A Major Project Report on

Air Mouse: AI-Powered Gesture Control

Submitted to

The Department of CSE (Artificial Intelligence & Machine Learning)
In partial fulfillment of the academic requirements of

Jawaharlal Nehru Technological University

For

The award of the degree of

Bachelor of Technology

In

Computer Science and Engineering (Artificial Intelligence & Machine Learning)

By

Gadige Srinivas	**21311A6625**
Sanda Rohan	**21311A6637**
Jamalpur Uday Kumar	**21311A6658**

Under the Guidance of

Mrs. N. Santhoshi
Asst. Professor

Sreenidhi Institute of Science and Technology

(An Autonomous Institution)
Yamnampet, Ghatkesar, R.R. District, Hyderabad – 501301

2024-2025

Sreenidhi Institute of Science and Technology

Yamnampet, Ghatkesar, R.R. District, Hyderabad - 501301

Department of CSE- **Artificial Intelligence & Machine Learning**

CERTIFICATE

This is to certify that this Major Project Report on "**Air Mouse: AI-Powered Gesture Control**", submitted by Sanda Rohan (21311A6637) in the year 2024-25 in partial fulfillment of the academic requirements of Jawaharlal Nehru Technological University for the award of the degree of Bachelor of Technology in CSE (**Artificial Intelligence & Machine Learning**), is a Bonafide work that has been carried out during B.Tech. IV year II Semester- CSE (**Artificial Intelligence & Machine Learning**), under our guidance. This report has not been submitted to any other institute or university for the award of any degree.

Mrs. N Santhoshi
Asst. Professor
Department of CSE-AI & ML
SNIST

Dr . K. Shirisha
Professor and Head,
Department of CSE- AI & ML
SNIST

External Examiner

Date:-

DECLARATION

We, **Gadige Srinivas (21311A6625), Sanda Rohan (21311A6637) and Jamalpur Uday Kumar (21311A6658) students** of **Sreenidhi Institute of Science And Technology, Yamnampet, Ghatkesar,** studying IV year II semester, **CSE (Artificial Intelligence & Machine Learning)** solemnly declare that the Major Project Report, titled "Air Mouse: AI-Powered Gesture Control" is submitted to **Sreenidhi Institute of Science and Technology** for partial fulfillment for the award of degree of Bachelor of Technology in **CSE (Artificial Intelligence & Machine Learning)**.

It is declared to the best of our knowledge that the work reported does not form part of any dissertation submitted to any other University or Institute for award of any degree

Gadige Srinivas	**21311A6625**
Sanda Rohan	**21311A6637**
Jamalpur Uday Kumar	**21311A6658**

ACKNOWLEDGEMENTS

I would like to take this opportunity to extend my heartfelt gratitude to all those who supported me in transforming a simple idea into a fully functional application. This project would not have been possible without the valuable contributions of several individuals working behind the scenes.

I am deeply thankful to my internal guide, **Mrs. N. Santhoshi**, and our **Major Project Coordinator, Mr. Ravi Gugulothu**, for their insightful technical guidance, constant encouragement, and unwavering support throughout the course of this project.

My sincere thanks go to **Dr. K. Shirisha**, Head of the Department of CSE - Artificial Intelligence & Machine Learning, for being a guiding light and an enduring source of inspiration. Her expert advice and motivational leadership played a vital role in shaping my work.

I would also like to express my deepest appreciation to my parents, whose unconditional love, sacrifices, and belief in me have been the driving force behind my achievements and dreams.

I am grateful to our respected Principal, **Dr. T. Ch. Siva Reddy**, for his visionary leadership and for fostering an academic environment that allowed me to explore and implement this project.

Finally, I extend my sincere thanks to all the faculty members and staff—both teaching and non-teaching—whose timely assistance and encouragement greatly facilitated the completion of this work. Their contributions, though often behind the scenes, were invaluable in ensuring the successful realization of this endeavor.

Sanda Rohan - 21311A6637

Abstract

AirMouse is an innovative, touchless cursor control system designed to replace traditional pointing devices by utilizing real-time hand-tracking and gesture recognition. It combines OpenCV's advanced computer vision capabilities with MediaPipe's precise hand landmark detection and PyAutoGUI's control interface to map intuitive hand movements—such as pinches, swipes, and palm motions—into standard mouse functions like clicking, dragging, scrolling, and cursor navigation. This allows users to interact with digital interfaces without physical contact, making it both convenient and futuristic.

The system enhances comfort by reducing strain from prolonged mouse usage and offers improved accessibility for individuals with motor impairments or in environments requiring high hygiene standards. Early tests show promising results, including over 90% tracking accuracy and sub-150ms click latency on mid-range hardware across varied lighting conditions. Future developments aim to expand gesture vocabulary with multi-finger recognition, implement adaptive learning for personalized gesture sensitivity, and explore AR-based visual feedback for an even more immersive user experience.

INDEX

1. INTRODUCTION

1. INTRODUCTION

1.1 Introduction

The world is full of technology driven factors in our day-to-day life. We have so many technologies, throughout the world computer technologies are growing simultaneously. They are used to perform various tasks which cannot be performed by humans. In fact they are ruling the human lives because they have a potential to do the tasks which cannot be done by humans. The interaction between human and computer can be done with output device like mouse. The mouse is a device used for interacting with a GUI which includes pointing, scrolling and moving etc. The hardware mouse in computers and touchpads in laptops will require a huge amount of time to perform complex tasks, in case we are carrying hardware mouse wherever we go it would be damaged sometimes.

After decades the technology has made the mouse functionality from wired into the wireless to improve the functionality and for the easy movements in hassle free manner. As the technologies started growing there came the speech recognition technique. This recognition is mainly used for the voice recognition purpose for searching something with the help of their voice and for translation purposes but it can take time for recognition to perform mouse functions. Later the human computer interaction evolved with the eye tracking techniques for controlling the cursor of the mouse. The major drawback of this technique is that some may wear contact lens or some may have long eyelashes so it may take some time to capture their eye movement.

Hand gesture controlled virtual mouse using artificial intelligence is a technology that allows users to control the movement of their computer mouse using hand gestures, without the advent of a physical mouse. This technology uses a camera vision based approach to track the movements of the user's hand and to perform mouse functions on the computer screen. The system works by capturing video input from a camera pointed at the user's hand. The computer vision algorithms then analyse the video feed to identify the user's hand and track its movement. This information is given to machine learning models which have been trained to

recognize specific hand gestures, such as pointing or swiping, and translate them into corresponding mouse movements.

1.2 PURPOSE

Our proposed Gesture-Controlled Virtual Mouse with Integrated Chatbot system aims to overcome these limitations by offering a seamless and intuitive user experience that combines gesture control technology with a conversational interface. By integrating gesture recognition capabilities with a robust chatbot framework, users can interact with their computing devices using natural hand gestures and voice commands, enabling greater accessibility, efficiency, and user satisfaction. Through this integrated approach, we seek to provide users with a versatile and inclusive interaction platform that enhances productivity, fosters innovation, and improves overall user experience.

Input Modalities

Incorporate multiple input modalities, such as gesture recognition, eye-tracking, braincomputer interfaces, or a combination there of.

Data Acquisition

Utilize sensors and device to capture user input data .These sensors should collect information about gestures, eye movements, or brain signals, depending on the chosen input modality.

Data Processing

Implement AI and machine learning algorithms to process and interpret the input data. For gesture recognition, this may involve computer vision techniques, while eye-tracking data could be analyzed to determine cursor movement.

User Intent Recognition

Develop algorithms that can recognize user intentions, such as moving the cursor, clicking, dragging, or performing other actions. Machine learning models can be trained to classify user input into these categories.

Cursor Control

Translate the recognized user intentions into cursor movements on the screen. Ensure smooth and accurate control, regardless of the chosen input modality.

2. LITERATURE SURVEY

2.1 Existing System

The current landscape of human-computer interaction predominantly relies on traditional input devices such as keyboards and mice, which may pose limitations for users with mobility impairments or those seeking more intuitive ways to interact with technology. While some alternative input methods exist, such as touchscreens and voice commands, they may not fully address the diverse needs and preferences of users. Moreover, standalone chatbots often lack seamless integration with gesture control systems, leading to disjointed user experiences and limited functionality.

Limitations Of Existing System

The limitations of the existing system include restricted accessibility for users with disabilities, a lack of natural and intuitive interaction methods, and disjointed user experiences when combining different interaction modalities. Additionally, standalone chatbots may have limited capabilities and may not fully integrate with gesture control systems, limiting their utility and effectiveness.

Disadvantages

Usually not as accurate as a mouse.
Not very user friendly.
Cannot provide high precision
Performance. Has specific surface
requirements to operate. Special hardware
required.

Again, specific surface requirements.

2.2 Proposed System

Our proposed Gesture-Controlled Virtual Mouse with Integrated Chatbot system aims to overcome these limitations by offering a seamless and intuitive user experience that combines gesture control technology with a conversational interface. By integrating gesture recognition capabilities with a robust chatbot framework, users can interact with their computing devices using natural hand gestures and voice commands, enabling greater accessibility, efficiency, and user satisfaction. Through this integrated approach, we seek to provide users with a versatile and inclusive interaction platform that enhances productivity, fosters innovation, and improves overall user experience.

Advantages

Eliminates the need for physical mouse.
Reduces physical strain.
Making the interaction with computers more intuitive and
user-friendly. Hand gestures are more intuitive and can be quickly
grasped by users.

3. SYSTEM ANALYSIS

3.1 Introduction

The AI Virtual Mouse is a groundbreaking technology that transforms human- computer interaction by enabling users to control a computer through hand gestures, eliminating the need for a physical mouse. Using artificial intelligence and computer vision, this system interprets hand movements captured by a camera, translating them into actions like moving the cursor, clicking, and scrolling on-screen. Through a thorough system analysis, developers can address core aspects such as gesture recognition accuracy, real-time responsiveness, hardware compatibility, and user-centric design, laying the foundation for a technology that can make digital interaction more intuitive and inclusive.

3.2 Analysis Model

The analysis model for an AI virtual mouse using computer vision focuses on translating hand gestures into mouse movements and actions through image processing and pattern recognition. Computer vision techniques enable the virtual mouse to interpret hand positions and gestures in real-time, allowing users to control the cursor and perform actions without a physical mouse.The analysis model can be segmented into key components are:

- **Functional Requirements Analysis**

Defines the core actions needed for user interaction, using computer vision to detect gestures and translate them into cursor control:

Cursor Movement

Tracks the position of the hand to move the cursor on the screen.

Click Actions

Recognizes specific hand gestures (e.g., pinching fingers together) to simulate left-click, right-click, and double-click actions.

Scrolling

Detects swiping gestures for vertical or horizontal scrolling.

Customizable Gestures

Allows users to set custom gestures for specialized commands, enhancing accessibility.

✞ System Architecture Using Computer Vision

The architecture involves several modules that work together to capture, process, and interpret gestures as cursor actions:

Input Hardware:

A camera (e.g., webcam) serves as the input device, continuously capturing video frames of the user's hand.

Software Modules:

Image Preprocessing Module

Enhances the raw video input for accurate gesture recognition. This step includes color filtering, background subtraction, and noise reduction to isolate the hand.

Hand Detection and Tracking

Uses computer vision algorithms to identify the hand region within each frame. Techniques like skin color segmentation, Haar cascades, or YOLO (You Only Look Once) object detection may be applied.

Gesture Recognition Module

An AI model, often based on CNN (Convolutional Neural Networks), processes the isolated hand image and classifies the gesture.

Cursor Control Module

Maps recognized gestures to cursor movement on the screen, translating x-y coordinates from the hand position into the cursor's on-screen location.

✞ Data Flow and Processing Model

Computer vision in the virtual mouse follows a structured data flow, transforming video input into actionable commands:

Input Stage

The camera captures live video frames, which serve as the raw data for gesture detection.

Preprocessing

Frames undergo image processing techniques, including color filtering to isolate skin tone, background removal to focus on the hand, and noise filtering to improve accuracy.

Gesture Detection

The preprocessed frames are passed to a trained neural network or classification algorithm that identifies specific gestures.

Cursor Mapping

Recognized gestures are mapped to screen coordinates for cursor movement or to specific commands for click and scroll actions.

Output

The mapped gestures generate actions that the operating system interprets as mouse commands, completing the data flow.

✞ Computer Vision Techniques in Gesture Recognition

Computer vision in the AI virtual mouse relies on specific algorithms and models to detect and classify gestures accurately:

Hand Segmentation

Methods such as color-based segmentation and background subtraction help separate the hand from the surrounding background.

Feature Extraction

Key features like fingertips or palm position are extracted to determine gestures and track hand movement.

Convolutional Neural Networks (CNN)

Used to classify hand shapes and movements in real-time, CNNs are trained on labeled datasets of hand gestures, making it possible to detect common gestures accurately.

Object Detection Models

Advanced models like YOLO or MobileNet can be used for robust hand detection, allowing the system to identify hand location and shape even under varying lighting or backgrounds.

- **Performance and Usability Metrics**

Evaluates the system's effectiveness, with a focus on real-time performance and ease of use:

Latency

Ensures the virtual mouse responds instantly to hand movements, creating a seamless user experience.

Accuracy

High recognition accuracy minimizes incorrect cursor movements or clicks.

Resource Efficiency

Optimizes CPU and memory usage for smooth operation, even on standard hardware.

User Adaptability

Allows customization of gestures, sensitivity, and interface preferences, adapting to individual user needs.

- **Security and Privacy Analysis**

Since the system captures live video, security is critical:

Data Protection

Ensures that video data is processed in real-time without being stored or transmitted, protecting user privacy.

Permissions and Access Control

Limits access to the camera, safeguarding the video feed from unauthorized use.

3.3 Modules of the system

Voice Assistant Features

The voice assistant feature has been included to launch gesture recognition through voice commands. And added certain features to improve the user engagement and they can assess

whatever they need with less amount of effort and in hassle free manner. The voice assistant features which can be performed through the voice commands are:

- To launch and end the gesture recognition
- To search for something over internet

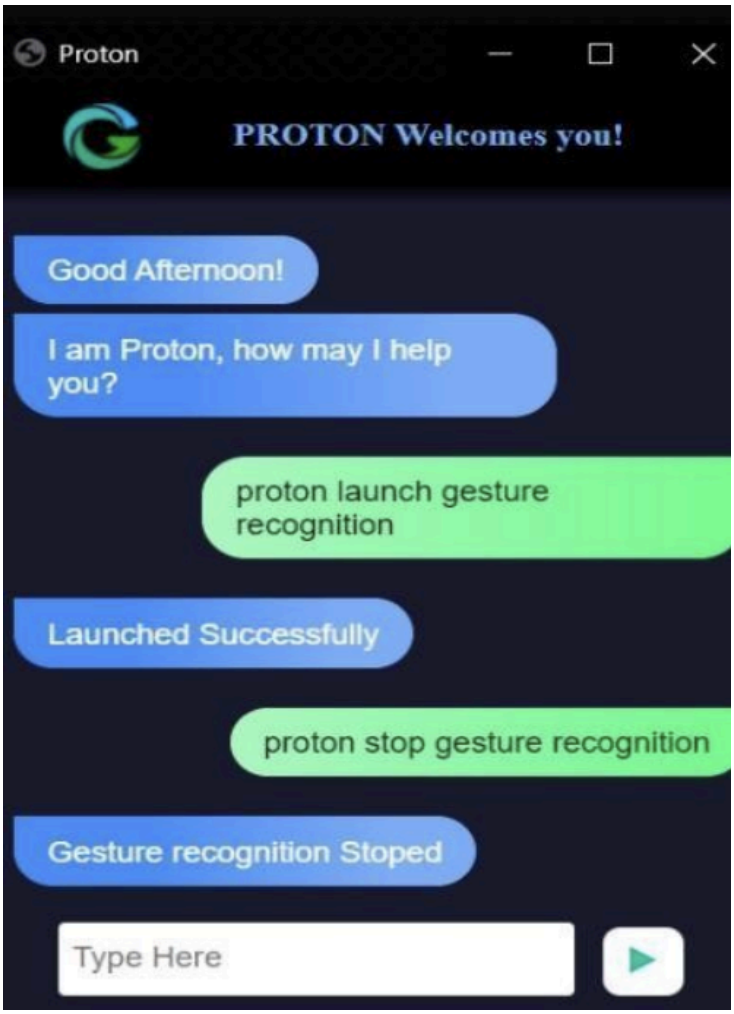

Fig: 3.3.1 Launch and End

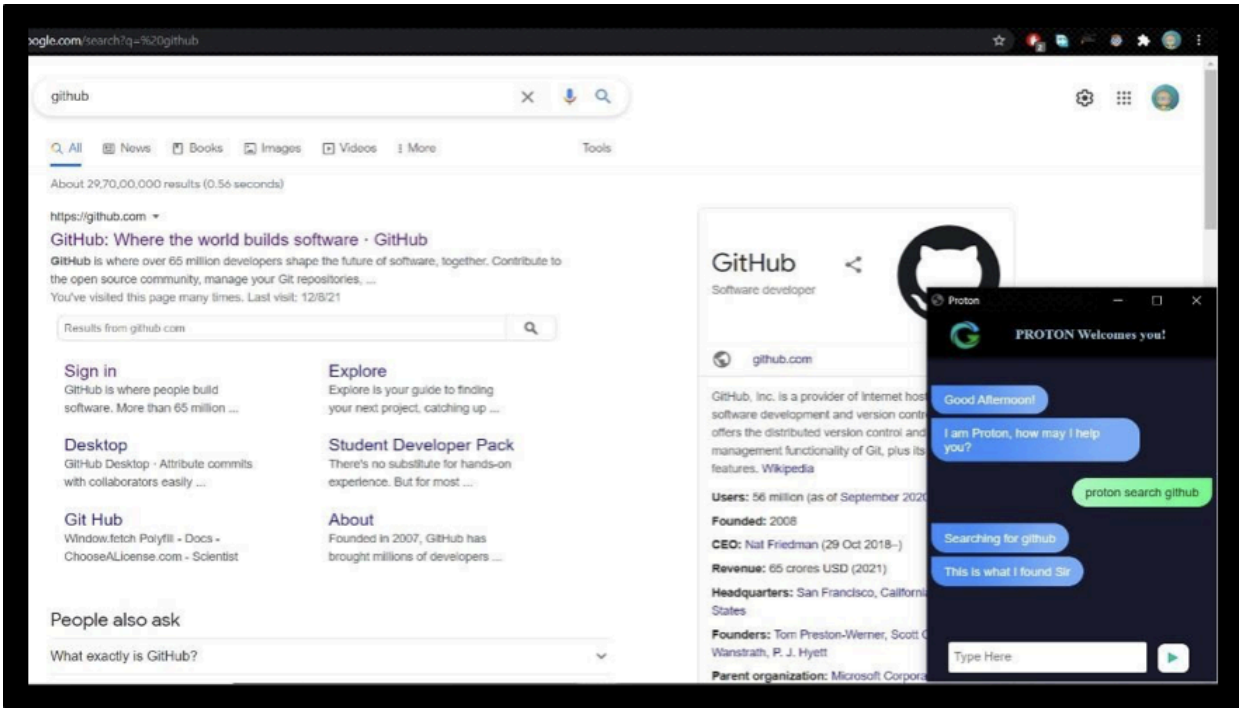

Fig: 3.3.2 Voice Assistant Search

.

4. FEASIBILITY STUDY

Feasibility Study

Feasibility study is made to see if the project on completion will serve the purpose of the organization for the amount of work, effort and the time that is spent on it. Feasibility study lets the developer foresee the future of the project and the usefulness. A feasibility study of a system proposal is according to its workability, which is the impact on the organization, ability to meet their user needs and effective use of resources. Thus when a new application is proposed it normally goes through a feasibility study before it is approved for development.

The document provide the feasibility of the project that is being designed and lists various areas that were considered very carefully during the feasibility study of this project such as Technical, Economic and Operational feasibilities.

Feasibility study encompasses the following thing:

1. Technical Feasibility

2. Economic Feasibility

3. Operational Feasibility

4.1 Technical Feasibility

The system must be evaluated from the technical point of view first. The assessment of this feasibility must be based on an outline design of the system requirement in the terms of input, output, programs and procedures. Having identified an outline system, the investigation must go on to suggest the type of equipment, required method developing the system, of running the system once it has been designed.

- Technical issues raised during the investigation are
- Does the existing technology sufficient for the suggested one?
- Can the system expand if developed?

The project should be developed such that the necessary functions and performance are achieved within the constraints. The project is developed within latest technology.

Through the technology may become obsolete after some period of time, due to the fact that never version of same software supports older versions, the system may still be used. So there are minimal constraints involved with this project. The system has been developed using Java the project is technically feasible for development.

4.2 Economic Feasibility

The developing system must be justified by cost and benefit. Criteria to ensure that effort is concentrated on project, which will give best, return at the earliest. One of the factors, which affect the development of a new system, is the cost it would require.

The following are some of the important financial questions asked during preliminary investigation:

- The costs conduct a full system investigation
- The cost of the hardware and software.
- The benefits in the form of reduced costs or fewer costly errors.

Since the system is developed as part of project work, there is no manual cost to spend for the proposed system. Also all the resources are already available, it give an indication of the system is economically possible for development.

4.3 Operational Feasibility

Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. Our proposed system overcomes all the problems related to present complex system and satisfies all the scope as defined.

5. SOFTWARE REQUIREMENTS

Software Requirements

This project presents finger movement gesture detection on our computer's window using camera & handling the whole system by just moving your one finger. Using finger detection methods for instant camera access and user-friendly user interface makes it more easily accessible. The system is used to implement motion tracking mouse, a signature input device and an application selector.

This system reduces the use of any physical mouse which saves time and also reduces effort.

5.1 Functional Requirements

Functional Requirement defines a function of a software system and how the system must behave when presented with specific inputs or conditions. These may include calculations, data manipulation and processing and other specific functionality.

In this system following are the functional requirements:

1. The system takes image of the hand through webcam and as the per the movements captured in the screen it performs cursor operations.
2. From a given point selected by the user, the movement along its path is captured and canvassing feature is applied.
3. The system uses points on the screen to activate keyboard functions.
4. It uses microphone to collect users voice to convert it into text.

5.2 Non-Functional Requirements

Performance

The system shall achieve a tracking accuracy of at least 95% under optimal lighting conditions.

The system shall respond to user input with a maximum latency of 100 milliseconds

Usability

The system shall provide a user-friendly interface with intuitive controls and clear feedback mechanisms.

The system shall support multiple languages and localization options to accommodate diverse user groups.

Security

The system shall encrypt user preferences and settings to protect sensitive information from unauthorized access.

The system shall comply with relevant data protection regulations and industry best practices for handling user data.

Compatibility

The system shall be compatible with a wide range of camera devices, including webcams, depth cameras, and infrared sensors.

The system shall support popular operating systems, including Windows, macOS, and Linux distributions.

Reliability

The system shall operate reliably under continuous use for extended periods without degradation in performance.

The system shall include error handling mechanisms to gracefully recover from unexpected failures or disruptions.

Maintainability

The system shall be modular and well-documented to facilitate future updates, modifications, and maintenance tasks.

The system shall include diagnostic tools and logging functionalities to assist developers in troubleshooting issues and identifying areas for improvement.

5.3 Performance Requirements

Performance term is mainly used to measure the parameters called time & space. This project uses verify less space and the actions up or operations performed are done very quickly in fraction of seconds. There is no issue of memory size out of bounds

Security

Security or authorization is one of the major parameters of all computerized applications. As details are confidential, no malicious user must be allowed to operate on.

Maintainability

The system shall be modular and well-documented to facilitate future updates, modifications, and maintenance tasks.

Reliability

The system shall be available 95% of the time.

Performance

This project uses less space and the action up or operations performed are done quickly. There is no issue in memory size out of bounds. The performance of this project will be at higher levels.

5.4 Hardware Requirements

The hardware requirements required to run and create the Virtual Mouse program is described below:

Webcam

To get an image, a webcam is required. Mouse sensitivity is constant with digicam clarity. The excellent consumer information is demonstrated whilst the digicam configuration is high enough. The camera is used for real-time images at any time the pc is on. The system will pick out the suitable movement primarily based on the contact and finger movement**.**

Computer desktop or laptop

The machine such as a desktop or laptop will be used to run a visual program that will display what the camera captured. To promote mobility, a notebook, which is a tiny, lightweight, and affordable laptop computer, is offered.

5.5 Software Requirements

The software requirements for an AI virtual mouse include:

Python Libraries:

Various Python libraries like OpenCV, NumPy, PyAutoGUI, and TensorFlow can be used for building the Al virtual mouse system.

OpenCV:

A computer vision library used for image and video processing, hand detection, and tracking.

NumPy:

This library is used for image and video processing, which can be used for hand detection and tracking.

PyAutoGUI:

PyAuto GUI is used to control mouse movements and clicks.

Mediapipe:

A cross-platform framework for building multi-modal applied machine learning pipelines.

Autopy:

Used to control mouse movement and click
An AI virtual mouse uses computer vision to track hand movements and translate them into cursor movements on the screen. It can be a useful alternative to a physical mouse, especially for people with disabilities or those who prefer a more natural way to interact with their computer.

6. SYSTEM DESIGN

6. SYSTEM DESIGN

6.1 Introduction

System design is the second phase of the software life cycle. The system goes through a logical and physical state of development. The user oriented performance specification is extended into a design specification, while designing the needed system. The design phase begins when the Requirement Specification document for the software to be developed is available. When the Requirement Specification activity is entirely in the problem domain, design is the first step to move from the problem domain to the solution domain. Design is essentially the bridge between the requirements specification and the final solution for satisfying these requirements.

6.2 UML diagrams

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system. Simply put, UML is a modern approach to modeling and documenting software. In fact, it's one of the most popular business process modelling techniques.

It is based on diagrammatic representations of software components. As the old proverb says: "a picture is worth a thousand words". By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

Use Case

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

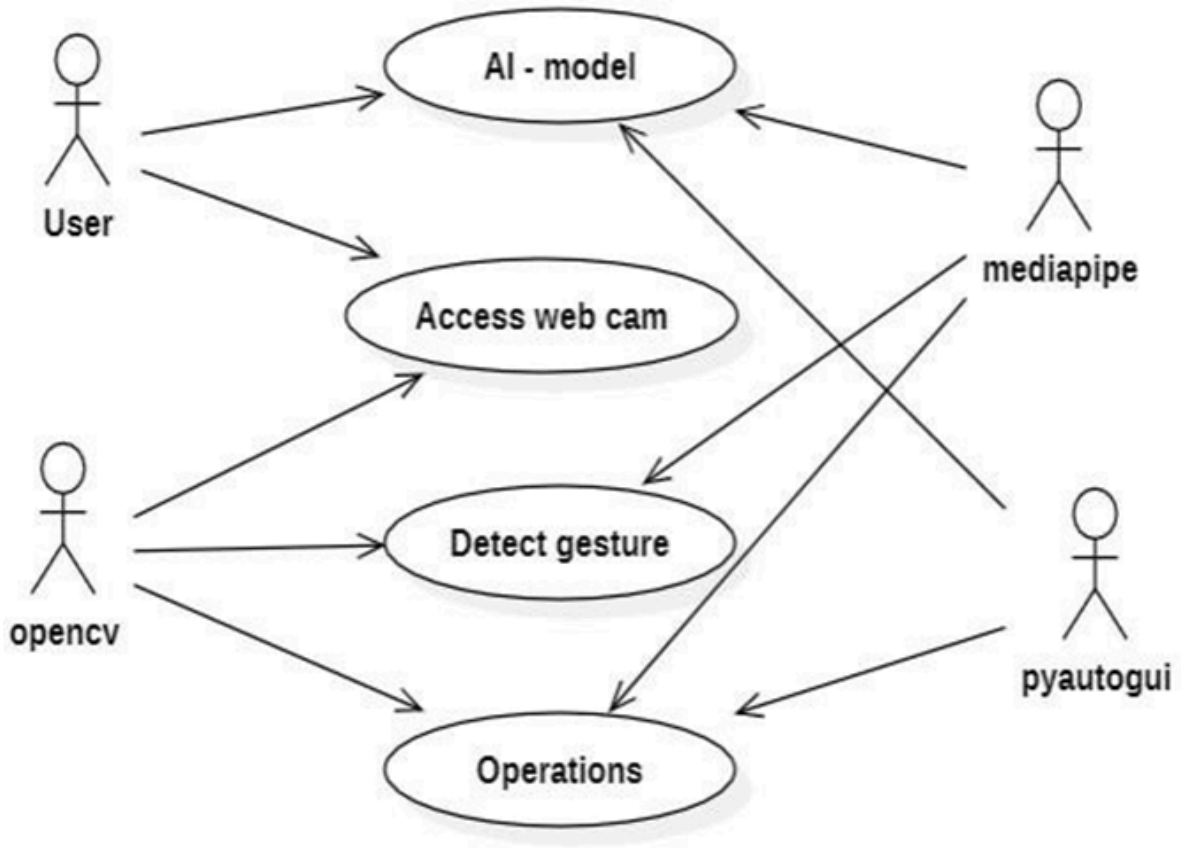

Fig: 5.2.1 Use Case Diagram

Class

Class diagram consists of classes, interfaces, associations, and collaboration. Class diagrams basically represent the object-oriented view of a system, which is static in nature. The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

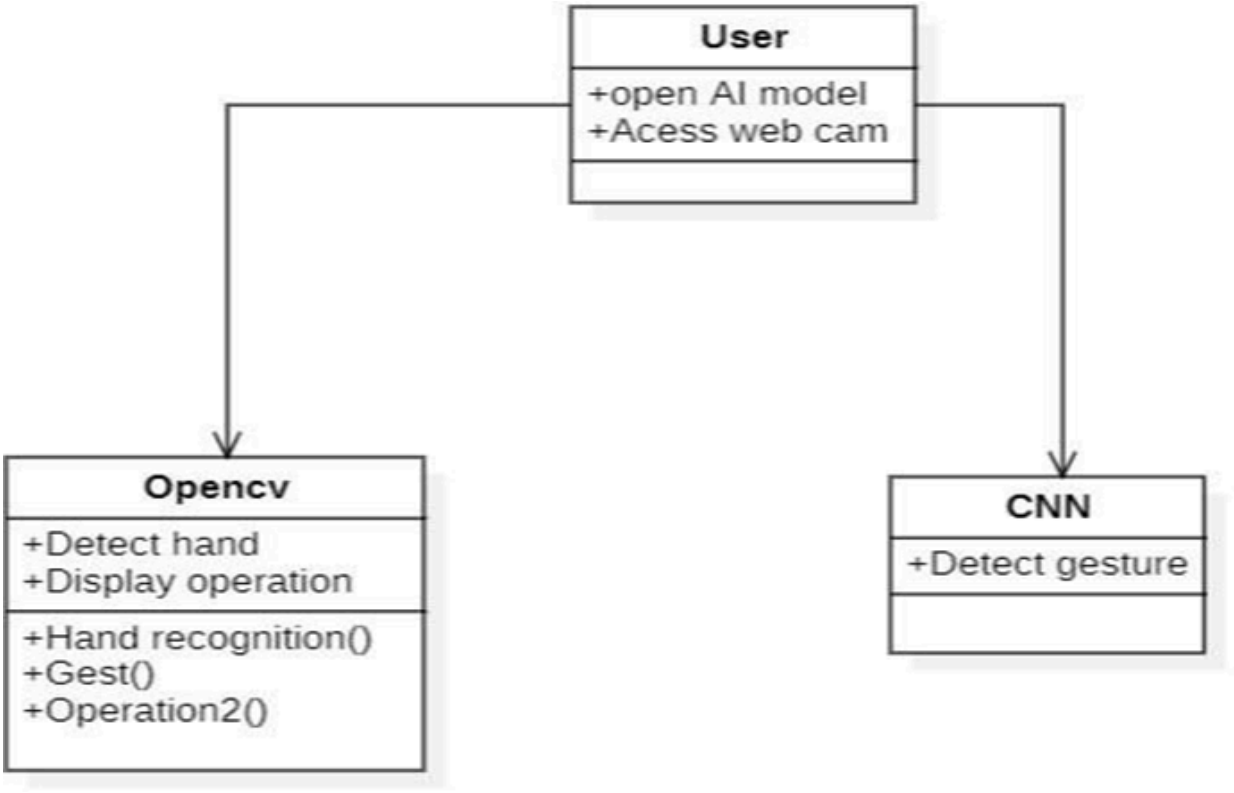

Fig: 5.2.2 Class diagram

Activity

An activity diagram is a behavioural diagram i.e. it depicts the behaviour of a system. An activity 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.

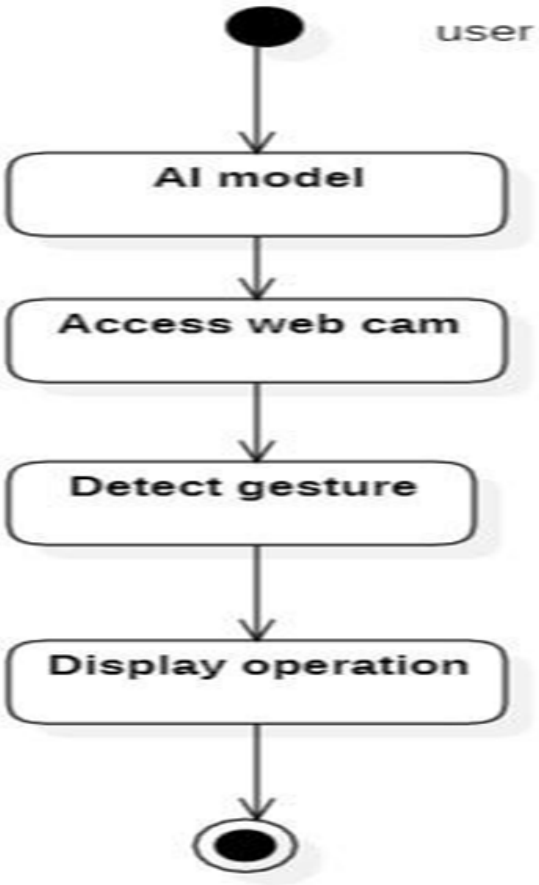

Fig 5.2.3 Activity Diagram

Block Diagram

A block diagram is helpful mainly in the preliminary stages of software development. A block diagram is similar to a UML package diagram in that it only shows very high level components of the design and how they interact.

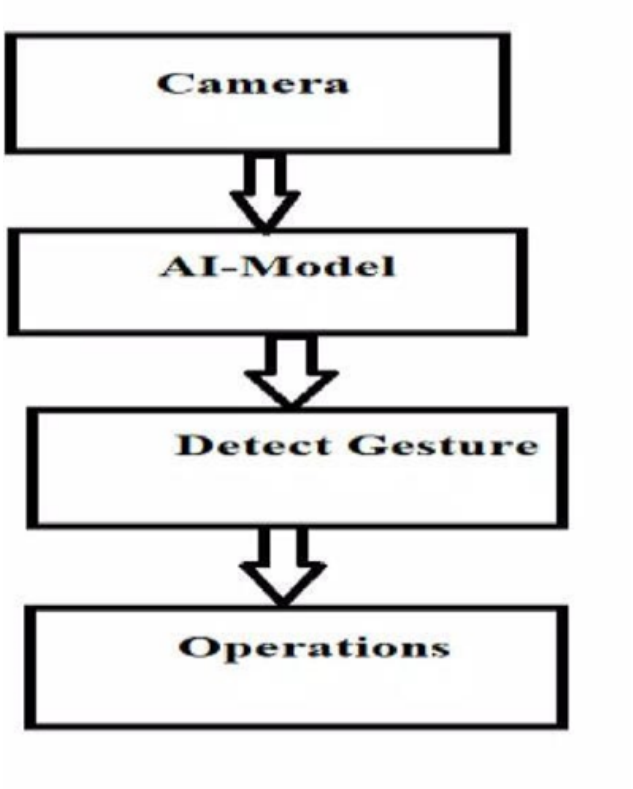

Fig:5.2.4 Block Diagram

7. Methodology

7.1 Algorithms and Tools Used

For the purpose of hand and finger detection we are using the one of the effective open source library mediapipe, it is one type of the framework based on the cross platform features which was developed by google and Opencv to perform some CV related tasks. This algorithm uses machine learning related concepts for detecting the hand gesture and to track their movements.

Mediapipe

The MediaPipe framework is used by developers to build and analyze systems through graphics and it has also been used to develop systems for application purposes. The MediaPipe library is used by developers to design and analyze various models graphically, and many of them have been used to create applications. MediaPipe Hands uses an ML pipeline consisting of multiple models that work together. The MediaPipe embedded model will work in pipeline mode. It mainly consists of graphs, nodes, streams and calculators. The MediaPipe framework is based on three basic parts; it is a benchmark, a framework for retrieving data from sensors and a set of components called computers and they are reusable. Computer and flow combine to create data flow diagrams; image created with MediaPipe where each node is a computer and the nodes are connected by threads. Mediapipe provides cross-platform and customizable open source ML solutions for live and streaming media. This is useful in many situations such as:

1. Selfie segmentation
2. Face mesh
3. Human pose detection and tracking
4. Holistic tracking
5. 3D object detection

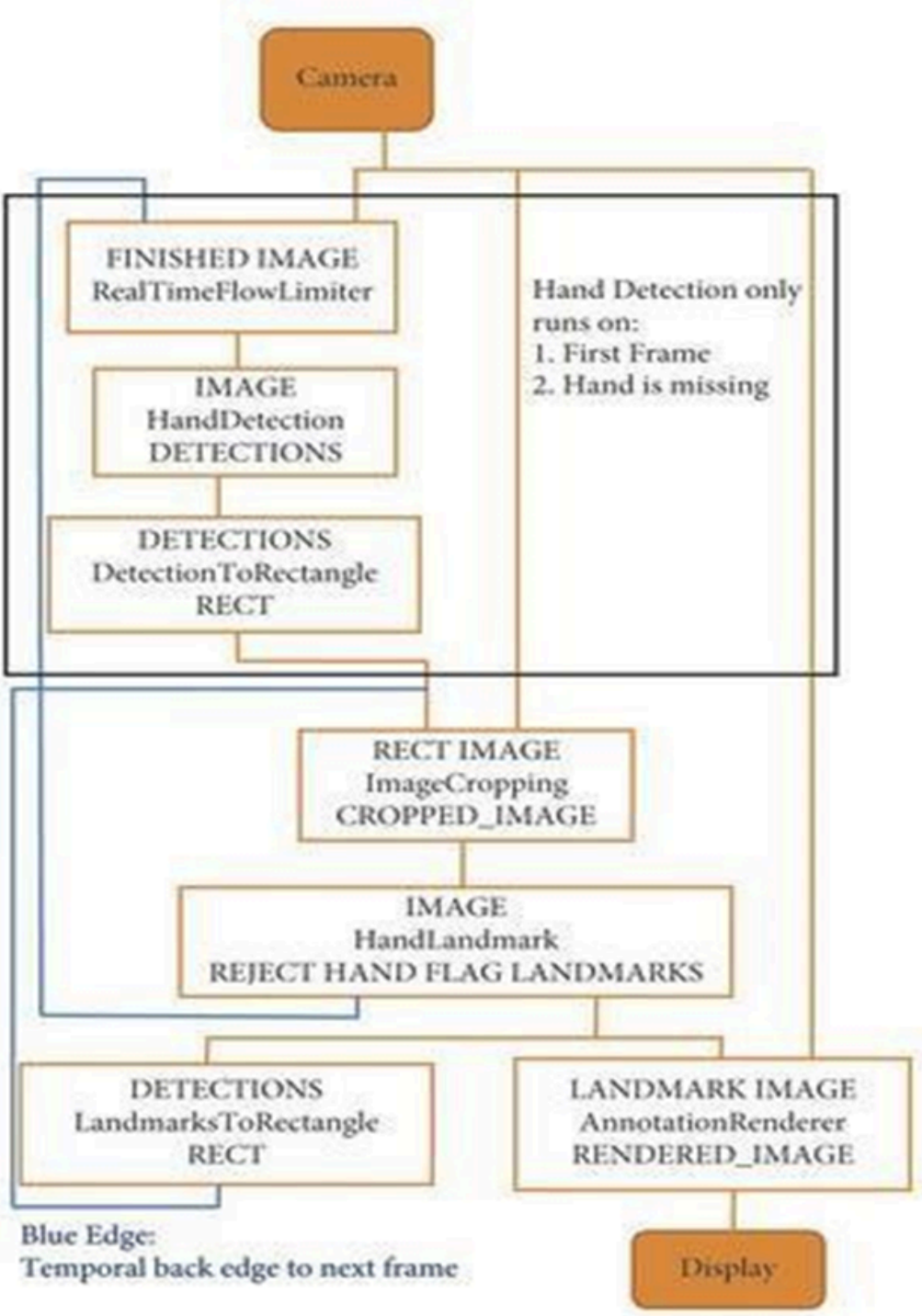

Fig: 7.1.1 MediaPipe hand recognition graph

Single-shot detector model is used for detecting and recognizing a hand or palm in real time. The single-shot detector model is used by the MediaPipe. First, in the hand detection module, it is first trained for a palm detection model because it is easier to train palms. Furthermore, the nonmaximum suppression works significantly better on small objects such as palms or fists. A model of hand landmark consists of locating 21 joint or knuckle co-ordinates in the hand region.

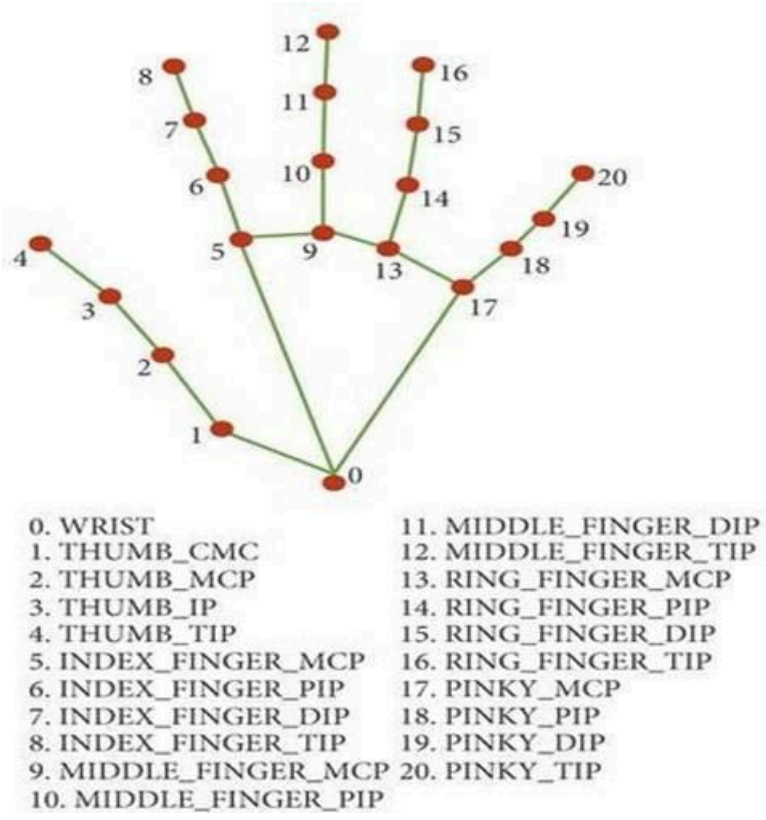

Fig: 7.1.2 Co-ordinates or land marks in the hand

Open-CV

OpenCV is a computer vision library which contains image-processing algorithms for object detection. OpenCV is a library of python programming language, and real-time computer vision applications can be developed by using the computer vision library. The OpenCV library is used in image and video processing and also analysis such as face detection and object detection.

7.2 Capturing video frames using camera

The overview of the hand gesture recognition the hand is detected using the background subtraction method and the result of this is transformed to a binary image. The fingers and palm are segmented to facilitate the finger recognition. The fingers are detected and recognized.

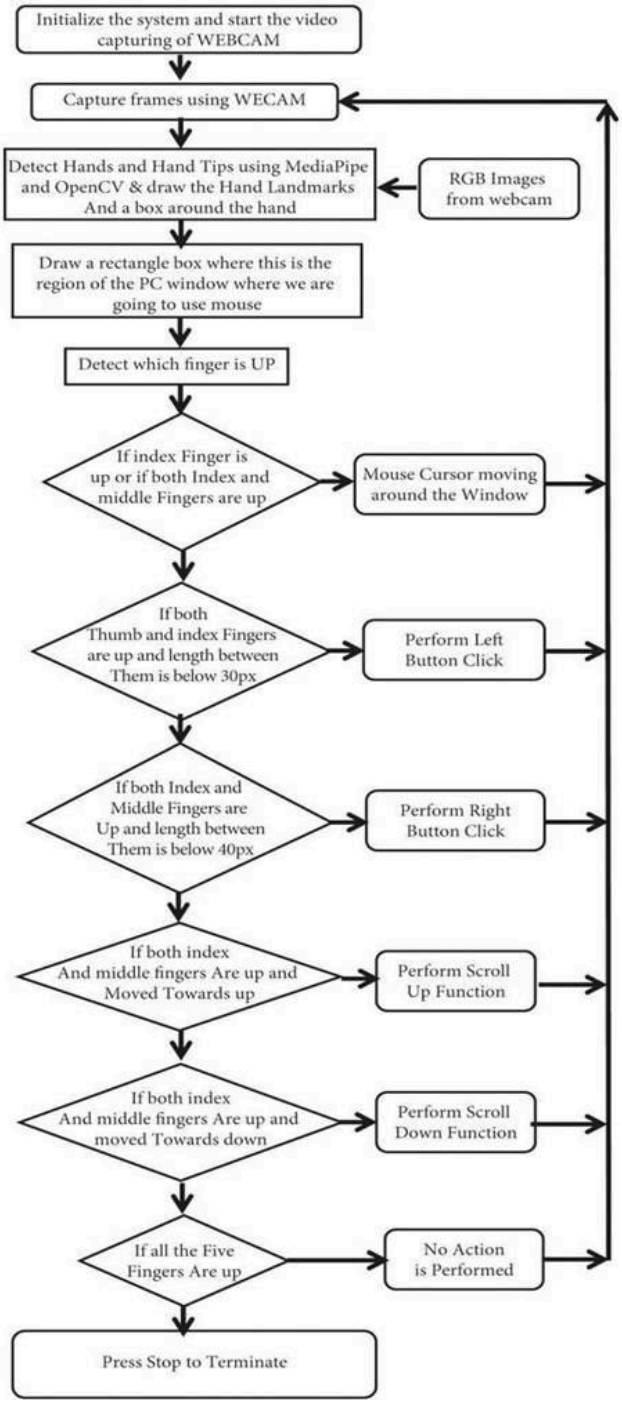

Fig: 7.1.3 Flowchart of the real-time AI virtual mouse system.

The Camera Used in the AI Virtual Mouse System

The proposed AI virtual mouse system is based on the frames that have been captured by the webcam in a laptop or PC. By using the Python computer vision library OpenCV, the video capture object is created and the web camera will start capturing video. The web camera captures and passes the frames to the AI virtual system.

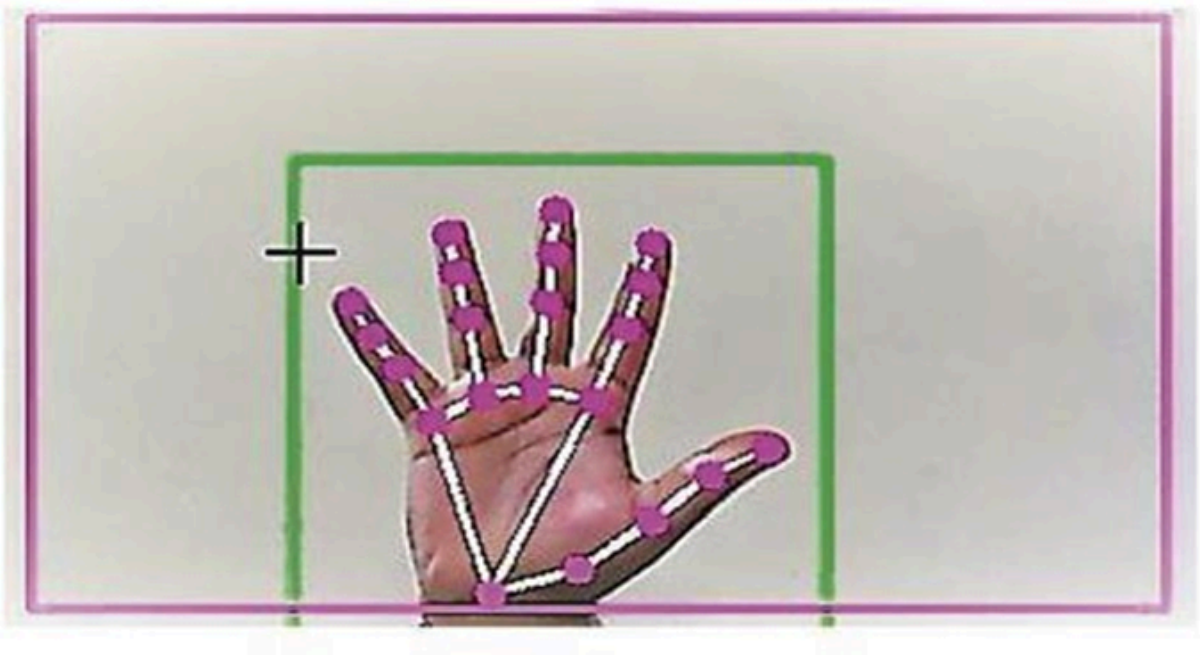

Fig: 7.1.4 Capturing video using the webcam (computer vision).

Capturing the Video and Processing

The AI virtual mouse system uses the webcam where each frame is captured till the termination of the program. The video frames are processed from BGR to RGB color space to find the hands in the video frame by frame as shown in the following code: def findHands(self, img, draw = True):

imgRGB = cv2.cvtColor(img,
cv2.COLOR_BGR2RGB) self.results =
self.hands.process(imgRGB)

Rectangular Region for Moving through the Window

The AI virtual mouse system makes use of the transformational algorithm, and it converts the coordinates of fingertip from the webcam screen to the computer window full screen for controlling the mouse. When the hands are detected and when we find which finger is up for performing the specific mouse function, a rectangular box is drawn with respect to the computer window in the webcam region where we move throughout the window using the mouse cursor.

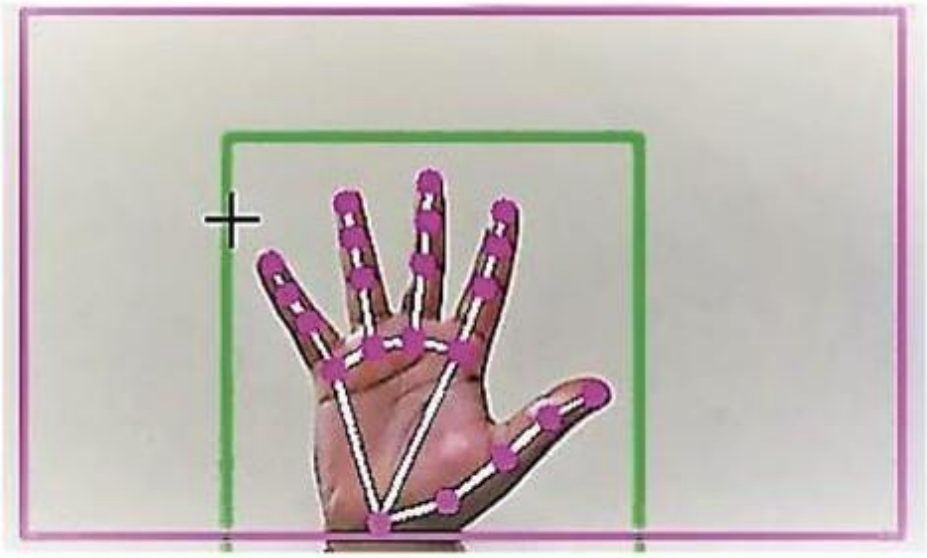

Fig: 7.1.5 Rectangular box for the area of the computer screen where we can move the cursor.

Detecting Which Finger Is Up and Performing the Particular Mouse Function

In this stage, we are detecting which finger is up using the tip Id of the respective finger that we found using the MediaPipe and the respective co-ordinates of the fingers that are up, and according to that, the particular mouse function is performed.

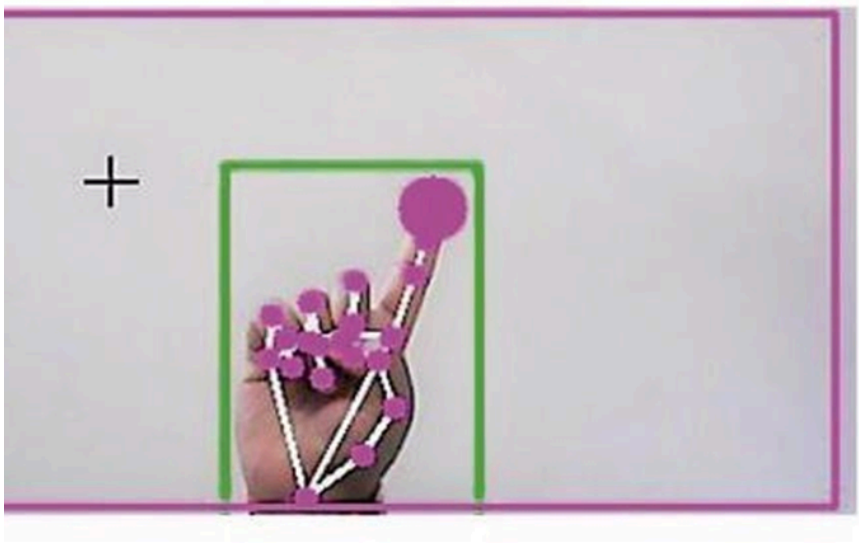

Fig: 7.1.6 Detection of which finger is up.

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

For the Mouse Cursor Moving around the Computer Window

If the index finger is up with tip Id = 1 or both the index finger with tip Id = 1 and the middle finger with tip Id = 2 are up, the mouse cursor is made to move around the window of the computer using the AutoPy package of Python.

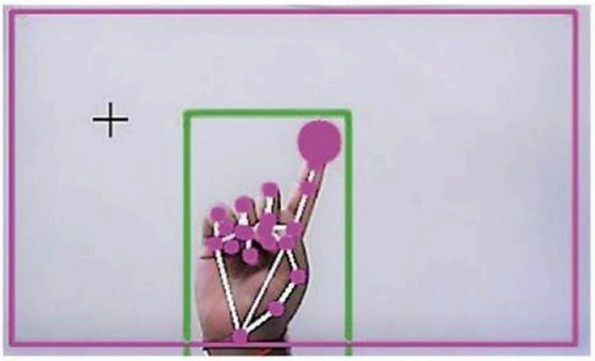

Fig: 7.1.7 Mouse cursor moving around the computer window.

For the Mouse to Perform Left Button Click

If both the index finger with tip Id = 1 and the thumb finger with tip Id = 0 are up and the distance between the two fingers is lesser than 30px, the computer is made to perform the left mouse button click using the pynput Python package.

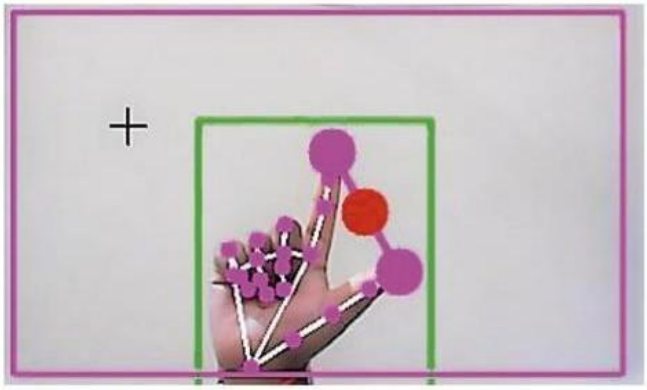

Fig: 7.1.8 Gesture for the computer to perform left button click.

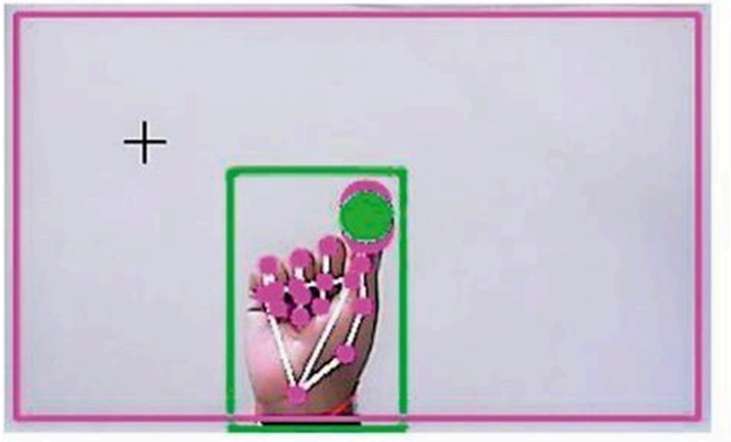

Fig: 7.1.9 Gesture for the computer to perform left button click.

For the Mouse to Perform Right Button Click

If both the index finger with tip Id = 1 and the middle finger with tip Id = 2 are up and the distance between the two fingers is lesser than 40 px, the computer is made to perform the right mouse button click using the pynput Python package.

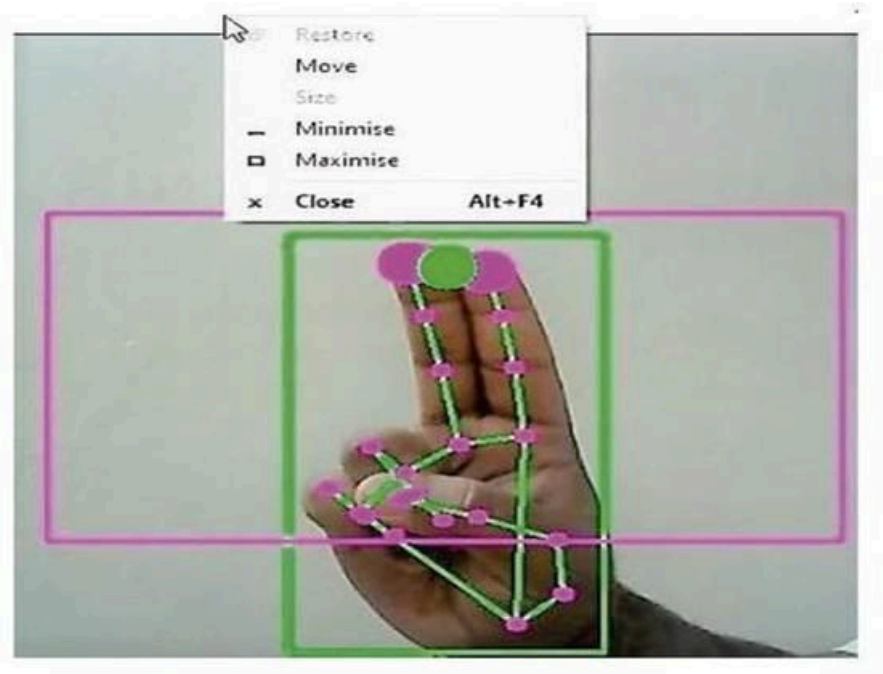

Fig: 7.1.10 Gesture for the computer to perform right button click.

For the Mouse to Perform Scroll up Function

If both the index finger with tip Id = 1 and the middle finger with tip Id = 2 are up and the distance between the two fingers is greater than 40 px and if the two fingers are moved up the page, the computer is made to perform the scroll up mouse function using the PyAutoGUI Python package.

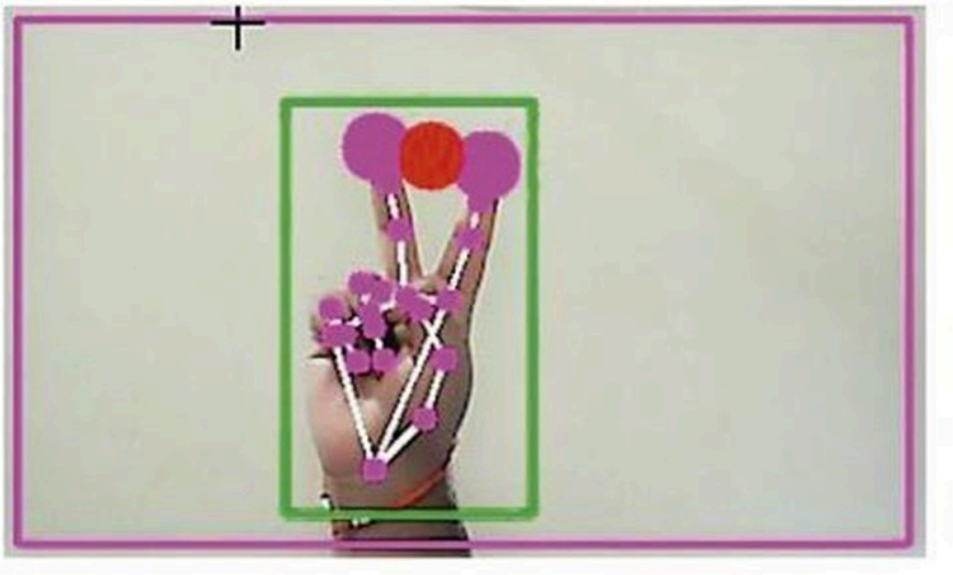

Fig:7.1.11 Gesture for the computer to perform scroll up function.

For the Mouse to Perform Scroll down Function

If both the index finger with tip Id = 1 and the middle finger with tip Id = 2 are up and the distance between the two fingers is greater than 40px and if the two fingers are moved down the page, the computer is made to perform the scroll down mouse function using the PyAutoGUI Python package.

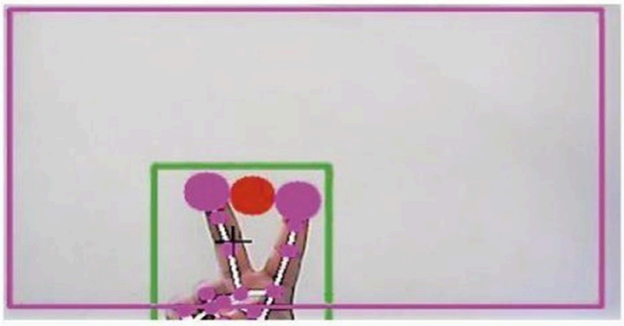

Fig: 7.1.12 Gesture for the computer to perform scroll down function.

For No Action to be Performed on the Screen

If all the fingers are up with tip Id = 0, 1, 2, 3, and 4, the computer is made to not perform any mouse events in the screen.

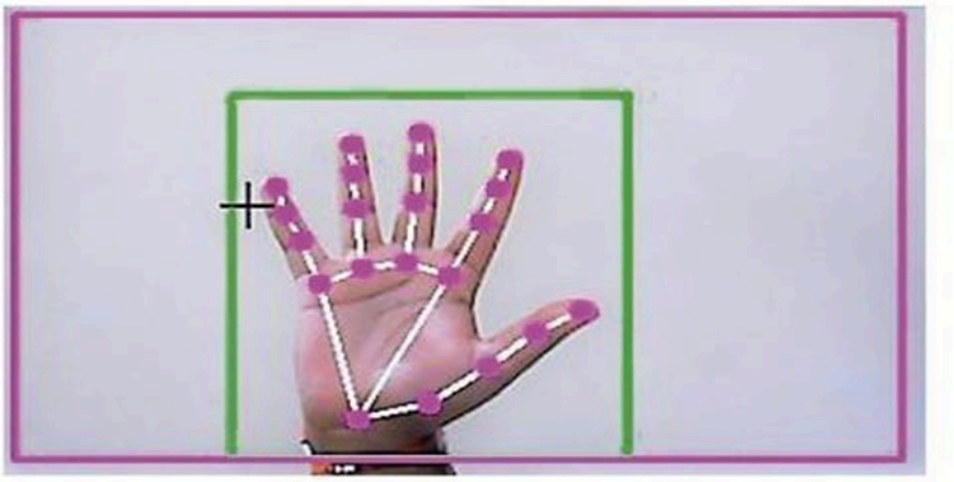

Fig: 7.1.13 Gesture for the computer to perform no action.

CODE FOR VIRTUAL MOUSE

```
# 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(sel
        f, 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(sel
        f, point):
        """

      returns euclidean distance between 'point'.

      Parameters
```

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

Returns

-
-
f
l
o
a
t
"
"
"

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(s
elf,point)
: """

returns absolute difference on z-axis between 'point'.

Parameters

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

Returns

-

-

f

l

o

a

t

"

"

"

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

    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_MIN
              OR 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_thresh
old = 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(currentVolu
meLv, 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.pinchdirectio
        nflag == True:
        controlHorizontal() #x
```

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


      lvx =
Controller.getpinchxlv(hand_resul
t) lvy =
Controller.getpinchylv(hand_resul
t)
      if abs(lvy) > abs(lvx) and abs(lvy) >
Controller.pinch_threshold:
Controller.pinchdirectionflag = False          if
abs(Controller.prevpinchlv - lvy) <
Controller.pinch_threshold:
          Controller.f
                rame
                count
                += 1
                else:
             Controller.prevpin
             chlv = lvy
             Controller.framec
             ount = 0



      elif abs(lvx) > Controller.pinch_threshold:
Controller.pinchdirectionflag = True
          if abs(Controller.prevpinchlv - lvx) <
Controller.pinch_threshold:
```

```
        Controller.f
                rame
                count
                += 1
                else:
            Controller.prevpin
            chlv = lvx
            Controller.framec
            ount = 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_re
sult)
    # 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 = Fals
        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.scrollHorizontal,
        Controller.scrollVertical)
    elif gesture ==
Gest.PINCH_MAJOR: if
Controller.pinchmajorflag ==
False:
            Controller.pinch_control_init(han
            d_result)
            Controller.pinchmajorflag = True
        Controller.pinch_control(hand_result,Controller.changesystembrightness,
        Controller.changesystemvolume)



'''
                                Main Class

Entry point of Gesture
Controller '''
  class
    GestureCo
    ntroller:
    """
  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_m
ode = 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 =
```

```
        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_m

ajor = left

GestureController.hr_mi

nor = right

7.3 PERFORMANCE ANALYSIS AND MEASURES:

Here's an outline of performance analysis and measures for our project:

Speed and Responsiveness:

- Measure the time taken for each step in your system, such as eye detection, gesture recognition, and cursor control.
- Assess the responsiveness of the system to user inputs, such as hand gestures or voice commands, by measuring the latency between input and action.
- Optimize algorithms and implementations to reduce latency and improve real-time performance.

Accuracy:

- Evaluate the accuracy of eye detection and tracking algorithms by comparing the detected eye positions with ground truth data.

- Measure the precision of gesture recognition by comparing the recognized gestures with the intended gestures performed by the user.
- Implement calibration routines to fine-tune the accuracy of cursor control based on the user's eye movements and gestures.

Robustness:

- Test the system under various lighting conditions, camera angles, and user environments to assess its robustness.
- Evaluate the system's ability to handle occlusions, such as partial obstruction of the user's face or hands, without significant degradation in performance.
- Implement error handling mechanisms to gracefully handle unexpected inputs or failures, ensuring the system remains stable and responsive.

Resource Utilization:

- Monitor the system's resource utilization, such as CPU and memory usage, to ensure efficient use of hardware resources.
- Profile the performance of individual components, such as image processing algorithms and gesture recognition modules, to identify bottlenecks and optimize resource usage.

User Experience:

- Gather feedback from users through surveys, interviews, or usability testing sessions to assess their satisfaction with the system's performance.
- Identify areas for improvement based on user feedback, such as interface design, responsiveness, and ease of use.
- Iterate on the design and implementation based on user feedback to enhance the overall user experience.

Scalability:

- Assess the scalability of the system to handle varying levels of user interactions, such as increasing the number of simultaneous users or supporting different input modalities.

- Evaluate the system's performance under load and identify any scalability limitations or constraints.

- Implement scalability enhancements, such as distributed processing or parallelization, to improve the system's ability to handle increased workload.

By conducting thorough performance analysis and measurement across these dimensions, you can ensure that your project meets its performance goals and delivers a reliable and responsive user experience.

A hand-gesture- controlled virtual mouse could provide an alternative method for people with disabilities who may have difficulty using a traditional mouse or keyboard. This technology can make it easier for them to interact with computers and other devices. A hand gesture-controlled virtual mouse could also be useful for people who prefer to work or play games without being tethered to a physical mouse touchpad. This model would allow them to control their devices without the need for a physical interface.

Depending on the technology used, a hand gesture-controlled virtual mouse may offer a higher degree of accuracy and speed than traditional mice or video editing. The success of this technology will depend on the user experience it provides. If the technology is easy to use, reliable and provides an intuitive interface, likely to be well-received. However, if the technology is difficult to use, unreliable, or unintuitive , users may quickly abandon it.

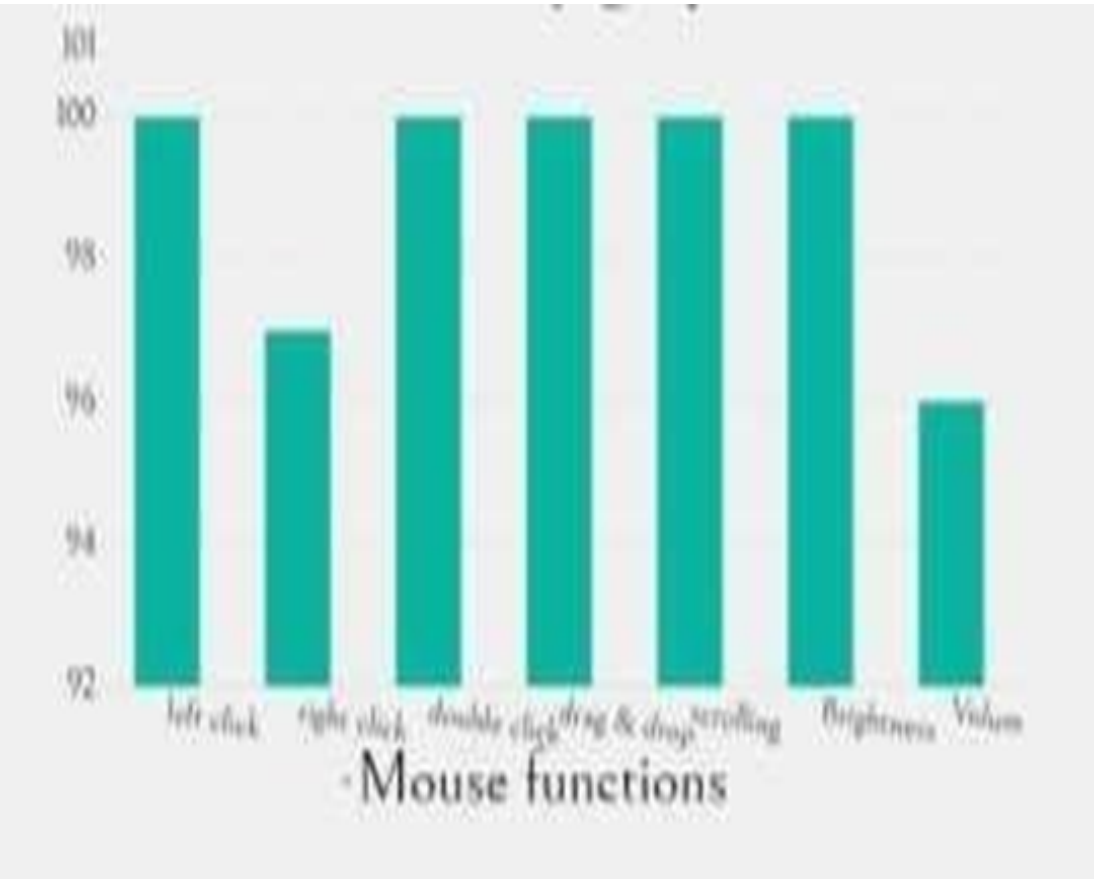

Fig: 7.3.1 Accuracy For Mouse Function.

8. TESTING

8. TESTING

Testing an AI Virtual Mouse System involves a systematic approach to ensure accuracy, responsiveness, and reliability across various scenarios and environments. Here's a structured guide to testing your AI virtual mouse:

Setup and Environment Preparation

Hardware Requirements:

A functional webcam or depth sensor with suitable resolution.

A computer with sufficient processing power and GPU capability (if required).

Software Setup:

Install the necessary AI models, libraries (e.g., OpenCV, TensorFlow, or PyTorch), and drivers.

Ensure the testing environment matches the intended deployment platform (e.g.Windows, macOS, Linux).

Lighting and Background:

Test under different lighting conditions (low light, bright light, natural light). Use varied backgrounds to check for noise handling and robustness.

Testing Scenarios

Cursor Movement

Smoothness and Precision:

Move your hand or gesture slowly and rapidly to test the system's responsiveness. Evaluate whether the cursor aligns accurately with your gestures.

Boundary Testing:

Test the cursor near screen edges to ensure it doesn't overshoot or lag.

Gesture Recognition

Basic Gestures:

Test core gestures like left-click, right-click, double-click, drag, and scroll.

Ensure gestures are consistently recognized across multiple attempts.

Complex Gestures:

Include multi-step or compound gestures to see if the AI differentiates them effectively.

Speed and Latency

Response Time:

Measure the time between a gesture and the corresponding cursor movement. Identify if delays occur under specific conditions.

User Adaptability

Multiple Users:

Test with users of different hand sizes, skin tones, and movement styles. Evaluate the system's ability to adapt or learn user-specific variations.

Edge Cases

Occlusions:

Test scenarios where parts of the hand or face are temporarily obscured. Assess recovery speed and accuracy after occlusion.

Distractions:

Introduce external movements (e.g., waving objects) to test the system's robustness. Ensure the AI focuses on intended gestures.

Unintended Gestures:

Perform random movements to check if the system misinterprets them as valid command.

Performance Metrics

Accuracy Rate:

Percentage of correctly recognized gestures.

False Positive/Negative Rate:

Rate of unintended actions or missed gestures.

Latency:

Average time taken to process input and reflect output.

Usability Rating:

User feedback on ease of use, comfort, and intuitive design.

Real-World Usability Testing

Deploy the system in actual usage scenarios such as:

i. Navigating websites.

ii. Playing games.

iii. Managing tasks like dragging files or interacting with productivity software.

iv. Gather feedback from real users to refine and improve the system.

Debugging and Optimization

i. Analyze logs to identify frequent failures or lag sources.

ii. Optimize algorithms for better performance on diverse hardware setups.

iii. Retrain the AI model with additional data if recognition accuracy is low.

Testing ensures your AI virtual mouse provides a seamless, intuitive, and inclusive experience for all users.

9. FUTURE ENHANCEMENTS

9. FUTURE ENHANCEMENTS

The application needs a number of features and enhancements to be more versatile, accurate, and user-friendly in a variety of settings. The required upgrades and functionalities are described as follows:

Intelligent Recognition Algorithm

An adjustable zoom-in/out function is needed to increase the covered range because the existing recognition mechanism can only detect objects within a 25 cm radius. This function can automatically modify the focus rate dependent on the proximity between both the operator and the video camera.

Increased Efficiency

The hardware of the device, which comprises the processor's processing speed and the amount of the memory space, has a significant impact on the download speed. RAM and the webcam's capabilities. As a result, the program might run more effectively when it's installed on a good computer with a webcam that works well in various lighting conditions.

10. CONCLUSION

10. CONCLUSION

The Proton Chatbot project represents a sophisticated and versatile virtual assistant designed to streamline user interactions with their digital environment. By seamlessly integrating voice recognition, gesture control, and a range of system commands, the chatbot offers users a convenient and intuitive interface for performing various tasks, from web searches to file navigation. With its robust architecture and user-friendly design, the chatbot stands as a testament to the power of AI-driven technologies in enhancing productivity and accessibility in everyday computing.

The success of the Proton Chatbot lies not only in its technical capabilities but also in its potential for further innovation and expansion. As technology continues to evolve, so too will the capabilities of the chatbot, opening up new possibilities for enhancing user experiences and addressing emerging needs. By harnessing the latest advancements in natural language processing, machine learning, and human-computer interaction, the chatbot can continue to adapt and evolve to meet the changing demands of its users.

11. BIBLIOGRAPHY

11. BIBLIOGRAPHY

1. OpenCV Documentation: https://opencv.org/documentation/

2. PyAutoGUI Documentation: https://pyautogui.readthedocs.io/en/latest/index.html

3. "Python Programming for Beginners: An Introduction to the Python Computer Language and Computer Programming" by Jason Cannon, Independently published, 2019.

4. Quam, D.L., et.al. (1990). Gesture Recognition with a Dataglove. In IEEE conference on Aerospace and Electronics (pp. 755-760).

5. Guoli Wang., et.al. (2015). Optical Mouse Sensor-Based Laser Spot Tracking for HCI input, Proceedings of the Chinese Intelligent Systems Conference (pp. 329-340).

6. Baldauf, M., and Frohlich, p. (2013). Supporting Hand Gesture Manipulation of Projected Content with mobile phones. In the European conference on computer vision (pp. 381-390).

7. Roshnee Matlani., Roshan Dadlani., Sharv Dumbre., Shruti Mishra., & Abha Tewari. (2021). Virtual Mouse Hand Gestures. In the International Conference on Technology Advancements and innovations (pp. 340-345).

8. Mayur, Yeshi., Pradeep, Kale., Bhushan, Yeshi., & Vinod Sonawane. (2016). Hand Gesture Recognition for Human-Computer Interaction. In the international journal of scientific development and research (pp. 9-13).

9. Shriram, S., Nagaraj, B., Jaya, J., Sankar, S., & Ajay, P. (2021). Deep Learning Based Real Time AI Virtual Mouse System Using Computer Vision to Avoid COVID-19 Spread. In the Journal of Healthcare Engineering (pp. 3076-3083).

10. Steven Raj, N., Veeresh Gobbur, S., Praveen., Rahul Patil., & Veerendra Naik. (2020). Implementing Hand Gesture Mouse Using OpenCV. In the International Research Journal of Engineering and Technology (pp. 4257-4261).

11. Sneha, U., Monika, B., & Ashwini, M. (2013). Cursor Control System Using Hand Gesture Recognition. In the International Journal of Advanced Research in Computer and Communication Engineering (pp. 2278-1021).

PAPER PUBLICATION

DOI: 10.55041/IJSREM48641

ISSN: 2582-3930

Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT

An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

N. Santhoshi

in recognition to the publication of paper titled

AIRMOUSE: AI-Powered Gesture Control

published in IJSREM Journal on Volume 09 Issue 05 May, 2025

www.ijsrem.com

Editor-in-Chief
IJSREM Journal

e-mail: editor@ijsrem.com

DOI: 10.55041/IJSREM48641

ISSN: 2582-3930

Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT

An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Srinivas Gadige

in recognition to the publication of paper titled

AIRMOUSE: AI-Powered Gesture Control

published in IJSREM Journal on Volume 09 Issue 05 May, 2025

www.ijsrem.com

Editor-in-Chief
IJSREM Journal

e-mail: editor@ijsrem.com

DOI: 10.55041/IJSREM48641

ISSN: 2582-3930

Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT

An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Sanda Rohan

in recognition to the publication of paper titled

AIRMOUSE: AI-Powered Gesture Control

published in IJSREM Journal on Volume 09 Issue 05 May, 2025

Editor-in-Chief
IJSREM Journal

www.ijsrem.com

e-mail: editor@ijsrem.com

DOI: 10.55041/IJSREM48641

ISSN: 2582-3930

Impact Factor: 8.586

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT

An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

Jamalpur Uday Kumar

in recognition to the publication of paper titled

AIRMOUSE: AI-Powered Gesture Control

published in IJSREM Journal on Volume 09 Issue 05 May, 2025

Editor-in-Chief
IJSREM Journal

www.ijsrem.com

e-mail: editor@ijsrem.com

Students' Roll Numbers	Students' names	Title	Internal Guide Name
21311A6625	Gadige Srinivas	AIR MOUSE: AI-Powered Gesture Control	Mrs. N Santhoshi
21311A6637	Sanda Rohan		
21311A6658	Jamalpur Uday Kumar		

ABSTRACT

AirMouse is an innovative, touchless cursor control system designed to replace traditional pointing devices by utilizing real-time hand-tracking and gesture recognition. It combines OpenCV's advanced computer vision capabilities with MediaPipe's precise hand landmark detection and PyAutoGUI's control interface to map intuitive hand movements—such as pinches, swipes, and palm motions—into standard mouse functions like clicking, dragging, scrolling, and cursor navigation. This allows users to interact with digital interfaces without physical contact, making it both convenient and futuristic.

The system enhances comfort by reducing strain from prolonged mouse usage and offers improved accessibility for individuals with motor impairments or in environments requiring high hygiene standards. Early tests show promising results, including over 90% tracking accuracy and sub-150ms click latency on mid-range hardware across varied lighting conditions. Future developments aim to expand gesture vocabulary with multi-finger recognition, implement adaptive learning for personalized gesture sensitivity, and explore AR-based visual feedback for an even more immersive user experience.

Domain of the Major Project

Domain	Tick the appropriate column	Nature of the Project (Product/Application/Research)
Privacy and Security		
Artificial Intelligence		
Machine Learning	√	Research
Web Development		
Mobile App Development		
Cloud Computing		
Data Engineering		
Networking		
If none above, specify any other		

Correlation of Major Project with Program Outcomes (Mention as L, M or H)
L: Low, M: Medium, H: High

PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12

Correlation of Major Project with Program Specific Outcomes (Mention as L, M or H)
L: Low, M: Medium, H: High

PSO 1	PSO 2

Internal Guide	Project Coordinator	Head of the Dept.
Mrs. N Santhoshi Asst. Professor Dept of CSE(AI & ML), SNIST	**Mr. Ravi Gugulothu,** Assistant Professor, Dept of CSE(AI & ML), SNIST	**Dr. K. Shirisha** Professor and Head, Dept of CSE(AI & ML), SNIST