is held.
- Drag-and-Drop: Pinch and hold gesture followed by movement.

5. System Control

The system must allow for adjustments of system settings such as:

- Screen Brightness: Controlled through specific gestures.
- Volume Control: Adjusted by predefined hand movements or actions.

Changes in settings must be reflected immediately.

6. Feedback Mechanism

The system should provide feedback to the user for gesture recognition, such as:

- Visual feedback on the screen (e.g., cursor visibility or confirmation pop-ups).
- Auditory feedback using text-to-speech or system sounds (optional).

7. Robustness and Error Handling

The system must handle common errors like:

- Incomplete gestures or unrecognizable movements.
- Temporary loss of hand detection due to obstructions.

The system should revert to standby mode when hands are not detected for a specified duration.

8. Real-Time Performance

Gesture recognition and action mapping must occur in real time with minimal delay (ideally under 100 milliseconds).

The system should maintain smooth performance even in dynamic environments.

7.5 IMPLEMENTATION

The implementation of the Gesture-Controlled Virtual Mouse involves integrating computer vision and machine learning techniques to enable real-time hand gesture recognition and system control. The project is developed using Python, leveraging libraries like OpenCV for image processing, MediaPipe for hand tracking, and pyautogui for simulating mouse and keyboard actions. A standard webcam captures the live video feed, which is processed frame-by-frame to detect hand landmarks using MediaPipe's pre-trained models. These landmarks are analyzed to identify specific gestures corresponding to actions such as cursor movement, left click, right click, and scrolling.

The system processes gestures by first applying image preprocessing techniques, such as grayscale conversion and thresholding, to enhance feature extraction. Hand landmarks are mapped to their respective positions, and their relative movements are used to control cursor actions. Python's pyautogui library translates these movements into system-level inputs, ensuring the virtual mouse operates seamlessly. Additionally, color filtering and contour detection are employed to improve the robustness of gesture detection under varying lighting conditions. The implementation focuses on creating a lightweight and efficient system that works in real-time, ensuring a smooth user experience.

This setup eliminates the need for specialized hardware, relying solely on a standard webcam and open-source libraries. By doing so, the implementation provides an accessible, cost-effective, and hygienic solution for touchless human-computer interaction.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENT

8.1 CONCLUSION

The Gesture-Controlled Virtual Mouse project demonstrates the power of combining computer vision and voice recognition to create an intuitive, hands-free interaction system for controlling a computer. By utilizing Media pipe for hand gesture recognition and Speech Recognition for voice commands, the project offers a seamless, interactive experience where users can control the mouse cursor and adjust system settings, such as volume, through simple hand movements and spoken instructions.

In conclusion, the Gesture-Controlled Virtual Mouse represents a significant step toward intuitive and interactive computing, pushing the boundaries of how humans interact with technology in a natural and hands-free manner.

8.2 FUTURE ENHANCEMENTS

- Improve the hand detection to work better in different lighting conditions or backgrounds.
- Add a calibration feature to adjust the system to different screen sizes or resolutions.
- Allow users to create their own gestures for specific actions, such as opening apps or controlling brightness.
- Connect the system with smart home devices, allowing users to control lights, fans, or other IoT devices with gestures or voice commands.
- Optimize the system to run more smoothly, especially on lower-end computers or battery-powered devices.

APPENDIX A

SOURCE CODE:

App.py:

```
import eel
import os
from queue import Queue
class ChatBot:
    started = False
    userinputQueue = Queue()
    def isUserInput():
        return not ChatBot.userinputQueue.empty()
    def popUserInput():
        return ChatBot.userinputQueue.get()
    def close_callback(route, websockets):
        # if not websockets:
        #     print('Bye!')
        exit()
    @eel.expose
    def getUserInput(msg):
        ChatBot.userinputQueue.put(msg)
        print(msg)
    def close():
        ChatBot.started = False
    def addUserMsg(msg):
        eel.addUserMsg(msg)
    def addAppMsg(msg):
        eel.addAppMsg(msg)
    def start():
```

```
    path = os.path.dirname(os.path.abspath(__file__))
    eel.init(path + r'\web', allowed_extensions=['.js', '.html'])
    try:
        eel.start('index.html', mode='chrome',
                                host='localhost',
                                port=27005,
                                block=False,
                                size=(350, 480),
                                position=(10,100),
                                disable_cache=True,
                                close_callback=ChatBot.close_callback)
        ChatBot.started = True
        while ChatBot.started:
            try:
                eel.sleep(10.0)
            except:
                #main thread exited
                break
    except:
        pass
```

Gesture_controller.py

```
# Imports
import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from google.protobuf.json_format import MessageToDict
import screen_brightness_control as sbcontrol
pyautogui.FAILSAFE = False
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
# Gesture Encodings
class Gest(IntEnum):
    # Binary Encoded
    """
    Enum for mapping all hand gesture to binary number.
    """
    FIST = 0
    PINKY = 1
    RING = 2
    MID = 4
    LAST3 = 7
    INDEX = 8
    FIRST2 = 12
    LAST4 = 15
    THUMB = 16
    PALM = 31
    # Extra Mappings
    V_GEST = 33
    TWO_FINGER_CLOSED = 34
    PINCH_MAJOR = 35
    PINCH_MINOR = 36
# Multi-handedness Labels
class HLabel(IntEnum):
```

```
def get_gesture(self):
    """
    returns int representing gesture corresponding to Enum 'Gest'.
    sets 'frame_count', 'ori_gesture', 'prev_gesture',
    handles fluctations due to noise.
   Returns
    int
    if self.hand_result == None:
        return Gest.PALM
    current_gesture = Gest.PALM
    if self.finger in [Gest.LAST3,Gest.LAST4] and self.get_dist([8,4]) < 0.05:
        if self.hand_label == HLabel.MINOR :
            current_gesture = Gest.PINCH_MINOR
        else:
            current_gesture = Gest.PINCH_MAJOR
    elif Gest.FIRST2 == self.finger :
        point = [[8,12],[5,9]]
        dist1 = self.get_dist(point[0])
        dist2 = self.get_dist(point[1])
        ratio = dist1/dist2
        if ratio > 1.7:
            current_gesture = Gest.V_GEST
        else:
            if self.get_dz([8,12]) < 0.1:
                current_gesture =  Gest.TWO_FINGER_CLOSED
            else:
                current_gesture =  Gest.MID
    else:
        current_gesture =  self.finger
    x coordinate of hand landmark when pinch gesture is started.
pinchstartycoord : int
    y coordinate of hand landmark when pinch gesture is started.
pinchdirectionflag : bool
    true if pinch gesture movment is along x-axis,
    otherwise false
prevpinchlv : int
    stores quantized magnitued of prev pinch gesture displacment, from
    starting position
pinchlv : int
    stores quantized magnitued of pinch gesture displacment, from
    starting position
framecount : int
    stores no. of frames since 'pinchlv' is updated.
prev_hand : tuple
    stores (x, y) coordinates of hand in previous frame.
pinch_threshold : float
    step size for quantization of 'pinchlv'
tx_old = 0
ty_old = 0
trial = True
flag = False
grabflag = False
pinchmajorflag = False
pinchminorflag = False
pinchstartxcoord = None
pinchstartycoord = None
pinchdirectionflag = None
prevpinchlv = 0
```

```
    def __init__(self):
        """Initilaizes attributes."""
        GestureController.gc_mode = 1
        GestureController.cap = cv2.VideoCapture(0)
        GestureController.CAM_HEIGHT =
GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
        GestureController.CAM_WIDTH =
GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH)
          cv2.imshow('Gesture Controller', image)
                if cv2.waitKey(5) & 0xFF == 13:
                    break
        GestureController.cap.release()
        cv2.destroyAllWindows()


# uncomment to run directly
gc1 = GestureController()
gc1.start()
```

Gesture_controller_gloved.py

```
import numpy as np
import cv2
import cv2.aruco as aruco
import os
import glob
import math
import pyautogui
import time
class Marker:
    def __init__(self, dict_type = aruco.DICT_4X4_50, thresh_constant = 1):
        self.aruco_dict = aruco.Dictionary_get(dict_type)
        self.parameters = aruco.DetectorParameters_create()
        self.parameters.adaptiveThreshConstant = thresh_constant
        self.corners = None # corners of Marker
        self.marker_x2y = 1 # width:height ratio
        self.mtx, self.dist = Marker.calibrate()
    def calibrate():
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
        objp = np.zeros((6*7,3), np.float32)
        objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)
        objpoints = [] # 3d point in real world space
        imgpoints = [] # 2d points in image plane.
        path = os.path.dirname(os.path.abspath(__file__))
        p1 = path + r'\calib_images\checkerboard\*.jpg'
        images = glob.glob(p1)
        for fname in images:
            img = cv2.imread(fname)
            gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            ret, corners = cv2.findChessboardCorners(gray, (7,6),None)
            if ret == True:
                objpoints.append(objp)
                corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
                imgpoints.append(corners2)
                img = cv2.drawChessboardCorners(img, (7,6), corners2,ret)

        ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)
        #mtx = [[534.34144579,0.0,339.15527836],[0.0,534.68425882,233.84359493],[0.0,0.0,1.0]]
        #dist = [[-2.88320983e-01, 5.41079685e-02, 1.73501622e-03, -2.61333895e-04, 2.04110465e-01]]
        else:
            sign = -1

        bot_rx = int(cx + self.roi_alpha2 * l * np.sqrt(1/(1+slope_12**2)))
        bot_ry = int(cy + self.roi_alpha2 * slope_12 * l * np.sqrt(1/(1+slope_12**2)))
        bot_lx = int(cx - self.roi_alpha1 * l * np.sqrt(1/(1+slope_12**2)))
        bot_ly = int(cy - self.roi_alpha1 * slope_12 * l * np.sqrt(1/(1+slope_12**2)))
        top_lx = int(bot_lx + sign * self.roi_beta * l * np.sqrt(1/(1+slope_14**2)))
        top_ly = int(bot_ly + sign * self.roi_beta * slope_14 * l * np.sqrt(1/(1+slope_14**2)))
        top_rx = int(bot_rx + sign * self.roi_beta * l * np.sqrt(1/(1+slope_14**2)))
        top_ry = int(bot_ry + sign * self.roi_beta * slope_14 * l * np.sqrt(1/(1+slope_14**2)))
        bot_lx = in_cam(bot_lx, 'x')
        bot_ly = in_cam(bot_ly, 'y')
        bot_rx = in_cam(bot_rx, 'x')
        bot_ry = in_cam(bot_ry, 'y')
        top_lx = in_cam(top_lx, 'x')
        top_ly = in_cam(top_ly, 'y')
        top_rx = in_cam(top_rx, 'x')
        top_ry = in_cam(top_ry, 'y')
    self.roi_corners = [(bot_lx,bot_ly), (bot_rx,bot_ry), (top_rx,top_ry), (top_lx,top_ly)
    def find_glove_hsv(self, frame, marker):
        rec_coor = marker.corners[0][0]
        c1 = (int(rec_coor[0][0]),int(rec_coor[0][1]))
        c2 = (int(rec_coor[1][0]),int(rec_coor[1][1]))
        c3 = (int(rec_coor[2][0]),int(rec_coor[2][1]))
        c4 = (int(rec_coor[3][0]),int(rec_coor[3][1]))
        l = np.absolute(ecu_dis(c1,c4))
        try:
            slope_12 = (c1[1]-c2[1])/(c1[0]-c2[0])
        glove = Glove()

        csrt_track = Tracker()

        mouse = Mouse()

        def __init__(self):

            GestureController.cap = cv2.VideoCapture(0)

            if GestureController.cap.isOpened():

                GestureController.cam_width  = int(
    GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH) )

                GestureController.cam_height = int(
    GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT) )

            else:

                print("CANNOT OPEN CAMERA")

            GestureController.gc_mode = 1

            GestureController.f_start_time = time.time()

            GestureController.f_now_time = time.time()

        def start(self):

            while (True):

                #mode checking

                #display frame

                cv2.imshow('frame',frame)

                if cv2.waitKey(1) & 0xFF == ord('q'):

                    break;

            # When everything done, release the capture

            GestureController.cap.release()

            cv2.destroyAllWindows()
```

APPENDIX B

OUTPUT SCREENSHOT:GESTURE RECOGNITION

MODULE 1: HAND DETECTION

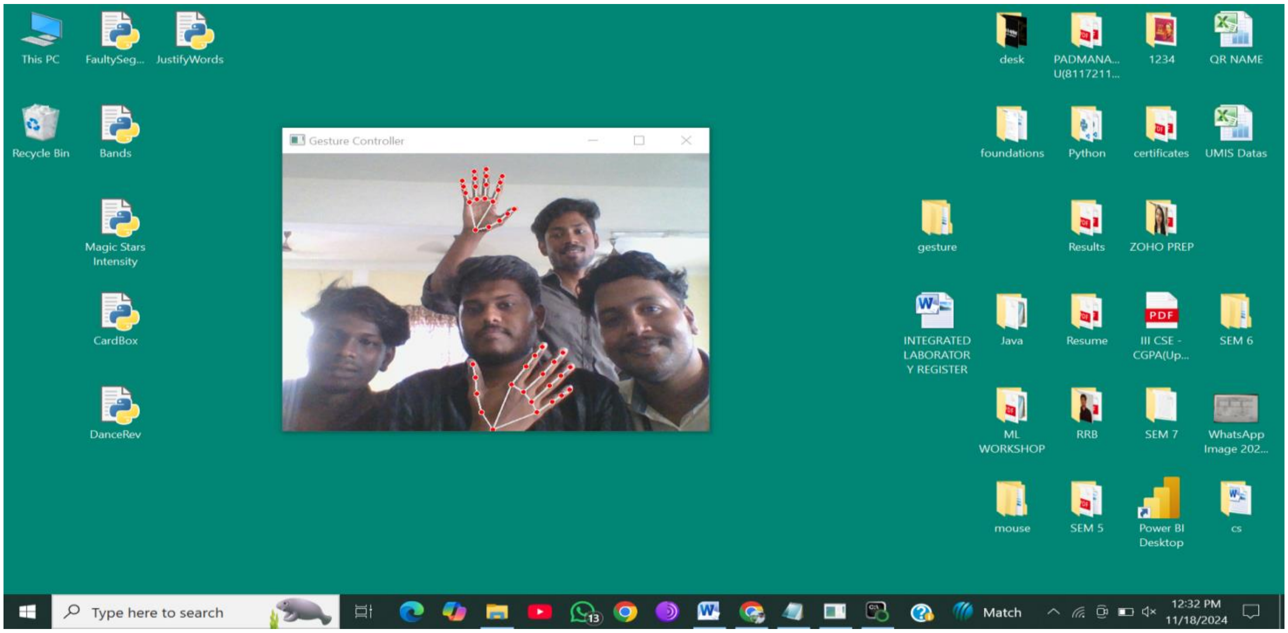
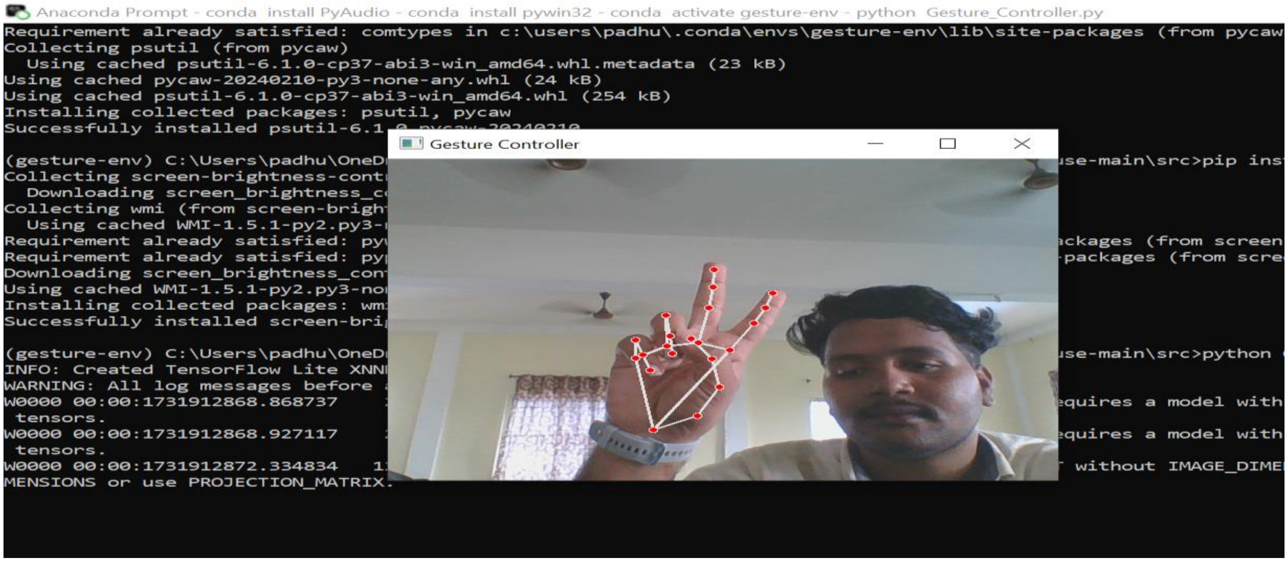


Fig B.1 HAND DETECTION

MODULE 2: CURSOR CONTROL


Anaconda Prompt - conda install PyAudio - conda install pywin32 - conda activate gesture-env - python Gesture_Controller.py
Gesture Controller

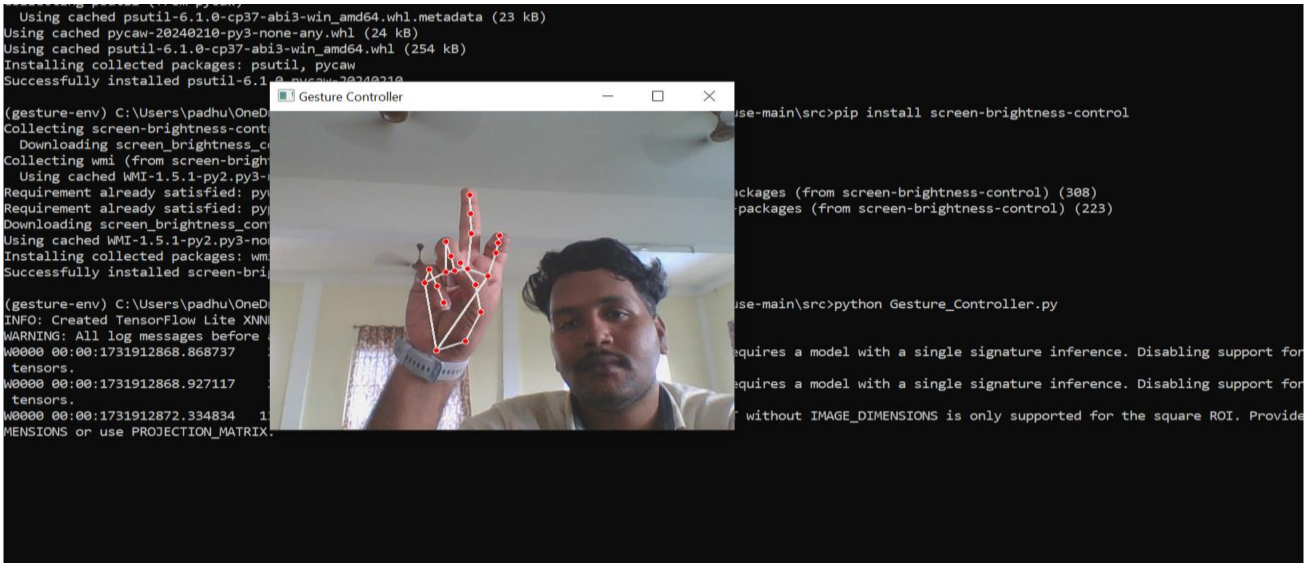
```
Requirement already satisfied: comtypes in c:\users\padhu\.conda\envs\gesture-env\lib\site-packages (from pycaw
Collecting psutil (from pycaw)
  Using cached psutil-6.1.0-cp37-abi3-win_amd64.whl.metadata (23 kB)
Using cached pycaw-20240210-py3-none-any.whl (24 kB)
Using cached psutil-6.1.0-cp37-abi3-win_amd64.whl (254 kB)
Installing collected packages: psutil, pycaw
Successfully installed psutil-6.1

(gesture-env) C:\Users\padhu\OneD                                                  use-main\src>pip ins
Collecting screen-brightness-cont
  Downloading screen_brightness_c
Collecting wmi (from screen-brigh
  Using cached WMI-1.5.1-py2.py3-
Requirement already satisfied: py                                                  ackages (from screen
Requirement already satisfied: py                                                  -packages (from scre
Downloading screen_brightness_con
Using cached WMI-1.5.1-py2.py3-no
Installing collected packages: wm
Successfully installed screen-bri

(gesture-env) C:\Users\padhu\OneD                                                  use-main\src>python
INFO: Created TensorFlow Lite XNN
WARNING: All log messages before
W0000 00:00:1731912868.868737                                                      equires a model with
 tensors.
W0000 00:00:1731912868.927117                                                      equires a model with
 tensors.
W0000 00:00:1731912872.334834   1                                                    without IMAGE_DIME
MENSIONS or use PROJECTION_MATRIX.
```

Fig B.2 CURSOR CONTROL

MODULE 3: MOUSE ACTIONS (LEFT CLICK)



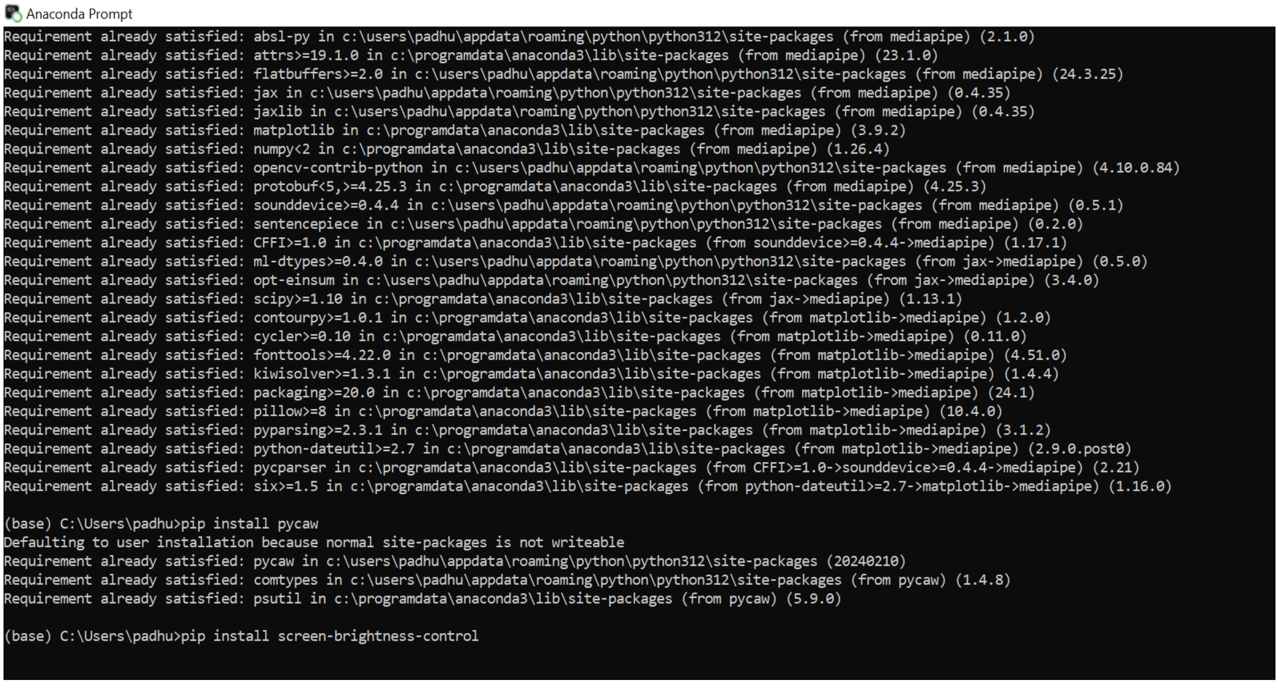
```
  Using cached psutil-6.1.0-cp37-abi3-win_amd64.whl.metadata (23 kB)
Using cached pycaw-20240210-py3-none-any.whl (24 kB)
Using cached psutil-6.1.0-cp37-abi3-win_amd64.whl (254 kB)
Installing collected packages: psutil, pycaw
Successfully installed psutil-6.1.0 pycaw-20240210

(gesture-env) C:\Users\padhu\OneD                                use-main\src>pip install screen-brightness-control
Collecting screen-brightness-cont
  Downloading screen_brightness_c
Collecting wmi (from screen-bright
  Using cached WMI-1.5.1-py2.py3-
Requirement already satisfied: py                                ackages (from screen-brightness-control) (308)
Requirement already satisfied: py                                -packages (from screen-brightness-control) (223)
Downloading screen_brightness_con
Using cached WMI-1.5.1-py2.py3-no
Installing collected packages: wm
Successfully installed screen-bri

(gesture-env) C:\Users\padhu\OneD                                use-main\src>python Gesture_Controller.py
INFO: Created TensorFlow Lite XNN
WARNING: All log messages before
W0000 00:00:1731912868.868737    1                               equires a model with a single signature inference. Disabling support for
 tensors.
W0000 00:00:1731912868.927117    1                               equires a model with a single signature inference. Disabling support for
 tensors.
W0000 00:00:1731912872.334834   1                                 without IMAGE_DIMENSIONS is only supported for the square ROI. Provide
MENSIONS or use PROJECTION_MATRIX.
```

Fig B.3 MOUSE ACTIONS

MODULE 4: SYSTEM INTEGRATIONS



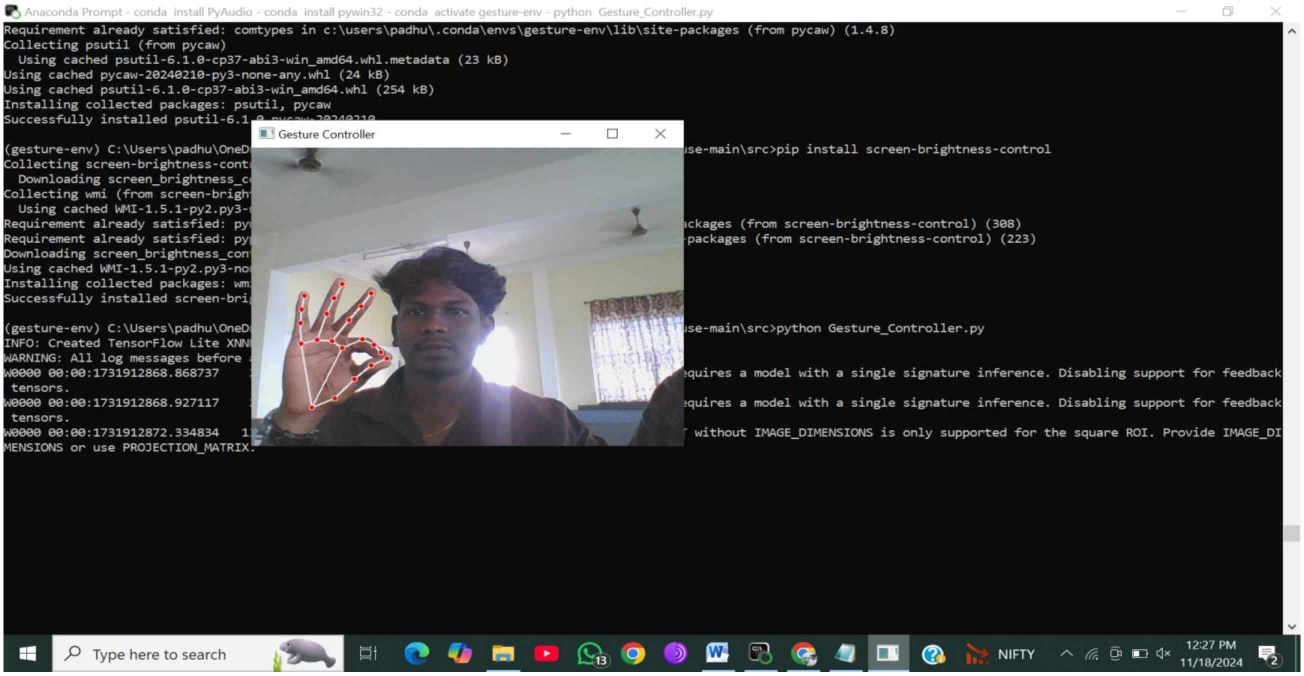
```
Requirement already satisfied: absl-py in c:\users\padhu\appdata\roaming\python\python312\site-packages (from mediapipe) (2.1.0)
Requirement already satisfied: attrs>=19.1.0 in c:\programdata\anaconda3\lib\site-packages (from mediapipe) (23.1.0)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\padhu\appdata\roaming\python\python312\site-packages (from mediapipe) (24.3.25)
Requirement already satisfied: jax in c:\users\padhu\appdata\roaming\python\python312\site-packages (from mediapipe) (0.4.35)
Requirement already satisfied: jaxlib in c:\users\padhu\appdata\roaming\python\python312\site-packages (from mediapipe) (0.4.35)
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-packages (from mediapipe) (3.9.2)
Requirement already satisfied: numpy<2 in c:\programdata\anaconda3\lib\site-packages (from mediapipe) (1.26.4)
Requirement already satisfied: opencv-contrib-python in c:\users\padhu\appdata\roaming\python\python312\site-packages (from mediapipe) (4.10.0.84)
Requirement already satisfied: protobuf<5,>=4.25.3 in c:\programdata\anaconda3\lib\site-packages (from mediapipe) (4.25.3)
Requirement already satisfied: sounddevice>=0.4.4 in c:\users\padhu\appdata\roaming\python\python312\site-packages (from mediapipe) (0.5.1)
Requirement already satisfied: sentencepiece in c:\users\padhu\appdata\roaming\python\python312\site-packages (from mediapipe) (0.2.0)
Requirement already satisfied: CFFI>=1.0 in c:\programdata\anaconda3\lib\site-packages (from sounddevice>=0.4.4->mediapipe) (1.17.1)
Requirement already satisfied: ml-dtypes>=0.4.0 in c:\users\padhu\appdata\roaming\python\python312\site-packages (from jax->mediapipe) (0.5.0)
Requirement already satisfied: opt-einsum in c:\users\padhu\appdata\roaming\python\python312\site-packages (from jax->mediapipe) (3.4.0)
Requirement already satisfied: scipy>=1.10 in c:\programdata\anaconda3\lib\site-packages (from jax->mediapipe) (1.13.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mediapipe) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mediapipe) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mediapipe) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mediapipe) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mediapipe) (24.1)
Requirement already satisfied: pillow>=8 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mediapipe) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mediapipe) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->mediapipe) (2.9.0.post0)
Requirement already satisfied: pycparser in c:\programdata\anaconda3\lib\site-packages (from CFFI>=1.0->sounddevice>=0.4.4->mediapipe) (2.21)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib->mediapipe) (1.16.0)

(base) C:\Users\padhu>pip install pycaw
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pycaw in c:\users\padhu\appdata\roaming\python\python312\site-packages (20240210)
Requirement already satisfied: comtypes in c:\users\padhu\appdata\roaming\python\python312\site-packages (from pycaw) (1.4.8)
Requirement already satisfied: psutil in c:\programdata\anaconda3\lib\site-packages (from pycaw) (5.9.0)

(base) C:\Users\padhu>pip install screen-brightness-control
```

Fig B.4 SYSTEM INTEGRATION

MODULE 5: GESTURE MAPPING



Anaconda Prompt - conda install PyAudio - conda install pywin32 - conda activate gesture-env - python Gesture_Controller.py
Gesture Controller
Type here to search
NIFTY
12:27 PM
11/18/2024

```
Requirement already satisfied: comtypes in c:\users\padhu\.conda\envs\gesture-env\lib\site-packages (from pycaw) (1.4.8)
Collecting psutil (from pycaw)
  Using cached psutil-6.1.0-cp37-abi3-win_amd64.whl.metadata (23 kB)
Using cached pycaw-20240210-py3-none-any.whl (24 kB)
Using cached psutil-6.1.0-cp37-abi3-win_amd64.whl (254 kB)
Installing collected packages: psutil, pycaw
Successfully installed psutil-6.1

(gesture-env) C:\Users\padhu\OneD                                                         use-main\src>pip install screen-brightness-control
Collecting screen-brightness-cont
  Downloading screen_brightness_c
Collecting wmi (from screen-brigh
  Using cached WMI-1.5.1-py2.py3-
Requirement already satisfied: py                                                         ackages (from screen-brightness-control) (308)
Requirement already satisfied: py                                                         -packages (from screen-brightness-control) (223)
Downloading screen_brightness_con
Using cached WMI-1.5.1-py2.py3-no
Installing collected packages: wm
Successfully installed screen-bri

(gesture-env) C:\Users\padhu\OneD                                                         use-main\src>python Gesture_Controller.py
INFO: Created TensorFlow Lite XNN
WARNING: All log messages before
W0000 00:00:1731912868.868737                                                             equires a model with a single signature inference. Disabling support for feedback
 tensors.
W0000 00:00:1731912868.927117                                                             equires a model with a single signature inference. Disabling support for feedback
 tensors.
W0000 00:00:1731912872.334834   1                                                         without IMAGE_DIMENSIONS is only supported for the square ROI. Provide IMAGE_DI
MENSIONS or use PROJECTION_MATRIX.
```

Fig B.5 GESTURE MAPPING

REFERENCES

[1]. Rautaray, S. S., & Agrawal, A. (2015), Hand Gesture Recognition for Human Computer Interaction: A Review.

[2]. K. Mitra, A. Acharya (2016), Gesture Recognition Using Deep Learning Techniques for Real-Time Applications.

[3]. J. Zhang, Y. Yao, T. Xiang (2017), A Survey on Gesture Recognition Systems: Methods and Applications.

[4]. R. C. Gupta, S. K. Singh (2017), An Overview of Vision-Based Hand Gesture Recognition Techniques.

[5]. S. L. Saha, K. K. M. Kumar (2018), Hand Gesture Recognition for Human-Computer Interaction.

[6]. H. Wang, J. Luo (2018), Real-Time Gesture Control Using Convolutional Neural Networks and OpenCV.

[7]. T. K. Sharma, M. P. Singh (2019), Hand Gesture Recognition Using Machine Learning Algorithms.

[8]. M. Sharma, K. P. Patel (2019), Comparative Analysis of Hand Gesture Recognition Algorithms for Human-Computer Interaction.

[9]. P. Bhattacharya, S. Pal, R. Sharma (2020), Real-Time Hand Gesture Recognition for Virtual Mouse Control Using OpenCV.

[10]. A. Patel, L. Rao (2020), Vision-Based Hand Gesture Recognition in Real-Time Human-Computer Interaction.

[11]. Khan, R. A., & Mohandes, M. (2020), Vision-Based Hand Gesture Recognition Systems: A Survey of Recent Developments.

[12]. P. Jain, A. Tiwari (2020), Gesture-Based Interfaces: Current Trends and Future Prospects.

[13]. S. H. Lee, J. Kim (2020), Hand Tracking and Gesture Recognition for Smart Device Control Using MediaPipe Framework.

[14]. R. D. Doshi, V. P. Patil, S. Patel (2021), Computer Vision-Based Gesture Recognition for Touchless Mouse Control.

[15]. A. Kumar, P. Roy, S. Banerjee (2021), Touchless Interaction Systems: A Study on Gesture-Controlled Devices.

[16]. J. Zhou, M. Li (2021), Gesture Recognition for Virtual Mouse Applications Using Computer Vision.

[17]. D. L. Zhang, T. X. Nguyen (2021), Human-Computer Interaction Through Vision-Based Gestures in Dynamic Environments.

[18]. P. Singh, R. Verma, A. Kaushik (2022), Deep Learning for Hand Gesture Recognition in Real-World Scenarios.

[19]. N. Chawla, S. P. Garg (2022), Design and Development of Virtual Mouse Using Hand Gestures and OpenCV.

[20]. M. Roy, H. Patel, R. Desai (2023), Advancements in Vision-Based Gesture Recognition Systems for AR and VR Applications.