ACKNOWLEDGEMENT

I undersigned, have great pleasure in giving my sincere thanks to those who have contributed their valuable time in helping me to achieve the success in my project work.

My heartfelt thanks to The Principle of the college **PROF.SIRAJUDDIN CHOUGLE** and the IT Department of College for helping in the project with words of encouragement and has shown full confidence in our abilities.

I would like to express my sincere thanks to **DR. SAIMA SHAIKH** head of I.T Department for her constant encouragement, which made this project a success.

I am indebted and thankful to our Project Guide **DR. SAIMA SHAIKH** to Whom I owe her piece of knowledge for her valuable and timely guidance, co-operation, encouragement and time spent for this project work. I would also like to thank our IT staff for providing us sufficient information, which helped us to complete our project successfully.

My sincere thanks to the Library staff for extending their help and giving me all the books for reference in a very short span of time. I also thank **MY PARENTS** and all my family members for their continued Support, without their support this project would not be possible.

Mohammed Adin Parvez Ansari

DECLARATION

I hereby declare that the project entitled, "**Gesture Control Virtual Mouse and Voice Assistant**" done at Mumbai, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of **MASTERS OF SCIENCE (INFORMATION TECHNOLOGY)** to be submitted as final semester project as part of our curriculum.

Name and Signature of the Student

ABSTRACT

The use of hand gesture recognition in controlling virtual devices has become popular due to the advancement of artificial intelligence technology. A hand gesture-controlled virtual mouse system that utilizes AI algorithms to recognize hand gestures and translate them into mouse movements is proposed in this paper. The system is designed to provide an alternative interface for people who have difficulty using a traditional mouse or keyboard. The proposed system uses a camera to capture images of the user's hand, which are processed by an AI algorithm to recognize the gestures being made. The system is trained using a dataset of hand gestures to recognize different gestures. Once the gesture is recognized, it is translated into a corresponding mouse movement, which is then executed on the virtual screen. The system is designed to be scalable and adaptable to different types of environments and devices. All the input operations can be virtually controlled by using dynamic/static hand gestures along with a voice assistant. In our work we make use of ML and Computer Vision algorithms to recognize hand gestures and voice commands, which works without any additional hardware requirements. The model is implemented using CNN and mediapipe framework. This system has potential applications like enabling hand-free operation of devices in hazardous environments and providing an alternative interface for hardware mouse. Overall, the hand gesture-controlled virtual mouse system offers a promising approach to enhance user experience and improve accessibility through human-computer interaction.

Keyword: Computer vision, hand gesture recognition, Media-pipe, virtual mouse.

Mohammed Adin Parvez Ansa

INDEX

Chapter 1
Introduction

1.1 Background and Motivation

In recent years, AI-based technologies have gained significant popularity and have revolutionized various industries and fields. One area that has seen remarkable advancements is human-computer interaction. Traditional input devices, such as keyboards and mice, have limitations in terms of natural and intuitive interaction. Users often require additional training to master complex interfaces, leading to reduced productivity and user experience.
The motivation behind this research is to develop a more efficient and user-friendly screen controller using AI and hand gestures. By leveraging computer vision and machine learning techniques, the aim is to create a system that allows users to interact with their screens effortlessly, mimicking natural hand movements. This would not only enhance user experience but also open up new possibilities for applications in areas such as gaming, design, and virtual reality.
Voice assistants like Siri, Google Assistant, and Amazon Alexa have revolutionized how users interact with technology. They offer hands-free control, natural language understanding, and seamless integration with various applications and devices. However, voice assistants primarily focus on executing commands and retrieving information, with limited capabilities in precise cursor control and detailed navigation tasks on a computer screen.

The motivation for developing the Gesture Controlled Virtual Mouse and Voice Assistant stems from the desire to combine the strengths of gesture recognition and voice control technologies to create a more versatile and user-friendly interface.

- **Enhanced Accessibility**
    - **Inclusive Design**: Create a system that is accessible to users with varying physical abilities. By providing multiple input modalities, the system can cater to users who may have difficulty using traditional input devices.
    - **User Independence**: Enable users with disabilities to interact with computers more independently and efficiently.

- **Improved Ergonomics**
    - **Reduced Physical Strain**: Minimize the physical strain associated with prolonged use of keyboards and mice. Gesture control can provide a more ergonomic alternative, reducing the risk of RSI and other musculoskeletal issues.
    - **Comfortable Interaction**: Allow users to interact with their computers in a more comfortable and natural manner, enhancing overall user experience.

- **Intuitive User Experience**
    - **Natural Interaction**: Utilize natural hand gestures and voice commands to provide a more intuitive and seamless user experience. This can be particularly beneficial for users who are not tech-savvy or are new to computing.
    - **Enhanced Productivity**: Combine the precision of gesture-based cursor control with the convenience of voice commands to streamline workflows and enhance productivity.

- **Innovation in HCI**

    - **Advanced HCI**: Contribute to the ongoing evolution of HCI by integrating cutting-edge technologies in gesture recognition and voice control. This project aims to push the boundaries of how users interact with digital devices.
    - **Future Applications**: Lay the groundwork for future innovations in HCI, including applications in virtual reality (VR), augmented reality (AR), and smart environments

1.2 Problem Statement

The traditional input devices used for screen control have several limitations. Keyboards and mouse, although widely used, may not provide the most intuitive and efficient interaction experience. Users often need to memorize complex shortcuts and gestures, which can be time- consuming and prone to errors. There is a need for a more natural and user-friendly interface that simplifies screen control and improves overall productivity. The current reliance on traditional input devices such as keyboards and mice presents significant barriers to accessibility, ergonomics, and intuitive interaction. Individuals with physical disabilities face challenges in using these devices, while prolonged use can lead to repetitive strain injuries and physical fatigue. Furthermore, the lack of natural interaction methods limits the intuitiveness and efficiency of human-computer interactions. There is a need for a more inclusive, ergonomic, and intuitive HCI solution that leverages gesture recognition and voice control technologies to enhance accessibility, reduce physical strain, and provide a more natural and efficient user experience.

1.3 Objectives

- **Develop a Gesture Recognition System**

    - Track hand movements using a camera.
    - Interpret gestures for mouse actions (e.g., move, click, drag, scroll).
    - Ensure real-time processing with minimal latency.

- **Integrate a Voice Assistant**

    - Recognize and interpret voice commands.
    - Use natural language processing to understand command intent.
    - Execute tasks like opening applications, web searches, and system control.

- **Enhance Accessibility**

    - Design the system to be accessible for users with physical disabilities.
    - Allow customization of gestures and voice commands.

- **Improve Ergonomics**
    - Reduce reliance on keyboards and mice to minimize physical strain.
    - Provide a comfortable interaction experience to reduce user fatigue.
- **Enhance User Experience**
    - Create an intuitive interface using natural gestures and speech.
    - Enable efficient multi-tasking with simultaneous gesture and voice commands.
- **Ensure Robustness and Accuracy**
    - Achieve high accuracy in gesture and voice recognition.
    - Adapt to different environments, handling lighting and background noise effectively.
- **Facilitate Integration and Compatibility**
    - Ensure compatibility with various devices (desktops, laptops, etc.).
    - Integrate seamlessly with existing software applications and operating systems.

1.4 Scope of the Research

The scope of the Gesture Controlled Virtual Mouse and Voice Assistant project encompasses the development, implementation, testing, and integration of a system that enhances human-computer interaction through gesture recognition and voice control technologies

- **Enhanced Gesture Recognition:** Continuous research and development in machine learning can lead to more accurate and robust gesture recognition algorithms. Consider exploring advanced techniques, such as deep learning models (e.g., deep neural networks) or hybrid models that combine CNNs and RNNs to improve the recognition of complex gestures.
- **Accessibility:** One of the primary applications of ML- based virtual mouse software is to provide accessibility for individuals with physical disability or motor impairments. It allows them to interact with computers and devices without the need for traditional hardware mouse, making computing tasks more accessible and inclusive.
- **Healthcare:** In medical settings, AI based virtual mouse software can be usedto control computers and devices without physical contact, reducing the risk of cross-contamination and promoting a more hygienic environment.
- **Hardware Optimization:** explore the use of specialized hardware or sensors, such as dept cameras (e.g., Intel RealSense or Kinect) or wearable devices (e.g., Smart Gloves), to improve the accuracy and versatility of gesture recognition.

- **Security and Privacy:** Consider the security and Privacy implications of hand gesture control, implement safeguards to prevent unintended actions and ensure that the system respects user privacy.

- **Voice Assistant**

  - **Voice Command Recognition**: Implement a system to accurately capture and interpret voice commands.
  - **Natural Language Processing**: Use NLP to understand and process the intent behind voice commands.
  - **Task Execution**: Enable the voice assistant to perform tasks such as opening applications, web searches, controlling system settings, and automating routines.
  - **Customization**: Allow users to customize voice commands for specific actions.

Chapter 2
System Analysis

2.1 Existing System

Overview

The existing systems typically rely on traditional input devices such as keyboards and mice for human-computer interaction (HCI). These systems, while effective, have limitations in terms of accessibility, ergonomics, and natural interaction.

Key Features

1. **Keyboard and Mouse Interaction**
    - Users interact primarily through typing on keyboards and navigating with mice.
    - Limited to manual input methods, requiring physical manipulation of devices.
2. **Voice Assistants**
    - Existing voice assistants like Siri, Google Assistant, and Alexa provide hands-free control and perform tasks based on voice commands.
    - Focus primarily on executing commands and retrieving information rather than precise control or integration with gesture-based interaction.
3. **Accessibility and Ergonomics**
    - Accessibility features are often limited, with few alternatives for users with physical disabilities beyond standard input devices.
    - Ergonomic issues include risks of RSI and physical strain due to prolonged use of keyboards and mice.

2.2 Proposed System

Overview

The Gesture Controlled Virtual Mouse and Voice Assistant project proposes an innovative system that combines gesture recognition and voice control technologies to enhance HCI. This system aims to provide a more intuitive, accessible, and ergonomic interaction method compared to existing systems.

Key Features

1. **Gesture Recognition Module**
    - **Hand Tracking**: Utilizes a camera to track hand movements and gestures.
    - **Gesture Interpretation**: Algorithms interpret gestures for precise mouse actions like cursor movement, clicking, dragging, and scrolling.
    - **Customization**: Users can define and customize gestures according to their preferences.
2. **Voice Assistant Module**
    - **Voice Command Recognition**: Captures and interprets voice commands using speech recognition technology.
    - **Natural Language Processing (NLP)**: Analyzes voice commands to understand intent and execute tasks such as application control, web searches, and system settings adjustments.
    - **Customization**: Allows users to personalize voice commands for specific actions and preferences.

3. **Benefits**
    - **Accessibility**: Enhances accessibility for users with physical disabilities by offering alternative input methods.
    - **Ergonomics**: Reduces ergonomic strain associated with traditional input devices, promoting user comfort and health.
    - **Intuitive Interaction**: Enables natural and intuitive interaction through gestures and voice commands, improving user experience and efficiency.

Implementation Plan

1. **Development and Integration**
    - Develop gesture recognition and voice assistant modules using technologies like OpenCV for image processing and Python for algorithm development.
    - Integrate modules into a unified system compatible with major operating systems (Windows, macOS, Linux).
2. **Testing and Refinement**
    - Conduct rigorous testing to ensure system accuracy, performance, and usability across diverse user scenarios.
    - Gather user feedback and iterate on the system design to optimize functionality and user satisfaction.
3. **Deployment and Support**
    - Release the finalized system along with comprehensive user documentation and support materials.
    - Provide ongoing updates and maintenance to address user feedback, technological advancements, and system improvements.

2.3 Gesture Recognition Techniques

Gesture recognition techniques play a crucial role in the field of human-computer interaction, enabling users to interact with computers and devices using hand gestures. There are various techniques employed for gesture recognition, including computer vision-based and machine learning-based approaches.

2.3.1 Computer Vision-based Gesture Recognition

Computer vision-based gesture recognition techniques utilize image processing and computer vision algorithms to detect and interpret hand gestures. These techniques typically involve extracting hand features, such as hand shape, position, and movement, from images or video frames. In the presented project, the Hand Tracking module employs computer vision techniques to detect and track hands in real-time using the Mediapipe library. It utilizes landmark detection to identify the positions of specific hand landmarks, allowing for gesture recognition based on hand poses and finger configurations.

2.3.2 Machine Learning-based Gesture Recognition

Machine learning-based gesture recognition techniques leverage the power of machine learning algorithms to recognize and classify hand gestures. These techniques involve training models on large datasets of hand gesture examples to learn patterns and relationships between input gestures and their corresponding outputs. The trained models can then be used to recognize and interpret new gestures.

2.4 AI-based Human-Computer Interaction

AI-based human-computer interaction refers to the application of artificial intelligence techniques in enhancing the interaction between humans and computers. In the context of the project, the integration of hand tracking and gesture recognition with an AI virtual mouse allows users to control the mouse cursor on the screen using hand movements and gestures. This AI-powered interaction enables a more intuitive and natural way of interacting with computers, offering users greater flexibility and convenience.

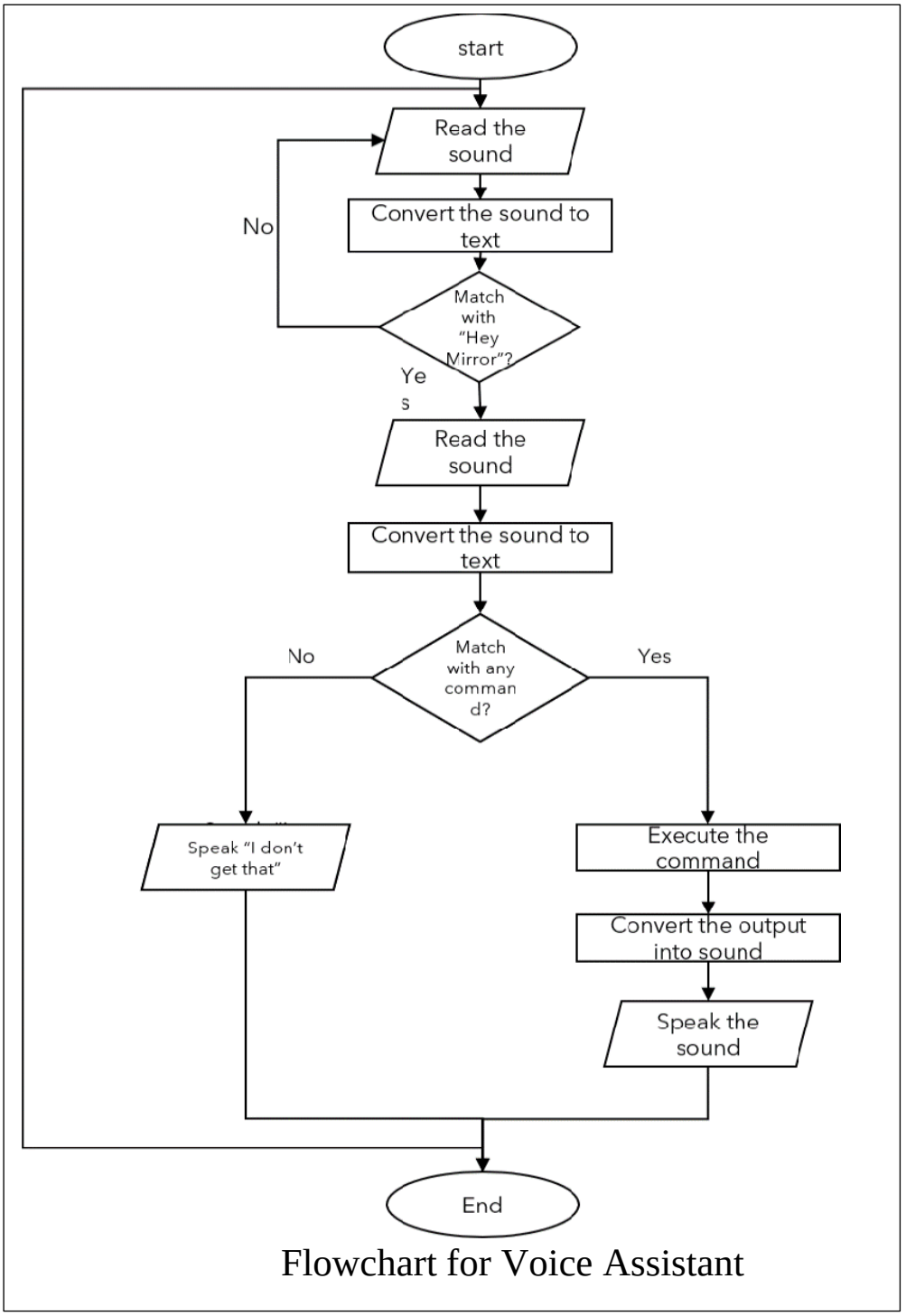
2.5 Hand Tracking and Detection

Hand tracking and detection form the foundation of the presented project. The Hand Tracking module utilizes computer vision techniques and the Mediapipe library to detect and track hands in real-time video frames. It extracts hand landmarks and calculates their positions, allowing for precise tracking and analysis of hand movements and gestures. The module employs a hand detection model and hand landmark models provided by Mediapipe, enabling accurate and robust hand tracking capabilities.

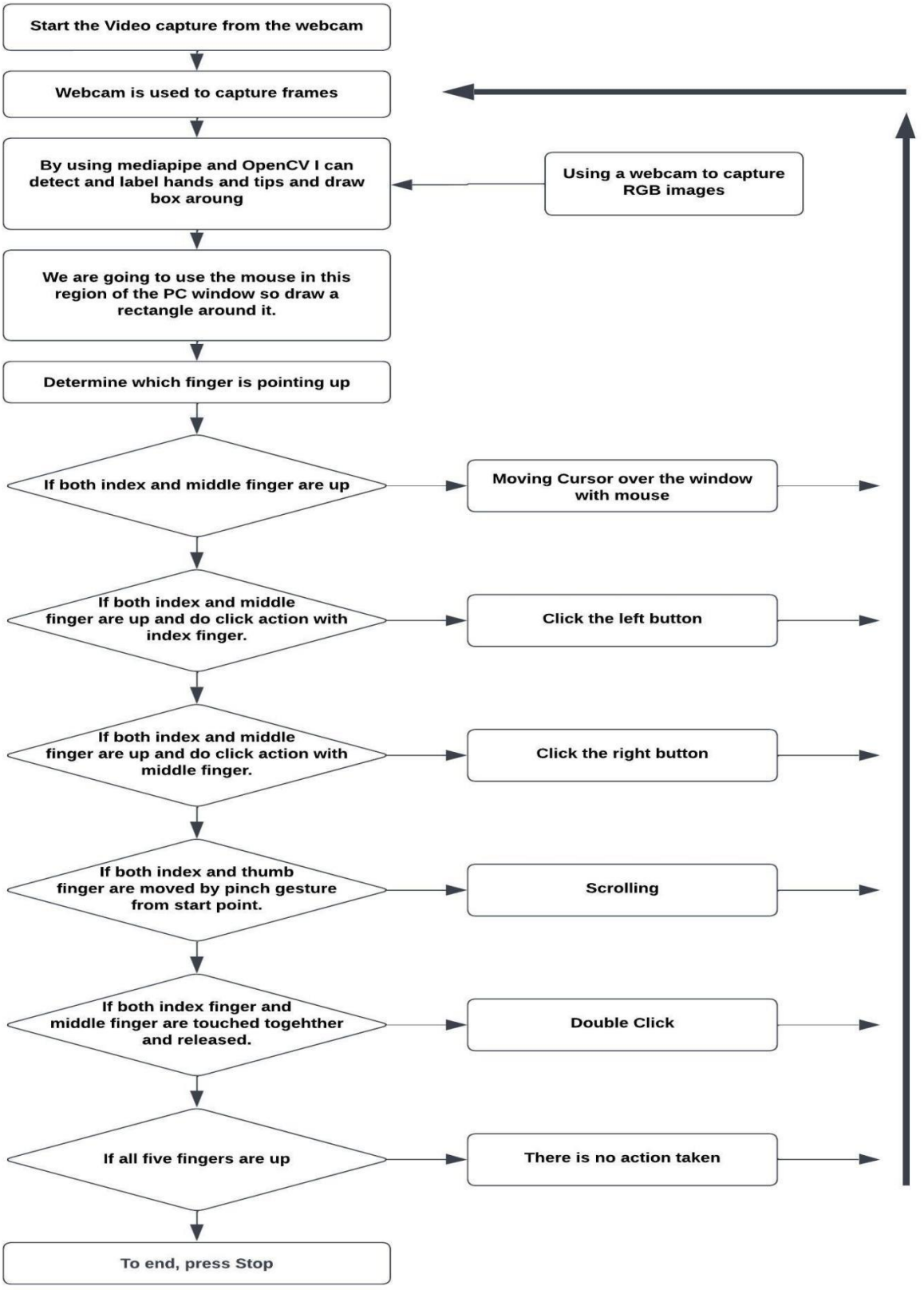
Chapter 3
System Architecture

3.1 System Overview

The AI-based screen controller using hand gestures is designed to enable intuitive screen control through the recognition of hand movements and gestures. The system comprises several key components, including data collection and preprocessing, hand tracking module, gesture recognition module, and AI-based screen control algorithms. These components work together to process input hand gestures and perform corresponding screen control actions.

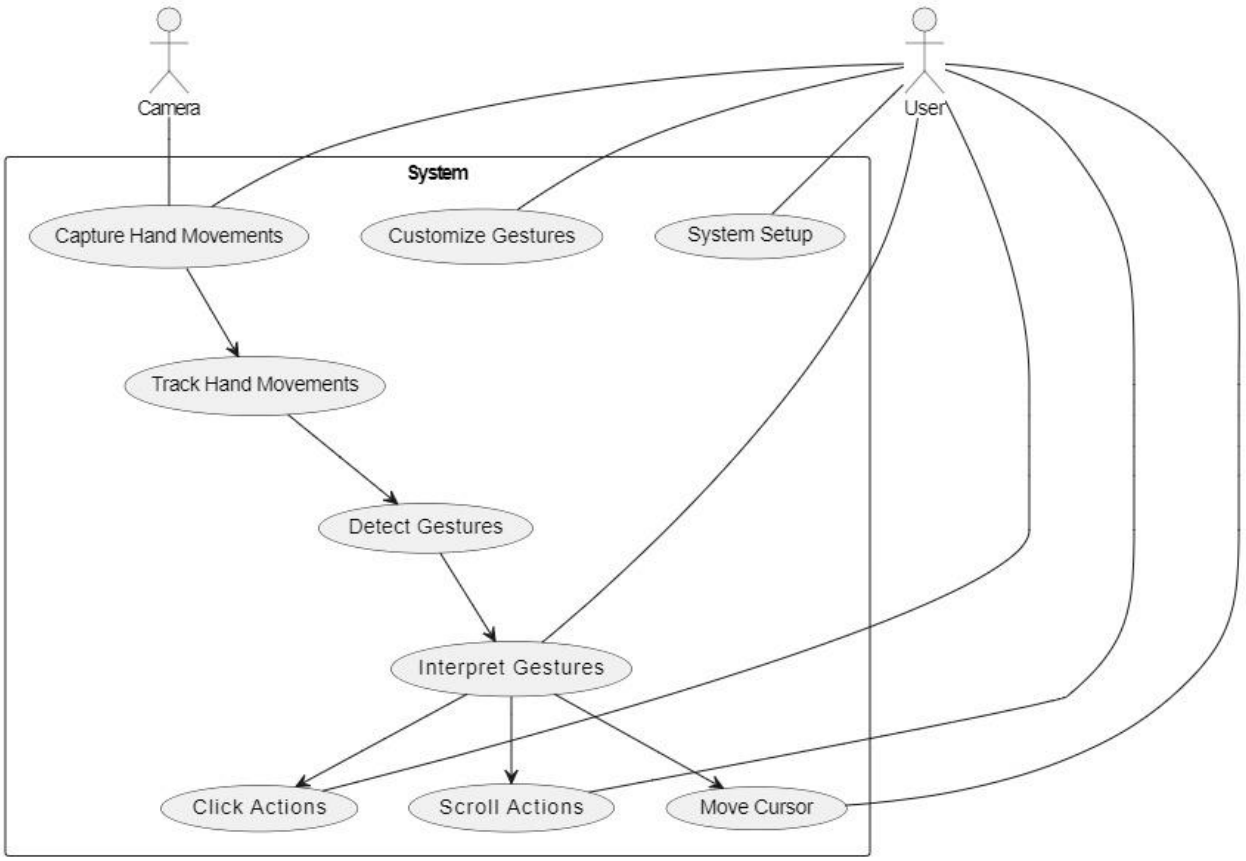


Flowchart for Voice Assistant

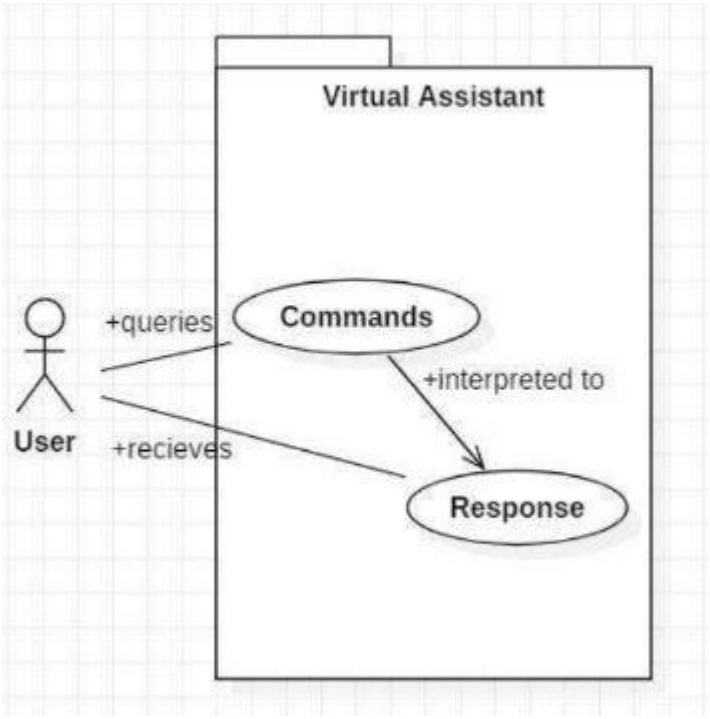


Flowchart for Virtual Mouse

3.2 Use Case Diagram

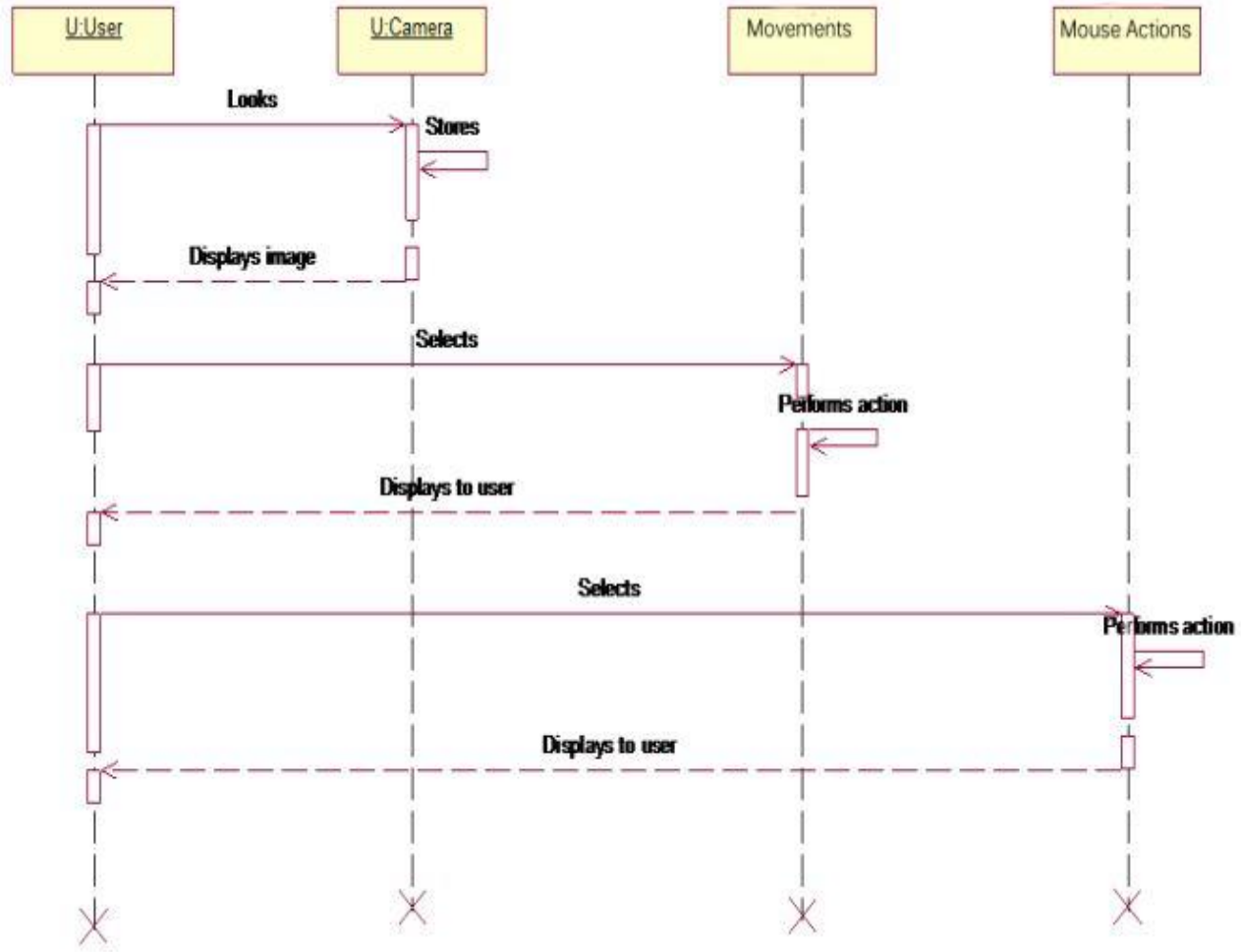


Use case for Virtual Mouse

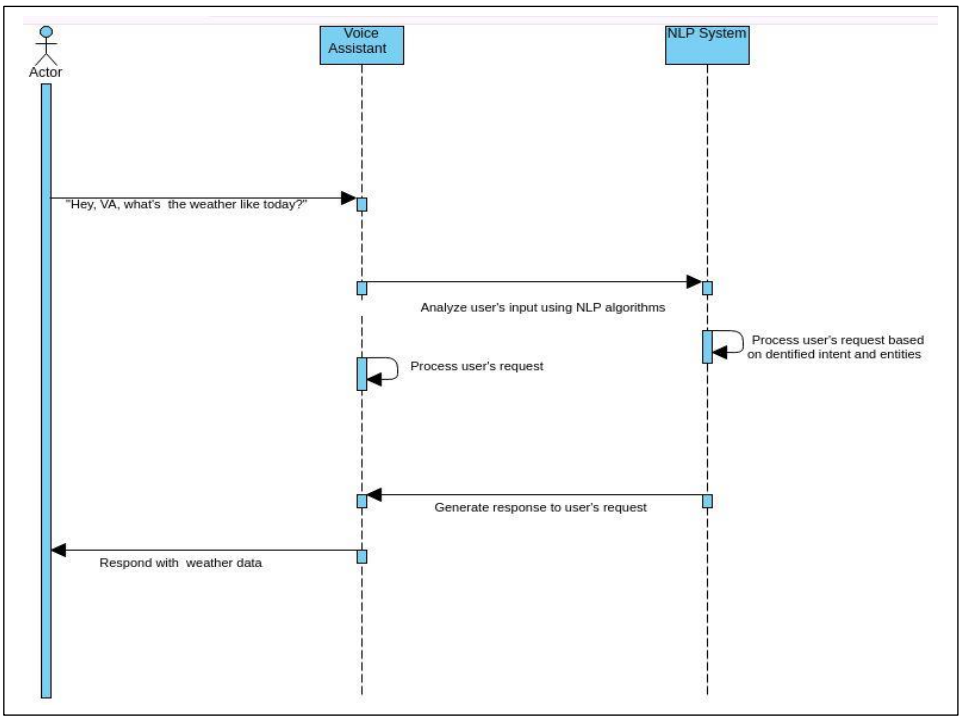


Use case for Voice Assistant

3.3 Sequence Diagram

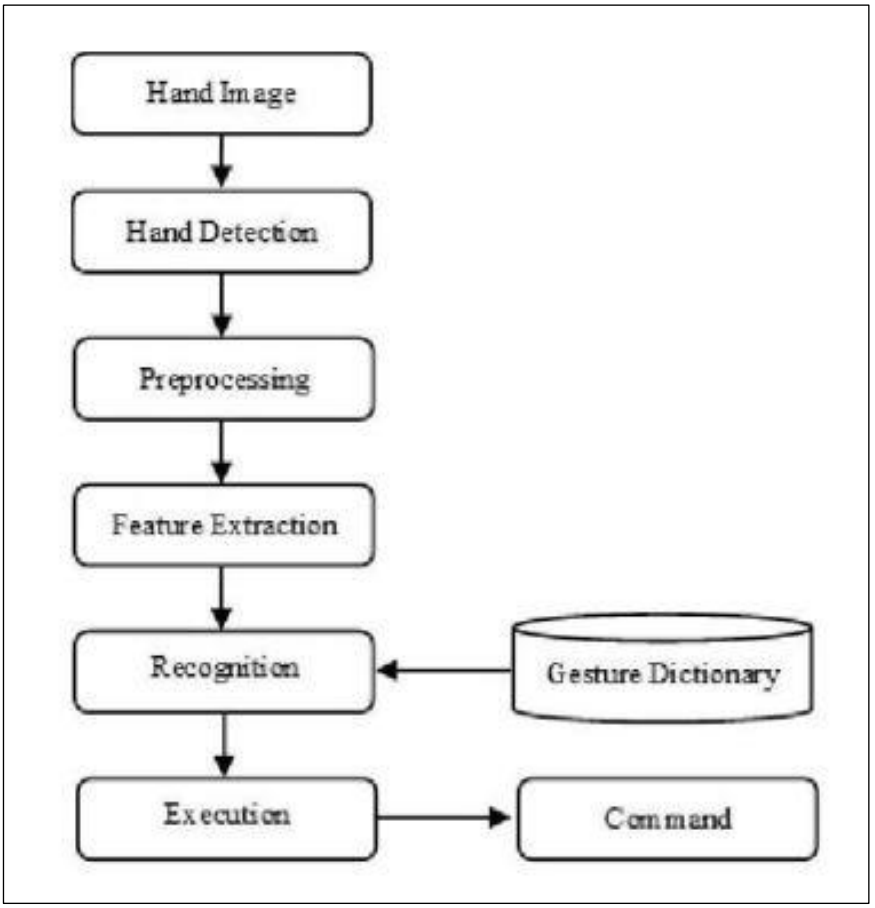


Sequence diagram for Virtual Mouse

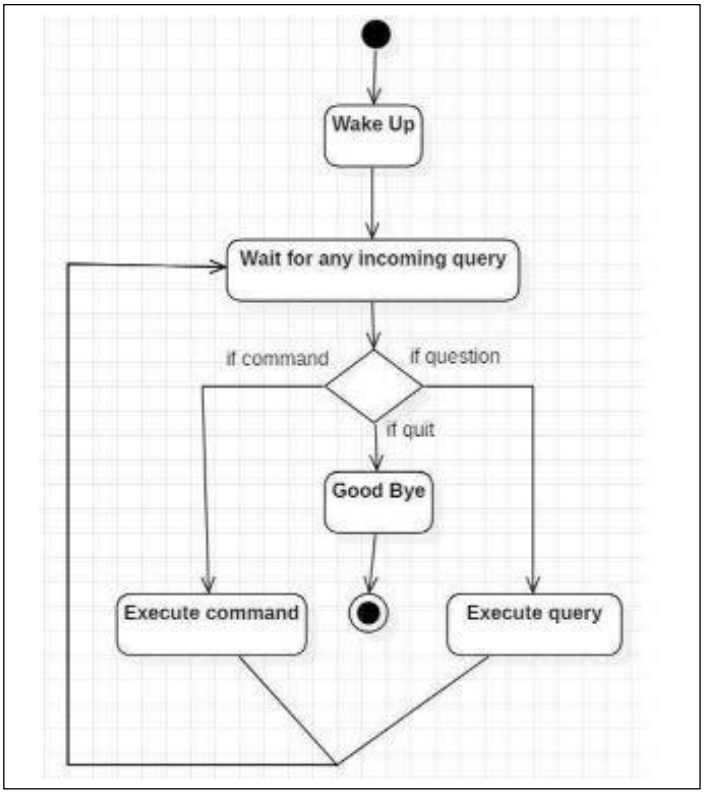


Sequence diagram for voice assistant

3.4 Activity Diagram



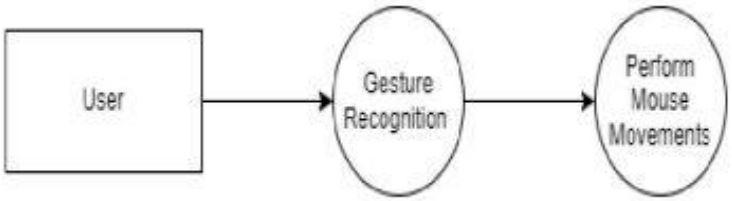
Activity diagram for Virtual mouse



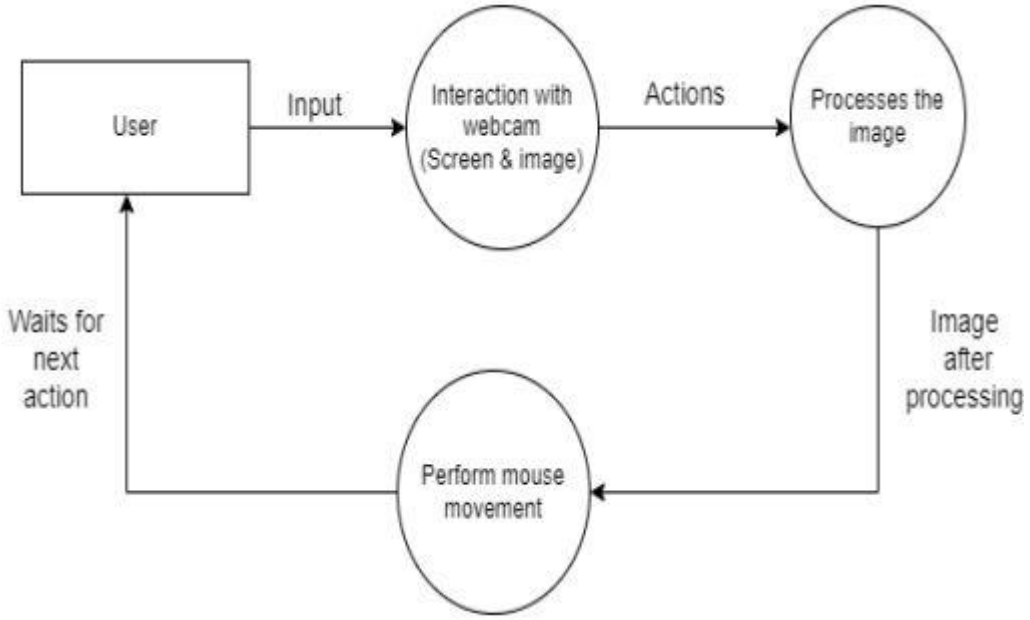
Activity diagram for Voice Assistant

3.5 Data Flow Diagram

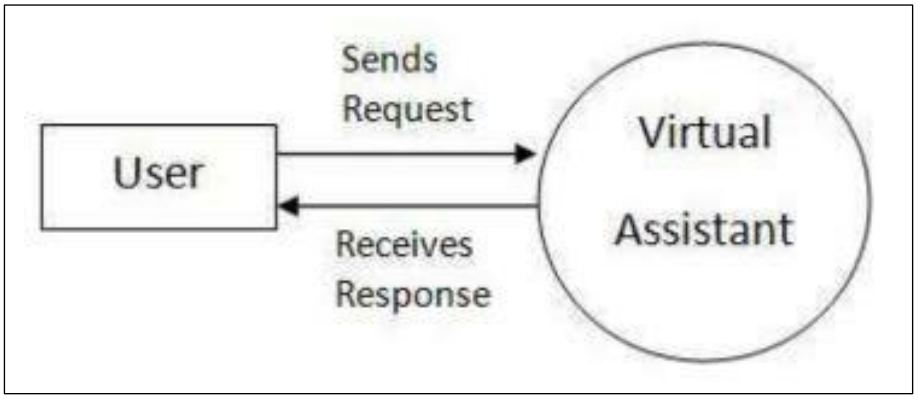
DFD level 0 for Virtual Mouse



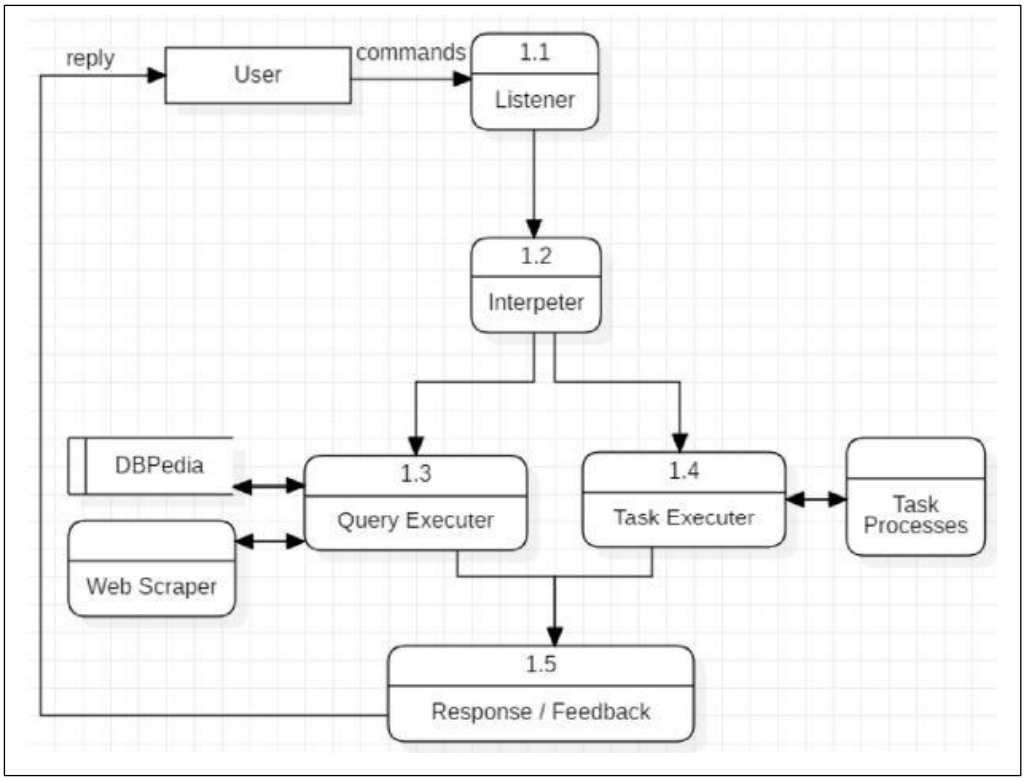
DFD level 1 for Virtual Mouse



DFD level 0 for Voice Assistant

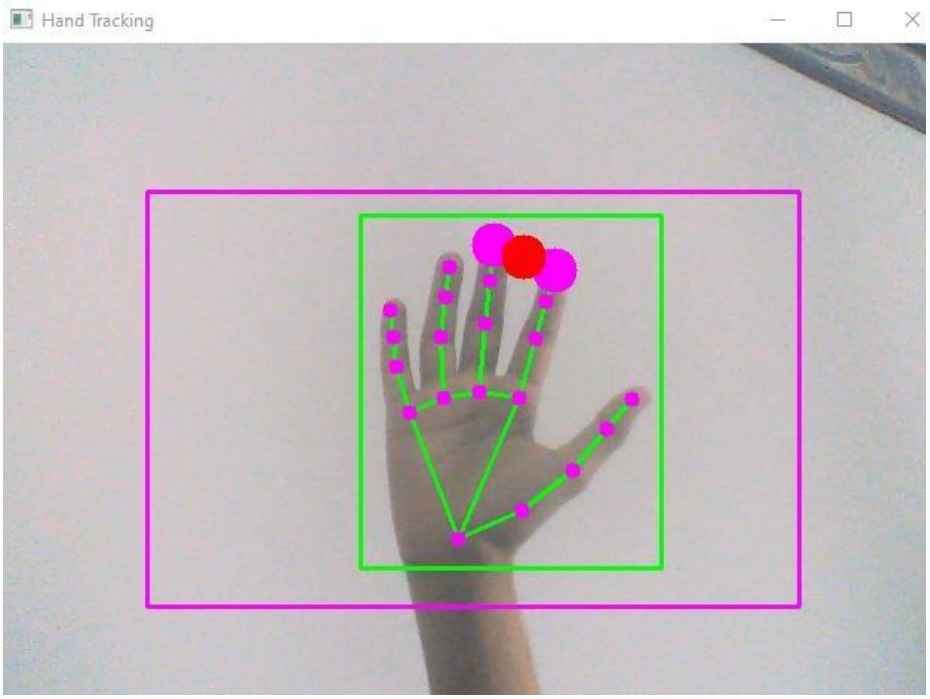


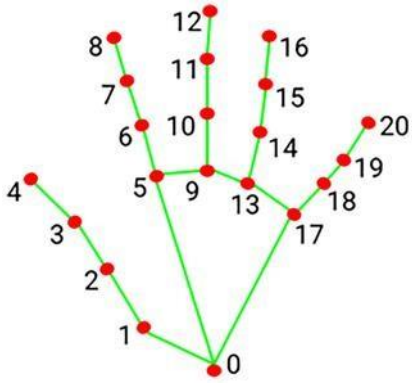
DFD level 1 for Voice Assistant



3.6 Hand Tracking Module

The hand tracking module is responsible for detecting and tracking hand movements in real-time. It utilizes computer vision techniques and libraries such as OpenCV and mediapipe. The module processes input images or frames to detect the presence of hands and track their movements. The module also extracts hand landmarks, which represent key points on the hand, such as fingertips and joints.



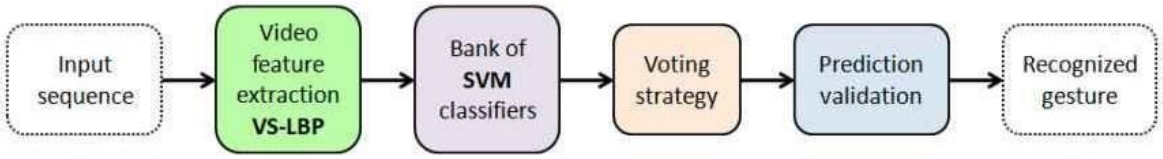


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Hand co-ordinates

3.7 Gesture Recognition Module

The gesture recognition module interprets the hand gestures detected by the hand tracking module and maps them to specific screen control actions. This module utilizes the finger positions and their configurations to recognize various gestures. The module employs machine learning techniques such as support vector machines (SVM) or deep learning algorithms for gesture classification.

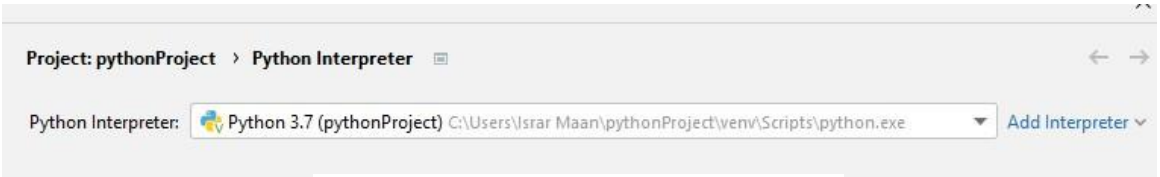


Chapter 4
Implementation
&
Testing

4.1 Tools and Language

- **Python 3.7 :**

Python programming language was the primary language used for developing the hand tracking mouse control system. Python offers simplicity, readability, and a vast array of
libraries that make it suitable for computer vision and machine learning tasks.



Python 3.7 as an interpreter

- **OpenCV:**

OpenCV (Open Source Computer Vision Library) is a popular open-source computer vision library used for image and video processing tasks. It provides various functions and algorithms for handling images, videos, and webcam inputs. In this project, OpenCV was used for video capture, image processing, and rendering.

- **Mediapipe:**

Mediapipe is a framework developed by Google that provides a comprehensive set of tools and algorithms for building applications involving real-time perception. The Hand Tracking Module used in your project is part of the Mediapipe library. It offers pre- trained models for hand detection and tracking, making it easy to extract hand landmarks and gestures.

- **Autopy:**

Autopy is a cross-platform library that allows interaction with the mouse, keyboard, and screen. It provides functionalities to control mouse movements, clicks, and keyboard inputs. In your project, Autopy was utilized for moving the mouse cursor and performing mouse clicks based on hand gestures.

- **Numpy:**

Numpy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. Numpy was used in your project for various mathematical calculations and array manipulations.

4.1.1 Libraries and their Purposes:

- OpenCV:

  Used for video capture, image processing, and rendering. It provides essential functions for handling image data, such as color space conversions, filtering, and drawing.

- **Mediapipe:**

  Integrated the Hand Tracking Module from Mediapipe, which offers pre-trained models for hand detection and tracking. It allows the extraction of hand landmarks and the ability to recognize hand gestures.

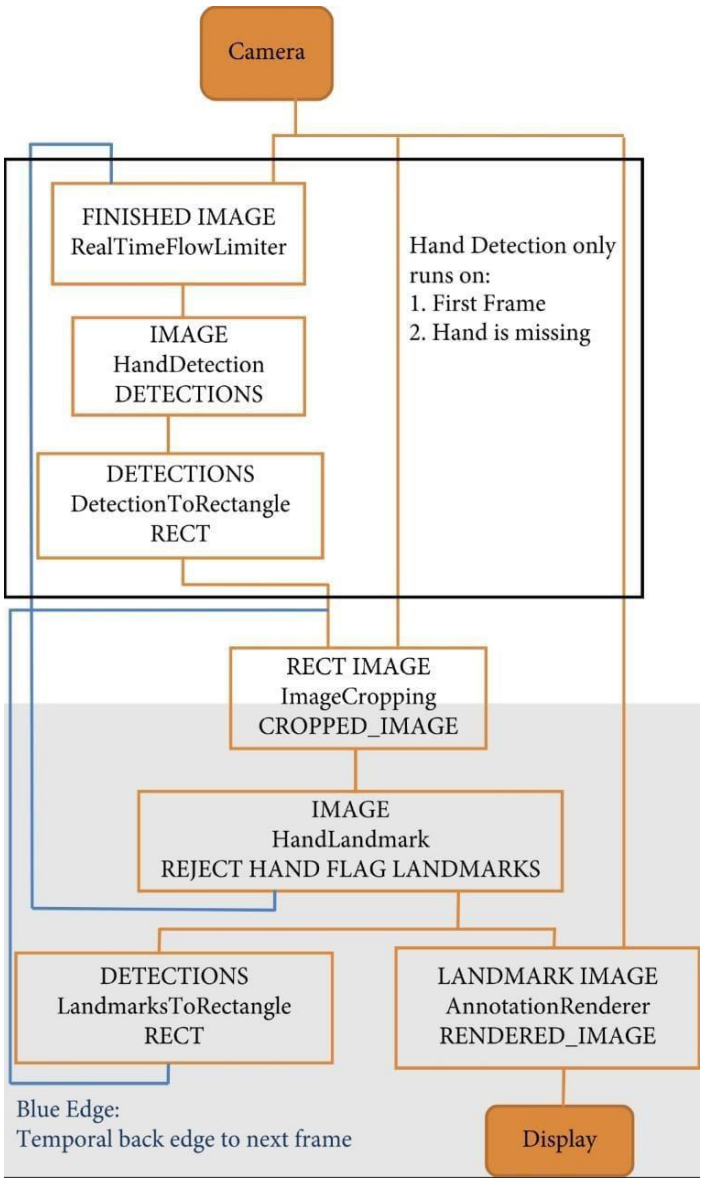


Fig: mediapipe hand recognition graph

- **Autopy:**

  Utilized Autopy library to control mouse movements and perform mouse clicks based on hand gestures. It provides an interface to interact with the mouse, keyboard, and screen.

- **Numpy:**

  Utilized Numpy for mathematical calculations, array manipulation, and handling multi-dimensional arrays. It simplifies complex mathematical operations in Python.

- **pyttsx3:**

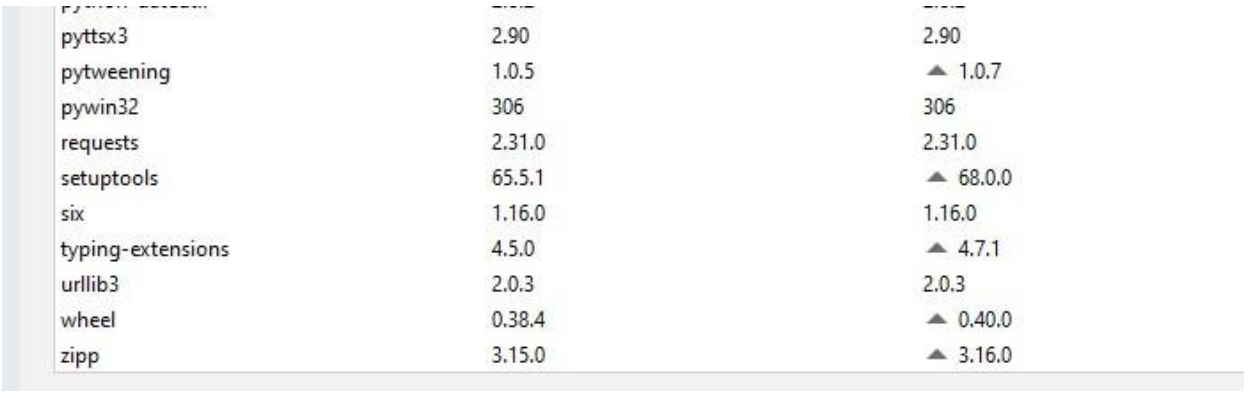
  Incorporated pyttsx3 library for text-to-speech synthesis, allowing the system to generate audio notifications or messages. It converts text-based messages into audible speech.

The combination of these tools, languages, and libraries enabled the development of an efficient hand tracking mouse control system. The integration of OpenCV and Mediapipe provided robust hand detection and tracking capabilities, while Autopy facilitated precise mouse control based on hand gestures. Numpy was instrumental in performing mathematical calculations, and pyttsx3 added audio notifications to enhance the user experience.

4.1.2 Python Packages table

Package	Version	Latest version
-lask	2.2.5	
HandTrackingModule	0.1	0.1
Jinja2	3.1.2	3.1.2
MarkupSafe	2.1.3	2.1.3
MouseInfo	0.1.3	0.1.3
Pillow	9.4.0	▲ 10.0.0
PyAutoGUI	0.9.53	▲ 0.9.54
PyGetWindow	0.0.9	0.0.9
PyMsgBox	1.0.9	1.0.9
PyQt5	5.15.9	5.15.9
PyQt5-Qt5	5.15.2	5.15.2
PyQt5-sip	12.12.1	12.12.1
PyRect	0.2.0	0.2.0
PyScreeze	0.1.28	▲ 0.1.29
SpeechRecognition	3.9.0	▲ 3.10.0
Werkzeug	2.2.3	▲ 2.3.6
absl-py	1.4.0	1.4.0
attrs	22.2.0	▲ 23.1.0
autopy	4.0.0	4.0.0
certifi	2023.5.7	2023.5.7
charset-normalizer	3.1.0	▲ 3.2.0
click	8.1.3	▲ 8.1.4
colorama	0.4.6	0.4.6

Package	Version	Latest version
comtypes	1.1.14	▲ 1.2.0
cycler	0.11.0	0.11.0
dataclasses	0.6	▲ 0.8
flatbuffers	23.1.21	▲ 23.5.26
fonttools	4.38.0	▲ 4.40.0
idna	3.4	3.4
importlib-metadata	6.7.0	▲ 6.8.0
itsdangerous	2.1.2	2.1.2
keyboard	0.13.5	0.13.5
kiwisolver	1.4.4	1.4.4
matplotlib	3.5.3	▲ 3.7.2
mediapipe	0.8.3.1	▲ 0.10.1
msvc-runtime	14.29.30133	▲ 14.34.31931
numpy	1.21.6	▲ 1.25.1
opencv-contrib-python	4.7.0.72	▲ 4.8.0.74
opencv-python	4.7.0.72	▲ 4.8.0.74
packaging	23.0	▲ 23.1
pip	23.1.2	23.1.2
protobuf	3.20.3	▲ 4.23.4
pyparsing	3.0.9	▲ 3.1.0
pyperclip	1.8.2	1.8.2
pypiwin32	223	223
python-dateutil	2.8.2	2.8.2



pyttsx3	2.90	2.90
pytweening	1.0.5	▲ 1.0.7
pywin32	306	306
requests	2.31.0	2.31.0
setuptools	65.5.1	▲ 68.0.0
six	1.16.0	1.16.0
typing-extensions	4.5.0	▲ 4.7.1
urllib3	2.0.3	2.0.3
wheel	0.38.4	▲ 0.40.0
zipp	3.15.0	▲ 3.16.0

All of these are the list of packages used in this project, here I also specify their latest versions and which version I used in this project.

4.2 Hand Tracking Module

4.2.1Code Overview

The Hand Tracking module is a crucial component of the project that enables the detection, tracking, and analysis of hand movements. It consists of a class called **handDetector** that encapsulates the functionality related to hand detection and tracking. The code is implemented using the OpenCV and Mediapipe libraries.
The **handDetector** class is initialized with parameters such as mode, maxHands, detectionCon, and trackCon, which control the behavior of the hand tracking algorithm. The **mpHands** and **mpDraw** objects are used from the Mediapipe library for hand tracking and landmark visualization, respectively. Additionally, the **tipIds** list is defined to identify specific landmarks on the hand.

4.2.2 Hand Detection and Tracking

In the **findHands** method of the **handDetector** class, the input image is processed to detect and track hands using the **hands.process** function. The processed results are then used to draw landmarks on the image if the hands are detected. The **draw_landmarks** function from **mpDraw** is employed for this purpose. The method returns the image with the drawn landmarks.

4.2.3 Landmark Detection and Visualization

The **findPosition** method in the **handDetector** class is responsible for extracting the positions of landmarks on the hand. It takes the processed image as input and retrieves the landmarks' coordinates by iterating through the **landmark** attribute of the detected hand. The x and y coordinates are normalized and converted to pixel values based on the image dimensions. These coordinates are stored in the **lmList** list.
Additionally, the method calculates the bounding box (bbox) that encompasses the hand by finding the minimum and maximum x and y values from the landmark coordinates. The bounding box is visualized on the image using the **rectangle** function from OpenCV. The method returns the **lmList** and the bounding box coordinates.

4.2.4 Finger Tracking and Gesture Recognition

The **fingersUp** method in the **handDetector** class determines which fingers are extended based on the landmark positions. It compares the y-coordinate of specific landmarks (stored in the **tipIds** list) to detect whether the corresponding fingers are up or down. The method returns a list of binary values indicating the state of each finger.

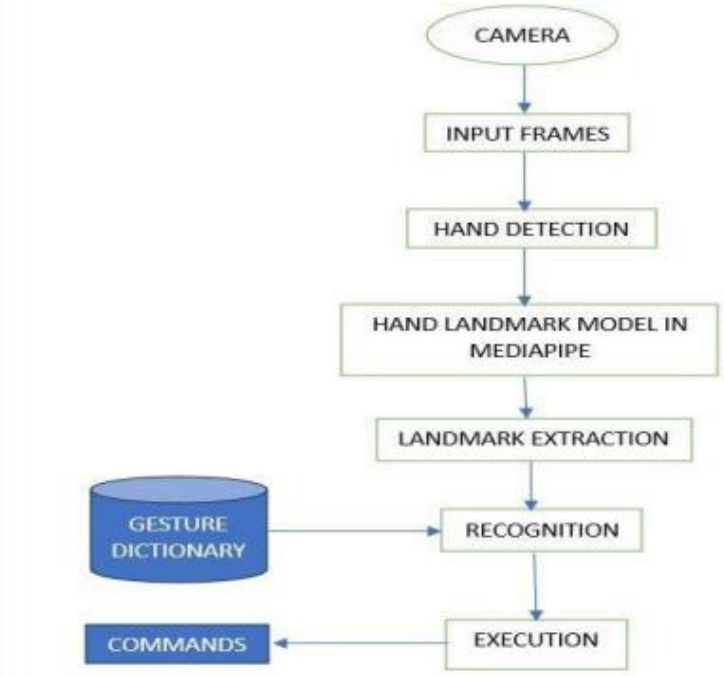
4.2.5Distance Calculation

The **findDistance** method in the **handDetector** class calculates the Euclidean distance between two specified landmarks on the hand. It takes the indices of the two landmarks, the image, and optional parameters for visualization (radius and line thickness). The method retrieves the coordinates of the landmarks from the **lmList** and calculates the distance using the **hypot** function from the math module. It also draws a line and circles representing the landmarks on the image if the **draw** parameter is set to true.

4.3 Virtual Mouse Implementation:

The Camera Used in the AI Virtual Mouse System :

The proposed system uses web camera for capturing images or video based on the frames. For capturing we are using CV library Opencv which is belongs to python web camera will start capturing the video and Opencv will create a object of video capture. To AI based virtual system the frames are passed from the captured web camera.



Capturing the Video and Processing:

The capturing of the frame was done with the AI virtual mouse system until the program termination.Then the video captured has to be processed to find the hands in the frame in each set. The processing takes places is it converts the BRG images into RGB images,which can be performed with the below code,

```
image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
image.flags.writeable = False
results = hands.process(image)
```

This code is used to flip the image in the horizontal direction then the resultant image is converted from the BRG scale to RGB scaled image.

Code:

Gesture_Controller.py

```
# Imports

import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from google.protobuf.json_format import MessageToDict
import screen_brightness_control as sbcontrol
```

```
pyautogui.FAILSAFE = False
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

# Gesture Encodings
class Gest(IntEnum):
    # Binary Encoded
    """
    Enum for mapping all hand gesture to binary number.
    """

    FIST = 0
    PINKY = 1
    RING = 2
    MID = 4
    LAST3 = 7
    INDEX = 8
    FIRST2 = 12
    LAST4 = 15
    THUMB = 16
    PALM = 31

    # Extra Mappings
    V_GEST = 33
    TWO_FINGER_CLOSED = 34
    PINCH_MAJOR = 35
    PINCH_MINOR = 36

# Multi-handedness Labels
class HLabel(IntEnum):
    MINOR = 0
    MAJOR = 1

# Convert Mediapipe Landmarks to recognizable Gestures
class HandRecog:
    """
    Convert Mediapipe Landmarks to recognizable Gestures.
    """

    def __init__(self, hand_label):
        """
        Constructs all the necessary attributes for the HandRecog object.

        Parameters
        ----------
            finger : int
                Represent gesture corresponding to Enum 'Gest',
                stores computed gesture for current frame.
            ori_gesture : int
                Represent gesture corresponding to Enum 'Gest',
                stores gesture being used.
            prev_gesture : int
                Represent gesture corresponding to Enum 'Gest',
                stores gesture computed for previous frame.
            frame_count : int
                total no. of frames since 'ori_gesture' is updated.
            hand_result : Object
                Landmarks obtained from mediapipe.
            hand_label : int
                Represents multi-handedness corresponding to Enum 'HLabel'.
        """

        self.finger = 0
```

```
        self.ori_gesture = Gest.PALM
        self.prev_gesture = Gest.PALM
        self.frame_count = 0
        self.hand_result = None
        self.hand_label = hand_label

    def update_hand_result(self, hand_result):
        self.hand_result = hand_result

    def get_signed_dist(self, point):
        """
        returns signed euclidean distance between 'point'.

        Parameters
        ----------
        point : list contaning two elements of type list/tuple which
represents
            landmark point.

        Returns
        -------
        float
        """
        sign = -1
        if self.hand_result.landmark[point[0]].y <
self.hand_result.landmark[point[1]].y:
            sign = 1
        dist = (self.hand_result.landmark[point[0]].x -
self.hand_result.landmark[point[1]].x)**2
        dist += (self.hand_result.landmark[point[0]].y -
self.hand_result.landmark[point[1]].y)**2
        dist = math.sqrt(dist)
        return dist*sign

    def get_dist(self, point):
        """
        returns euclidean distance between 'point'.

        Parameters
        ----------
        point : list contaning two elements of type list/tuple which
represents
            landmark point.

        Returns
        -------
        float
        """
        dist = (self.hand_result.landmark[point[0]].x -
self.hand_result.landmark[point[1]].x)**2
        dist += (self.hand_result.landmark[point[0]].y -
self.hand_result.landmark[point[1]].y)**2
        dist = math.sqrt(dist)
        return dist

    def get_dz(self,point):
        """
        returns absolute difference on z-axis between 'point'.

        Parameters
        ----------
        point : list contaning two elements of type list/tuple which
represents
```

```
            landmark point.

        Returns
        -------
        float
        """
        return abs(self.hand_result.landmark[point[0]].z -
self.hand_result.landmark[point[1]].z)

    # Function to find Gesture Encoding using current finger_state.
    # Finger_state: 1 if finger is open, else 0
    def set_finger_state(self):
        """
        set 'finger' by computing ratio of distance between finger tip
        , middle knuckle, base knuckle.

        Returns
        -------
        None
        """
        if self.hand_result == None:
            return

        points = [[8,5,0],[12,9,0],[16,13,0],[20,17,0]]
        self.finger = 0
        self.finger = self.finger | 0 #thumb
        for idx,point in enumerate(points):

            dist = self.get_signed_dist(point[:2])
            dist2 = self.get_signed_dist(point[1:])

            try:
                ratio = round(dist/dist2,1)
            except:
                ratio = round(dist1/0.01,1)

            self.finger = self.finger << 1
            if ratio > 0.5 :
                self.finger = self.finger | 1


    # Handling Fluctations due to noise
    def get_gesture(self):
        """
        returns int representing gesture corresponding to Enum 'Gest'.
        sets 'frame_count', 'ori_gesture', 'prev_gesture',
        handles fluctations due to noise.

        Returns
        -------
        int
        """
        if self.hand_result == None:
            return Gest.PALM

        current_gesture = Gest.PALM
        if self.finger in [Gest.LAST3,Gest.LAST4] and self.get_dist([8,4])
< 0.05:
            if self.hand_label == HLabel.MINOR :
                current_gesture = Gest.PINCH_MINOR
            else:
                current_gesture = Gest.PINCH_MAJOR
```

```
        elif Gest.FIRST2 == self.finger :
            point = [[8,12],[5,9]]
            dist1 = self.get_dist(point[0])
            dist2 = self.get_dist(point[1])
            ratio = dist1/dist2
            if ratio > 1.7:
                current_gesture = Gest.V_GEST
            else:
                if self.get_dz([8,12]) < 0.1:
                    current_gesture =  Gest.TWO_FINGER_CLOSED
                else:
                    current_gesture =  Gest.MID

        else:
            current_gesture =  self.finger

        if current_gesture == self.prev_gesture:
            self.frame_count += 1
        else:
            self.frame_count = 0

        self.prev_gesture = current_gesture

        if self.frame_count > 4 :
            self.ori_gesture = current_gesture
        return self.ori_gesture

# Executes commands according to detected gestures
class Controller:
    """
    Executes commands according to detected gestures.

    Attributes
    ----------
    tx_old : int
        previous mouse location x coordinate
    ty_old : int
        previous mouse location y coordinate
    flag : bool
        true if V gesture is detected
    grabflag : bool
        true if FIST gesture is detected
    pinchmajorflag : bool
        true if PINCH gesture is detected through MAJOR hand,
        on x-axis 'Controller.changesystembrightness',
        on y-axis 'Controller.changesystemvolume'.
    pinchminorflag : bool
        true if PINCH gesture is detected through MINOR hand,
        on x-axis 'Controller.scrollHorizontal',
        on y-axis 'Controller.scrollVertical'.
    pinchstartxcoord : int
        x coordinate of hand landmark when pinch gesture is started.
    pinchstartycoord : int
        y coordinate of hand landmark when pinch gesture is started.
    pinchdirectionflag : bool
        true if pinch gesture movment is along x-axis,
        otherwise false
    prevpinchlv : int
        stores quantized magnitued of prev pinch gesture displacment, from
        starting position
    pinchlv : int
        stores quantized magnitued of pinch gesture displacment, from
        starting position
```

```
    framecount : int
        stores no. of frames since 'pinchlv' is updated.
    prev_hand : tuple
        stores (x, y) coordinates of hand in previous frame.
    pinch_threshold : float
        step size for quantization of 'pinchlv'.
    """

    tx_old = 0
    ty_old = 0
    trial = True
    flag = False
    grabflag = False
    pinchmajorflag = False
    pinchminorflag = False
    pinchstartxcoord = None
    pinchstartycoord = None
    pinchdirectionflag = None
    prevpinchlv = 0
    pinchlv = 0
    framecount = 0
    prev_hand = None
    pinch_threshold = 0.3

    def getpinchylv(hand_result):
        """returns distance beween starting pinch y coord and current hand
position y coord."""
        dist = round((Controller.pinchstartycoord -
hand_result.landmark[8].y)*10,1)
        return dist

    def getpinchxlv(hand_result):
        """returns distance beween starting pinch x coord and current hand
position x coord."""
        dist = round((hand_result.landmark[8].x -
Controller.pinchstartxcoord)*10,1)
        return dist

    def changesystembrightness():
        """sets system brightness based on 'Controller.pinchlv'."""
        currentBrightnessLv = sbcontrol.get_brightness(display=0)/100.0
        currentBrightnessLv += Controller.pinchlv/50.0
        if currentBrightnessLv > 1.0:
            currentBrightnessLv = 1.0
        elif currentBrightnessLv < 0.0:
            currentBrightnessLv = 0.0
        sbcontrol.fade_brightness(int(100*currentBrightnessLv) , start =
sbcontrol.get_brightness(display=0))

    def changesystemvolume():
        """sets system volume based on 'Controller.pinchlv'."""
        devices = AudioUtilities.GetSpeakers()
        interface = devices.Activate(IAudioEndpointVolume._iid_,
CLSCTX_ALL, None)
        volume = cast(interface, POINTER(IAudioEndpointVolume))
        currentVolumeLv = volume.GetMasterVolumeLevelScalar()
        currentVolumeLv += Controller.pinchlv/50.0
        if currentVolumeLv > 1.0:
            currentVolumeLv = 1.0
        elif currentVolumeLv < 0.0:
            currentVolumeLv = 0.0
        volume.SetMasterVolumeLevelScalar(currentVolumeLv, None)
```

```
    def scrollVertical():
        """scrolls on screen vertically."""
        pyautogui.scroll(120 if Controller.pinchlv>0.0 else -120)


    def scrollHorizontal():
        """scrolls on screen horizontally."""
        pyautogui.keyDown('shift')
        pyautogui.keyDown('ctrl')
        pyautogui.scroll(-120 if Controller.pinchlv>0.0 else 120)
        pyautogui.keyUp('ctrl')
        pyautogui.keyUp('shift')

    # Locate Hand to get Cursor Position
    # Stabilize cursor by Dampening
    def get_position(hand_result):
        """
        returns coordinates of current hand position.

        Locates hand to get cursor position also stabilize cursor by
        dampening jerky motion of hand.

        Returns
        -------
        tuple(float, float)
        """
        point = 9
        position = [hand_result.landmark[point].x
,hand_result.landmark[point].y]
        sx,sy = pyautogui.size()
        x_old,y_old = pyautogui.position()
        x = int(position[0]*sx)
        y = int(position[1]*sy)
        if Controller.prev_hand is None:
            Controller.prev_hand = x,y
        delta_x = x - Controller.prev_hand[0]
        delta_y = y - Controller.prev_hand[1]

        distsq = delta_x**2 + delta_y**2
        ratio = 1
        Controller.prev_hand = [x,y]

        if distsq <= 25:
            ratio = 0
        elif distsq <= 900:
            ratio = 0.07 * (distsq ** (1/2))
        else:
            ratio = 2.1
        x , y = x_old + delta_x*ratio , y_old + delta_y*ratio
        return (x,y)

    def pinch_control_init(hand_result):
        """Initializes attributes for pinch gesture."""
        Controller.pinchstartxcoord = hand_result.landmark[8].x
        Controller.pinchstartycoord = hand_result.landmark[8].y
        Controller.pinchlv = 0
        Controller.prevpinchlv = 0
        Controller.framecount = 0

    # Hold final position for 5 frames to change status
    def pinch_control(hand_result, controlHorizontal, controlVertical):
        """
```

```
        calls 'controlHorizontal' or 'controlVertical' based on pinch
flags,
        'framecount' and sets 'pinchlv'.

        Parameters
        ----------
        hand_result : Object
            Landmarks obtained from mediapipe.
        controlHorizontal : callback function assosiated with horizontal
            pinch gesture.
        controlVertical : callback function assosiated with vertical
            pinch gesture.

        Returns
        -------
        None
        """
        if Controller.framecount == 5:
            Controller.framecount = 0
            Controller.pinchlv = Controller.prevpinchlv

            if Controller.pinchdirectionflag == True:
                controlHorizontal() #x

            elif Controller.pinchdirectionflag == False:
                controlVertical() #y

        lvx =  Controller.getpinchxlv(hand_result)
        lvy =  Controller.getpinchylv(hand_result)

        if abs(lvy) > abs(lvx) and abs(lvy) > Controller.pinch_threshold:
            Controller.pinchdirectionflag = False
            if abs(Controller.prevpinchlv - lvy) <
Controller.pinch_threshold:
                Controller.framecount += 1
            else:
                Controller.prevpinchlv = lvy
                Controller.framecount = 0

        elif abs(lvx) > Controller.pinch_threshold:
            Controller.pinchdirectionflag = True
            if abs(Controller.prevpinchlv - lvx) <
Controller.pinch_threshold:
                Controller.framecount += 1
            else:
                Controller.prevpinchlv = lvx
                Controller.framecount = 0

    def handle_controls(gesture, hand_result):
        """Impliments all gesture functionality."""
        x,y = None,None
        if gesture != Gest.PALM :
            x,y = Controller.get_position(hand_result)

        # flag reset
        if gesture != Gest.FIST and Controller.grabflag:
            Controller.grabflag = False
            pyautogui.mouseUp(button = "left")

        if gesture != Gest.PINCH_MAJOR and Controller.pinchmajorflag:
            Controller.pinchmajorflag = False

        if gesture != Gest.PINCH_MINOR and Controller.pinchminorflag:
```

```
            Controller.pinchminorflag = False

        # implementation
        if gesture == Gest.V_GEST:
            Controller.flag = True
            pyautogui.moveTo(x, y, duration = 0.1)

        elif gesture == Gest.FIST:
            if not Controller.grabflag :
                Controller.grabflag = True
                pyautogui.mouseDown(button = "left")
            pyautogui.moveTo(x, y, duration = 0.1)

        elif gesture == Gest.MID and Controller.flag:
            pyautogui.click()
            Controller.flag = False

        elif gesture == Gest.INDEX and Controller.flag:
            pyautogui.click(button='right')
            Controller.flag = False

        elif gesture == Gest.TWO_FINGER_CLOSED and Controller.flag:
            pyautogui.doubleClick()
            Controller.flag = False

        elif gesture == Gest.PINCH_MINOR:
            if Controller.pinchminorflag == False:
                Controller.pinch_control_init(hand_result)
                Controller.pinchminorflag = True
            Controller.pinch_control(hand_result,Controller.scrollHorizonta
l, Controller.scrollVertical)

        elif gesture == Gest.PINCH_MAJOR:
            if Controller.pinchmajorflag == False:
                Controller.pinch_control_init(hand_result)
                Controller.pinchmajorflag = True
            Controller.pinch_control(hand_result,Controller.changesystembri
ghtness, Controller.changesystemvolume)

'''
---------------------------------------  Main Class  ----------------------
-------------------
    Entry point of Gesture Controller
'''


class GestureController:
    """
    Handles camera, obtain landmarks from mediapipe, entry point
    for whole program.

    Attributes
    ----------
    gc_mode : int
        indicates weather gesture controller is running or not,
        1 if running, otherwise 0.
    cap : Object
        object obtained from cv2, for capturing video frame.
    CAM_HEIGHT : int
        highet in pixels of obtained frame from camera.
    CAM_WIDTH : int
        width in pixels of obtained frame from camera.
    hr_major : Object of 'HandRecog'
```

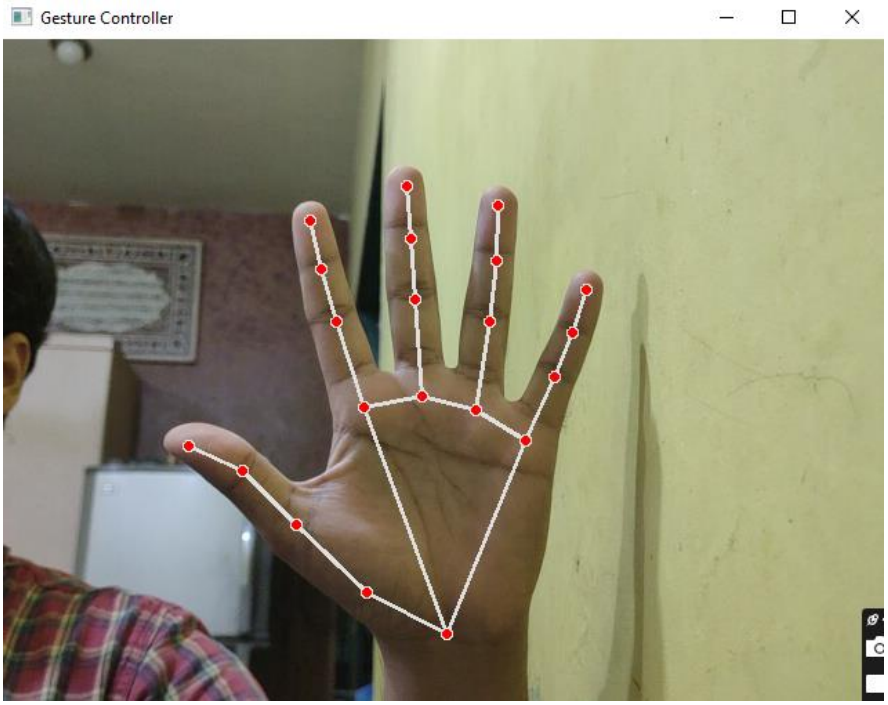
```
        object representing major hand.
    hr_minor : Object of 'HandRecog'
        object representing minor hand.
    dom_hand : bool
        True if right hand is domaniant hand, otherwise False.
        default True.
    """
    gc_mode = 0
    cap = None
    CAM_HEIGHT = None
    CAM_WIDTH = None
    hr_major = None # Right Hand by default
    hr_minor = None # Left hand by default
    dom_hand = True

    def __init__(self):
        """Initilaizes attributes."""
        GestureController.gc_mode = 1
        GestureController.cap = cv2.VideoCapture(0)
        GestureController.CAM_HEIGHT =
GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
        GestureController.CAM_WIDTH =
GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH)

    def classify_hands(results):
        """
        sets 'hr_major', 'hr_minor' based on classification(left, right) of
        hand obtained from mediapipe, uses 'dom_hand' to decide major and
        minor hand.
        """
        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):
        """
        Entry point of whole programm, caputres video frame and passes,
obtains
        landmark from mediapipe and passes it to 'handmajor' and
'handminor' for
        controlling.
```

```
        """

        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()
```

Test Cases for Gesture Controlled Virtual Mouse

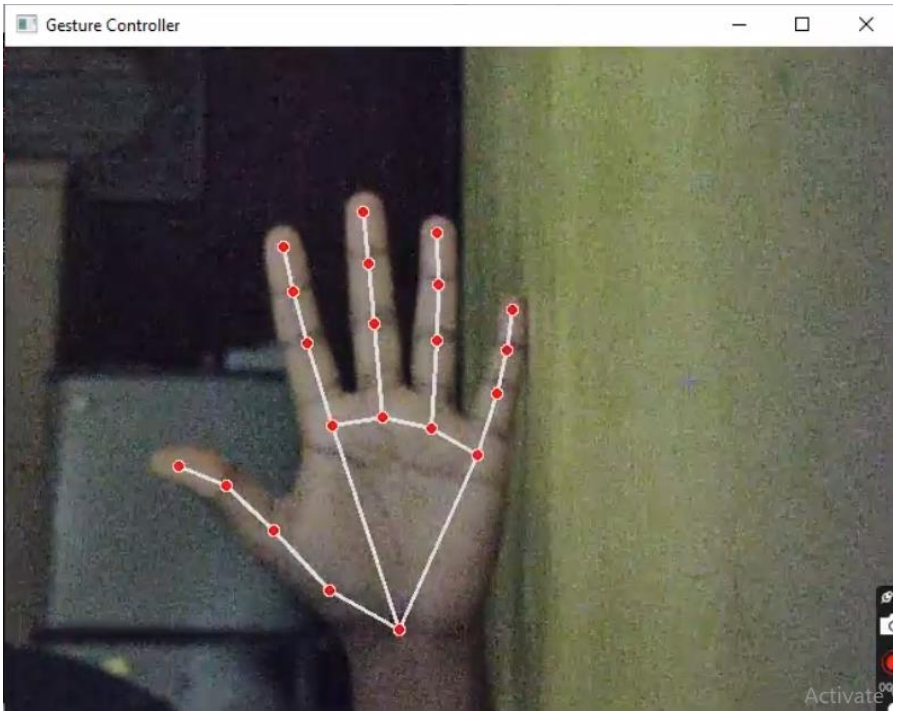
Test Case 1: Hand Detection in Various Lighting Conditions

- **Objective**: Verify hand detection accuracy in different lighting conditions.
- **Preconditions**: Proper lighting setup.
- **Steps**:
    1. Set up the camera in a well-lit room.



    2. Perform gestures in front of the camera.

3. Record the detection accuracy.
4. Repeat steps in a dimly-lit room



- **Expected Result**: Hand detection should be accurate in various lighting conditions.

Test Case 2: Gesture Recognition for Cursor Movement

- **Objective**: Validate gesture recognition for moving the cursor.
- **Preconditions**: System is initialized and hand detection is active.
- **Steps**:
    1. Perform the gesture for moving the cursor

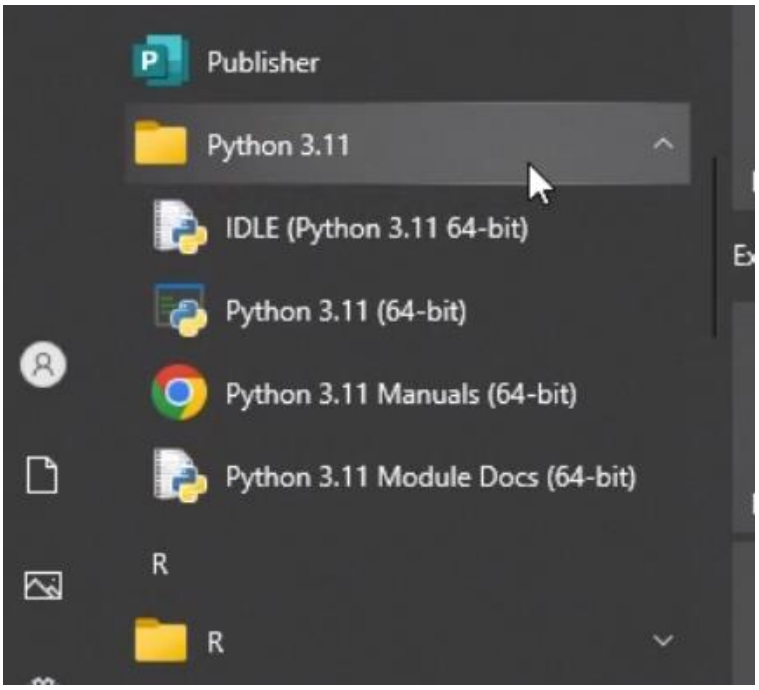
2. Observe the cursor movement on the screen.

- **Expected Result**: Cursor should move according to the hand gesture.

Test Case 3: Gesture Recognition for Click Action

- **Objective**: Verify the recognition of a clicking gesture.
- **Preconditions**: System is initialized and hand detection is active.
- **Steps**:
    1. Perform the gesture for a left-click.

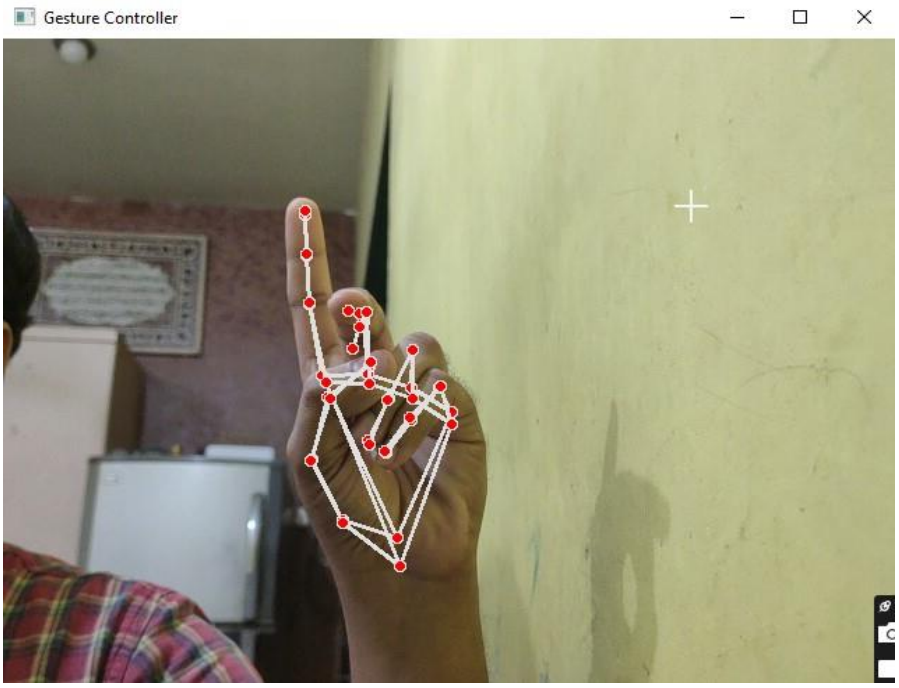
    2. Observe if the click action is performed on the screen.



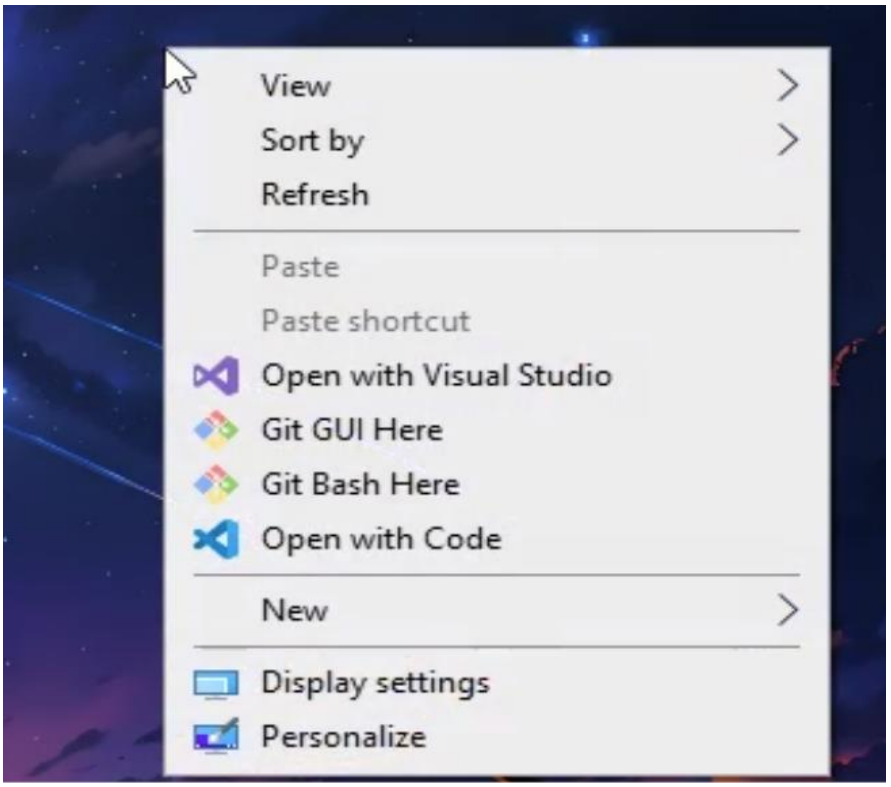
- **Expected Result**: The system should accurately recognize the click gesture and perform the click action.

Test Case 4: Gesture Recognition for Right-Click Action

- **Objective**: Verify the recognition of a right-click gesture.
- **Preconditions**: System is initialized and hand detection is active.
- **Steps**:
    1. Perform the gesture for a right-click.



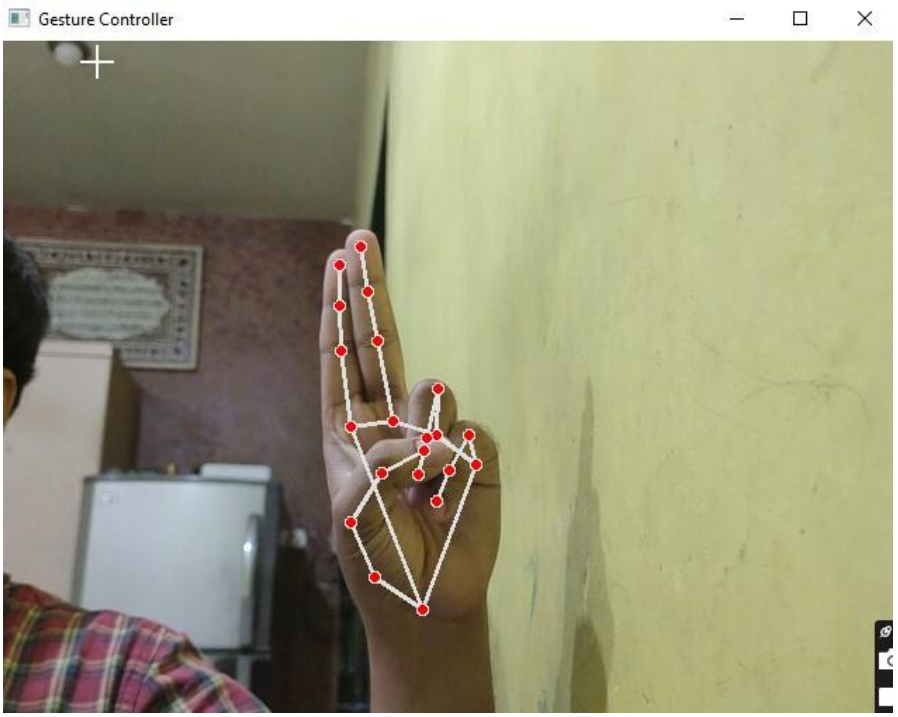
    2. Observe if the right-click action is performed on the screen.



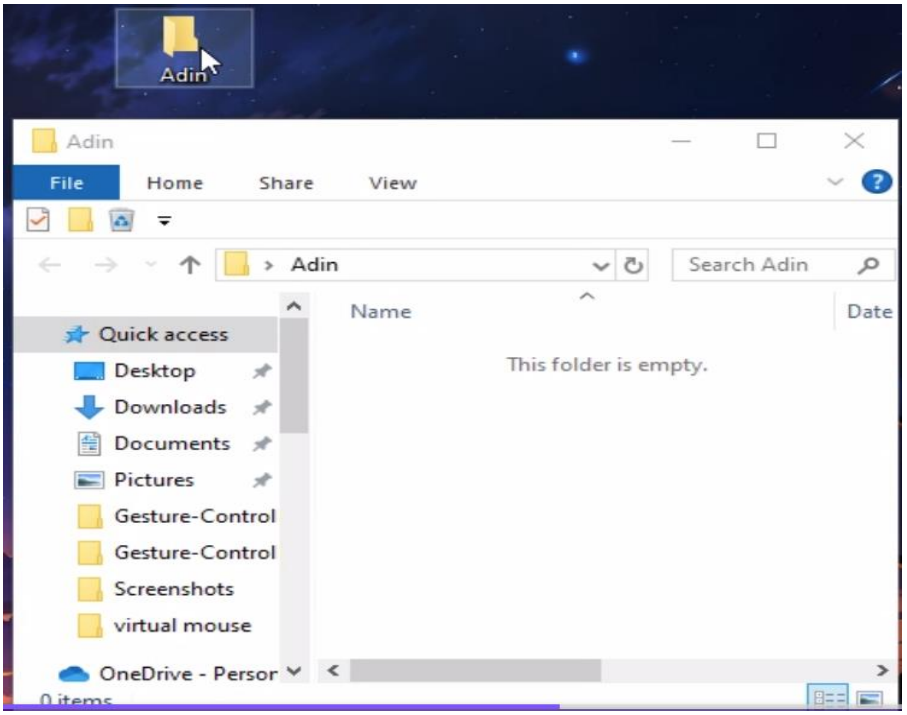
- **Expected Result**: The system should accurately recognize the right-click gesture and perform the right-click action.

Test Case 5: Double Click Gesture

- **Objective**: Validate the gesture for performing a double click.
- **Preconditions**: System is initialized, webcam is active, and hand detection is running.
- **Steps**:
    1. Position your hand in front of the webcam.
    2. Perform the double click gesture.



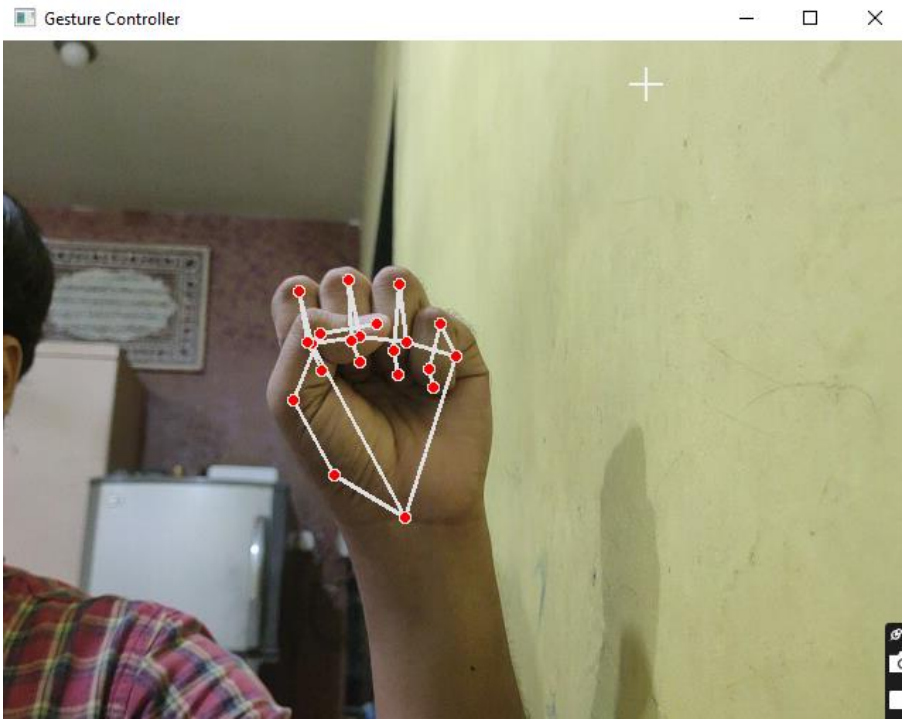
3. Observe if the system performs a double click action on the screen.



- **Expected Result**: The system should accurately recognize the double click gesture and perform a double click action.

Test Case 6: Multiple Item Selection Gesture

- **Objective**: Validate the gesture for selecting multiple items.
- **Preconditions**: System is initialized, webcam is active, and hand detection is running.
- **Steps**:
    1. Position your hand in front of the webcam.



    2. Perform the gesture for multiple item selection.
    3. Observe the selection of multiple items on the screen.

- **Expected Result**: The system should accurately recognize the multiple item selection gesture and select multiple items on the screen.

4.4 **Voice Assistant Implementation**

Code:

```
import pyttsx3
import speech_recognition as sr
from datetime import date
import time
import webbrowser
import datetime
from pynput.keyboard import Key, Controller
import pyautogui
import sys
import os
from os import listdir
from os.path import isfile, join
import smtplib
import wikipedia
import Gesture_Controller
#import Gesture_Controller_Gloved as Gesture_Controller
import app
from threading import Thread


# -------------Object Initialization---------------
today = date.today()
r = sr.Recognizer()
keyboard = Controller()
engine = pyttsx3.init('sapi5')
engine = pyttsx3.init()
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[0].id)

# -----------------Variables------------------------
file_exp_status = False
files =[]
path = ''
is_awake = True  #Bot status

# ------------------Functions----------------------
def reply(audio):
    app.ChatBot.addAppMsg(audio)

    print(audio)
    engine.say(audio)
    engine.runAndWait()


def wish():
    hour = int(datetime.datetime.now().hour)

    if hour>=0 and hour<12:
        reply("Good Morning!")
    elif hour>=12 and hour<18:
        reply("Good Afternoon!")
    else:
        reply("Good Evening!")

    reply("I am Proton, how may I help you?")

# Set Microphone parameters
with sr.Microphone() as source:
        r.energy_threshold = 500
        r.dynamic_energy_threshold = False
```

```
# Audio to String
def record_audio():
    with sr.Microphone() as source:
        r.pause_threshold = 0.8
        voice_data = ''
        audio = r.listen(source, phrase_time_limit=5)

        try:
            voice_data = r.recognize_google(audio)
        except sr.RequestError:
            reply('Sorry my Service is down. Plz check your Internet
connection')
        except sr.UnknownValueError:
            print('cant recognize')
            pass
        return voice_data.lower()


# Executes Commands (input: string)
def respond(voice_data):
    global file_exp_status, files, is_awake, path
    print(voice_data)
    voice_data.replace('proton','')
    app.eel.addUserMsg(voice_data)

    if is_awake==False:
        if 'wake up' in voice_data:
            is_awake = True
            wish()

    # STATIC CONTROLS
    elif 'hello' in voice_data:
        wish()

    elif 'what is your name' in voice_data:
        reply('My name is Proton!')

    elif 'date' in voice_data:
        reply(today.strftime("%B %d, %Y"))

    elif 'time' in voice_data:
        reply(str(datetime.datetime.now()).split(" ")[1].split('.')[0])

    elif 'search' in voice_data:
        reply('Searching for ' + voice_data.split('search')[1])
        url = 'https://google.com/search?q=' +
voice_data.split('search')[1]
        try:
            webbrowser.get().open(url)
            reply('This is what I found Sir')
        except:
            reply('Please check your Internet')

    elif 'location' in voice_data:
        reply('Which place are you looking for ?')
        temp_audio = record_audio()
        app.eel.addUserMsg(temp_audio)
        reply('Locating...')
        url = 'https://google.nl/maps/place/' + temp_audio + '/&amp;'
        try:
            webbrowser.get().open(url)
            reply('This is what I found Sir')
```

```
        except:
            reply('Please check your Internet')

    elif ('bye' in voice_data) or ('by' in voice_data):
        reply("Good bye Sir! Have a nice day.")
        is_awake = False

    elif ('exit' in voice_data) or ('terminate' in voice_data):
        if Gesture_Controller.GestureController.gc_mode:
            Gesture_Controller.GestureController.gc_mode = 0
        app.ChatBot.close()
        #sys.exit() always raises SystemExit, Handle it in main loop
        sys.exit()


    # DYNAMIC CONTROLS
    elif 'launch gesture recognition' in voice_data:
        if Gesture_Controller.GestureController.gc_mode:
            reply('Gesture recognition is already active')
        else:
            gc = Gesture_Controller.GestureController()
            t = Thread(target = gc.start)
            t.start()
            reply('Launched Successfully')

    elif ('stop gesture recognition' in voice_data) or ('top gesture
recognition' in voice_data):
        if Gesture_Controller.GestureController.gc_mode:
            Gesture_Controller.GestureController.gc_mode = 0
            reply('Gesture recognition stopped')
        else:
            reply('Gesture recognition is already inactive')

    elif 'copy' in voice_data:
        with keyboard.pressed(Key.ctrl):
            keyboard.press('c')
            keyboard.release('c')
        reply('Copied')

    elif 'page' in voice_data or 'pest'  in voice_data or 'paste' in
voice_data:
        with keyboard.pressed(Key.ctrl):
            keyboard.press('v')
            keyboard.release('v')
        reply('Pasted')

    # File Navigation (Default Folder set to C://)
    elif 'list' in voice_data:
        counter = 0
        path = 'C://'
        files = listdir(path)
        filestr = ""
        for f in files:
            counter+=1
            print(str(counter) + ':  ' + f)
            filestr += str(counter) + ':  ' + f + '<br>'
        file_exp_status = True
        reply('These are the files in your root directory')
        app.ChatBot.addAppMsg(filestr)

    elif file_exp_status == True:
        counter = 0
        if 'open' in voice_data:
```

```
            if isfile(join(path,files[int(voice_data.split(' ')[-1])-1])):
                os.startfile(path + files[int(voice_data.split(' ')[-1])-
1])
                file_exp_status = False
            else:
                try:
                    path = path + files[int(voice_data.split(' ')[-1])-1] +
'//'
                    files = listdir(path)
                    filestr = ""
                    for f in files:
                        counter+=1
                        filestr += str(counter) + ':  ' + f + '<br>'
                        print(str(counter) + ':  ' + f)
                    reply('Opened Successfully')
                    app.ChatBot.addAppMsg(filestr)

                except:
                    reply('You do not have permission to access this
folder')

        if 'back' in voice_data:
            filestr = ""
            if path == 'C://':
                reply('Sorry, this is the root directory')
            else:
                a = path.split('//')[:-2]
                path = '//'.join(a)
                path += '//'
                files = listdir(path)
                for f in files:
                    counter+=1
                    filestr += str(counter) + ':  ' + f + '<br>'
                    print(str(counter) + ':  ' + f)
                reply('ok')
                app.ChatBot.addAppMsg(filestr)

    else:
        reply('I am not functioned to do this !')

# ------------------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:
            #Handle sys.exit()
```

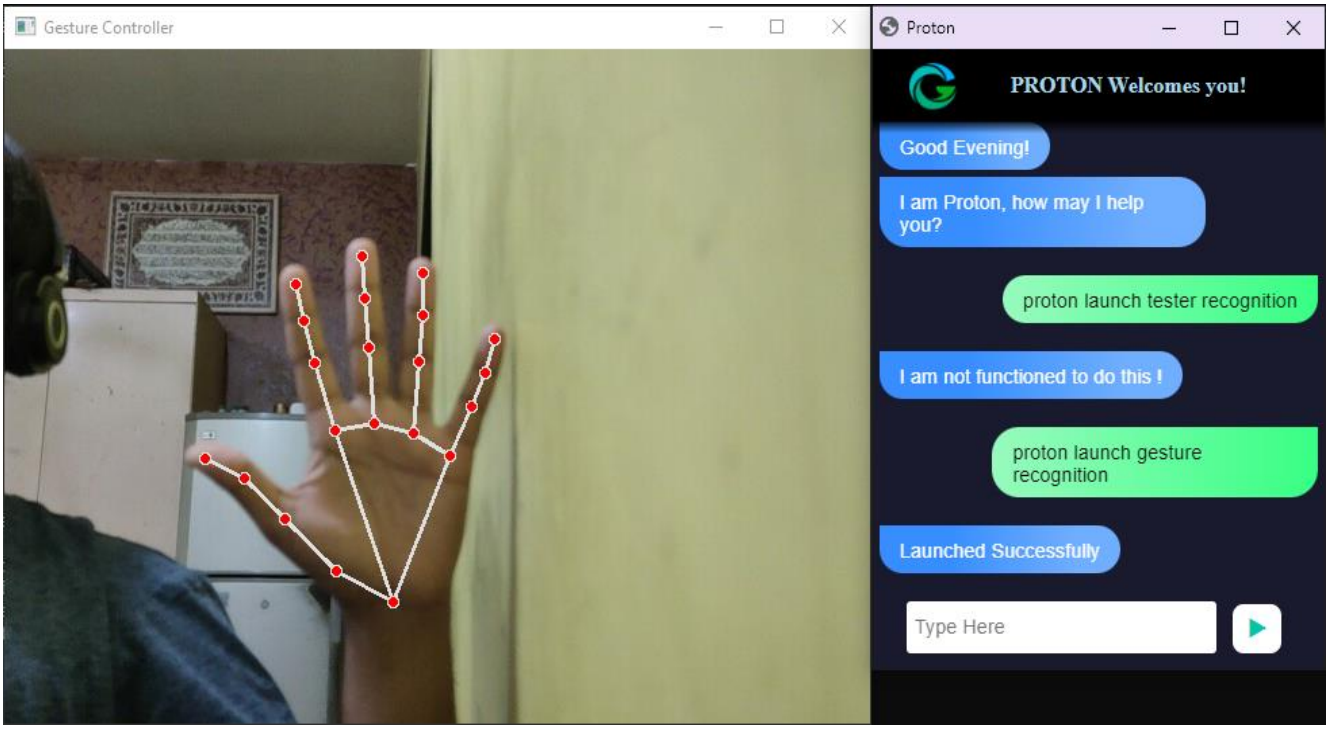
```
            respond(voice_data)
        except SystemExit:
            reply("Exit Successfull")
            break
        except:
            #some other exception got raised
            print("EXCEPTION raised while closing.")
            break
```

Test Cases for Voice Assistant:

Below are the detailed test cases for each of the specified voice commands, including where to insert images for illustration. You can add actual images in your document or presentation as needed.

Test Case 1: Launch Gesture Recognition

- **Objective**: Validate the voice command to start gesture recognition.
- **Preconditions**: System is initialized and the microphone is active.
- **Steps**:
    1. Issue the voice command "Proton Launch Gesture Recognition".
    2. Observe if the webcam turns on and starts recognizing hand gestures.

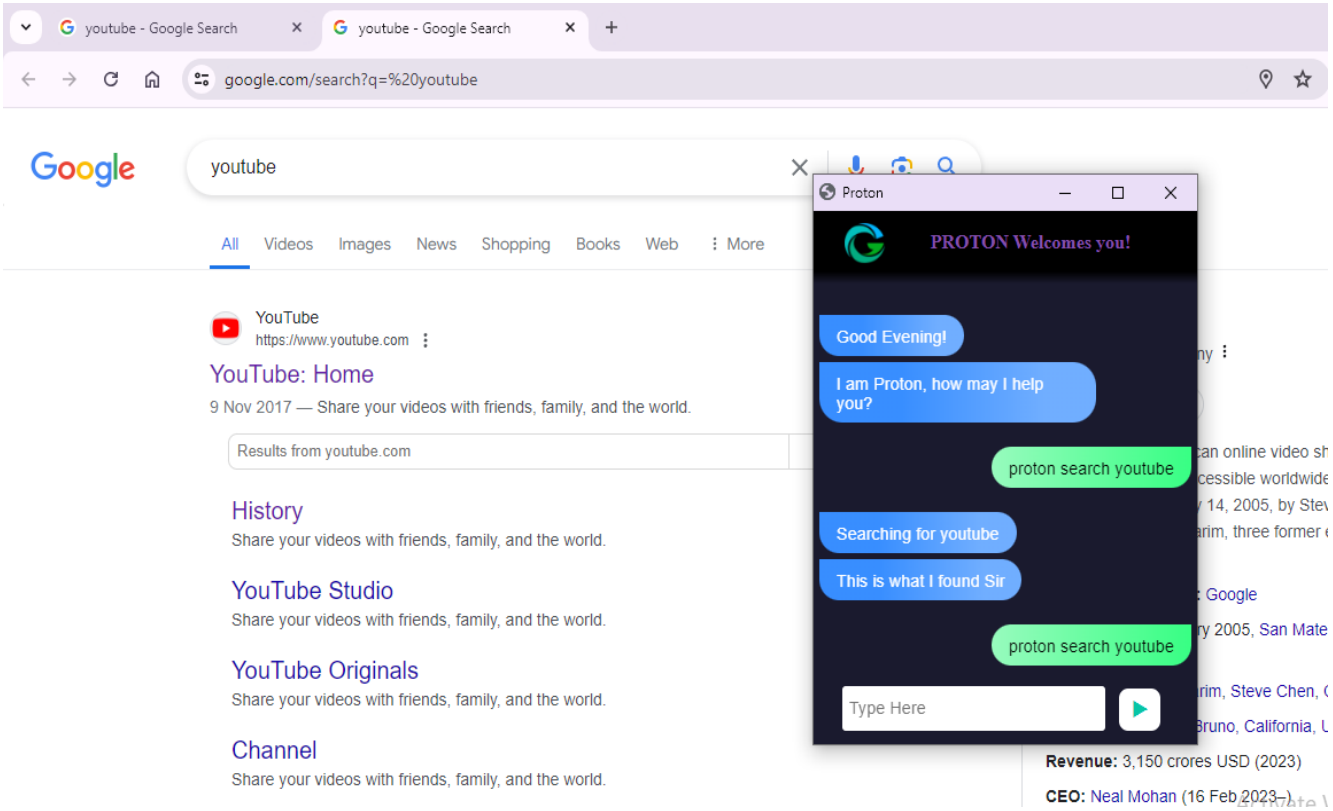


- **Expected Result**: The webcam should turn on and start recognizing hand gestures.

Test Case 2: Google Search

- **Objective**: Validate the voice command to search on Google.
- **Preconditions**: System is initialized and the microphone is active.

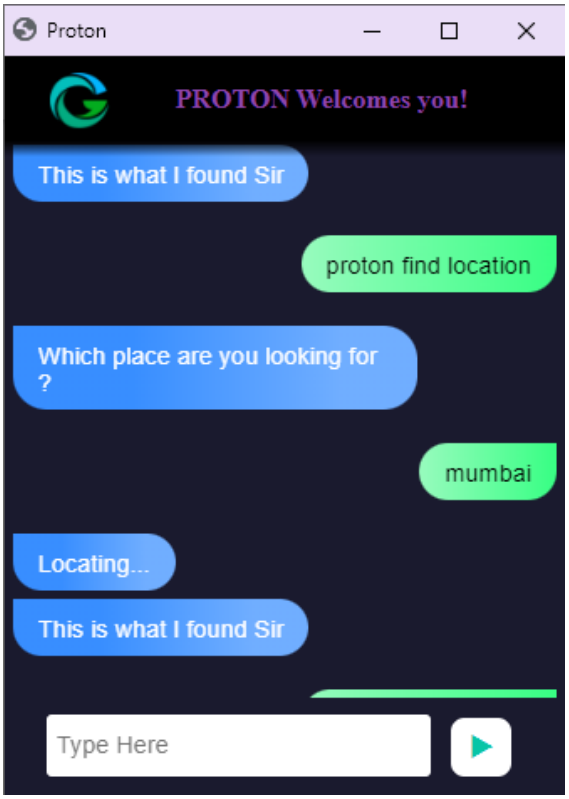
- **Steps**:
    1. Issue the voice command "Proton search {text_you_wish_to_search}" (e.g., "Proton search Youtube").
    2. Observe if a new tab or window opens in Chrome and performs the search.



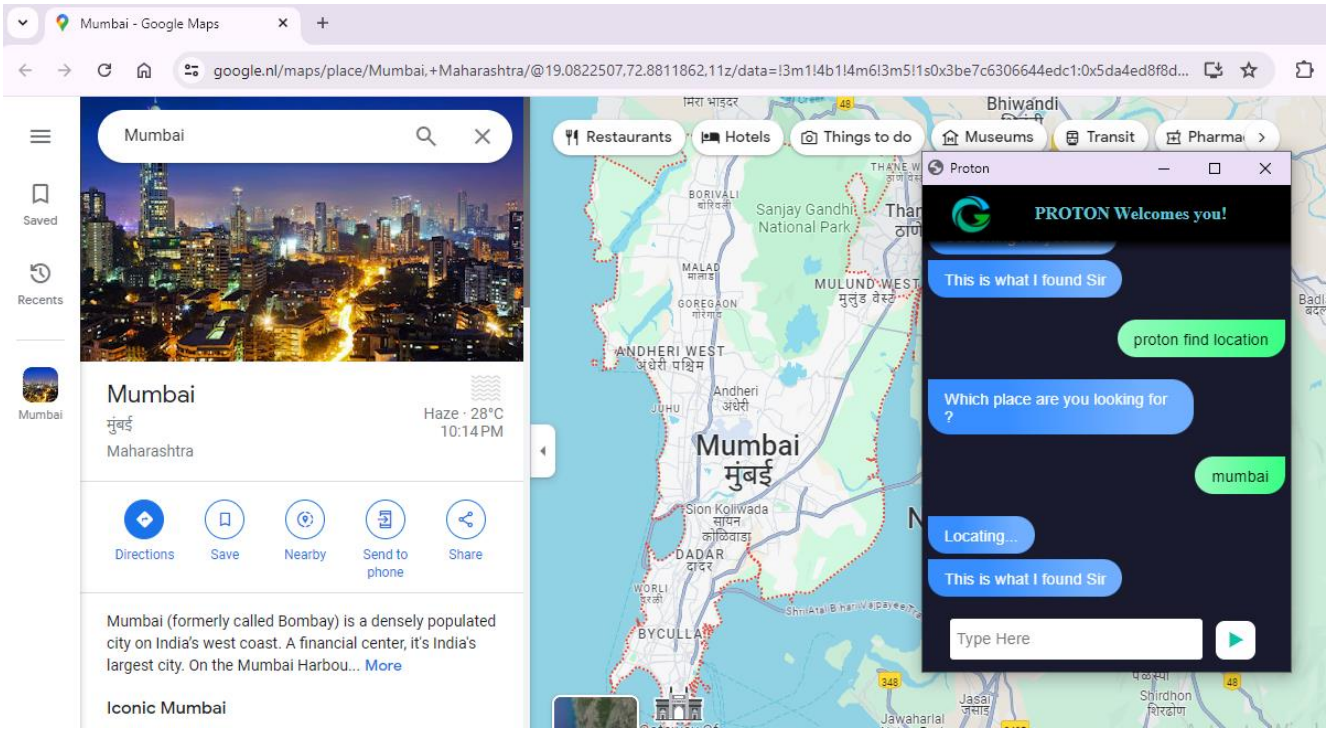
- **Expected Result**: A new tab or window should open in Chrome and display the search results.

Test Case 3: Find a Location on Google Maps

- **Objective**: Validate the voice command to find a location on Google Maps.
- **Preconditions**: System is initialized and the microphone is active.
- **Steps**:
    1. Issue the voice command "Proton Find a Location".
    2. Respond to the prompt with the location (e.g., "Mumbai").



3. Observe if a new tab in Chrome opens with the location on Google Maps.

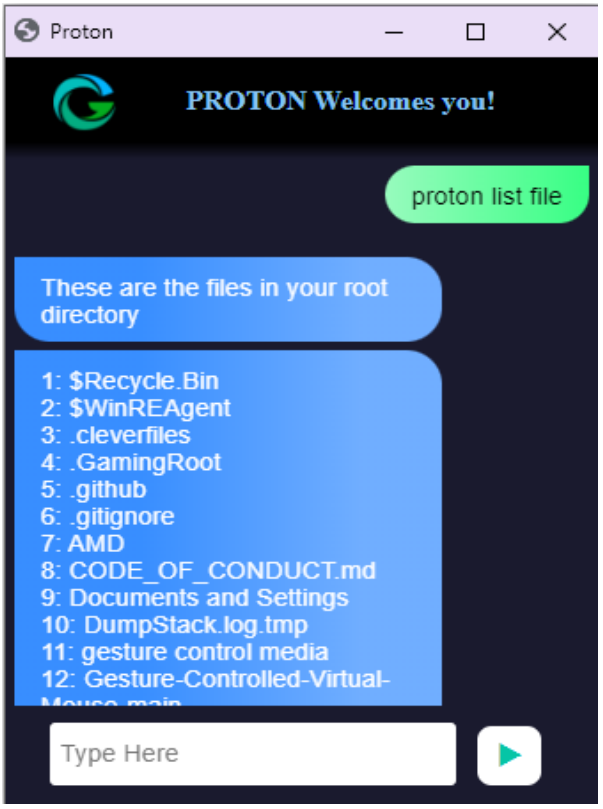


- **Expected Result**: A new tab should open in Chrome displaying the location on Google Maps.

Test Case 4: List Files

- **Objective**: Validate the voice command to list files in the current directory.

- **Preconditions**: System is initialized and the microphone is active.
- **Steps**:
    1. Issue the voice command "Proton list files".
    2. Observe if the system lists the files and their respective file numbers.



- **Expected Result**: The system should list the files and their respective file numbers.

Test Case 5: Open a File

- **Objective**: Validate the voice command to open a file by its number.
- **Preconditions**: Files are listed in the current directory.
- **Steps**:
    1. Issue the voice command "Proton open {file_number}" (e.g., "Proton open 13").
    2. Observe if the specified file or directory opens.

- **Expected Result**: The specified file or directory should open.

Test Case 6: Current Date

- **Objective**: Validate the voice command to get the current date.
- **Preconditions**: System is initialized and the microphone is active.
- **Steps**:
    1. Issue the voice command "Proton what is today's date" or "Proton date".
    2. Observe if the system returns the current date.

- **Expected Result**: The system should return the current date.

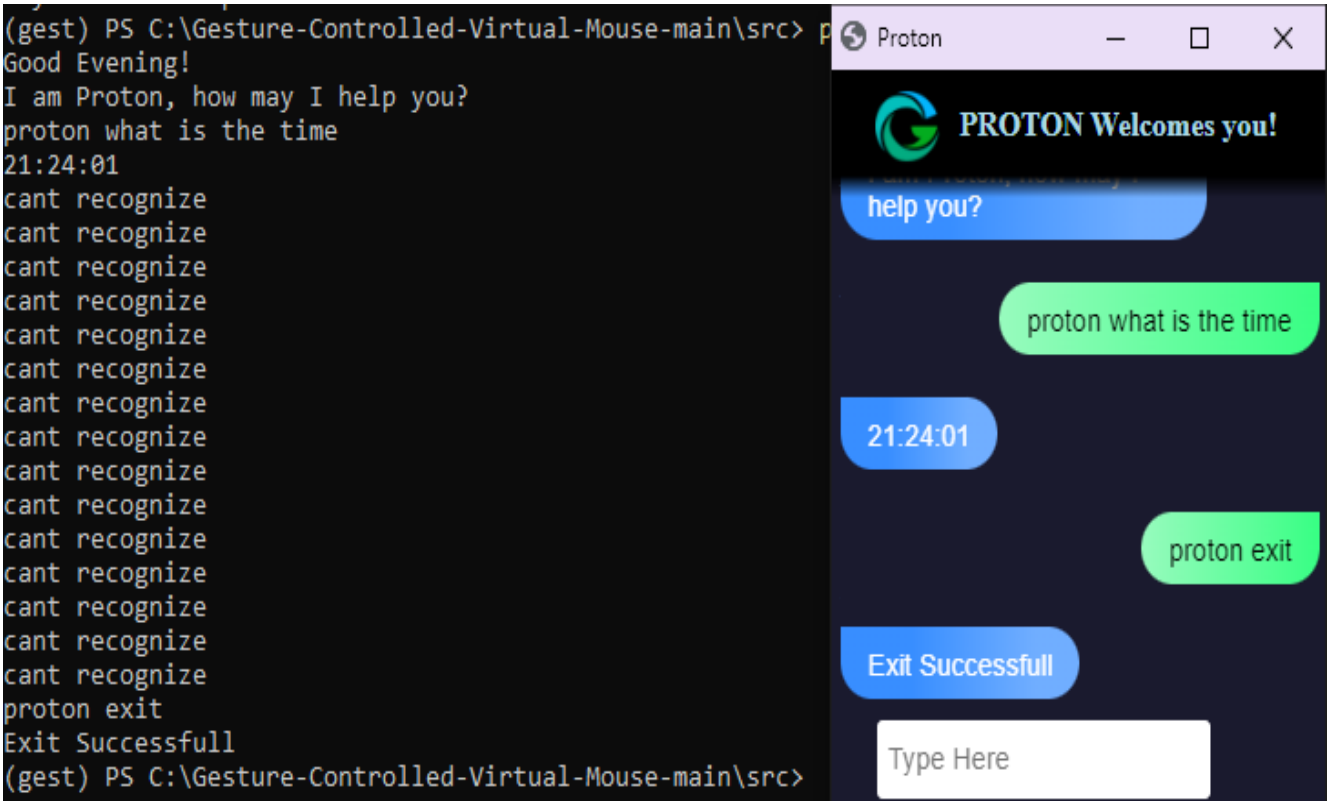
Test Case 7: Current Time

- **Objective**: Validate the voice command to get the current time.
- **Preconditions**: System is initialized and the microphone is active.
- **Steps**:
    1. Issue the voice command "Proton what is the time" or "Proton time".
    2. Observe if the system returns the current time.

- **Expected Result**: The system should return the current time.

Test Case 8: Exit the Voice Assistant

- **Objective**: Validate the voice command to exit the Voice Assistant.
- **Preconditions**: System is initialized and the microphone is active.
- **Steps**:
    1. Issue the voice command "Proton Exit".
    2. Observe if the Voice Assistant exits.



- **Expected Result**: The Voice Assistant should be closed.

Chapter 5
User Guide

5.1 Introduction

The user guide provides detailed instructions on how to use the Hand Tracking Module for various hand gestures and actions. This chapter will guide users through the setup, usage, and available features of the module.

5.2 System Requirements

Operating System: 64-bit operating system, x64-based processor

Python: The module requires Python 3.6 or later.

Anaconda Distribution

OPENCV

MEDIAPIPE:

Hand-Tracking module will be importing from mediapipe package where we will be having the record of landmarks.

AUTOPY:

AutoPy is a simple, cross-platform GUI automation library for Python. It includes functions for controlling the keyboard and mouse, finding colors and bitmaps on- screen, and displaying alerts. Currently supported on macOS, windows and X11 with the XTest extension. AutoPy includes a number of functions for controlling the mouse.

PYAUTOGUI:

PyAutoGUI is a Python automation library used to click, drag, scroll, move, etc. It can be used to click at an exact position.

PyAutoGUI works across Windows, MacOS X and Linux.

Anaconda:

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012. As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition, while other products from the company are Anaconda Team Edition and Anaconda Enterprise Edition, neither of which is free.

5.3 Getting Started

To start using theVirtual Mouse, follow these steps:

1. Ensure that your webcam is connected and functional.
2. Open a terminal or command prompt.
3. Activate the Python virtual environment (if applicable).
4. Navigate to the directory containing the code.
5. Run the program
6. The module will start capturing video from the webcam and detecting hand gestures.

Chapter 6
Conclusion
&
Future Work

6.1 Limitations and Challenges:

While the proposed system offers a promising solution for hand gesture recognition and AI- based human-computer interaction, it is important to acknowledge the limitations and challenges that need to be addressed:

- **Occlusion issues:**

  One of the primary challenges faced by the system is handling occlusion. When the hand is partially or fully occluded, such as when fingers overlap or objects obstruct the hand, it can lead to inaccurate hand tracking and gesture recognition. To overcome this limitation, advanced techniques such as 3D hand tracking or multi-camera setups can be explored to improve the system's robustness in occlusion scenarios.

- **Lighting conditions**:

  Variations in lighting conditions can significantly impact the system's performance. Low light conditions can lead to difficulties in detecting hand landmarks accurately, while strong backlighting can result in overexposed images, making hand tracking challenging. To address this, adaptive algorithms that dynamically adjust the image processing parameters based on the lighting conditions can be implemented. Additionally, exploring techniques like infrared-based hand tracking can provide more reliable results in diverse lighting environments.

- **Complex gestures:**

  Recognizing complex gestures involving multiple fingers or intricate hand movements can be a challenging task. Certain gestures may have similar hand configurations, making it difficult to distinguish them accurately. To overcome this, incorporating advanced machine learning algorithms, such as deep learning models, can help capture more complex patterns and improve gesture recognition accuracy. Additionally, integrating temporal information by considering the sequence of hand poses over time can enhance the system's ability to recognize dynamic gestures.

6.2 Future Enhancements:

To further advance the proposed system and address its limitations, the following future enhancements can be considered:

- **Advanced machine learning algorithms:**

  Expanding the system's capabilities by exploring and integrating advanced machine learning techniques, such as deep learning models, can significantly improve the accuracy and robustness of gesture recognition. These models can learn complex patterns and representations from large-scale training data, enabling the system to recognize a wider range of gestures accurately.

- **Refinement of hand tracking module**:

  Continuously refining the hand tracking module is crucial for achieving more accurate and reliable hand tracking. Improvements can be made to handle occlusion and lighting challenges by exploring novel algorithms that leverage multiple sensors or depth information. Additionally, incorporating hand pose estimation techniques can provide more detailed information about hand articulation, enabling precise tracking even in challenging conditions.

- **User customization and adaptation**:

  Providing options for users to calibrate the system according to their hand shape and size can enhance the user experience and improve gesture recognition accuracy. By allowing users to input their hand profiles or leveraging user-specific training, the system can adapt to individual variations, ensuring personalized interaction.

- **Multi-user support:**

  Extending the system's capabilities to support multiple users simultaneously can enable collaborative interactions and expand its applicability in scenarios such as interactive presentations or group activities. Techniques like multi-camera setups or advanced depth sensing can be explored to differentiate between multiple users' hands and enable simultaneous tracking and interaction.

- **Integration of additional features:**

  To enhance the overall usability and flexibility of the system, integrating additional features can be considered. For example, incorporating voice commands alongside hand gestures can provide users with multiple interaction modalities.

Additionally, integrating hand gesture shortcuts for common tasks or applications can further streamline the user experience and improve efficiency.

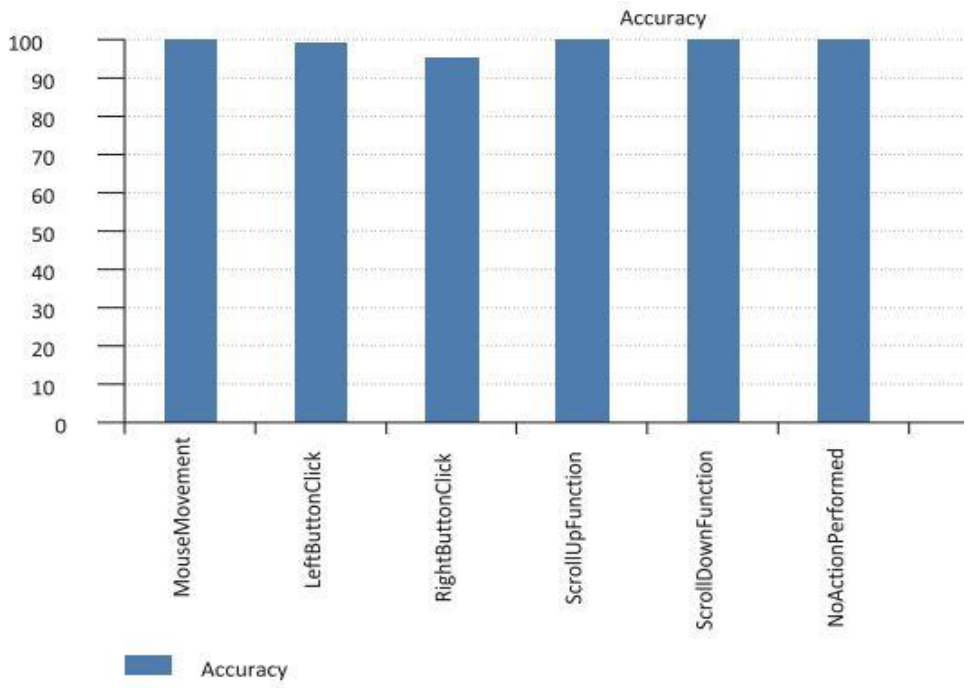
- **Cross-platform compatibility:**

  Expanding the system's compatibility to various platforms and operating systems can increase its accessibility and reach. Supporting different devices, such as smartphones, tablets, or wearable devices, can enable users to interact with a wide range of digital interfaces beyond traditional desktop environments. This cross-platform compatibility can be achieved through platform-specific adaptations and optimizations.

By addressing these limitations and exploring the suggested future enhancements, the proposed system can further improve its accuracy, usability, and adaptability, making it a more powerful tool for AI-based human-computer interaction.

6.3 Accuracy and Precision:

One of the critical factors in assessing the effectiveness of a virtual mouse is its accuracy and precision in tracking hand movements. The AI algorithms should be capable of accurately interpreting hand gestures and translating them into precise cursor movements on the screen. Evaluating the system's performance in different scenarios, such as varying lighting conditions or complex hand motions, can provide insights into its reliability.



6.4 Summary:

The Gesture Controlled virtual mouse system presented in this research holds great promise for revolutionizing the way we interact with computers. It offers a novel and accessible input modality, opening up new possibilities for individuals with disabilities and providing an alternative option for all users. By further refining the system, exploring new applications, and addressing user feedback, we can continue to advance the field of human-computer interaction and create more inclusive and intuitive computing experiences for everyone.

The voice assistant component of the Gesture Controlled Virtual Mouse and Voice Assistant project aims to enhance human-computer interaction through voice control. This module addresses the limitations of traditional input devices by providing an intuitive and accessible way to execute commands and perform tasks.

By integrating advanced gesture recognition and voice control technologies, the Gesture Controlled Virtual Mouse and Voice Assistant project aims to redefine human-computer interaction. This system offers a more inclusive, ergonomic, and intuitive way for users to interact with their computers, enhancing overall user experience and productivity.

6.5 Implications and Applications:

The Gesture Controlled Virtual Mouse component of the project transforms human-computer interaction by introducing intuitive gesture recognition technology. It offers users an alternative to traditional input devices like keyboards and mice, allowing them to control cursor movements, clicks, drags, and scrolls through natural hand gestures captured by a camera. This approach enhances accessibility by accommodating users with physical disabilities who may find traditional input methods challenging. Moreover, it improves ergonomics by reducing the physical strain associated with prolonged use of conventional devices, promoting a more comfortable and sustainable interaction experience. In educational and gaming contexts, the Gesture Controlled Virtual Mouse enhances engagement and immersion by enabling precise control and interaction within digital environments. Additionally, it supports research in HCI by exploring innovative ways to integrate gesture recognition with computing interfaces, aiming to redefine how users interact with technology in various applications.

The voice assistant component of the Gesture Controlled Virtual Mouse and Voice Assistant project revolutionizes human-computer interaction by integrating advanced voice control technologies. It significantly enhances accessibility by providing a hands-free alternative input method, particularly beneficial for users with physical disabilities. This approach not only improves ergonomics by reducing reliance on traditional input devices like keyboards and mice but also enhances productivity through efficient execution of tasks via voice commands. In educational settings, it supports interactive learning experiences, while in gaming, it enhances immersion with natural voice-based interactions for gameplay control. Furthermore, the voice assistant facilitates seamless integration with smart home devices, enabling users to manage their environments effortlessly with intuitive voice commands.