Gesture Controlled Virtual Mouse & Voice Assistant

by

Deepanshu Gupta	**(1900270120019)**
Devansh Kumar	**(1900270120020)**
Harsh Harit	**(1900270120025)**
Jigyansh Varshney	**(1900270120033)**

Submitted

to

Department of Computer Science & Engineering

in partial fulfillment of the requirements

for the degree

of

Bachelor of Technology

in

Computer Science & Engineering

Ajay Kumar Garg Engineering College, Ghaziabad

Dr. APJ Abdul Kalam Technical University, Lucknow

June, 2023

TABLE OF CONTENTS

DECLARATION…………………………………………………………………………..…...iv

CERTIFICATE………………………………………………………….….…………………v

ACKNOWLEDGEMENT……….……………………………………………...…………...vi

ABSTRACT…………………………………………………………..………………vii

LIST OF FIGURES………………………………………………………..………………viii

LIST OF ABBREVIATIONS……………………………………………………..............x

CHAPTER 1 (INTRODUCTION)

1.1 INTRODUCTION………………………………….…………………………...11

1.2 METHODOLOGY……………………………….……………….…………..11

1.3 OBJECTIVE……………………………………...…..……………………13

CHAPTER 2 (SOFTWARE REQUIREMENTS SPECIFICATIONS)

2.1 PURPOSE…………………………..…………………………………….…...…14

2.2 SCOPE OF PROJECT………...……………………………….………...……14

2.3 REFERENECES…………………………………..........………….……...14

2.4 OVERVIEW OF DOCUMENT…………………………………….….….…14

2.5 OVERALL DESCRIPTION……………………………………………….....15

2.6 FUNCTION REQUIREMENTS……………………………….………..16

2.7 NON-FUNCTIONAL REQUIREMENTS………………………………...…31

CHAPTER 3 (SOFTWARE DESIGN DOCUMENT)

3.1 DEPLOYMENT DESIGN…………….………………………….……..…32

3.2 ARCHITECTURE DESIGN………..………………………………….…..……33

3.3 DATA FLOW DIAGRAM.…………………………...……………….……..35

3.4 SEQUENCE DIAGRAM…….…………………………....……..………...38

3.5 ER DIAGRAM……………………………………………………..............39

3.6 FLOWCHART……………...……………………………..........………40

3.7 TECHNOLOGICAL DESCRIPTION……..…………………………...……….43

CHAPTER 4 (IMPLEMENTATION AND SYSTEM INTEGRATION)

4.1 PSEUDOCODE FOR GESTURE RECOGNITION……………………………….45

4.2 PSEUDOCODE FOR VOICE ASSISTANT……………………………….......…….53

CHATPER 5 (TESTCASES AND SNAPSHOTS)

5.1 TESTCASES OF GESTURE RECOGNITION………………………………………55

5.2 TESTCASES OF VOICE ASSISTANT………………………………………………65

CHAPTER 6 (FUTURE SCOPE)

6.1 FUTURE SCOPE OF GESTURE RECOGNITION………………………………….70

6.2 FUTURE SCOPE OF VOICE ASSISTANT…………………………………………..72

6.3 LIMITATIONS………………………………………………………………………73

CHATPER 7 (CONCLUSION)

7.1 CONCLUSION………………………………………………………………………75

REFERENCES……………..…………………………...……………………………………...77

APPENDIX A……………………………………………………………………………………78

RESUME OF EACH STUDENT…………………………………………………………....80

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature:

Name: Deepanshu Gupta

Roll No: 1900270120019

Date: 01/06/2023

Signature:

Name: Devansh Kumar

Roll No: 1900270120020

Date: 01/06/2023

Signature:

Name: Harsh Harit

Roll No: 1900270120025

Date: 01/06/2023

Signature:

Name: Jigyansh Varshney

Roll No: 1900270120033

Date: 01/06/2023

CERTIFICATE

This is to certify that Project Report entitled "**Gesture Controlled Virtual Mouse and Voice Assistant**" which is submitted by **Deepanshu Gupta (1900270120019), Devansh Kumar (1900270120020), Harsh Harit (1900270120025), Jigyansh Varshney (1900270120033)** in partial fulfilment of the requirement for the award of degree B. Tech. in Department of Computer Science and Engineering of Dr. APJ Abdul Kalam Technical University, is a record of the candidate own work carried out by him under our supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

Supervisor

Ms Harnit Saini

(Assistant Professor)

CSE Department

Date: 01/06/2023

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the report of the B. Tech Project undertaken during B. Tech. Final Year. We owe special debt of gratitude to ***Ms Harnit Saini****, Department of Computer Science & Engineering, Ajay Kumar Garg Engineering College, Ghaziabad for her constant support and guidance throughout the course of our work. Her sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only her cognizant efforts that our endeavours have seen light of the day.*

We also take the opportunity to acknowledge the contribution of ***Professor Sunita Yadav****, Head, Department of Computer Science & Engineering, Ajay Kumar Garg Engineering College, Ghaziabad for his full support and assistance during the development of the project.*

We also do not like to miss the opportunity to acknowledge the contribution of all faculty and staff members of the department for their kind assistance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.

Signature:

Name: Deepanshu Gupta

Roll No: 1900270120019

Date: 01/06/2023

Signature:

Name: Devansh Kumar

Roll No: 1900270120020

Date: 01/06/2023

Signature:

Name: Harsh Harit

Roll No: 1900270120025

Date: 01/06/2023

Signature:

Name: Jigyansh Varshney

Roll No: 1900270120033

Date: 01/06/2023

ABSTRACT

The Gesture Controlled Virtual Mouse and Voice Assistant project aims to provide an alternative and more accessible way of controlling a computer mouse using hand gestures and voice commands. This project leverages computer vision and speech recognition technologies to enable users to control a virtual mouse cursor on the computer screen through natural and intuitive interactions.

The project's background and context highlight the limitations of traditional computer mice for certain users, such as those with physical disabilities or mobility limitations, and the increasing demand for alternative input methods in emerging technologies like virtual reality. The project builds on existing research and advancements in computer vision and speech recognition, as well as related fields like human-computer interaction and accessibility.

The project's key components include a computer vision module for detecting hand gestures and a speech recognition module for interpreting voice commands. These modules work in tandem to translate hand gestures and voice commands into mouse cursor movements and clicks. The system also offers customization options, allowing users to configure sensitivity, set custom gestures and voice commands, and adapt the system to their preferences.

The project's potential practical applications include gaming, virtual reality, and presentations, where hand gestures and voice commands can provide a more immersive, interactive, and efficient way of interacting with computers. The project aims to develop a functional prototype that demonstrates the feasibility and potential of a gesture and voice controlled virtual mouse system, and contribute to the advancement of the field.

Overall, the Gesture Controlled Virtual Mouse and Voice Assistant project aims to enhance accessibility and usability in computer interaction by leveraging gesture and voice recognition technologies, and has the potential to offer a more intuitive and enjoyable user experience in various domains.

LIST OF FIGURES

Figure No.		Page No
Fig 2.1	System Environment	15
Fig 2.2	Use Case Diagram : Gesture Recognition	16
Fig 2.3	Neutral Gesture	17
Fig 2.4	Move Cursor	18
Fig 2.5	Left Click	19
Fig 2.6	Right Click	20
Fig 2.7	Double Click	21
Fig 2.8	Scrolling	22
Fig 2.9	Drag and Drop	23
Fig 2.10	Multiple Item Selection	24
Fig 2.11	Volume Control	25
Fig 2.12	Brightness Control	26
Fig 2.13	Use Case Diagram : Voice Assistant	26
Fig 2.14	Launch and Stop Gesture Recognition	27
Fig 2.15	Google Search	28
Fig 2.16	Current Date and Time	28
Fig 2.17	Copy and Paste	29
Fig 2.18	Find location on Google Maps	29
Fig 2.19	File Navigation	30
Fig 2.20	Sleep and Wakeup	30
Fig 2.21	Exit	31
Fig 3.1	Deployment Diagram	32
Fig 3.2	Architecture Diagram	33
Fig 3.3	Model View Controller	34

Fig 3.4	Activity Diagram	35
Fig 3.5	Data Flow Diagram	36
Fig 3.6	Level-1 DFD	36
Fig 3.7	Level-2 DFD	37
Fig 3.8	Sequence Diagram	38
Fig 3.9	ER Diagram	39
Fig 3.10	Gesture Recognition Flowchart	41
Fig 3.11	Voice Assistant Flowchart	42

LIST OF ABBREVIATIONS

GVCM	Gesture and Voice Controlled Virtual Mouse
CV	Computer Vision
SR	Speech Recognition
VR	Virtual Reality
HCI	Human-Computer Interaction

CHAPTER 1
INTRODUCTION

1.1 Introduction

The use of computer input devices such as the traditional mouse and keyboard have been the standard method of interfacing with computers for many years. However, these input methods can sometimes be cumbersome and unintuitive, especially for individuals who are not familiar with computers. This has led to the development of alternative input methods such as touchscreens, styluses, and voice recognition systems.

Gesture and voice recognition virtual mouse is a system that provides an alternative method of controlling a computer cursor using hand gestures and voice commands. The system is designed to improve user experience by allowing users to interact with their computers in a more natural and intuitive way. This technology has numerous potential applications, including gaming, accessibility for individuals with disabilities, and improved user experience for everyday computer use.

The project aims to develop a low-cost, user-friendly, and reliable virtual mouse that can be used by anyone with a computer and a webcam. By leveraging computer vision and speech recognition techniques, the system is designed to provide accurate and reliable cursor movements based on user commands. The virtual mouse technology has the potential to transform the way people interact with their computers, making the process more intuitive, accessible, and enjoyable.

The traditional computer mouse has been the standard input device for decades, but its limitations and discomfort have inspired researchers and developers to explore alternative input methods. Gesture and voice recognition virtual mouse is an innovative project that allows users to control a computer cursor using hand gestures and voice commands. This system aims to provide a more intuitive and natural user interface and to offer an alternative to traditional computer mouse input.

Gesture recognition controlled virtual mouse is a system that allows users to control their computer's mouse cursor using hand gestures instead of a physical mouse. This technology leverages computer vision algorithms to track and interpret the movements of the user's hand, enabling them to perform various mouse-related tasks by simply gesturing in the air. This project report presents the development and implementation of a gesture recognition controlled virtual mouse system.

1.2 Methodology

The project was implemented using a combination of computer vision and speech recognition techniques. OpenCV, a popular computer vision library, was used to detect hand gestures and track their movement. OpenCV provides several image processing techniques that help in detecting and tracking hands and fingers. The voice recognition module was developed using Google Speech

API, which is an online speech recognition service. The system was implemented in Python programming language and used the PyAutoGUI library to simulate mouse clicks and movements.

The hand gesture detection and tracking module uses a combination of skin color segmentation, contour detection, and convex hull algorithms. The module first detects the hand region in the image by applying a skin colour filter. Then, it detects the contours of the hand region and computes the convex hull of the contours. The convex hull is used to extract the hand's boundary and to detect the fingers' tips. The module then calculates the hand's centroid and uses it to control the cursor's movement.

The voice recognition module uses the Google Speech API to recognize voice commands. The module captures the user's speech and sends it to the API for recognition. Once the API returns the recognized text, the module performs the appropriate mouse actions, such as moving the cursor or clicking.

The project utilizes a combination of hardware and software components to achieve its objectives. The following steps outline the methodology used in the project:

Hardware Setup:

- A camera (such as a webcam) is connected to the computer to capture video input.
- The camera is positioned to have a clear view of the user's hand gestures.

Gesture Recognition Algorithm:

- The captured video frames are processed using computer vision techniques to detect and track hand movements.
- The algorithm analyzes the hand movements and extracts relevant features for gesture recognition.
- Machine learning algorithms, such as convolutional neural networks (CNNs), are trained to classify different hand gestures based on the extracted features.

Mouse Control System:

- The recognized gestures are mapped to corresponding mouse movements and actions.
- For example, a swipe gesture to the right could translate to a mouse movement to the right, while a pinch gesture could simulate a mouse click.
- The mouse control system translates the recognized gestures into appropriate mouse commands to control the cursor on the computer screen.

User Interface:

- A user interface is developed to provide users with options for configuring and customizing the gesture recognition system.
- Users can set gesture preferences, define custom gestures, and adjust sensitivity settings.
- The interface also provides visual feedback to the user, indicating the recognized gestures and corresponding mouse actions.

1.3 Objective

The project's main goal is to create a hands-free virtual Mouse system that focuses on a few key applications in development. This project aims to eliminate the need for a physical mouse while allowing users to interact with the computer system via webcam and speech using various image and audio processing techniques. This project seeks to create a Virtual Mouse programme that can be used in a variety of contexts and on a variety of surfaces.

The following are the objectives of the project:

- The cursor is assigned to a certain screen position when the hand gesture/motion is converted into a mouse operation. The Virtual Mouse application is set up to identify the position of the mouse pointers by detecting the position of the fingertips and knuckles on a defined hand colour and texture.

- Develop a multi user independent speech recognition system that captures voice in real-time with the help of a microphone and is able to retrieve folders, sub-folders, documents, copy, paste, left click, right click and double click by taking voice command and checking its validity.

- Create a voice-activated mouse system that works in tandem with the gesture-activated system.

- Design for mouse operation with the aid of a webcam. The Virtual Mouse technology works with the help of a webcam, which takes real-time photos and photographs. A webcam is required for the application to function.

CHAPTER 2
SOFTWARE REQUIREMENTS SPECIFICATIONS

2.1. Purpose

The purpose of this document is to present a detailed description of the Gesture and Voice Controlled Virtual Mouse. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both the stakeholders and the developers of the system.

2.2. Scope of Project

This software aims to eliminate the need for a physical mouse while allowing users to interact with the computer system via webcam and speech using various image and audio processing techniques. This project seeks to create a Virtual Mouse programme that can be used in a variety of contexts and on a variety of surfaces.

Design for mouse operation with the aid of a webcam. The Virtual Mouse technology works with the help of a webcam, which takes real-time photos and photographs. A webcam is required for the application to function. Develop a multi user independent speech recognition system that captures voice in real-time with the help of a microphone and is able to retrieve folders, sub-folders, documents, copy, paste, left click, right click and double click by taking voice command and checking its validity.

2.3. References

IEEE. *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.* IEEE Computer Society, 1998.

2.4. Overview of Document

The next chapter, the Overall Description section, of this document gives an overview of the functionality of the product. It describes the informal requirements and is used to establish a context for the technical requirements specification in the next chapter.
The third chapter, Requirements Specification section, of this document is written primarily for the developers and describes in technical terms the details of the functionality of the product. Both sections of the document describe the same software product in its entirety, but are intended for different audiences and thus use different language.

2.5. Overall Description

2.5.1 System Environment

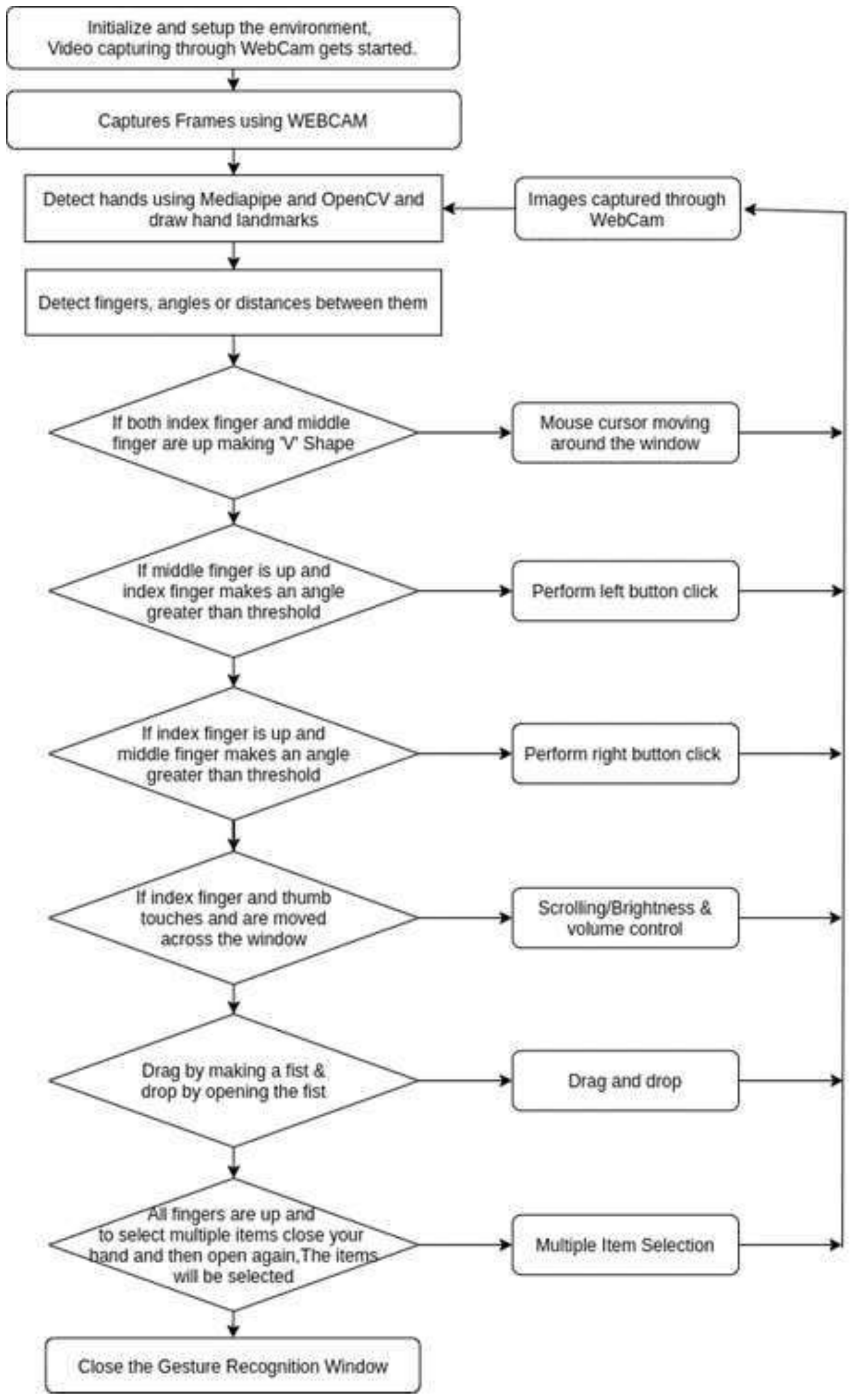

Initialize and setup the environment, Video capturing through WebCam gets started.
Captures Frames using WEBCAM
Detect hands using Mediapipe and OpenCV and draw hand landmarks
Images captured through WebCam
Detect fingers, angles or distances between them
If both index finger and middle finger are up making 'V' Shape
Mouse cursor moving around the window
If middle finger is up and index finger makes an angle greater than threshold
Perform left button click
If index finger is up and middle finger makes an angle greater than threshold
Perform right button click
If index finger and thumb touches and are moved across the window
Scrolling/Brightness & volume control
Drag by making a fist & drop by opening the fist
Drag and drop
All fingers are up and to select multiple items close your hand and then open again,The items will be selected
Multiple Item Selection
Close the Gesture Recognition Window

Fig 2.1 System Environment

The Gesture and Voice Controlled Virtual Mouse has only one active actor and one cooperation system. The user accessing the computer has control over the mouse and he performs the actions

on the computer using the mouse. These actions are performed either by the gestures as shown by the user to the camera being attached to the computer system or the user can give voice commands to the computer to which the computer interprets and responds with the desired
results to the user.

2.6 Functional Requirements Specification

This section outlines the use cases for each of the active users separately.

End User Use Case (For Gesture Recognition)

Use Case: **Perform Mouse Functions:**

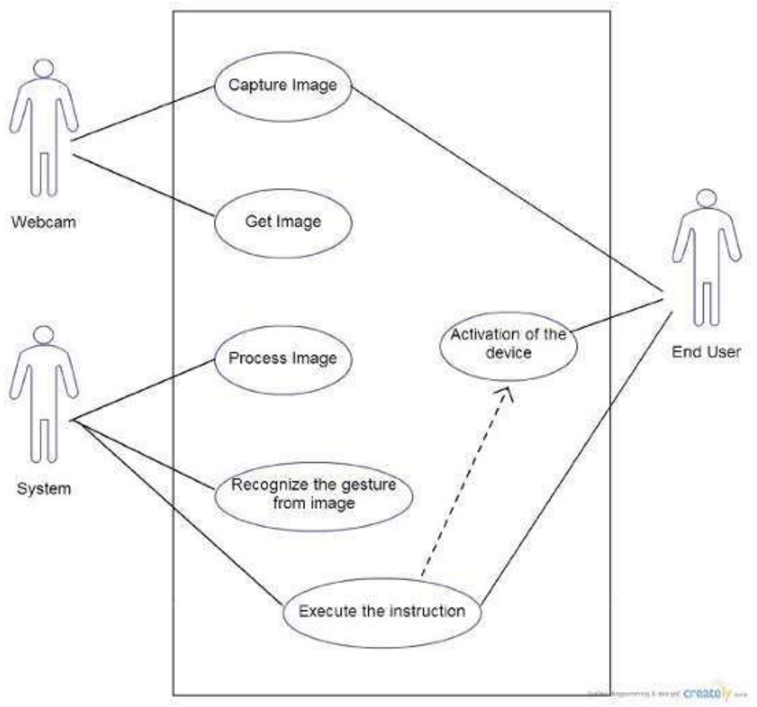


Fig 2.2 Use Case Diagram : Gesture Recognition

Brief Description

The user shows gestures in front of the screen, the webcam captures the movement of the hands and performs the appropriate function as per the recognized gesture.

Initial Step-By-Step Description

1. The user plans which action is to executed with the mouse.
2. The user then performs the gestures as per his/her needs.
3. The webcam tracks the movement of the hands.
4. The system interprets the gestures of the hands captured by the webcam.
5. The system maps the recognized gesture with the specified action for that particular gesture.
6. The system replies back by performing the desired operation as per the gesture shown by the user in front of the webcam.

Mouse Functions Depending on the Hand Gestures and Hand Tip Detection Using Computer Vision.

1. <u>Neutral Gesture</u>: The neutral gesture, in the context of a gesture recognition virtual mouse project, refers to the default or resting position of the hand that is not associated with any specific mouse action or command. It is typically used as a reference point or starting point for recognizing other gestures.

When developing a gesture recognition virtual mouse, you would need to define the neutral gesture as a baseline or reference for detecting and differentiating it from other gestures. The neutral gesture could be identified based on certain criteria such as hand position, palm orientation, or absence of specific hand movements.

By recognizing the neutral gesture, the system can distinguish when the user intends to perform a gesture command rather than simply keeping their hand in a neutral position. This allows for more precise and accurate gesture recognition and control of the virtual mouse.

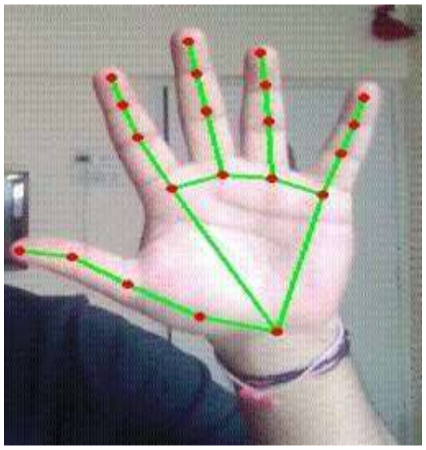


Fig 2.3 Neutral Gesture

2. <u>Move Cursor</u>: The movement of the cursor is mapped to the **V** position formed by the index and middle finger. To move the cursor using a **V** gesture, you can follow these steps:

i. **Hand Tracking:** Utilize a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Gesture Detection:** Implement an algorithm to detect the V gesture. This algorithm can involve analysing the hand's shape, finger positions, or hand movements. For example, you may check if the user's hand forms a V shape by examining the relative positions of the fingertips or the angle between the fingers.

iii. **Cursor Movement:** Once the V gesture is detected, map the gesture to cursor movement. Determine the direction and speed of the cursor movement based on the hand's movement. For instance, if the hand moves upwards, move the cursor upwards on the screen. The mouse cursor is used to navigate the computer window. If the index finger with tip Id1 and the middle finger with tip Id 2 are both up, the mouse cursor is made to move around the computer window.

iv. **Update Cursor Position:** Use appropriate libraries or APIs to update the cursor position on the screen accordingly. This could involve simulating mouse events or directly manipulating the cursor position in the operating system.

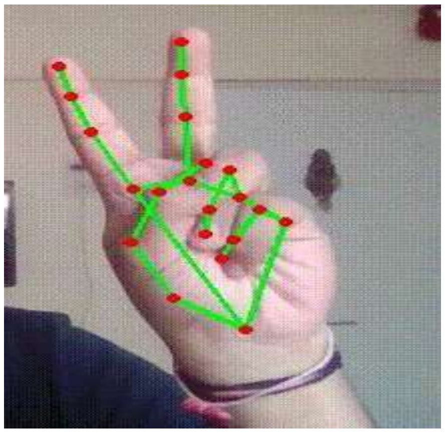


Fig 2.4 Move Cursor

3. <u>Left Click</u>: This functionality is mapped to the movement of the index finger. Performing a left click by bending the index finger involves the following steps:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Finger Detection:** Identify the index finger within the tracked hand region. This can be done by analyzing the hand's shape, finger positions, or using hand skeleton tracking algorithms.

iii. **Bend Detection:** Determine if the index finger is bent or in a curved position. This can be achieved by calculating the angle between the finger segments or by comparing the position of the fingertip relative to the hand. In the first frame, the index finger with Tip ID 1 and the middle finger with Tip ID 2 must be up, followed by Index Finger (Tip Id 1) down and Middle finger (Tip id2) up in the second frame, with both producing an angle greater than 33.5°.

iv. **Left Click Action:** Once the bent index finger is detected, map this gesture to a left-click action. Simulate a left-click event, emulating the action of pressing the left mouse button.

v. **Trigger Left Click:** Send a command or event to the system or application to perform the left-click action. This can be done by using appropriate libraries or APIs to simulate the left-click event.

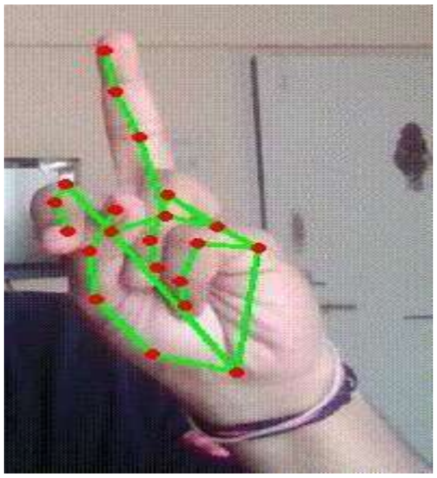


Fig 2.5 Left Click

3. <u>Right Click</u>: This functionality is mapped to the movement of the middle finger. Performing a right click involves the following steps:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Finger Detection:** Identify the middle finger within the tracked hand region. This can be done by analyzing the hand's shape, finger positions, or using hand skeleton tracking algorithms.

iii. **Bend Detection:** Determine if the middle finger is bent or in a curved position. Calculate the angle between the finger segments or compare the position of the fingertip relative to the hand to detect the bending.

iv. **Right Click Action:** Once the bent middle finger is detected, map this gesture to a right-click action. Simulate a right-click event, emulating the action of pressing the right mouse button.

v. **Trigger Right Click:** Send a command or event to the system or application to perform the right-click action. This can be done by using appropriate libraries or APIs to simulate the right-click event.

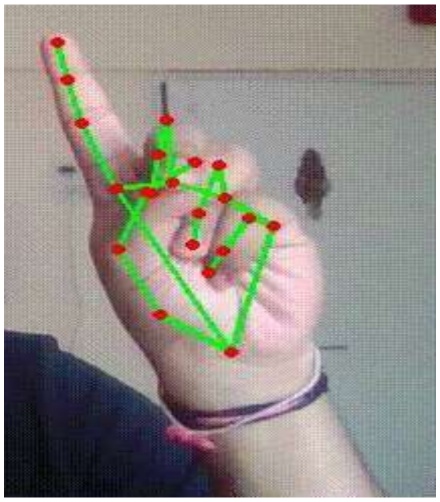


Fig 2.6 Right Click

4. <u>Double Click</u>: This functionality is mapped to the movement of both the index and the middle finger. Performing a double click involves the following steps:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Finger Detection:** Identify the index and middle fingers within the tracked hand region. Analyze the hand's shape, finger positions, or use hand skeleton tracking algorithms to detect and track the desired fingers.

iii. **Bend Detection:** Determine if both the index and middle fingers are bent or in a curved position. Calculate the angle between the finger segments or compare the position of the fingertips relative to the hand to detect the bending.

iv. **Double Click Action:** Once both fingers are detected as bent, map this gesture to a double-click action. Simulate a double-click event, emulating the action of quickly pressing and releasing the left mouse button twice.

v. **Trigger Double Click:** Send a command or event to the system or application to perform the double-click action. Use appropriate libraries or APIs to simulate the double-click event.

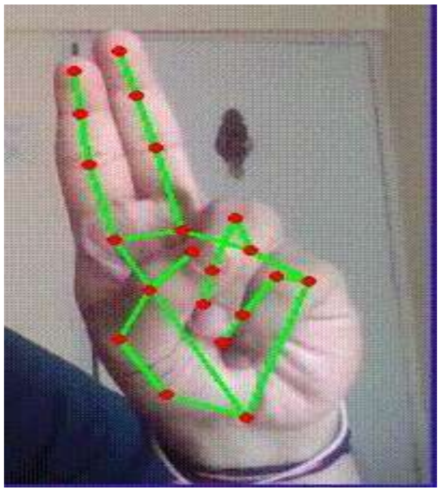


Fig 2.7 Double Click

5. Scrolling: Performing scrolling by making a pinch with the index finger and thumb and moving up and down as the scrolling action involves the following steps:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Finger Detection:** Identify the index finger and thumb within the tracked hand region. Analyse the hand's shape, finger positions, or use hand skeleton tracking algorithms to detect and track the desired fingers.

iii. **Pinch Gesture Detection:** Determine if the index finger and thumb are brought close together to form a pinch gesture. This can be achieved by analyzing the distance between the fingertips or by comparing the relative positions of the fingers.

iv. **Scroll Action:** Once the pinch gesture is detected, map this gesture to a scrolling action. Track the movement of the pinched fingers in an upward or downward direction to determine the scrolling speed and direction.

v. **Perform Scroll:** Send a command or event to the system or application to perform the scrolling action. Use appropriate libraries or APIs to simulate the scroll event and adjust the scroll amount based on the finger movement.

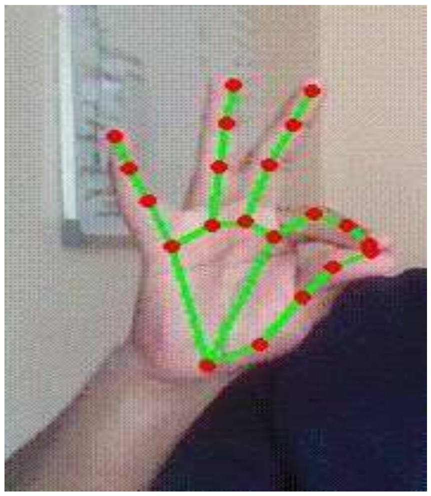


Fig 2.8 Scrolling

7. <u>Drag and Drop</u>: This functionality is mapped to the fist movement and dropping the items by opening the fist and showing the palm. Performing drag and drop action involves the following steps:

i. **Object Selection:** Implement a mechanism to select items for drag and drop. This could involve using a pointing device or another gesture to indicate the selection of the items.

ii. **Fist Gesture Detection:** Track the user's hand and detect the fist gesture. Utilize computer vision techniques to recognize the closed fist pose.

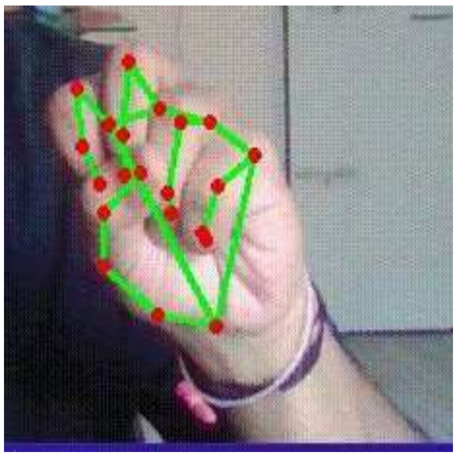
iii. **Item Grabbing:** Once the fist gesture is detected, associate it with grabbing the selected items for drag and drop. The closed fist represents holding onto the items.

iv. **Dragging Gesture:** Monitor the movement of the fist gesture to determine the desired drag direction. Capture the changes in hand position to calculate the movement vector for dragging.

v. **Item Movement:** Translate the movement of the fist gesture into the movement of the selected items on the screen. Update the position of the items relative to the movement of the fist.

vi. **Fist Opening Detection:** Detect the moment when the fist is opened to show the complete hand.

vii. **Drop Action:** Once the complete hand is detected, associate it with the drop action. Release the held items at the current location.



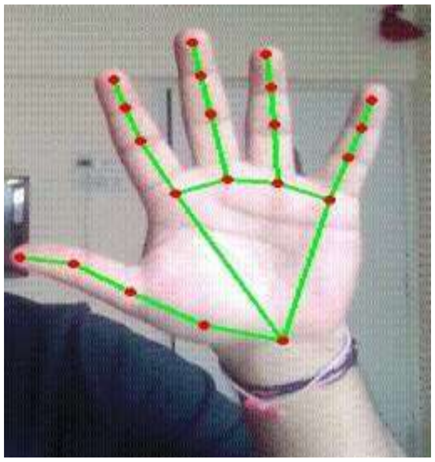


Fig 2.9 Drag and Drop

8. <u>Multiple Item Selection</u>: This functionality can be implemented by making a fist and moving over the items to be selected, then opening the fist to show the complete palm and terminate the action in a gesture recognition virtual mouse project. The required steps are as follows:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Fist Gesture Detection:** Track the user's hand and detect the closed fist gesture. Utilize computer vision techniques to recognize the closed fist pose.

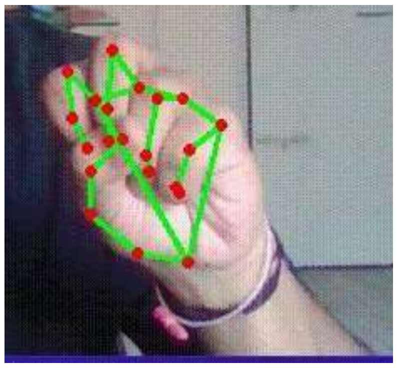
iii. **Item Selection:** When a closed fist is detected, associate it with the selection mode. As the user moves their fist over the items of interest, track the movement and identify the items that fall within the region covered by the fist.

iv. **Visual Feedback:** Provide visual feedback to indicate the items that are being selected. This could be highlighting the selected items or changing their appearance.

v. **Fist Opening Detection:** Detect the moment when the fist is opened to show the complete palm. Recognize the transition from the closed fist gesture to an open hand.

vi. **Multiple Item Selection:** When the complete palm is detected, consider all the items that were covered by the fist as selected. Store the selected items or mark them for further actions.

vii. **Termination of Action:** Once the complete palm is detected, terminate the selection action. The system can respond by deselecting the items, clearing the selection, or allowing the user to proceed with the selected items for further interactions.



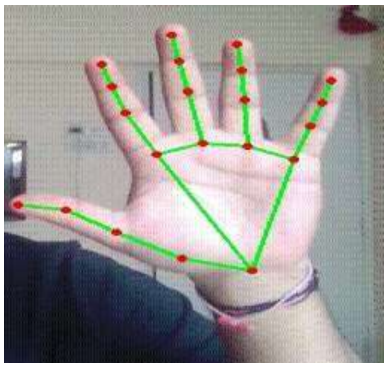


Fig 2.10 Multiple Item Selection

9. <u>Volume Control:</u> To control volume of the system, make a pinch gesture of the index finger and thumb and move it up to increase the volume while lower it down to decrease it. It can be implemented through the following steps:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Pinch Gesture Detection:** Track the positions of the index finger and thumb and detect when they are brought close together to form a pinch gesture. Analyse the distance between the fingertips or compare the relative positions of the fingers to detect the pinch gesture.

iii. **Vertical Movement Detection:** Monitor the vertical movement of the pinch gesture. Calculate the change in position of the pinch gesture as it moves up or down.

iv. **Volume Control Mapping:** Map the vertical movement of the pinch gesture to volume control actions. For example, moving the pinch gesture upwards can be associated with increasing the volume, while moving it downwards can be associated with decreasing the volume.

v. **Adjust Volume:** Based on the detected pinch gesture and its vertical movement, send commands to the system or application to adjust the volume accordingly. Utilize appropriate libraries or APIs to simulate volume control events, such as increasing or decreasing the volume.

vi. **Continuous Tracking:** Continuously track the pinch gesture and its movement to provide real-time volume control. Update the volume based on the current position of the pinch gesture.

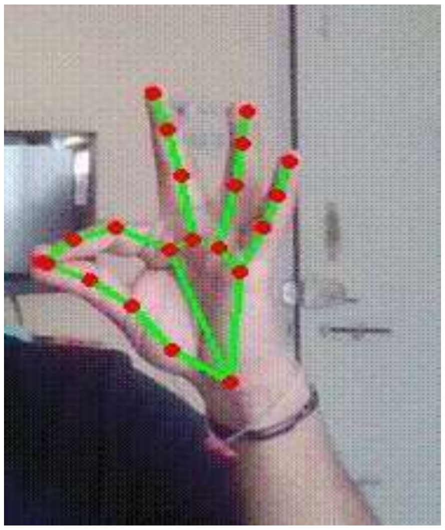


Fig 2.11 Volume Control

10. <u>Brightness Control</u>: To control brightness, make a pinch gesture of the index finger and thumb and slide it to the right to increase the brightness while slide it to the left to decrease. Following are the steps for the implementation of the functionality:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Pinch Gesture Detection:** Track the positions of the index finger and thumb and detect when they are brought close together to form a pinch gesture. Analyze the distance between the fingertips or compare the relative positions of the fingers to detect the pinch gesture.

iii. **Horizontal Sliding Detection:** Monitor the horizontal movement of the pinch gesture. Calculate the change in position of the pinch gesture as it slides to the right or left.

iv. **Brightness Control Mapping:** Map the horizontal movement of the pinch gesture to brightness control actions. For example, sliding the pinch gesture to the right can be associated with increasing the brightness, while sliding it to the left can be associated with decreasing the brightness.

v. **Adjust Brightness:** Based on the detected pinch gesture and its horizontal movement, send commands to the system or application to adjust the brightness accordingly. Utilize appropriate libraries or APIs to simulate brightness control events, such as increasing or decreasing the brightness.

vi. **Continuous Tracking:** Continuously track the pinch gesture and its movement to provide real-time brightness control. Update the brightness based on the current position of the pinch gesture.

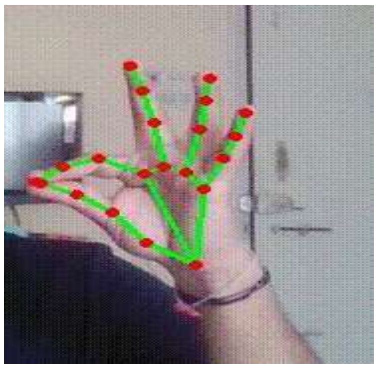


Fig 2.12 Brightness Control

*End User Use Case (For Voice Assistant)***2**

Use Case: **Perform specified functions by recognizing user's voice:**

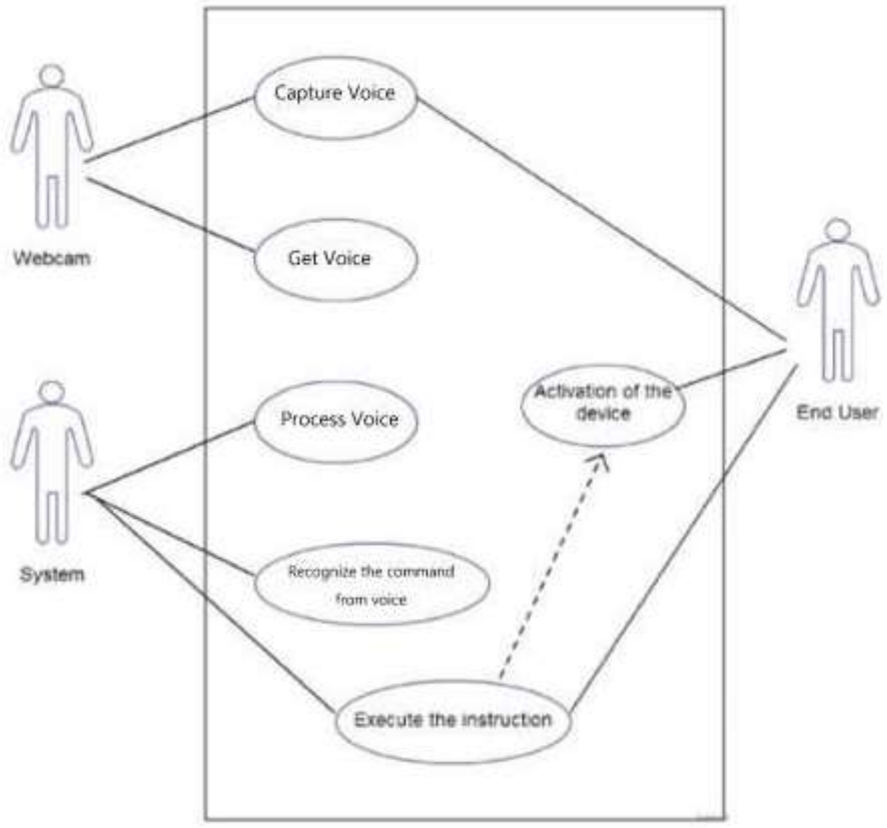


Fig 2.13 Use Case Diagram : Voice Assistant

Brief Description

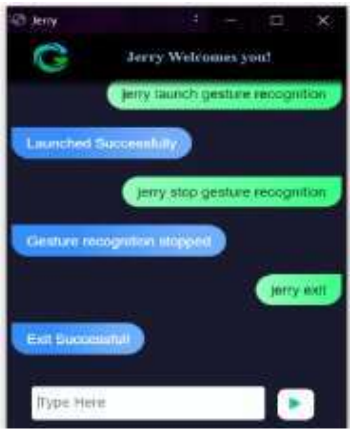
The user speaks the commands in the microphone attached to the computer system, the microphone captures the voice of the user and performs the appropriate function as per the recognized voice command.

Initial Step-By-Step Description

1. The user plans which action is to executed with the mouse.
2. The user then gives the voice command as per his/her needs.
3. The microphone tracks the voice command given by the user.
4. The system interprets the voice command given by the user to the computer.
5. The system maps the recognized voice command with the specific action for that specific voice command.
6. The system replies back by performing the desired operation as per the voice command given by the user to the computer system.

<u>Voice Assistant Features</u>

1. <u>Launch/Stop Gesture Recognition</u>: This will simply invoke the Gesture Controller to control perform the virtual mouse functions.



jerry launch gesture recognition

Launched Successfully

jerry stop gesture recognition

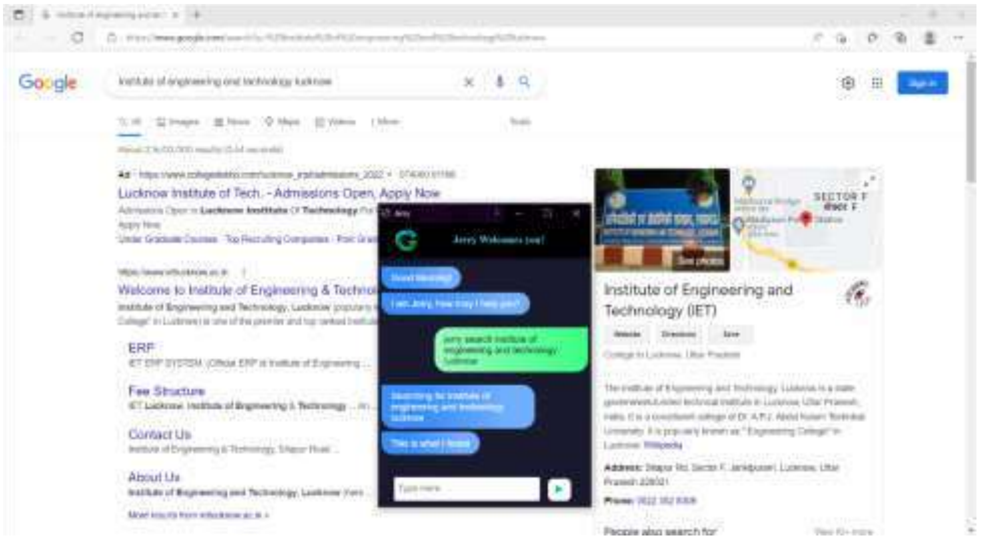
Gesture recognition stopped

jerry exit

Exit Successfull

Fig 2.14 Launch and Stop Gesture Recognition

2. <u>Google Search</u>: User can do google searches by just invoking the assistant and searching for the thing they want in the specified voice format.


Jerry Welcomes you!

Good Morning!

I am Jerry, How may I help you?

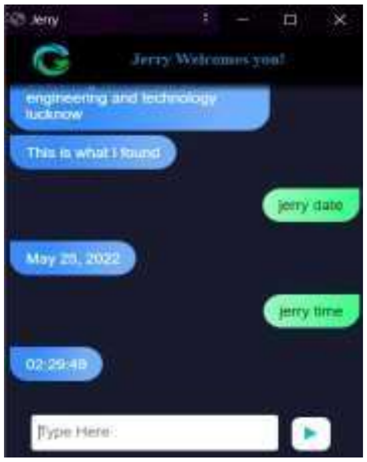
jerry search institute of engineering and technology lucknow

Searching for institute of engineering and technology lucknow

This is what I found

Fig 2.15 Google Search

3. **Current Date and Time**: The assistant is able to tell the current Date and Time to the user upon asking in the specified voice format.


Jerry Welcomes you!

engineering and technology lucknow

This is what I found

jerry date

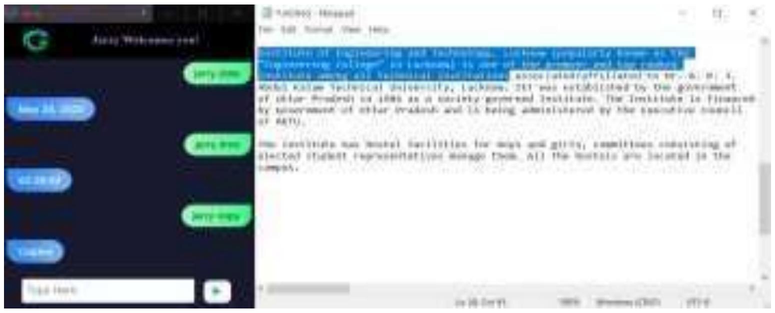
May 25, 2022

jerry time

02:29:48

Fig 2.16 Current Date and Time

4. **Copy and Paste**: The assistant is capable to perform the copy and paste action. This is initiated by manually selecting the content to be copied and then giving the copy command in the specified voice format. Later, placing the cursor to the desired location where the content needs to be pasted by giving the paste command in the specified voice format.



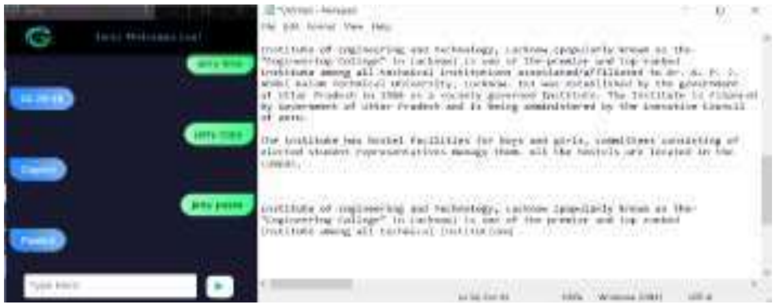


Fig 2.17 Copy and Paste

5. **<u>Find location on Google Maps</u>**: The user is able to search and find a location on Google Maps by just mentioning the desired location in the specified command

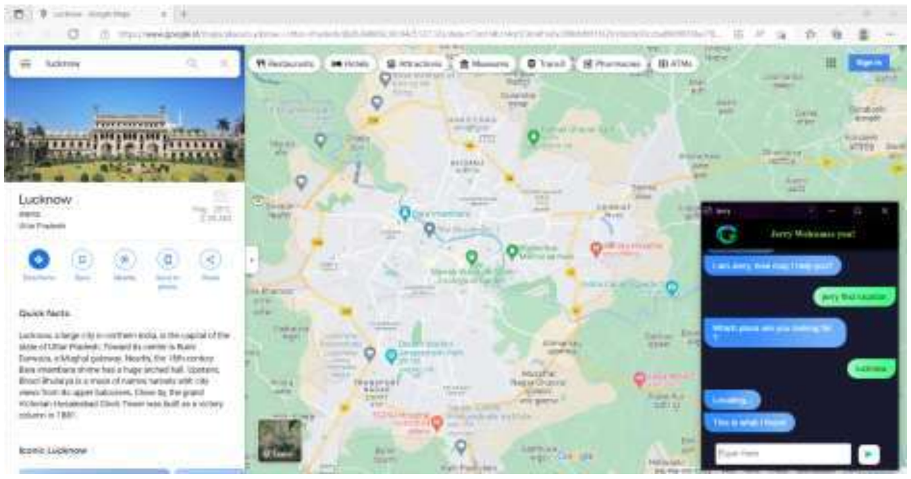
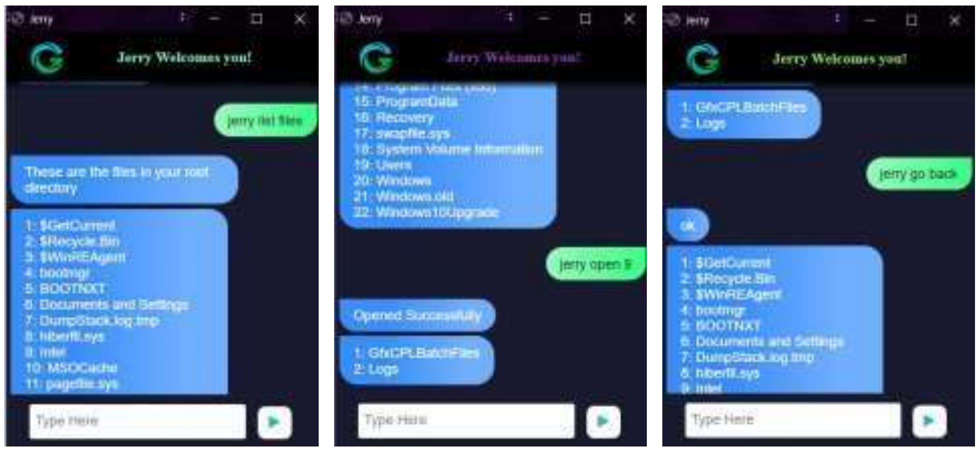


Fig 2.18 Find location on Google Maps

6. <u>File Navigation</u>: Navigating through the system files and directories is possible with the help of this assistant. It is able to move in and out of a directory and also can tell the location of the present working directory.



jerry list files

These are the files in your root directory

1: $GetCurrent
2: $Recycle.Bin
3: $WinREAgent
4: bootmgr
5: BOOTNXT
6: Documents and Settings
7: DumpStack.log.tmp
8: hiberfil.sys
9: Intel
10: MSOCache
11: pagefile.sys

15: ProgramData
16: Recovery
17: swapfile.sys
18: System Volume Information
19: Users
20: Windows
21: Windows.old
22: Windows10Upgrade

jerry open 9

Opened Successfully

1: GfxCPLBatchFiles
2: Logs

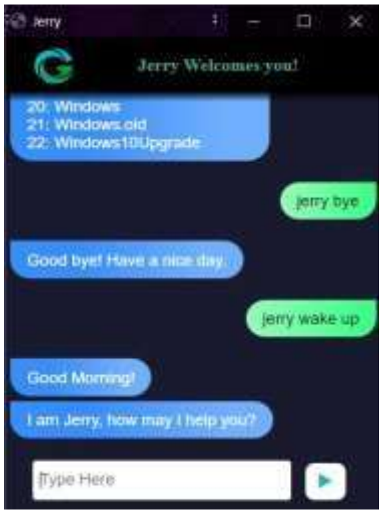
jerry go back

ok

1: $GetCurrent
2: $Recycle.Bin
3: $WinREAgent
4: bootmgr
5: BOOTNXT
6: Documents and Settings
7: DumpStack.log.tmp
8: hiberfil.sys
9: Intel

Fig 2.19 File Navigation

7. <u>Sleep/Wakeup</u>: You can make the voice assistant sleep by simply giving the *SLEEP* command and later wake it up whenever required by simply giving the *WAKEUP* command.



20: Windows
21: Windows.old
22: Windows10Upgrade

jerry bye

Good bye! Have a nice day.

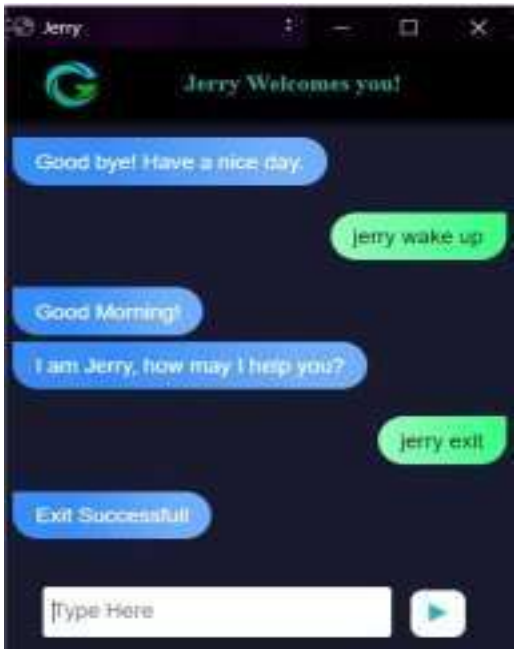
jerry wake up

Good Morning!

I am Jerry, how may I help you?

Fig 2.20 Sleep and Wakeup

8. <u>Exit</u>: The voice assistant can be closed by simply giving the *EXIT* command.


Jerry

Jerry Welcomes you!

Good bye! Have a nice day.

jerry wake up

Good Morning!

I am Jerry, how may I help you?

jerry exit

Exit Successfull

Fig 2.21 Exit

2.6.1 User Characteristics

After developing this application user should be able to access their system through Motion Tracker Application. User are able to use this application to maintain eye distance between their device and himself. Also, during streaming he/she can able to access their system without any movement of their body. User should able to easily install in their computer. User should able to use feature of Drag & Drop.
This application will also enable the users to make use of voice commands in order to perform certain operations in the computer system. User just need to speak up the specific voice commands as per his/her needs and this voice command is mapped to certain specific actions which are performed after begin interpreted by the computer system.

2.7 Non-Functional Requirements

The system database will be on a server with high-speed Internet capability. The physical machine to be used will be determined by the company but should have minimum of windows XP installed.. The software developed here assumes the use of a tool such as PyCharm or any other IDE for smooth functioning. The speed of the end user computer system owner connection will depend on the hardware used rather than characteristics of this system.

Some non-functional requirements include – **Speed, Security, Portability, Compatibility, Reliability, Usability.**

CHAPTER 3
SOFTWARE DESIGN DOCUMENT

3.1. Deployment Design

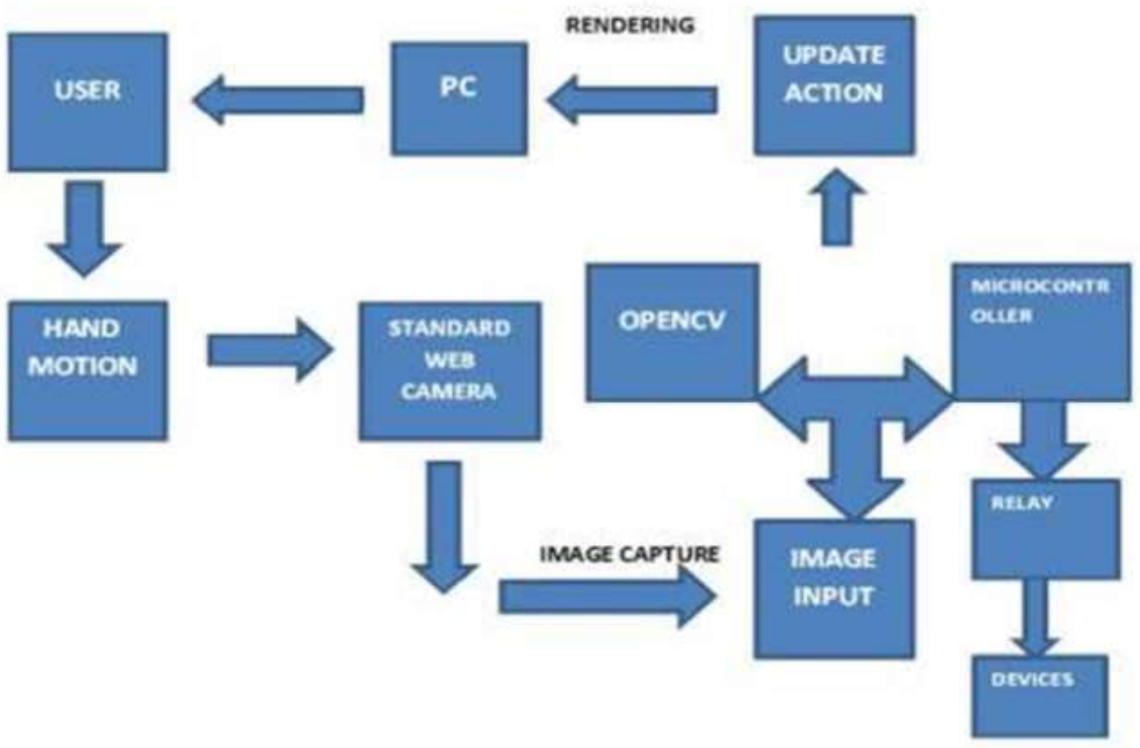


Fig 3.1 Deployment Diagram

The deployment diagram explains the series of steps and processes being followed and executed for an operation to take place. This involves processes being depicted as a flow diagram which are performed in accordance to the specified rules.

A user accesses the application in any local computer system. After accessing the application, the user shows any gestures in front of the system. This activity is being tracked by the webcam. The webcam interprets this movement activity through a series of steps to map it to the appropriate action to be performed against the shown gesture. When the action is mapped to the gesture, the correct mouse action is being executed in the computer system.

3.2. Architectural Design

3.2.1 System Architecture

The architecture of the proposed system is as displayed in the figure below. The major components of the architecture are as follows: Display area, input area, gesture capture, voice capture, gesture and voice recognition, gesture/voice input, and the end user.

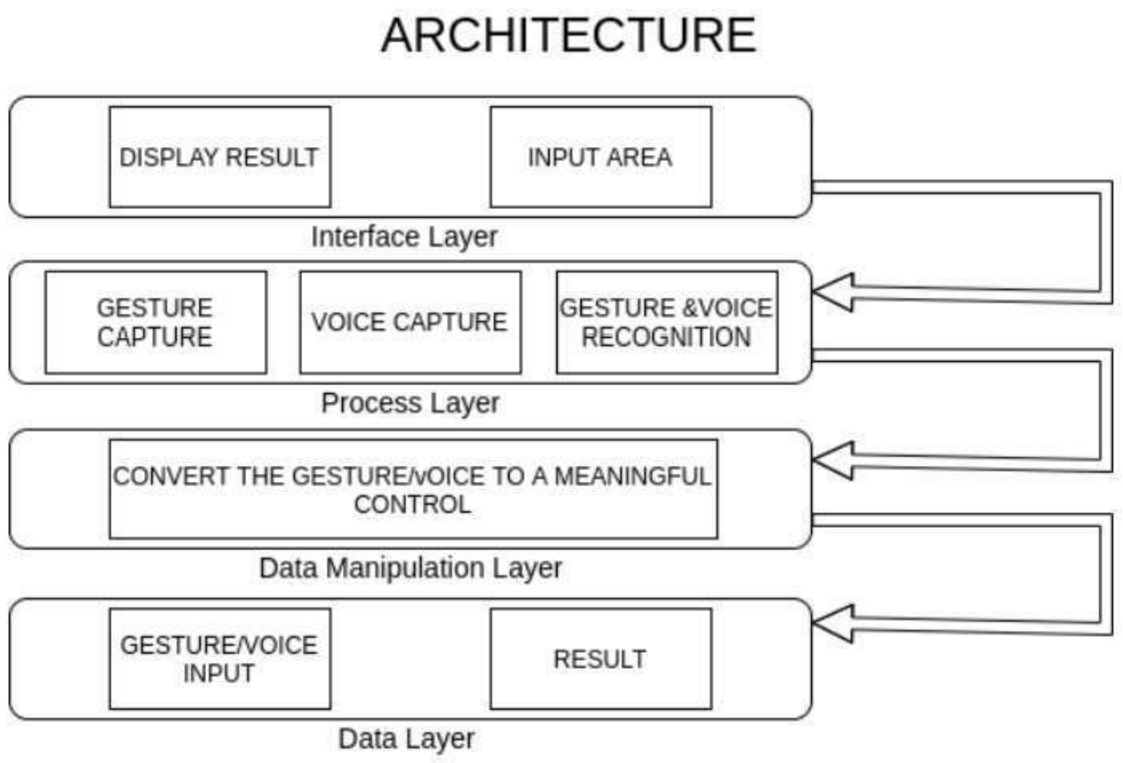


Fig 3.2 Architecture Diagram

3.2.2 Model View Controller Architecture for GUI

The following figure describes the process of MVC architecture. Here there are three components which are model, view and controller. The model component is responsible of the operation and management of the data. The view component is responsible for the presentation of the data to the user. The controller component is responsible for user interaction.

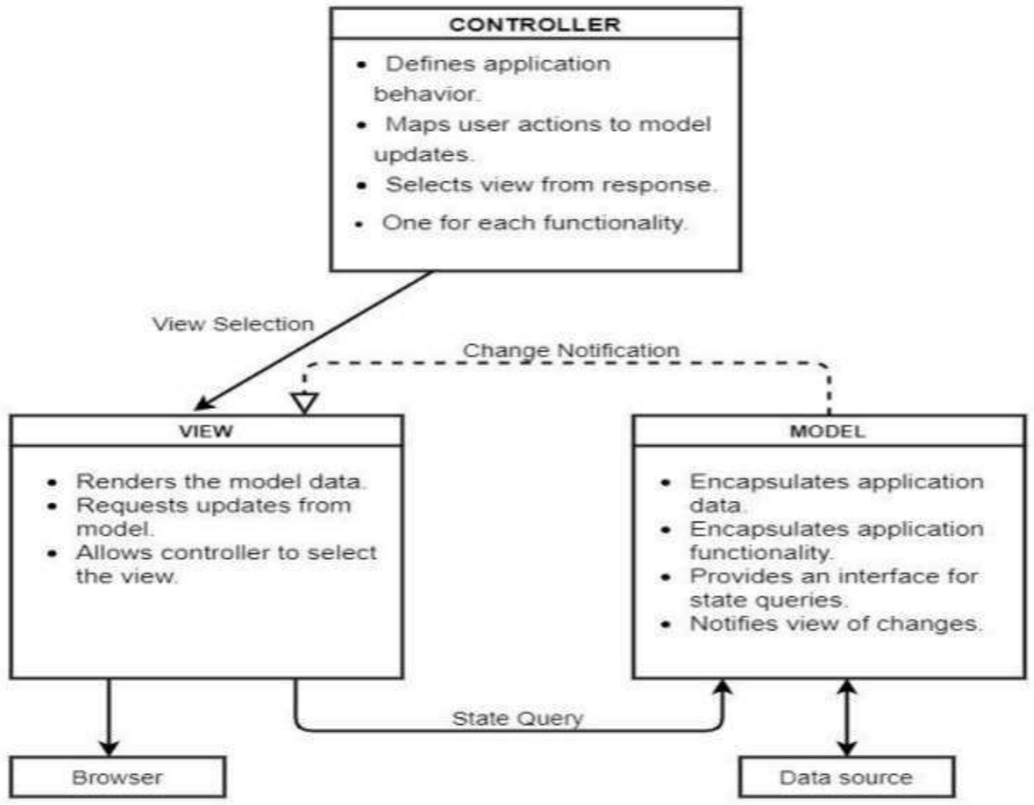

CONTROLLER
Defines application behavior.
Maps user actions to model updates.
Selects view from response.
One for each functionality.
View Selection
Change Notification
VIEW
Renders the model data.
Requests updates from model.
Allows controller to select the view.
MODEL
Encapsulates application data.
Encapsulates application functionality.
Provides an interface for state queries.
Notifies view of changes.
State Query
Browser
Data source

Fig 3.3 Model View Controller

3.2.3 Activity Diagram

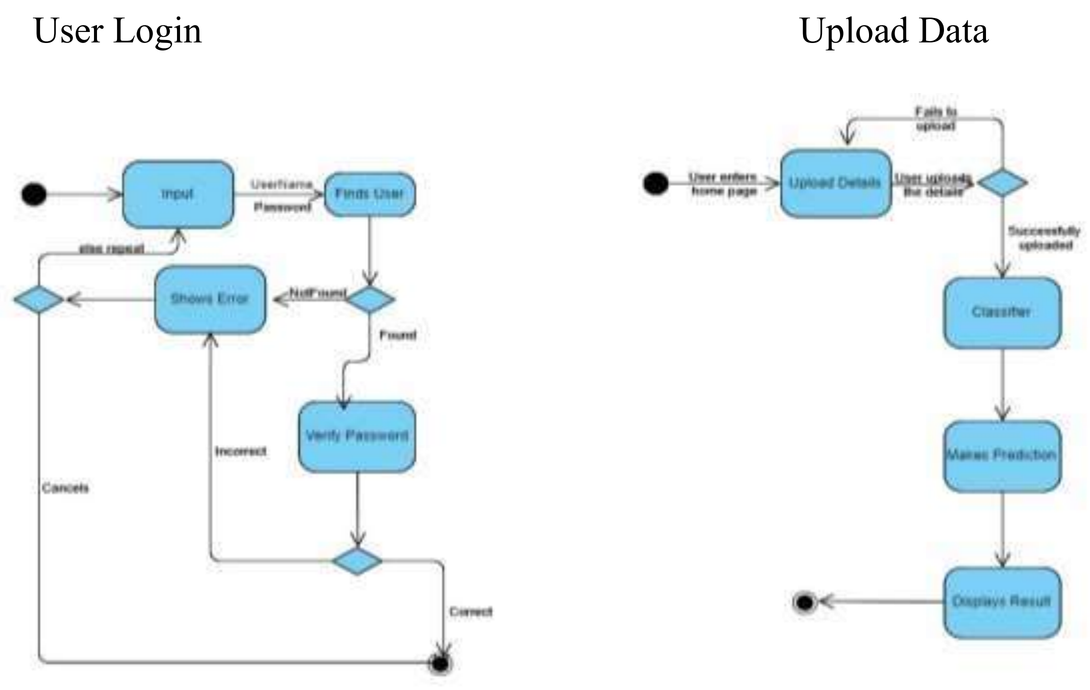


Fig 3.4 Activity Diagram

The activity diagram for login system and upload image is shown in the above figures respectively. The diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. For example, if a user wants to login, the input in this activity will be the username and the password and once the user enters its credentials in the application the database will find the user and verify its password. Once verified the activity will stop their but, in the case, where the user is not found in the database then an error will be shown.

3.3. Data Flow Diagram:

The entire working or the flow of the data can be divided into three groups for better understanding. They are- 1. DFD-L0 2. DFD-L1 3. DFD-L2

This is the initial idea for the flow of the data. The user will show gestures in front of the webcam. The gesture tracked by the webcam is being recognized by the OpenCV library and the corresponding mouse operation being interpreted from it. After the interpretation the desired mouse action is being executed on the computer system.

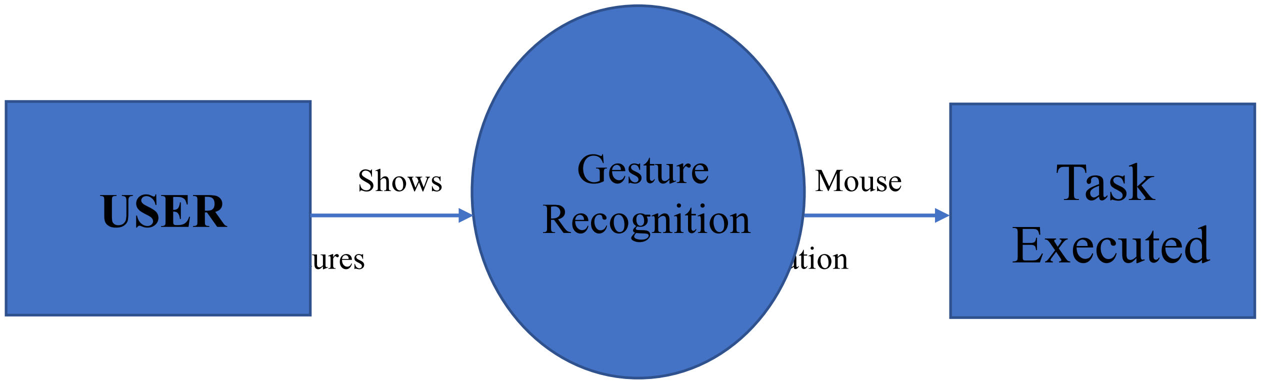


Fig 3.5 Data Flow Diagram

3.3.1. First Level Data Flow Diagram

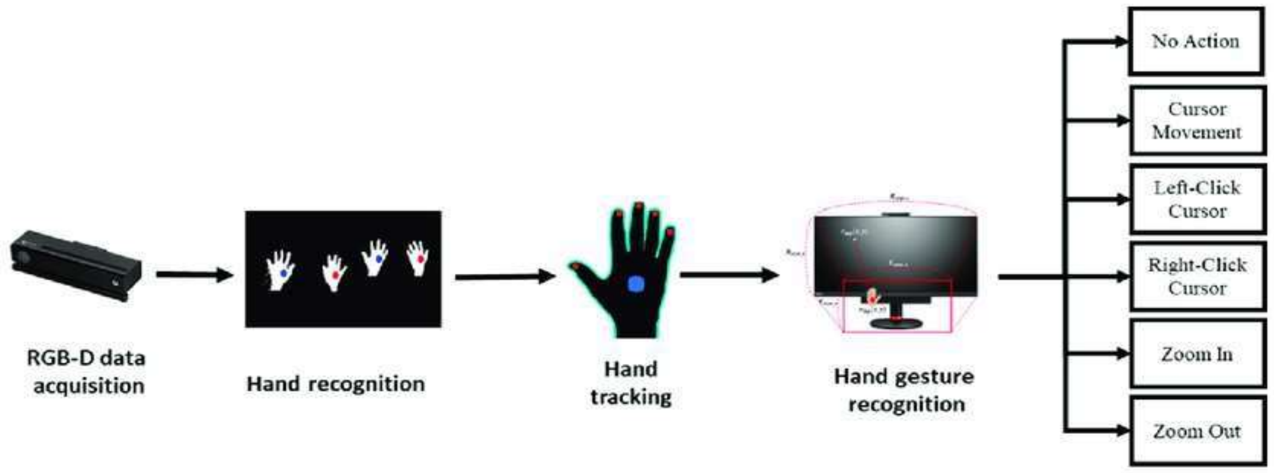


Fig 3.6 Level-1 DFD

Data acquisition is being done is Level-1 DFD. Hand movement is being tracked in the image captured by the webcam. The hand movements are recognized by the various structural points on the fingers of the hands. The way of movement of fingers differentiates the various gestures mapped to the mouse actions. The recognized hand gesture gets mapped to the appropriate mouse action which is then being executed on the computer system.

3.3.2. Second Level Data Flow Diagram

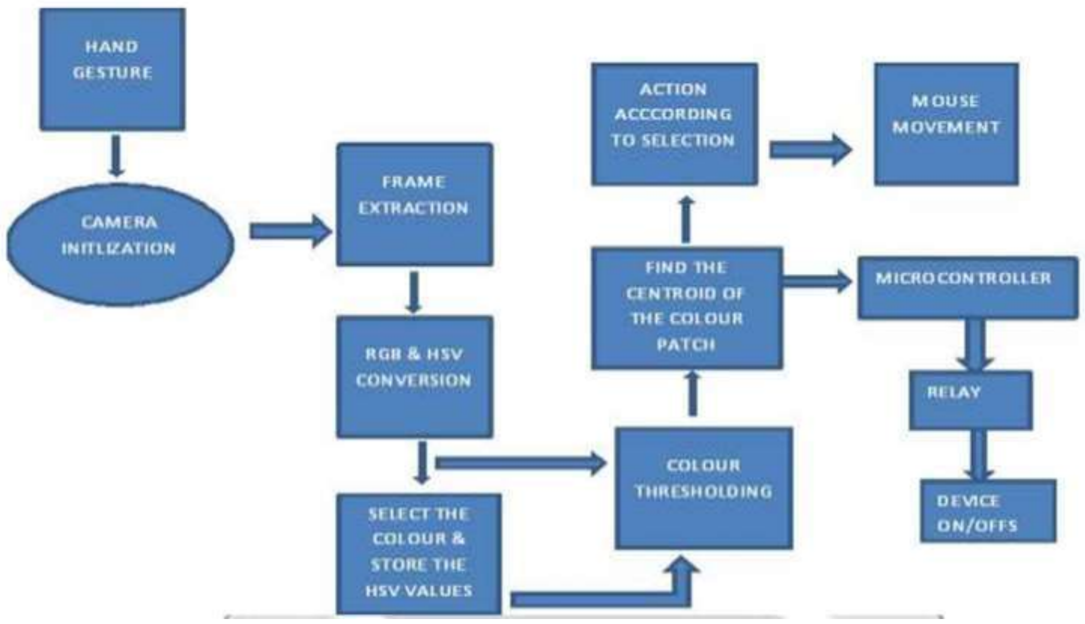


Fig 3.7 Level-2 DFD

The basic requirement of any system are its sensors. So in this system too a sensor (webcam) is used to interact with the environment. Its functionality is to record the live video which is taken as input through hand gesture by the user. It processes this input and then sends it to OpenCV . Here, in CV a code is generated which is used to convert the live video into frames of images.

This activity is technically termed as “slicing of video”. Then this frame of images are processed for color recognition process, where only images with colors mentioned in the code are kept. Rest images are discarded by the system.

The speed at which output images are displayed is equal to the speed at which slicing of the video is done. This output display seems like a movie running where input is the physical world while output are only those colors which are present on the fingertips of the user. These colors signify mouse cursor on the screen. As these colors move similarly mouse cursor on the screen moves. And in this way output is produced and the mouse click is replaced by finger clicks, gesture recognition and image processing.

3.4 Sequence Diagram

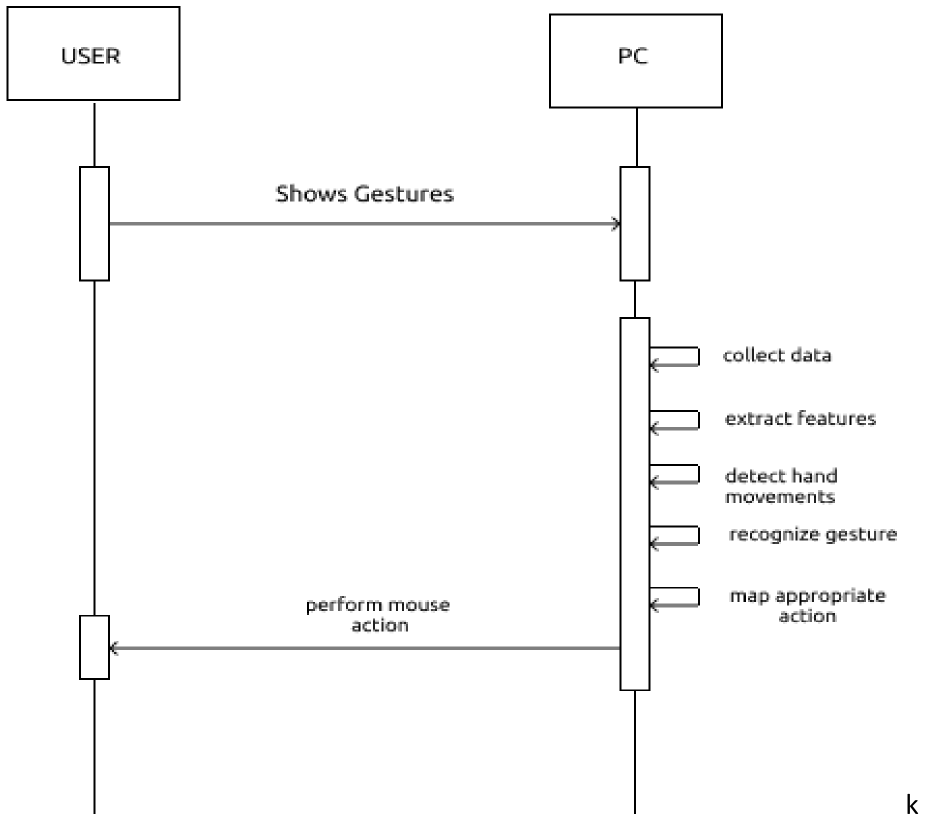


Fig 3.8 Sequence Diagram

A sequence diagram is a type of interaction diagram in software engineering that shows the flow of messages or interactions between different objects or components within a system. It depicts the chronological sequence of events and the order in which the interactions occur.

Sequence diagrams are primarily used to visualize and describe the dynamic behavior of a system or a specific scenario. They provide a clear representation of how objects or components collaborate to accomplish a specific task or fulfill a particular requirement. Sequence diagrams are commonly used during the design phase of software development to model the interactions between different parts of a system.

In a sequence diagram, the participants or objects involved in the interactions are represented as vertical lines called lifelines. The lifelines are ordered from top to bottom to indicate the sequence of events. Arrows are used to represent messages sent between the participants, indicating the flow of communication.

Additionally, sequence diagrams can include various elements such as activation and deactivation markers to indicate when a participant is active or inactive during an interaction, as well as loops, conditionals, and parallel execution to depict more complex scenarios.

Overall, sequence diagrams provide a visual representation of the behavior of a system, helping developers, designers, and stakeholders to understand and communicate the flow of interactions and the overall system behavior.

3.5 ER Diagram

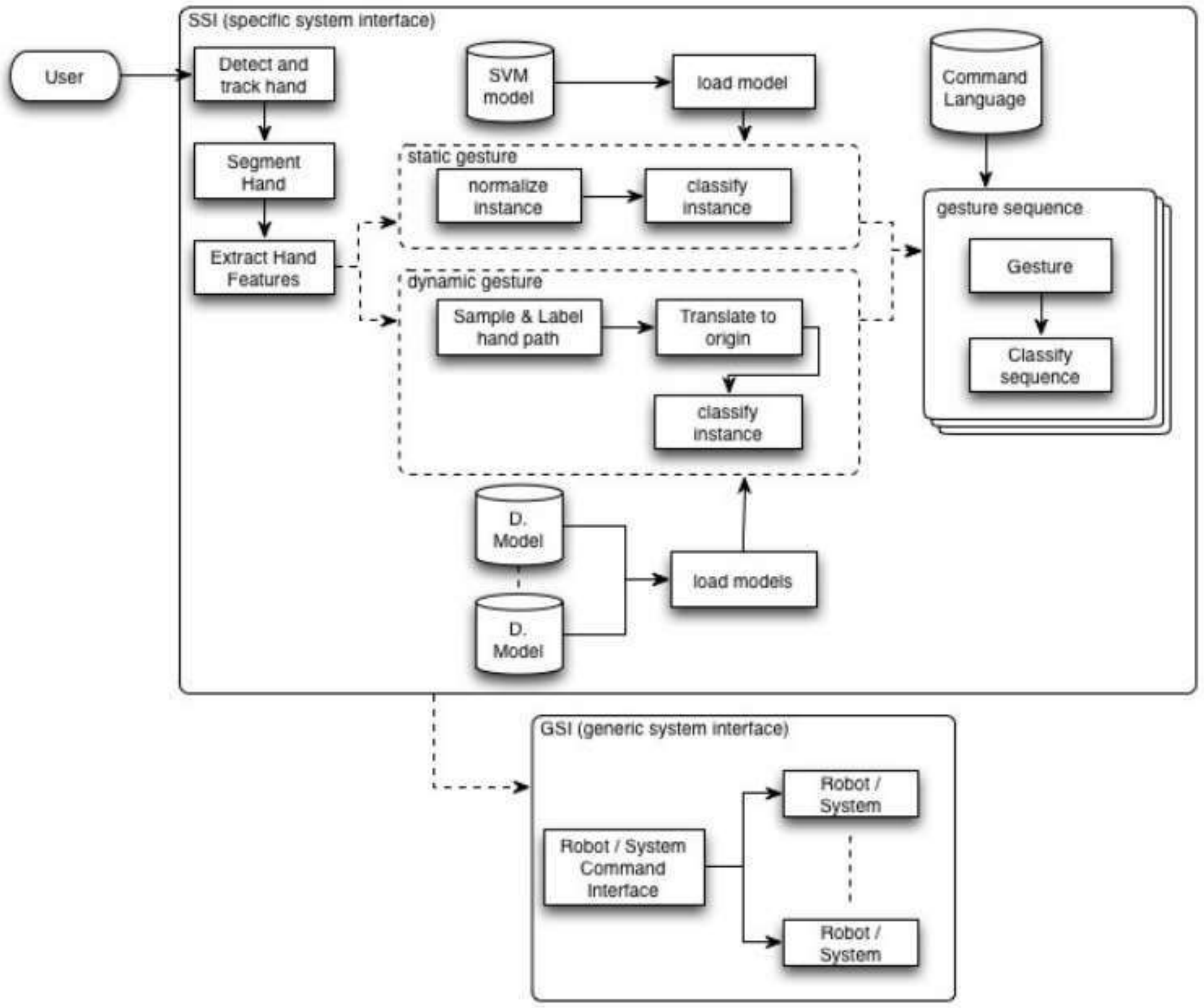


Fig 3.9 ER Diagram

An Entity-Relationship (ER) diagram is a visual representation that depicts the relationships between entities (objects) in a system or database. It is a popular tool used in software engineering and database design to model the structure and organization of data.

In an ER diagram, entities are represented as rectangles, and relationships between entities are depicted as lines connecting them. The key components of an ER diagram are:

1. Entities: An entity represents a real-world object or concept, such as a person, place, thing, or event, which is relevant to the system being modeled. Each entity is depicted as a rectangle, and its name is written inside the rectangle.

2. Attributes: Attributes are the properties or characteristics of an entity. They provide additional information about the entity. Attributes are typically listed inside the entity rectangle.

3. Relationships: Relationships show how entities are connected or associated with each other. They are represented by lines connecting the participating entities. The relationship line usually has a label to describe the nature of the relationship, such as "works for," "owns," or "is a part of." The relationship can also have additional properties or attributes associated with it.

4. Cardinality: Cardinality describes the number of instances of one entity that can be associated with the instances of another entity in a relationship. It indicates the minimum and maximum number of occurrences. Common cardinality notations include "1" (exactly one occurrence), "0..1" (zero or one occurrence), "0..*" (zero or more occurrences), and "1..*" (one or more occurrences).

ER diagrams help in understanding the relationships and dependencies between entities in a system or database. They provide a clear and concise representation of the data model, aiding in the design, development, and documentation of a database system. ER diagrams are commonly used in the early stages of software development to capture the requirements and structure of the data, facilitating effective communication between stakeholders, designers, and developers.

3.6 Flowchart

A flowchart is a graphical representation of a process, algorithm, or workflow. It uses a set of symbols and arrows to illustrate the sequence of steps and decisions involved in completing a task or achieving a specific outcome. Flowcharts are widely used in various fields, including software development, business process management, system analysis, and problem-solving.

Flowcharts typically consist of the following elements:

1. **Start/End Symbol:** This symbol represents the beginning or end of a flowchart and is usually depicted as an oval or rounded rectangle.
2. **Process Symbol:** This symbol represents a specific action or process that needs to be performed. It is usually depicted as a rectangle with rounded corners.
3. **Decision Symbol:** This symbol represents a decision point where a choice needs to be made. It is typically depicted as a diamond-shaped symbol with arrows coming in and going out, indicating the different paths based on the decision outcome.
4. **Input/Output Symbol:** This symbol represents input or output operations, such as reading data or displaying information. It is usually depicted as a parallelogram.
5. **Connector Symbol:** This symbol is used to join different parts of the flowchart together when the process is too complex to fit on a single page. It is depicted as a small, labeled circle or a larger circle with a number inside to indicate the connection point.

6. **Arrows:** Arrows are used to show the flow or sequence of steps in the process. They indicate the direction of the flow and connect the symbols to form a logical flow of the process.

Flowcharts provide a visual representation of a process, making it easier to understand and analyze complex procedures. They help identify bottlenecks, potential issues, or areas for improvement in a process. Flowcharts can be used for documentation, communication, process analysis, and as a tool for problem-solving and decision-making.

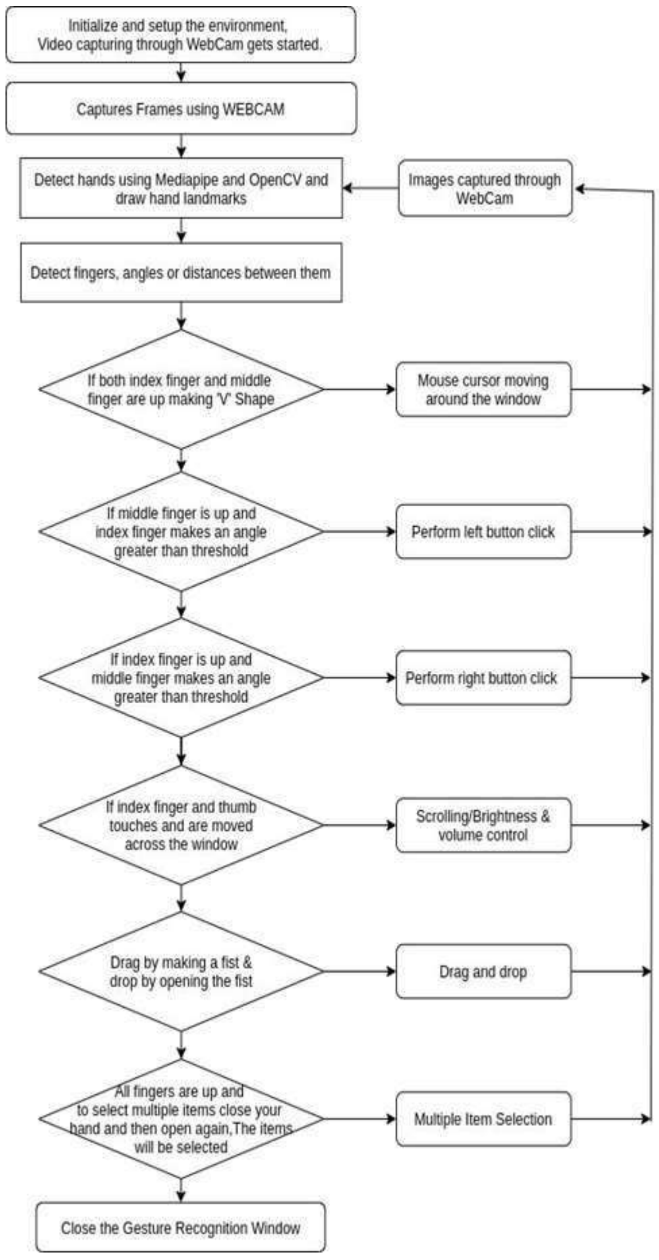

Initialize and setup the environment, Video capturing through WebCam gets started.
Captures Frames using WEBCAM
Detect hands using Mediapipe and OpenCV and draw hand landmarks
Images captured through WebCam
Detect fingers, angles or distances between them
If both index finger and middle finger are up making 'V' Shape
Mouse cursor moving around the window
If middle finger is up and index finger makes an angle greater than threshold
Perform left button click
If index finger is up and middle finger makes an angle greater than threshold
Perform right button click
If index finger and thumb touches and are moved across the window
Scrolling/Brightness & volume control
Drag by making a fist & drop by opening the fist
Drag and drop
All fingers are up and to select multiple items close your hand and then open again,The items will be selected
Multiple Item Selection
Close the Gesture Recognition Window

Fig 3.10 Gesture Recognition Flowchart

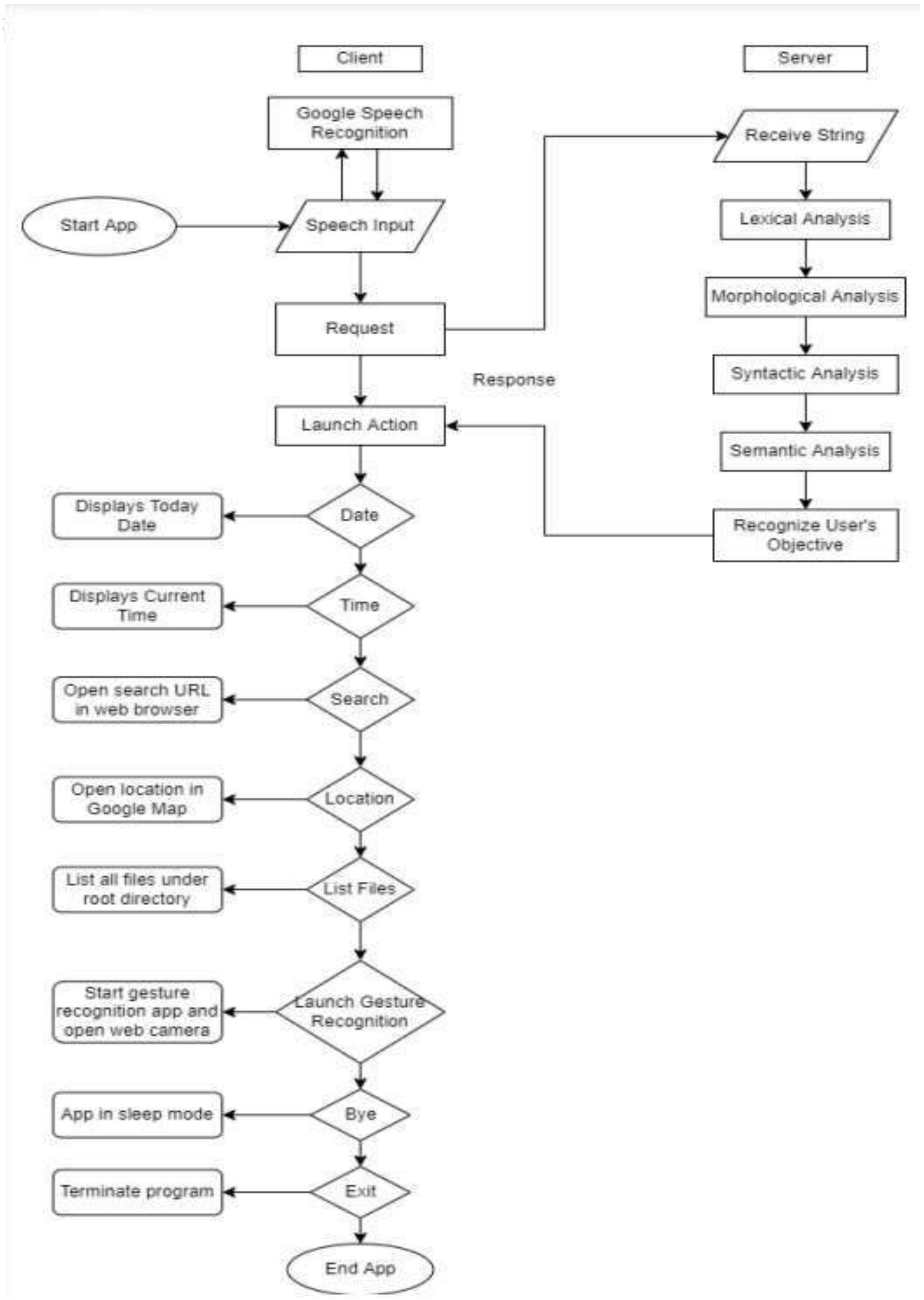


Fig 3.11 Voice Assistant Flowchart

3.7 Technological Description

OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source library that includes several hundreds of computer vision algorithms. The document describes the so-called OpenCV 2.x API, which is essentially a C++ API, as opposed to the C-based OpenCV 1.x API (C API is deprecated and not tested with "C" compiler since OpenCV 2.4 releases)

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

1. **Core functionality** (**core**) - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.

2. **Image Processing** (**imgproc**) - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.

3. **Video Analysis** (**video**) - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.

4. **Camera Calibration and 3D Reconstruction** (**calib3d**) - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.

5. **2D Features Framework** (**features2d**) - salient feature detectors, descriptors, and descriptor matchers.

6. **Object Detection** (**objdetect**) - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).

7. **High-level GUI** (**highgui**) - an easy-to-use interface to simple UI capabilities.

8. **Video I/O** (**videoio**) - an easy-to-use interface to video capturing and video codecs.

9. ... some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

ML Pipeline (MediaPipe)

MediaPipe offers ready-to-use yet customizable Python solutions as a prebuilt Python package. The MediaPipe Python framework grants direct access to the core components of the MediaPipe C++ framework such as Timestamp, Packet, and CalculatorGraph, whereas the ready-to-use Python solutions hide the technical details of the framework and simply return the readable model inference results back to the callers.

MediaPipe framework sits on top of the pybind11 library. The C++ core framework is exposed in Python via a C++/Python language binding. The content below assumes that the reader already has a basic understanding of the MediaPipe C++ framework. Otherwise, you can find useful information in Framework Concepts.

CHAPTER 4
IMPLEMENTATION AND SYSTEM INTEGRATION

4.1 Pseudocode for Implementation of Gesture Recognition:

4.1.1 Segment of code initializing the environment

```
Import necessary libraries and modules

Set pyautogui's failsafe mode to False

Initialize mediapipe's drawing utilities and hands modules

Create an enumeration class for gesture encodings (Gest) with various
hand gestures mapped to binary numbers

Create an enumeration class for multi-handedness labels (HLabel) with
labels for minor and major hands

Initialize the main program or function

    Initialize the webcam or video source for capturing frames

    Create a while loop to continuously read frames from the video source

        Read a frame from the video source

        Convert the frame to grayscale

        Use mediapipe's hands module to detect hands in the frame

        If hands are detected

            For each detected hand

                Use mediapipe's drawing utilities to draw landmarks on
the hand in the frame

                Determine the gesture based on the landmarks and assign
it to a variable

                Perform actions based on the detected gesture

        Display the annotated frame
        Check for any keyboard interrupt or exit condition

    Release the video source

Define a function to perform actions based on the detected gesture

Initialize the program or call the main function
```

4.1.2 Class which converts Mediapipe landmarks to recognizable gestures

```
Create a class named HandRecog

    Define the constructor method __init__ that takes a hand_label parameter

        Initialize the attributes:
            - finger: an integer representing the gesture corresponding to Enum 'Gest', set to 0
            - ori_gesture: an integer representing the gesture corresponding to Enum 'Gest', set to Gest.PALM
            - prev_gesture: an integer representing the gesture corresponding to Enum 'Gest', set to Gest.PALM
            - frame_count: an integer representing the total number of frames since 'ori_gesture' was updated, set to 0
            - hand_result: an object to store landmarks obtained from mediapipe, set to None
            - hand_label: an integer representing multi-handedness corresponding to Enum 'HLabel'

    Define a method named update_hand_result that takes a hand_result parameter

        // Code Here

    Define a method named get_signed_dist that takes a point parameter

       // Code Here

    Define a method named get_dist that takes a point parameter

       // Code Here

    Define a method named get_dz that takes a point parameter
```

```
        // Code Here

    Define a method named set_finger_state

        // Code Here
    Define a method named get_gesture

        // Code Here
```

4.1.3 Class controlling the execution of commands

The pseudocode for the **Controller** class:

```
class Controller:
    Initialize attributes for controlling gestures

    Define 'getpinchylv' function to calculate the vertical distance between pinch start and current hand position
    Define 'getpinchxlv' function to calculate the horizontal distance between pinch start and current hand position

    Define 'changesystembrightness' function to adjust system brightness based on pinch gesture
    Define 'changesystemvolume' function to adjust system volume based on pinch gesture
    Define 'scrollVertical' function to scroll vertically on the screen
    Define 'scrollHorizontal' function to scroll horizontally on the screen

    Define 'get_position' function to get the current hand position and stabilize cursor motion
    Define 'pinch_control_init' function to initialize attributes for pinch gesture
    Define 'pinch_control' function to control pinch gesture based on frame count and direction
    Define 'handle_controls' function to implement functionality for each gesture
```

4.1.4 Pseudocode for the functions and methods within the Controller class:

```
# Define 'getpinchylv' function
def getpinchylv(hand_result):
    Calculate the vertical distance between pinch start and current hand
position
    Return the distance

# Define 'getpinchxlv' function
def getpinchxlv(hand_result):
    Calculate the horizontal distance between pinch start and current
hand position
    Return the distance

# Define 'changesystembrightness' function
def changesystembrightness():
    Get the current system brightness level
    Adjust the brightness level based on 'Controller.pinchlv'
    Set the new brightness level using appropriate system control
    Return

# Define 'changesystemvolume' function
def changesystemvolume():
    Get the current system volume level
    Adjust the volume level based on 'Controller.pinchlv'
    Set the new volume level using appropriate system control
    Return

# Define 'scrollVertical' function
def scrollVertical():
    Scroll vertically on the screen based on 'Controller.pinchlv'
    Return

# Define 'scrollHorizontal' function
def scrollHorizontal():
    Scroll horizontally on the screen based on 'Controller.pinchlv'
    Return

# Define 'get_position' function
def get_position(hand_result):
    Get the current hand position from 'hand_result'
    Calculate the corresponding cursor position based on the screen size
    Apply dampening to stabilize the cursor motion
    Return the cursor position

# Define 'pinch_control_init' function
def pinch_control_init(hand_result):
    Set 'Controller.pinchstartxcoord' and 'Controller.pinchstartycoord' 
based on 'hand_result'
    Reset 'Controller.pinchlv', 'Controller.prevpinchlv', and 
'Controller.framecount'
    Return

# Define 'pinch_control' function
def pinch_control(hand_result, controlHorizontal, controlVertical):
    Determine the direction of pinch gesture (horizontal or vertical) 
based on distances
    Update 'Controller.pinchlv' and 'Controller.prevpinchlv' based on 
distances
    Update 'Controller.framecount'
    If the frame count reaches the threshold:
        Execute 'controlHorizontal' or 'controlVertical' based on the 
direction
    Return

# Define 'handle_controls' function
def handle_controls(gesture, hand_result):
    Get the current cursor position from 'hand_result'
    If the gesture is not FIST and 'Controller.grabflag' is True:
        Release the mouse button
        Set 'Controller.grabflag' to False

    If the gesture is not PINCH_MAJOR and 'Controller.pinchmajorflag' is 
True:
        Set 'Controller.pinchmajor
If the gesture is not PINCH_MINOR and 'Controller.pinchminorflag' is 
True:
        Set 'Controller.pinchminorflag' to False

    If the gesture is V_GEST:
        Set 'Controller.flag' to True
        Move the cursor to the current position

    If the gesture is FIST:
        If 'Controller.grabflag' is False:
            Set 'Controller.grabflag' to True
            Press the left mouse button
        Move the cursor to the current position

    If the gesture is MID and 'Controller.flag' is True:
        Perform a left click
        Set 'Controller.flag' to False

    If the gesture is INDEX and 'Controller.flag' is True:
        Perform a right click
        Set 'Controller.flag' to False

    If the gesture is TWO_FINGER_CLOSED and 'Controller.flag' is True:
        Perform a double click
        Set 'Controller.flag' to False

    If the gesture is PINCH_MINOR:
        If 'Controller.pinchminorflag' is False:
            Call 'pinch_control_init' to initialize attributes
            Set 'Controller.pinchminorflag' to True
        Call 'pinch_control' with appropriate callback functions

    If the gesture is PINCH_MAJOR:
        If 'Controller.pinchmajorflag' is False:
            Call 'pinch_control_init' to initialize attributes
            Set 'Controller.pinchmajorflag' to True
        Call 'pinch_control' with appropriate callback functions
```

4.1.4 Main driver code of Gesture Controller

```
class GestureController:
    gc_mode = 0
    cap = None
    CAM_HEIGHT = None
    CAM_WIDTH = None
    hr_major = None
    hr_minor = None
    dom_hand = True

    def __init__(self):
        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)

    def classify_hands(results):
        left, right = None, None

        try:
            handedness_dict                                          = 
MessageToDict(results.multi_handedness[0])
            if handedness_dict['classification'][0]['label'] == 'Right':
                right = results.multi_hand_landmarks[0]
            else:
                left = results.multi_hand_landmarks[0]
        except:
            pass

        try:
            handedness_dict                                          = 
MessageToDict(results.multi_handedness[1])
            if handedness_dict['classification'][0]['label'] == 'Right':
                right = results.multi_hand_landmarks[1]
            else:
                left = results.multi_hand_landmarks[1]
        except:
            pass

        if GestureController.dom_hand == True:
            GestureController.hr_major = right
            GestureController.hr_minor = left
        else:
            GestureController.hr_major = left
            GestureController.hr_minor = right

    def start(self):
        handmajor = HandRecog(HLabel.MAJOR)
        handminor = HandRecog(HLabel.MINOR)

        with                            mp_hands.Hands(max_num_hands=2, 
min_detection_confidence=0.5, min_tracking_confidence=0.5) as hands:
            while         GestureController.cap.isOpened()         and 
GestureController.gc_mode:
                success, image = GestureController.cap.read()

                if not success:
                    print("Ignoring empty camera frame.")
                    continue
                image      =      cv2.cvtColor(cv2.flip(image,      1),
cv2.COLOR_BGR2RGB)
                image.flags.writeable = False
                results = hands.process(image)
                image.flags.writeable = True
                image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

                if results.multi_hand_landmarks:
                    GestureController.classify_hands(results)

handmajor.update_hand_result(GestureController.hr_major)

handminor.update_hand_result(GestureController.hr_minor)
                    handmajor.set_finger_state()
                    handminor.set_finger_state()
                    gest_name = handminor.get_gesture()

                    if gest_name == Gest.PINCH_MINOR:
                        Controller.handle_controls(gest_name,
handminor.hand_result)
                    else:
                        gest_name = handmajor.get_gesture()
                        Controller.handle_controls(gest_name,
handmajor.hand_result)

                    for hand_landmarks in results.multi_hand_landmarks:
                        mp_drawing.draw_landmarks(image,
hand_landmarks, mp_hands.HAND_CONNECTIONS)
                else:
                    Controller.prev_hand = None

                cv2.imshow('Gesture Controller', image)
                if cv2.waitKey(5) & 0xFF == 13:
                    break

        GestureController.cap.release()
        cv2.destroyAllWindows()

gc1 = GestureController()
gc1.start()
```

4.2 Pseudocode for Implementation of Voice Assistant:

```
# Object Initialization

# Variables Initialisation


# Functions that the voice assistant will perform upon listening and
interpretating the voice of the user as per the given command

def reply(audio):
    # Display response in the application

def wish():
    # Greet the user based on the time of the day


def record_audio():
    # Record audio from the microphone and convert it to text


def respond(voice_data):
    # Process user's voice input and provide appropriate response


# Driver Code
t1 = Thread(target=app.ChatBot.start)
t1.start()

# Lock main thread until Chatbot has started
while not app.ChatBot.started:
    time.sleep(0.5)

wish()
voice_data = None
while True:
    if app.ChatBot.isUserInput():
        # Take input from GUI
        voice_data = app.ChatBot.popUserInput()
    else:
        # Take input from Voice
        voice_data = record_audio()
```

```
# Process voice_data
if 'proton' in voice_data:
    try:
        respond(voice_data)
    except SystemExit:
        reply("Exit Successful")
        break
    except:
        print("EXCEPTION raised while closing.")
        break
```

CHATPER 5

TESTCASES AND SNAPSHOTS

5.1 Testcases of Gesture Recognition:

When testing a gesture recognition virtual mouse, it is important to cover a range of scenarios and gestures to ensure its accuracy and reliability. Here are some potential test cases to consider:

1. **Basic Gestures:**

- Single tap/click: Verify that a single tap gesture is correctly recognized as a left-click action.
- Double tap/click: Confirm that a double tap gesture is recognized as a double-click action.
- Long press: Test if a prolonged touch or press gesture is detected as a right-click action.
- Swipe: Check if swiping gestures in different directions (up, down, left, right) are accurately recognized.

2. **Custom Gestures:**

- Custom shape recognition: Define specific custom shapes or patterns and verify that the mouse correctly identifies them as assigned actions. For example, drawing a circle could trigger a specific command.

3. **Combination Gestures:**

- Two-finger swipe: Test the recognition of two-finger swipe gestures in various directions.
- Pinch/spread: Verify if the mouse recognizes pinch/spread gestures correctly, such as zoom in or out actions.

4. **Sensitivity and Precision:**

- Varying speed: Test the mouse's response to gestures performed at different speeds, from slow to fast, to ensure accurate recognition.
- Small movements: Verify if the mouse can detect subtle hand movements accurately.

5. **Environmental Factors:**

- Lighting conditions: Test the mouse's performance under different lighting conditions, such as bright light or dimly lit environments.
- Background interference: Assess how well the mouse filters out background movements or interference, such as other hand movements or objects.

6. **Compatibility:**

- Application integration: Test the mouse's compatibility and gesture recognition within various applications, such as web browsers, productivity software, or gaming environments.

- Operating system compatibility: Verify that the mouse functions correctly across different operating systems (e.g., Windows, macOS, Linux).

7. **Error Handling:**

- False positives/negatives: Intentionally perform gestures similar to predefined actions to verify if the mouse handles false positives (incorrectly recognizing a gesture) or false negatives (not recognizing a correct gesture) appropriately.

8. **User Experience:**

- Ergonomics and comfort: Assess the comfort and usability of the virtual mouse during extended use, considering factors such as hand fatigue and ease of performing gestures.

Mouse Functions Depending on the Hand Gestures and Hand Tip Detection Using Computer Vision.

1. <u>Neutral Gesture</u>: The neutral gesture, in the context of a gesture recognition virtual mouse project, refers to the default or resting position of the hand that is not associated with any specific mouse action or command. It is typically used as a reference point or starting point for recognizing other gestures.

When developing a gesture recognition virtual mouse, you would need to define the neutral gesture as a baseline or reference for detecting and differentiating it from other gestures. The neutral gesture could be identified based on certain criteria such as hand position, palm orientation, or absence of specific hand movements.

By recognizing the neutral gesture, the system can distinguish when the user intends to perform a gesture command rather than simply keeping their hand in a neutral position. This allows for more precise and accurate gesture recognition and control of the virtual mouse.

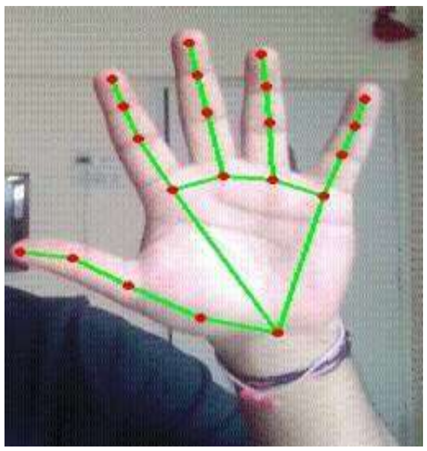


Fig 2.3 Neutral Gesture

2. Move Cursor: The movement of the cursor is mapped to the **V** position formed by the index and middle finger. To move the cursor using a **V** gesture, you can follow these steps:

i. **Hand Tracking:** Utilize a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Gesture Detection:** Implement an algorithm to detect the V gesture. This algorithm can involve analysing the hand's shape, finger positions, or hand movements. For example, you may check if the user's hand forms a V shape by examining the relative positions of the fingertips or the angle between the fingers.

iii. **Cursor Movement:** Once the V gesture is detected, map the gesture to cursor movement. Determine the direction and speed of the cursor movement based on the hand's movement. For instance, if the hand moves upwards, move the cursor upwards on the screen. The mouse cursor is used to navigate the computer window. If the index finger with tip Id1 and the middle finger with tip Id 2 are both up, the mouse cursor is made to move around the computer window.

iv. **Update Cursor Position:** Use appropriate libraries or APIs to update the cursor position on the screen accordingly. This could involve simulating mouse events or directly manipulating the cursor position in the operating system.

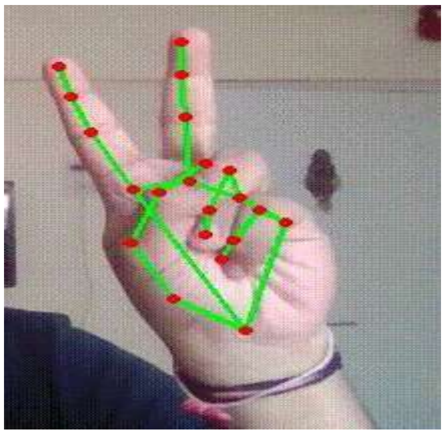


Fig 2.4 Move Cursor

3. Left Click: This functionality is mapped to the movement of the index finger. Performing a left click by bending the index finger involves the following steps:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Finger Detection:** Identify the index finger within the tracked hand region. This can be done by analyzing the hand's shape, finger positions, or using hand skeleton tracking algorithms.

iii. **Bend Detection:** Determine if the index finger is bent or in a curved position. This can be achieved by calculating the angle between the finger segments or by comparing the position of the fingertip relative to the hand. In the first frame, the index finger with Tip ID 1 and the middle finger with Tip ID 2 must be up, followed by Index Finger (Tip Id 1) down and Middle finger (Tip id2) up in the second frame, with both producing an angle greater than 33.5°.

iv. **Left Click Action:** Once the bent index finger is detected, map this gesture to a left-click action. Simulate a left-click event, emulating the action of pressing the left mouse button.

v. **Trigger Left Click:** Send a command or event to the system or application to perform the left-click action. This can be done by using appropriate libraries or APIs to simulate the left-click event.

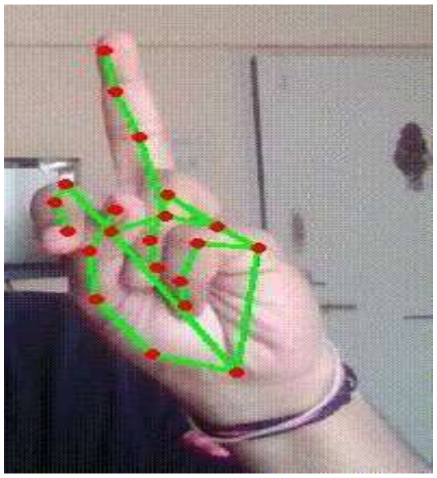


Fig 2.5 Left Click

3. <u>Right Click</u>: This functionality is mapped to the movement of the middle finger. Performing a right click involves the following steps:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Finger Detection:** Identify the middle finger within the tracked hand region. This can be done by analyzing the hand's shape, finger positions, or using hand skeleton tracking algorithms.

iii. **Bend Detection:** Determine if the middle finger is bent or in a curved position. Calculate the angle between the finger segments or compare the position of the fingertip relative to the hand to detect the bending.

iv. **Right Click Action:** Once the bent middle finger is detected, map this gesture to a right-click action. Simulate a right-click event, emulating the action of pressing the right mouse button.

v. **Trigger Right Click:** Send a command or event to the system or application to perform the right-click action. This can be done by using appropriate libraries or APIs to simulate the right-click event.

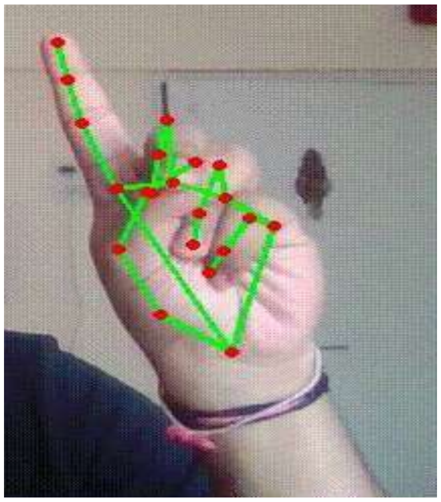


Fig 2.6 Right Click

4. <u>Double Click</u>: This functionality is mapped to the movement of both the index and the middle finger. Performing a double click involves the following steps:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Finger Detection:** Identify the index and middle fingers within the tracked hand region. Analyze the hand's shape, finger positions, or use hand skeleton tracking algorithms to detect and track the desired fingers.

iii. **Bend Detection:** Determine if both the index and middle fingers are bent or in a curved position. Calculate the angle between the finger segments or compare the position of the fingertips relative to the hand to detect the bending.

iv. **Double Click Action:** Once both fingers are detected as bent, map this gesture to a double-click action. Simulate a double-click event, emulating the action of quickly pressing and releasing the left mouse button twice.

v. **Trigger Double Click:** Send a command or event to the system or application to perform the double-click action. Use appropriate libraries or APIs to simulate the double-click event.

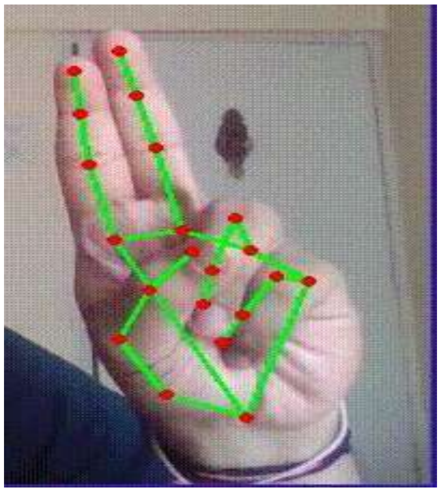


Fig 2.7 Double Click

5. <u>Scrolling</u>: Performing scrolling by making a pinch with the index finger and thumb and moving up and down as the scrolling action involves the following steps:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Finger Detection:** Identify the index finger and thumb within the tracked hand region. Analyse the hand's shape, finger positions, or use hand skeleton tracking algorithms to detect and track the desired fingers.

iii. **Pinch Gesture Detection:** Determine if the index finger and thumb are brought close together to form a pinch gesture. This can be achieved by analyzing the distance between the fingertips or by comparing the relative positions of the fingers.

iv. **Scroll Action:** Once the pinch gesture is detected, map this gesture to a scrolling action. Track the movement of the pinched fingers in an upward or downward direction to determine the scrolling speed and direction.

v. **Perform Scroll:** Send a command or event to the system or application to perform the scrolling action. Use appropriate libraries or APIs to simulate the scroll event and adjust the scroll amount based on the finger movement.

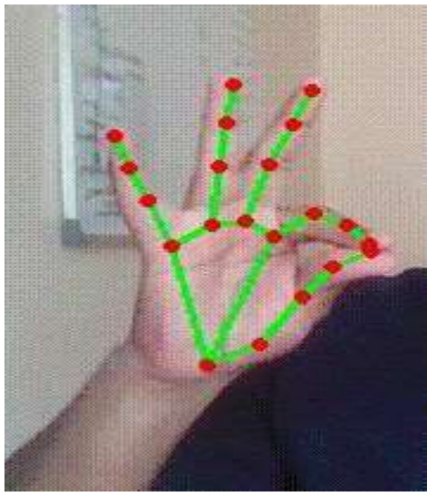


Fig 2.8 Scrolling

7. <u>Drag and Drop</u>: This functionality is mapped to the fist movement and dropping the items by opening the fist and showing the palm. Performing drag and drop action involves the following steps:

i. **Object Selection:** Implement a mechanism to select items for drag and drop. This could involve using a pointing device or another gesture to indicate the selection of the items.

ii. **Fist Gesture Detection:** Track the user's hand and detect the fist gesture. Utilize computer vision techniques to recognize the closed fist pose.

iii. **Item Grabbing:** Once the fist gesture is detected, associate it with grabbing the selected items for drag and drop. The closed fist represents holding onto the items.

iv. **Dragging Gesture:** Monitor the movement of the fist gesture to determine the desired drag direction. Capture the changes in hand position to calculate the movement vector for dragging.

v. **Item Movement:** Translate the movement of the fist gesture into the movement of the selected items on the screen. Update the position of the items relative to the movement of the fist.

vi. **Fist Opening Detection:** Detect the moment when the fist is opened to show the complete hand.

vii. **Drop Action:** Once the complete hand is detected, associate it with the drop action. Release the held items at the current location.

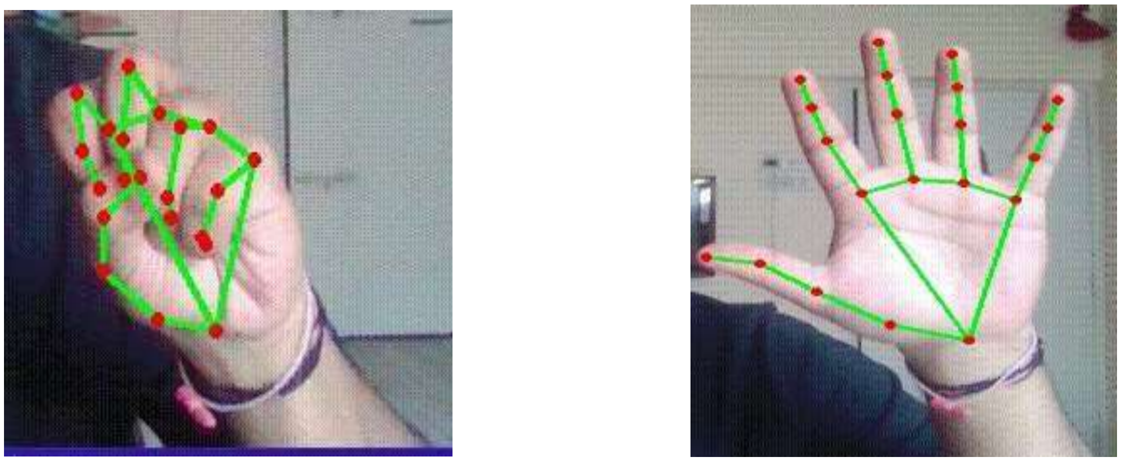


Fig 2.9 Drag and Drop

8. <u>Multiple Item Selection</u>: This functionality can be implemented by making a fist and moving over the items to be selected, then opening the fist to show the complete palm and terminate the action in a gesture recognition virtual mouse project. The required steps are as follows:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Fist Gesture Detection:** Track the user's hand and detect the closed fist gesture. Utilize computer vision techniques to recognize the closed fist pose.

iii. **Item Selection:** When a closed fist is detected, associate it with the selection mode. As the user moves their fist over the items of interest, track the movement and identify the items that fall within the region covered by the fist.

iv. **Visual Feedback:** Provide visual feedback to indicate the items that are being selected. This could be highlighting the selected items or changing their appearance.

v. **Fist Opening Detection:** Detect the moment when the fist is opened to show the complete palm. Recognize the transition from the closed fist gesture to an open hand.

vi. **Multiple Item Selection:** When the complete palm is detected, consider all the items that were covered by the fist as selected. Store the selected items or mark them for further actions.

vii. **Termination of Action:** Once the complete palm is detected, terminate the selection action. The system can respond by deselecting the items, clearing the selection, or allowing the user to proceed with the selected items for further interactions.

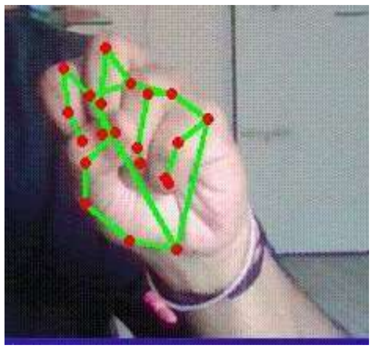
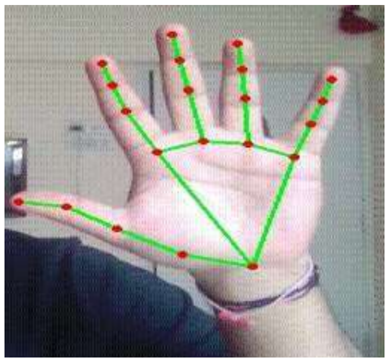



Fig 2.10 Multiple Item Selection

9. <u>Volume Control:</u> To control volume of the system, make a pinch gesture of the index finger and thumb and move it up to increase the volume while lower it down to decrease it. It can be implemented through the following steps:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Pinch Gesture Detection:** Track the positions of the index finger and thumb and detect when they are brought close together to form a pinch gesture. Analyse the distance between the fingertips or compare the relative positions of the fingers to detect the pinch gesture.

iii. **Vertical Movement Detection:** Monitor the vertical movement of the pinch gesture. Calculate the change in position of the pinch gesture as it moves up or down.

iv. **Volume Control Mapping:** Map the vertical movement of the pinch gesture to volume control actions. For example, moving the pinch gesture upwards can be associated with increasing the volume, while moving it downwards can be associated with decreasing the volume.

v. **Adjust Volume:** Based on the detected pinch gesture and its vertical movement, send commands to the system or application to adjust the volume accordingly. Utilize appropriate libraries or APIs to simulate volume control events, such as increasing or decreasing the volume.

vi. **Continuous Tracking:** Continuously track the pinch gesture and its movement to provide real-time volume control. Update the volume based on the current position of the pinch gesture.

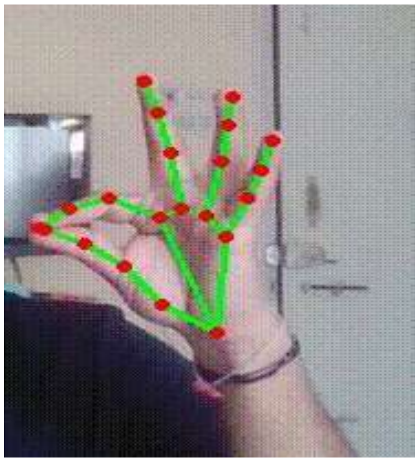


Fig 2.11 Volume Control

10. <u>Brightness Control</u>: To control brightness, make a pinch gesture of the index finger and thumb and slide it to the right to increase the brightness while slide it to the left to decrease. Following are the steps for the implementation of the functionality:

i. **Hand Tracking:** Use a camera or depth sensor to track the user's hand in real-time. Apply computer vision techniques to extract the hand region from the captured video feed.

ii. **Pinch Gesture Detection:** Track the positions of the index finger and thumb and detect when they are brought close together to form a pinch gesture. Analyze the distance between the fingertips or compare the relative positions of the fingers to detect the pinch gesture.

iii. **Horizontal Sliding Detection:** Monitor the horizontal movement of the pinch gesture. Calculate the change in position of the pinch gesture as it slides to the right or left.

iv. **Brightness Control Mapping:** Map the horizontal movement of the pinch gesture to brightness control actions. For example, sliding the pinch gesture to the right can be associated with increasing the brightness, while sliding it to the left can be associated with decreasing the brightness.

v. **Adjust Brightness:** Based on the detected pinch gesture and its horizontal movement, send commands to the system or application to adjust the brightness accordingly. Utilize appropriate libraries or APIs to simulate brightness control events, such as increasing or decreasing the brightness.

vi. **Continuous Tracking:** Continuously track the pinch gesture and its movement to provide real-time brightness control. Update the brightness based on the current position of the pinch gesture.

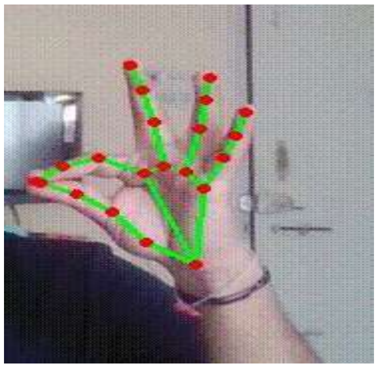
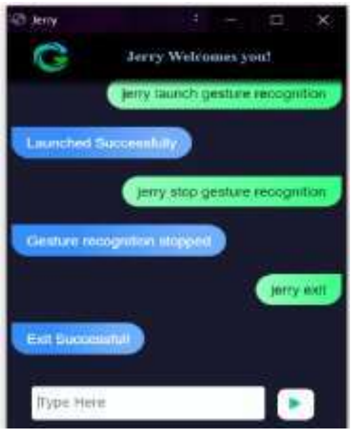


Fig 2.12 Brightness Control

5.2 Testcases of Voice Assistant:

Every action that has to be performed by the voice assistant, the action name has to be preceded by the word ***PROTON***. Upon hearing this word, the assistant get invokes and is ready to hear the command as specified.

1. <u>Launch/Stop Gesture Recognition</u>: This will simply invoke the Gesture Controller to control perform the virtual mouse functions.



jerry launch gesture recognition

Launched Successfully

jerry stop gesture recognition

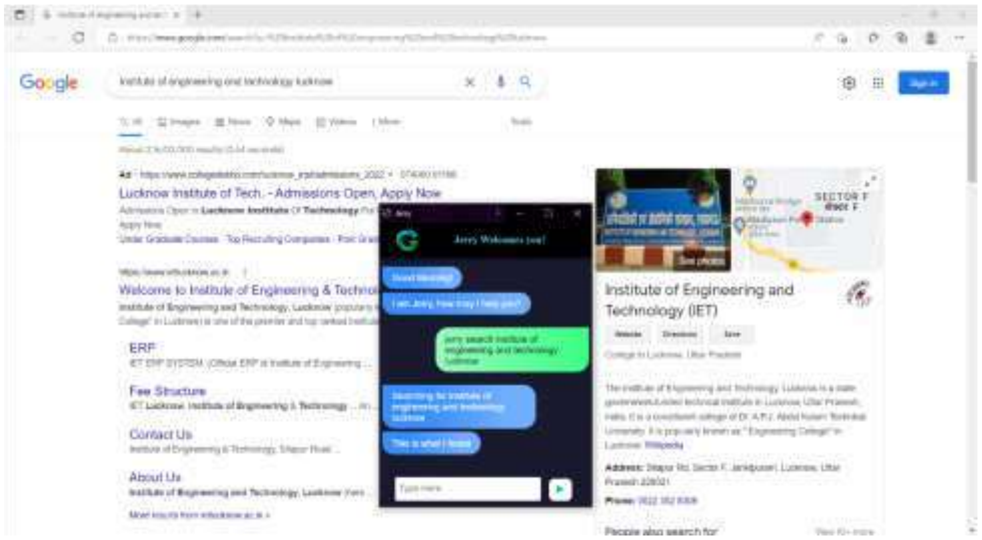
Gesture recognition stopped

jerry exit

Exit Successfull

Fig 2.14 Launch and Stop Gesture Recognition

2. <u>Google Search</u>: User can do google searches by just invoking the assistant and searching for the thing they want in the specified voice format.


Good Morning!

I am Jerry. How may I help you?

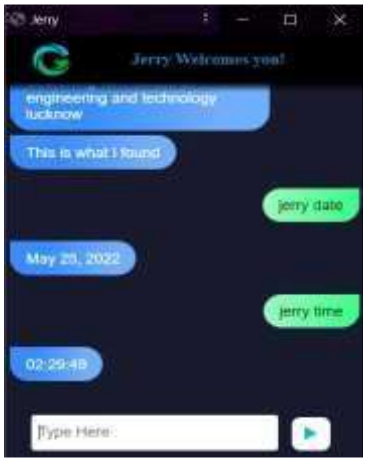
jerry search institute of engineering and technology lucknow

Searching for institute of engineering and technology lucknow

This is what I found

Fig 2.15 Google Search

3. Current Date and Time: The assistant is able to tell the current Date and Time to the user upon asking in the specified voice format.



engineering and technology lucknow

This is what I found

jerry date

May 25, 2022

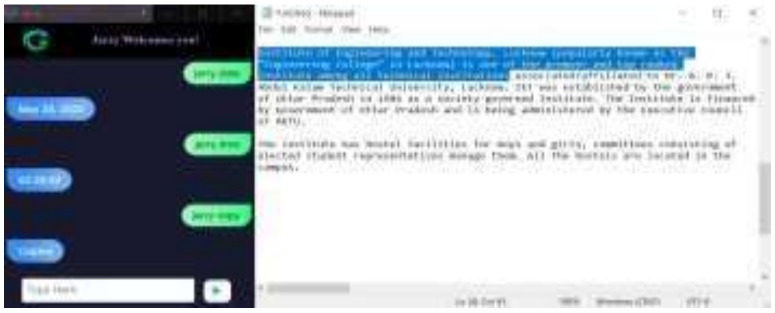
jerry time

02:29:48

Type Here

Fig 2.16 Current Date and Time

4. Copy and Paste: The assistant is capable to perform the copy and paste action. This is initiated by manually selecting the content to be copied and then giving the copy command in the specified voice format. Later, placing the cursor to the desired location where the content needs to be pasted by giving the paste command in the specified voice format.



Jerry Welcomes you!

jerry date

June 28, 2022

jerry time

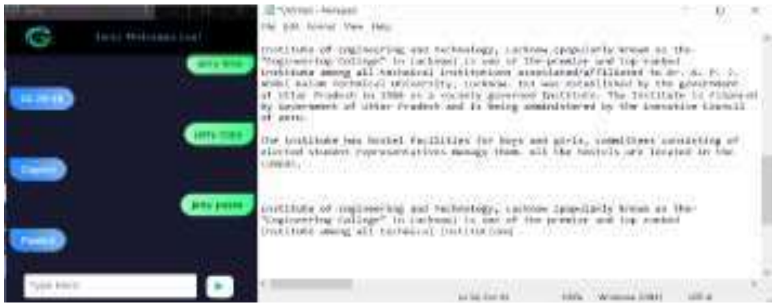
12:28:02

jerry copy

Copied

Institute of Engineering and Technology, Lucknow (popularly known as the "Engineering College" in Lucknow) is one of the premier and top ranked institute among all technical institutions associated/affiliated to Dr. A. P. J. Abdul Kalam Technical University, Lucknow. IET was established by the government of Uttar Pradesh in 1984 as a society governed Institute. The Institute is financed by Government of Uttar Pradesh and is being administered by the executive council of AKTU.

The institute has hostel facilities for boys and girls, committees consisting of elected student representatives manage them. All the hostels are located in the campus.



Jerry Welcomes you!

jerry time

12:28:02

jerry copy

Copied

jerry paste

Pasted

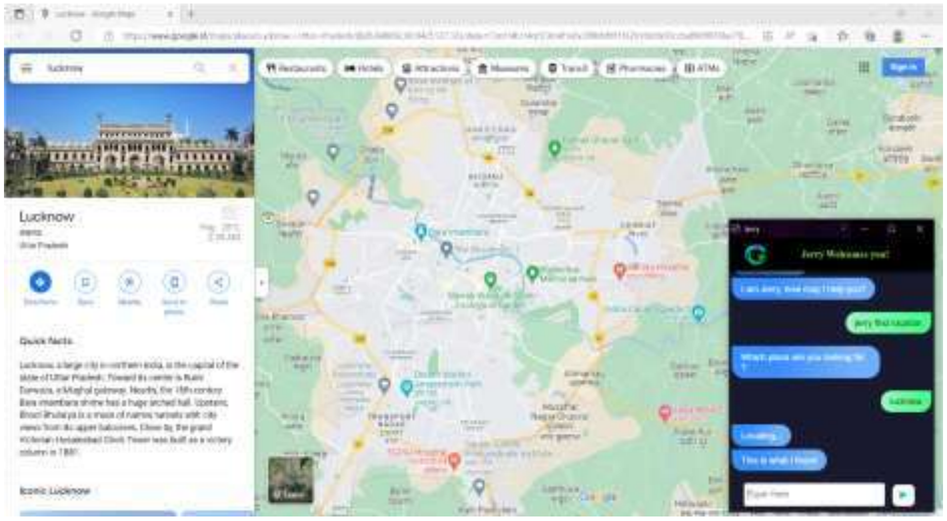
Institute of Engineering and Technology, Lucknow (popularly known as the "Engineering College" in Lucknow) is one of the premier and top ranked institute among all technical institutions associated/affiliated to Dr. A. P. J. Abdul Kalam Technical University, Lucknow. IET was established by the government of Uttar Pradesh in 1984 as a society governed Institute. The Institute is financed by Government of Uttar Pradesh and is being administered by the executive council of AKTU.

The institute has hostel facilities for boys and girls, committees consisting of elected student representatives manage them. All the hostels are located in the campus.

Institute of Engineering and Technology, Lucknow (popularly known as the "Engineering College" in Lucknow) is one of the premier and top ranked institute among all technical institutions

Fig 2.17 Copy and Paste

5. <u>Find location on Google Maps</u>: The user is able to search and find a location on Google Maps by just mentioning the desired location in the specified command..



Jerry Welcomes you!

I am Jerry, how may I help you?

jerry find location

Which place are you looking for ?

Lucknow

Locating...

This is what I found

Lucknow

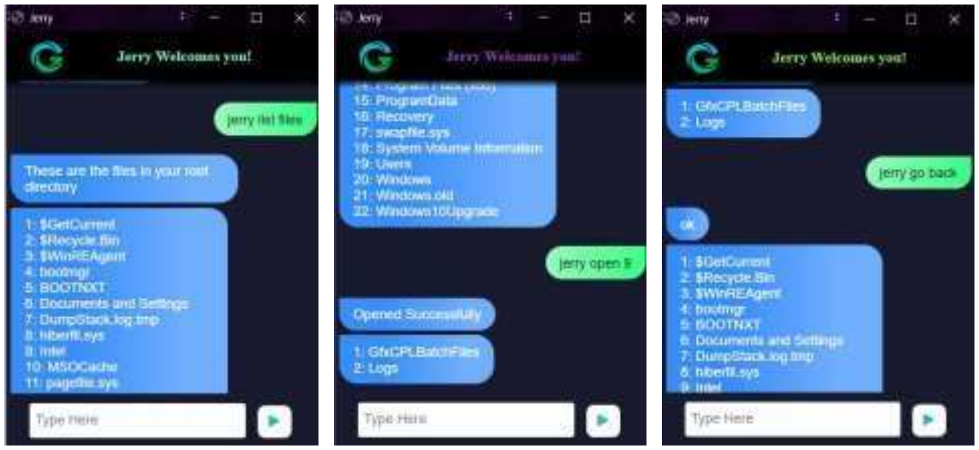
Quick facts

Lucknow, a large city in northern India, is the capital of the state of Uttar Pradesh. Toward its center is Rumi Darwaza, a Mughal gateway. Nearby, the 18th-century Bara Imambara shrine has a huge arched hall. Upstairs, Bhool Bhulaiya is a maze of narrow tunnels with city views from its upper balconies. Close by, the grand Victorian Husainabad Clock Tower was built as a victory column in 1881.

Iconic Lucknow

Fig 2.18 Find location on Google Maps

6. <u>File Navigation</u>: Navigating through the system files and directories is possible with the help of this assistant. It is able to move in and out of a directory and also can tell the location of the present working directory.



jerry list files

These are the files in your root directory

1: $GetCurrent
2: $Recycle.Bin
3: $WinREAgent
4: bootmgr
5: BOOTNXT
6: Documents and Settings
7: DumpStack.log.tmp
8: hiberfil.sys
9: Intel
10: MSOCache
11: pagefile.sys

15: ProgramData
16: Recovery
17: swapfile.sys
18: System Volume Information
19: Users
20: Windows
21: Windows.old
22: Windows10Upgrade

jerry open 9

Opened Successfully

1: GfxCPLBatchFiles
2: Logs

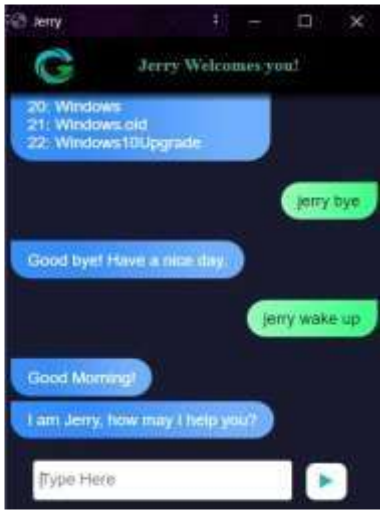
jerry go back

ok

1: $GetCurrent
2: $Recycle.Bin
3: $WinREAgent
4: bootmgr
5: BOOTNXT
6: Documents and Settings
7: DumpStack.log.tmp
8: hiberfil.sys
9: Intel

Fig 2.19 File Navigation

7. <u>Sleep/Wakeup</u>: You can make the voice assistant sleep by simply giving the *SLEEP* command and later wake it up whenever required by simply giving the *WAKEUP* command.



20: Windows
21: Windows.old
22: Windows10Upgrade

jerry bye

Good bye! Have a nice day.

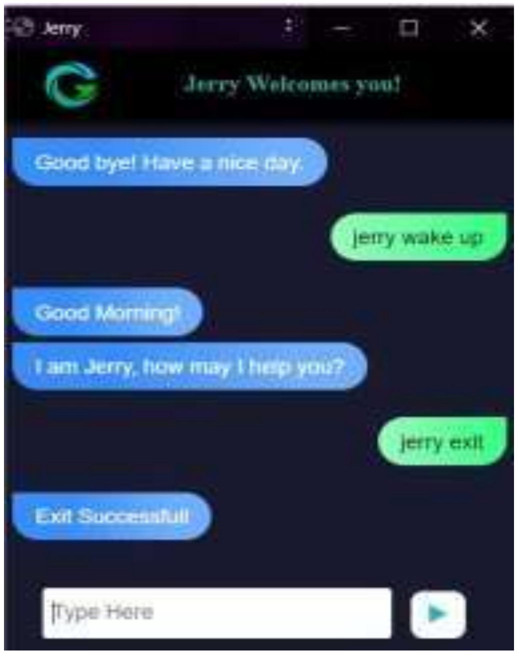
jerry wake up

Good Morning!

I am Jerry, how may I help you?

Fig 2.20 Sleep and Wakeup

8. <u>Exit</u>: The voice assistant can be closed by simply giving the *EXIT* command.



Good bye! Have a nice day.

jerry wake up

Good Morning!

I am Jerry, how may I help you?

jerry exit

Exit Successfull

Fig 2.21 Exit

CHAPTER 6

FUTURE SCOPE

Virtual Mouse will be introduced soon to replace the conventional computer mouse, making it easier for users to connect with and administer their computers. In order to correctly track the user's gesture, the software must be fast enough to capture and process every image and speech command. Other features and improvements could be added to make the application more user-friendly, accurate, and adaptable in different contexts. The following are the enhancements and functionalities that are required:

- **Smart Recognition Algorithm:** Using the palm and numerous fingers, additional functions such as enlarging and reducing the window, and so on, can be implemented.

- **Better Performance:** The response time is largely influenced by the machine's hardware, which includes the processor's processing speed, the amount of RAM available, and the webcam's characteristics. As a result, when the software is performed on a respectable machine with a webcam that operates well in various lighting conditions and a better quality microphone that can detect voice instructions correctly and rapidly, the programme may perform better.

6.1 Future Scope of Gesture Controlled Virtual Mouse

The Gesture Controlled Virtual Mouse project opens up several avenues for future development and expansion. Here are some potential areas for further exploration:

6.1.1 Advanced Gesture Recognition

While the project focuses on basic hand gestures, there is room for advancing the gesture recognition capabilities. More complex gestures, such as pinch, rotate, and multi-finger gestures, can be incorporated to provide a richer interaction experience. Additionally, incorporating depth-sensing technologies like 3D cameras or depth sensors could enable more precise gesture tracking and recognition.

6.1.2 Natural Language Processing

The voice recognition module can be enhanced with natural language processing (NLP) techniques. By analyzing the context and intent behind voice commands, the system can provide a more sophisticated and intuitive control experience. This could include understanding commands like "open a new tab" or "search for XYZ on the web" to perform specific actions without explicit commands for each function.

6.1.3 Adaptive Learning

Implementing machine learning algorithms could enable the system to learn and adapt to individual user preferences and behavior patterns. By continuously analyzing user input and feedback, the system could improve gesture and voice recognition accuracy and customize the control experience based on individual user needs.

6.1.4 Expanded Platform Support

Currently, the project targets popular operating systems, but future development could expand support to other platforms such as mobile devices and gaming consoles. This would allow users to control their smartphones or play games using gestures and voice commands, further enhancing accessibility and user experience.

6.1.5 Integration with Virtual Reality

Integrating the Gesture and Voice Controlled Virtual Mouse with virtual reality (VR) systems could create immersive and intuitive interactions within virtual environments. Users could navigate VR interfaces, interact with objects, and control virtual elements using natural hand gestures and voice commands, providing a seamless and immersive VR experience.

6.1.6 Collaboration and Multi-User Support

Expanding the system to support multiple users simultaneously would enable collaboration scenarios, where users can interact with the same system using gestures and voice commands. This feature could be useful in team-based environments, educational settings, or during presentations, where multiple users need to interact with a shared virtual environment.

6.1.7 Accessibility Enhancements

Continued development can focus on improving accessibility features, catering to individuals with specific disabilities or impairments. For example, incorporating eye-tracking technology could enable users with limited mobility to control the virtual mouse solely using eye movements, expanding accessibility options for a wider range of users.

6.1.8 Integration with Smart Home Devices

Integrating the gesture and voice recognition capabilities with smart home devices could provide a seamless control experience for various IoT devices. Users could control lights, appliances, and other smart devices using hand gestures and voice commands, creating a unified and intuitive home automation system.

6.1.9 User Feedback and Iterative Improvements

Collecting user feedback and continuously improving the system based on real-world usage will be crucial. Conducting usability studies, soliciting user suggestions, and implementing iterative improvements will ensure the system remains user-centric and meets the evolving needs of the user base.

By exploring these future avenues, the Gesture and Voice Controlled Virtual Mouse project can continue to evolve, providing innovative and efficient ways for users to interact with computers and other digital devices.

6.2 Future Scope of Voice Assistant

The future scope of a voice assistant is vast and promising, with continuous advancements in natural language processing (NLP), artificial intelligence (AI), and voice recognition technologies. Here are some key areas where voice assistants can expand their capabilities and find new applications in the future:

1. **Enhanced Personalization:** Voice assistants will become more personalized, adapting to individual users' preferences, behaviors, and needs. They will learn from user interactions, analyze data, and provide more accurate and tailored responses and recommendations.
2. **Multilingual Support:** Voice assistants will continue to improve their ability to understand and respond in multiple languages, making them accessible to a global audience. This will enable seamless communication and interaction across different cultures and languages.

3. **Contextual Understanding:** Future voice assistants will focus on better contextual understanding. They will be able to comprehend and analyze the user's context, such as location, previous queries, and current situation, to provide more relevant and helpful responses.

4. **Expanded Skill Sets:** Voice assistants will continue to acquire new skills and capabilities through integrations with third-party services and APIs. They will become more adept at performing tasks like making reservations, ordering food, managing finances, and controlling smart home devices.

5. **Improved Natural Language Processing:** NLP algorithms will advance, allowing voice assistants to understand and interpret human language more accurately. They will excel at handling complex queries, understanding nuances, and grasping the user's intent even in ambiguous situations.

6. **Emotional Intelligence:** Future voice assistants may develop emotional intelligence to recognize and respond to human emotions. They could provide empathy, support, and encouragement in various scenarios, such as mental health support or personal coaching.

7. **Seamless Integration across Devices:** Voice assistants will seamlessly integrate across a wide range of devices, including smartphones, tablets, smart speakers, cars, appliances, and wearable devices. This integration will enable a connected ecosystem where users can interact with their voice assistants from anywhere, anytime.

8. **Increased Accessibility:** Voice assistants have the potential to bridge the accessibility gap by providing assistance to people with disabilities. With improved speech recognition and specialized features, they can enhance the lives of visually impaired individuals, those with motor disabilities, or individuals with cognitive impairments.

9. **Business and Industry Applications:** Voice assistants will find extensive applications in various industries, including customer service, healthcare, education, finance, retail, and more. They can streamline processes, improve efficiency, and enhance the overall user experience in these domains.

10. **Natural Voice Interactions:** As voice assistant technology advances, interactions will become more natural and conversational. They will be capable of understanding and responding to complex queries, engaging in multi-turn conversations, and providing detailed information in a human-like manner.

11. **Security and Privacy:** Future voice assistants will prioritize security and privacy to ensure user data is protected. They will employ robust encryption techniques, provide granular control over data sharing, and implement advanced authentication mechanisms to prevent unauthorized access.

12. **Voice Assistant Ecosystem:** The future will witness the growth of an ecosystem around voice assistants, with developers creating and integrating voice-enabled applications, services, and devices. This ecosystem will drive innovation, offering new opportunities for businesses and users alike.

In summary, the future of voice assistants is brimming with possibilities. They will become more personalized, intelligent, context-aware, and seamlessly integrated into our daily lives. With advancements in technology and increasing user adoption, voice assistants will continue to evolve, transforming how we interact with technology and enhancing our overall productivity and convenience.

6.3 Limitation

There are certain existing environmental issues in this project that may obstruct the outcomes of gesture and voice recognition.

Extreme darkness or brightness can cause the targeted locations on the hand to be overlooked in the captured frames, making the gesture identification process particularly sensitive to light levels. Furthermore, because the current detection region can only handle a radius of 50cm, any display of hand that exceeds this distance will be considered noise and filtered out.

For voice recognition some background noises can hinder the detection of intended command. Identifying different types of accents and performing the exact commands is another tough task.

Furthermore, the program's performance is significantly reliant on the user's hardware, as processing speed and/or resolutions acquired by the webcam/mic may affect the program's load. As a result, the longer it takes to perform a single command, the slower the processing speed and/or the higher the resolutions are.

CHAPTER 7
CONCLUSION

7.1 Conclusions

In conclusion, the Gesture Controlled Virtual Mouse project has successfully achieved its objectives of developing a robust gesture recognition algorithm and implementing a virtual mouse system controlled by hand gestures. Throughout the project, various phases were executed, including data collection, gesture recognition model training, system development, user interface design, and testing and evaluation.

The project has resulted in the creation of a functional and intuitive system that allows users to control the computer mouse cursor using hand gestures. By utilizing machine learning or deep learning techniques, the gesture recognition algorithm demonstrates high accuracy in interpreting and classifying hand gestures in real-time.

The virtual mouse system seamlessly translates recognized gestures into precise mouse cursor movements, providing users with a natural and alternative input method. Additionally, the system incorporates click and drag functionalities, enabling users to perform tasks such as selecting, dragging, and dropping objects on the screen.

The user interface of the system is designed with simplicity and customization in mind. Users can easily configure and personalize the gesture recognition system according to their preferences, adjusting sensitivity, defining custom gestures, and calibrating the system as needed. Feedback mechanisms, such as visual cues or sounds, enhance the user experience and improve usability.
Thorough testing and evaluation have been conducted to ensure the accuracy, reliability, and performance of the gesture recognition system. The system's performance has been measured by comparing recognized gestures with the intended gestures, and user feedback has been gathered to identify areas of improvement and address usability issues.

The Gesture Recognition Controlled Virtual Mouse project holds great potential for enhancing accessibility, particularly for individuals with disabilities who may face challenges with conventional input devices. The system offers an intuitive and inclusive means of interacting with computers, enabling a wider range of users to navigate digital interfaces with ease.

As a result of this project, valuable insights and knowledge have been gained in the fields of computer vision, machine learning, and human-computer interaction. Future enhancements could involve expanding the range of recognized gestures, integrating additional functionalities, or exploring new applications for gesture-based control in various domains.

Overall, the Gesture Recognition Controlled Virtual Mouse project successfully demonstrates the feasibility and effectiveness of using hand gestures as a control mechanism for computer mouse operations. This technology opens up new possibilities for intuitive human-computer interaction and has the potential to revolutionize the way we interact with digital interfaces in the future.

In conclusion, the future of voice assistant projects holds immense potential for transforming our daily lives and reshaping the way we interact with technology. The advancements in natural language processing, artificial intelligence, and voice recognition technologies will pave the way for voice assistants to become more personalized, intelligent, and contextually aware.

The scope of voice assistant projects encompasses various aspects, including enhanced personalization, multilingual support, contextual understanding, expanded skill sets, improved natural language processing, emotional intelligence, seamless integration across devices, increased accessibility, business and industry applications, natural voice interactions, and prioritizing security and privacy.

As voice assistants continue to evolve, they will provide tailored and accurate responses, adapt to individual preferences, and seamlessly integrate across a wide range of devices. This will enable users to interact with voice assistants from anywhere, at any time, fostering a connected ecosystem. Additionally, voice assistants will find applications in diverse industries, improving efficiency, enhancing user experiences, and bridging accessibility gaps.

Furthermore, voice assistants will become more proficient in understanding human emotions, offering empathy and support in various scenarios. They will enhance communication and make technology more inclusive for people with disabilities.

To ensure user trust and data security, future voice assistants will prioritize robust encryption, granular control over data sharing, and advanced authentication mechanisms.

Overall, the future of voice assistant projects is promising and exciting. As they become more capable, intelligent, and seamlessly integrated into our lives, voice assistants will undoubtedly revolutionize how we interact with technology, improve our productivity, and enhance our overall convenience. The continuous advancements in this field will contribute to the growth of an ecosystem around voice assistants, fostering innovation and opening up new opportunities for businesses and users alike.

REFERENCES

1. Himanshu Bansal, Rijwan Khan, "**A review paper on human computer interaction**" International Journals of advanced research in Computer Science and Software Engineering, Volume 8, Issue 4, April 2018, https://www.researchgate.net/publication/325534924_A_Review_Paper_on_Human_Computer_Interaction

2. N. Subhash Chandra, T. Venu, P. Srikanth, "**A Real-Time Static & Dynamic Hand Gesture Recognition System**" International Journal of Engineering Inventions Volume 4, Issue 12, August 2015, http://www.ijeijournal.com/papers/Vol.4-Iss.12/P04129398.pdf

3. S. Shriram, B. Nagaraj, J. Jaya, "**Deep learning based real time AI Virtual Mouse system using computer vision to avoid COVID-19 spread**", Hindawi Journal of Healthcare Engineering, October 2021, https://www.researchgate.net/publication/355586801_Deep_Learning-Based_Real-Time_AI_Virtual_Mouse_System_Using_Computer_Vision_to_Avoid_COVID-19_Spread

4. Hritik Joshi, Nitin Waybhase, Ratnesh Litoria, "**Towards controlling mouse through hand gestures: A novel and efficient approach**", Medi-caps University, May 2022, https://journals.christuniversity.in/index.php/mapana/article/view/3420

5. Mohhamad Rafi, Khan Sohail, Shaikh Huda, "**Control mouse and computer system using voice commands**", International Journal of Research in Engineering and Technology", Volume 5, Issue 3, March 2016, https://ijret.org/volumes/2016v05/i03/IJRET20160503077.pdf

6. Python **OpenCV** Library, https://docs.opencv.org/4.x/

7. Python **Mediapipe** Library, https://developers.google.com/mediapipe/

APPENDIX A

ML Pipeline(Mediapipe) for Hand Tracking and Gesture Recognition

Mediapipe is a Machine Learning system built on the collaboration of pipeline models.

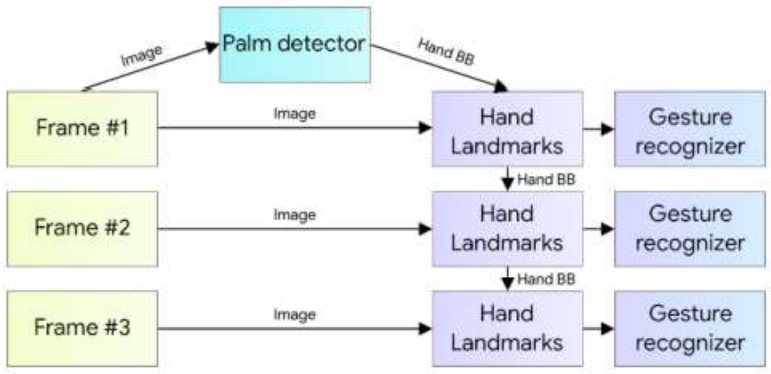
<u>What is ML Pipeline</u>?

A pipeline joins several stages together so that the output of one is used as the input for the next. Pipeline makes it simple to train and test using the same preprocessing.

<u>The hand tracking method makes use of a machine learning pipeline that consists of two models that work together:</u>

• A palm detector that uses an aligned hand bounding box to locate palms on a whole input image.
• A hand landmark model that uses the palm detector's clipped hand bounding box to produce high-fidelity results. landmarks in 2.5D.

The following is a summary of the pipeline:



<u>Palm Detector Model</u>

Hand detection is a tedious process as it requires identifying hands of various sizes, shapes, with deformities, etc. It is more complex than face detection as the contrast in features is far less than that in face. We use palm detection model first as detecting palm or a fist is much easier than detecting a full hand with articulated fingers. Also palms are smaller therefore non suppression algorithm works better for it.

<u>Hand Landmark Model</u>

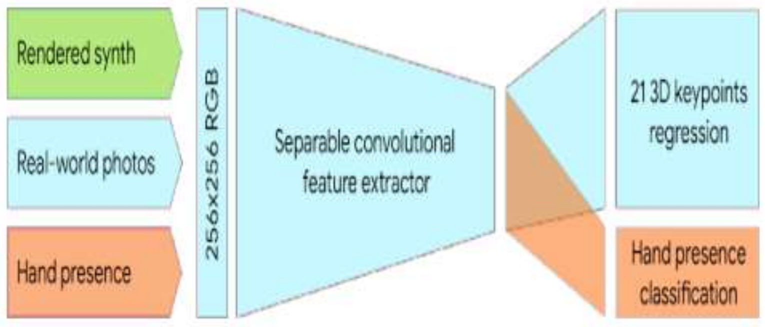
After detecting the palm using a palm detection model next a hand landmark model is used to detect 21 landmark points in 2.5 dimension. The Z depth is analysed using a Image Depth Map. The model recognises both partially and fully acclusioned hands perfectly.

The model has three outputs:

1. 21 hand landmarks consisting of x, y, and relative depth.
2. A hand flag indicates the existence of a hand in the input image.
3. A binary classification of handedness, e.g. left or right hand.

The topology is the same as for the 21 landmarks. To avoid performing hand detection over and over for the entire frame, the probability of hand presence in a bounded crop is determined. The detector is triggered to reset tracking if the score falls below a threshold. We constructed a binary classification head to predict whether the input hand is left or right. Only the first frame or when the hand prediction shows that the hand is lost is the detector used.

For our project, the fingers are given Ids from 0 to 4.


Rendered synth
Real-world photos
Hand presence
256x256 RGB
Separable convolutional feature extractor
21 3D keypoints regression
Hand presence classification

DEEPANSHU GUPTA

Phone: +91-81718700659847 404933 | Email: deepanshugupta9128@gmail.com| LinkedIn: deepanshu-gupta-6222aa219 | GitHub: deepanshu823

EDUCATION

Ajay Kumar Garg Engineering College, Ghaziabad (2019 – 2023)
Bachelor of Technology in Computer Science **CGPA : 6.97**

DAV Shreshta Vihar, Delhi (2018 - 2019)
Senior Secondary (Class XII) | Central Board of Secondary Education **Percentage : 80.25%**

SKILLS

Technical : C++, Python, SQL, HTML, Data Structure and Algorithm, Machine Learning, Problem Solving

TECHNICAL PROJECTS

Movie Suggestion System

- Implemented a machine learning model which takes users information as input and recommends movies to the user through a filtering process that is based on user preferences and browsing history
- In order to build the suggestion system, we have used the MovieLens Dataset. This data consists of 105339 ratings applied over 10329 movies.

Bomb Catcher Game

- Designed and Developed an interactive mini game using Unity game development engine. The aim of the game is to catch the bombs falling from the sky into a container before bomb touching the ground else the bomb will explode

ACHIEVEMENTS

- Solved 200+ Problems on GeeksForGeeks
- Solved 150+ Problems on LeetCode
- Certified by Coursera for completing Data Structure and Algorithms course
- Participation in various Competitive Coding contest
- Coordination in organizing technical event in college

DEVANSH KUMAR

Phone: +91-8171404933| Email: kumardevansh@gmail.com | LinkedIn: devanshkumar524 | GitHub: devansh511

EDUCATION

Ajay Kumar Garg Engineering College, Ghaziabad (2019 – 2023)
Bachelor of Technology in Computer Science **CGPA : 8.7**

Krishna Public Collegiate, Kashipur (2018 – 2019)
Senior Secondary (Class XII) | Central Board of Secondary Education **Percentage : 94.80%**

SKILLS

Technical : C++, C, Data Structures, Algorithms, Object Oriented Programming (OOPS), Design and Analysis of Algorithms, ReactJS, SQL, HTML, CSS, Git, GitHub, Windows
Non-Technical : Problem Solving, Communication, Teamwork, Leadership, Versatility

TECHNICAL PROJECTS

vTube

- **vTube** is an online video streaming web application, a YouTube clone built using **ReactJS** for frontend and **Google Firebase** for backend.
- Implemented various functionalities from YouTube using **YouTube Data API v3.**
- Technologies include **ReactJS**, **React-Redux**, **Google Firebase.**

Real Time Voice Chat App (DiscussIT)

- **DiscussIT** is a MERN stack-based web application which enables the authenticated users to join any voice channel or create their own channel and discuss on the specific topics.
- Worked on complete frontend of the application using **React.js** and **React Redux.**

PROGRAMMING ACHIEVEMENTS

- Rated **Specialist** on Codeforces with max rating **1405**.
- Rated **4 Stars** on CodeChef with max rating **1865**.
- Solved 800+ problems on LeetCode with max contest rating **1891**.
- Global Rank **749** in TCS Codevita Season 10.
- Global Rank **995**(out of 21k+ participants) in Codeforces Round #797.
- Global Rank **1124**(out of 20k+ participants) in Codeforces Round #790.

CODING PROFILE HANDLES

Codeforces - **_accepted_524**	LeetCode - **devansh__005**
CodeChef - **flamboi005**	HackerRank - **kumardevansh8**

HARSH HARIT

Phone: +91- 7983948935 | Email:harsh1911019@akgec.ac.in | LinkedIn: harshharit |
GitHub : harshharit

Education

Ajay Kumar Garg Engineering College, Ghaziabad (2019 – 2023)
Bachelor of Technology in Computer Science **CGPA : 7.24**

BDS International School, Meerut (2018-2019)
Senior Secondary (Class XII) | Central Board of Secondary Education **Percentage : 72.6%**

Skills

Languages - C/C++ , Python, HTML, CSS, JavaScript, Dart.

Technologies/Framework/Libraries - Flutter , ReactJS, Firebase, Google Cloud, MySQL.

Technical Projects

Weather App - (Oct 2020)

- Developed a standard weather app using Flutter and Dart.
- Implemented a simple API in the app.

Smart Chat Bot - (May 2021)

- Developed a smart chat bot using python and NLTK framework.

Portfolio Responsive Website - (June 2022)

- Developed a portfolio website of myself using React.
- It also uses Node.js and SASS.

Achievements

- Ethical Hacking and Networking Certificate , Udemy. (June 2019)
- Google Cloud Ready Certificate.(August 2020)
- Data Structures and Algorithms in C++,Coding Ninjas Certificate of Excellence.(May 2022)

Coding Profile Handles

LeetCode - **harshharitt**
CodeStudio - **https://www.codingninjas.com/codestudio/profile/8e24fc1b-54f3-4eb6-85a9-7747c9e815f1**

JIGYANSH VARSHNEY

Phone: +91-8171762485 | Email: varshneyjigyansh@gmail.com | LinkedIn : jigyansh-varshney-27500418a | GitHub : jigyansh

EDUCATION

Ajay Kumar Garg Engineering College, Ghaziabad (2019 – 2023)
Bachelor of Technology in Computer Science **CGPA : 7.56**

RRK Sr Secondary School (2018 – 2019)
Senior Secondary (Class XII) | Central Board of Secondary Education **Percentage : 89.4%**

SKILLS

Technical : HMTL, CSS, JavaScript, OOPS, Computer Networks, Data Structures, C, C++, React.js, Node.js

TECHNICAL PROJECTS

Messaging App

- Messenger clone using React and Node.js would involve building a real-time chat application with a frontend built using React, a backend built using Node.js, and a database to store chat messages.
- The UI components will be created using React, and API endpoints will be created to handle message retrieval and sending.
- **ReactJs and NodeJs.**

Crud Application

- CRUD project involves building a software application that allows users to perform operations on data stored in a database with user friendly interface, API endpoints to handle CRUD operations, and a user authentication system.
- **Nodejs , Express , MongoDb , Mongoose.**

ACHIEVEMENTS

- Solved around 200+ problems on GFG.
- Solved around 250+ problems on LeetCode.
- Finalist in school level badminton singles tournament.

CODING PROFILES

GeeksForGeeks : **varshneyjigyansh**
LeetCode : **jigyansh**