GESTURE RECOGNITION SYSTEM FOR VIRTUAL MOUSE CONTROL

A PROJECT REPORT

Submitted by

PRABURAM.K. V	**1904034**
RAKSHEKA.R	**1904039**
SWETHA.S	**1904060**
YUGASHINI.A	**1904062**

Under the guidance of

Ms. K. Harini,
Assistant Professor
Department of Electronics and Communication Engineering

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

COIMBATORE INSTITUTE OF TECHNOLOGY

(Government Aided Autonomous Institution affiliated to Anna University)

COIMBATORE – 641 014

ANNA UNIVERSITY: CHENNAI 600025

NOVEMBER 2022

COIMBATORE INSTITUTE OF TECHNOLOGY

(Government aided Autonomous Institution Affiliated to Anna University)
COIMBATORE – 641014

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this report **"GESTURE RECOGNITION SYSTEM FOR VIRTUAL MOUSE CONTROL"** is the Bonafide work of **PRABURAM.K.V (1904034), RAKSHEKA.R (1904039), SWETHA.S (1904060), YUGASHINI.A (1904062)** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Ms. K. Harini M.E.,	**Dr. A. RAJESWARI, M.E. Ph.D.,**
SUPERVISOR	**PRINCIPAL AND HoD**
Assistant Professor	Department of Electronics and
Department of Electronics and	Communication Engineering,
Communication Engineering,	Coimbatore Institute of Technology,
Coimbatore Institute of Technology,	Coimbatore – 641 014.
Coimbatore – 641 014.	

Submitted for **19EC76 Project Phase-I** held on

_____

INTERNAL EXAMINER	EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We wholeheartedly thank the management of Coimbatore Institute of Technology for providing us the necessary infrastructure which was very much helpful to complete our project.

We sincerely thank our Principal and Head of Electronics and Communication Engineering Department **Dr. A. Rajeswari, M.E., Ph.D.,** Coimbatore Institute of Technology, for providing us with the necessary facilities in completing our project efficiently.

We also express our profound gratitude to our supervisor **Ms. K. Harini, M.E.**, Assistant Professor, Department of Electronics and Communication Engineering, Coimbatore Institute of Technology, for providing valuable ideas in completing our project successfully.

We also express our sincere thanks to our Tutors, Committee Members and Project Coordinators, Department of Electronics and Communication Engineering, Coimbatore Institute of Technology, for their guidance and motivation towards completion of our project.

We also extend our thanks to all our teaching faculty and non-teaching staffs of our department for their kind attitude towards completion of our project work.

ABSTRACT

The evolution of Human Computer Interaction (HCI) has been developed from punch cards to keyboards and mouses and it is now moving towards graphical-based user interfaces. The objective of this project is to develop a graphical-based AI controlled mouse pointer using hand gesture recognition with the help of image processing and computer vision. This work aims to focus on the use of web cameras to develop a virtual HCI device in a cost-effective manner. Based on simple dynamic hand gestures and movements, the machine would be able to track and respond accordingly. The hand motion tracking is done using Artificial Intelligence (AI) and computer vision (CV) by laying over anchor points around the hand. This could reduce the use of mechanical devices and wires, overcoming the difficulties of optical and wireless mice. This project aims to create a vision-based system to control various mouse activities using hand gestures to make the interaction more efficient and reliable and to use this model for various other applications.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS & SYMBOLS

ABBREVATION	EXPANSION
HCI	Human computer interaction
CV	Computer Vision
AR	Augmented reality
VR	Virtual reality
HGR	Hand gesture recognition
AI	Artificial Intelligence
LLP	Landmark localization points

CHAPTER 1

INTRODUCTION

This chapter gives an overview of the Gesture based mouse control system, presents the need and objective of the project.

1.1 OVERVIEW

The hand gesture-based virtual mouse is a software program that allows users to provide mouse inputs to a device without using a physical mouse. This research presents a computer creative hand gesture based virtual mouse device that produces, using hand gestures and hand tip detection, for performing mouse activities on the computer. The major purpose of the suggested device is to perform laptop mouse cursor functions using a webcam or a built-in digital camera within the laptop rather than a conventional mouse device. A computer web camera is used in conjunction with various image processing techniques to create a hand gesture-based digital mouse.

In this research hand movements of a user are used as mouse inputs. A web camera is a set of cameras that indefinitely take photos, and most laptops now include them. Webcams have also been used by security applications that use face recognition to harness the potential of face detection. To fully utilize the power of a system camera, it can be used for Vision-based CC is frequently used, which eliminates the need for such a computer mouse and mouse pad. They can also be used in HCI applications such as motion controllers and sign language databases, which can benefit greatly from using a system camera.
A wireless mouse is used to control a system camera. A wireless mouse or a Bluetooth mouse takes several components to work, including a mouse, a

dongle, and a battery, but in this project, the client will operate the computer mouse using hand gestures using a built-in camera or a web camera.

A real-time on-device hand tracking solution that predicts a hand skeleton of a human from a single RGB camera for various applications. The pipeline consists of two models: 1) a palm detector, that is providing a bounding box of a hand to, 2) a hand landmark model, that is predicting the hand skeleton. It is implemented via MediaPipe, a framework for building cross-platform ML solutions. The proposed model and pipeline architecture demonstrate real-time gesture recognition that virtually controls the mouse with high prediction quality.

1.2 NEED FOR THE PROJECT

HCI is crucial in designing intuitive interfaces that people with different abilities and expertise usually access. Most importantly, human-computer interaction is helpful for communities lacking knowledge and formal training on interacting with specific computing systems.The existing mouses' accuracy can be reduced if the open hole leading to the laser area is dirty. Mechanical mice require a hard, flat and frictional surface for proper functioning.This makes the virtual mouse much more convenient to use on the go. Mechanical mouses require constant stream of energy in the form of batteries or through wires so that it can constantly relate the mouse's position to the cursor. So, making things virtual will solve this hinderance.

1.3 OBJECTIVE

To develop a hand tracking and gesture recognition system for human computer interaction. The goal of this work is to develop a computation model to classify various dynamic hand gestures and use it in navigation of mouse pointers.

1.4 PROBLEM STATEMENT

To operate a software method to resolve the challenge, keep in mind the issue. The goal here is to devise the most efficient process for humans to interact with a laptop without having any physical interface with it. Many concepts had been suggested, but they all required hardware motion. Given that hand gesture and hand Tip detection areas are used to handle the laptop mouse functions using a webcam or digital camera, AI digital mouse is frequently used to overcome these challenges Using a simple web digital cam, this research aims to reduce costs and improve the robustness of the suggested machine.

To design a computer vision model that can interact with the computer using dynamic hand gestures without adding much computation weight on the computer and could recognize Hand gestures using a webcam.

1.5 ORGANISATION OF REPORT

CHAPTER 1: Gives an overview of the Gesture recognition system for virtual mouse control, states the need and objective of the project

CHAPTER 2: Discusses the literature review

CHAPTER 3: Explains the concept and working of the Gesture recognition system.

CHAPTER 4: Shows the simulation results of the proposed system model

CHAPTER 5: Derives the conclusion of the project

CHAPTER 2

LITERATURE SURVEY

This chapter gives an overview of research carried out related to the project "GESTURE RECOGNITION SYSTEM FOR VIRTUAL MOUSE CONTROL".

As trendy technology of human pc interactions become necessary in our everyday lives, sorts of a mouse of quite shapes and sizes were unread, from an inform workplace mouse to a hard-core diversion mouse. However, there are some limitations to this hardware as they're not as environmentally friendly as it.

Sande et al.[1] suggested The present virtual mouse control system comprises generally mouse operations that control the hand gesture-based virtual mouse, left-click, right-click, and scrap-down, among other things, using a hand gesture detection system. Although there are several hand recognition systems, the one they chose was static hand recognition, which is merely a recognition of the shape created by the hand and therefore the definition of action for each shape made, which is confined to a few defined actions and generates a lot of confusion. There are more and additional alternatives to using a mouse as technology progresses.

Thakur et al.[2] suggested A hand gesture-based system to handle various mouse actions such as eft and right-clicking, scrolling up and down, and other mouse actions using hand gestures to provide interaction, additional efficiency, and reliability. This paper delineates a hand gesture-based interface for regulating a computer mouse via 2D hand gestures. Colour detection algorithms based on cameras are used to detect hand movements. This technique primarily focuses on the effective usage of a Web Camera to create a virtual device. Each

input image's centroid is located. Because hand movement directly moves the centroid, it is the sensing principle for changing the pointer on a computer screen. The left and right-click scroll up down functions of a mouse are implemented by folding the first and middle fingers of the hand respectively, and developing So, comparing the length of fingers images with those in the image gives an idea about the functionality performed by the hand gesture-based virtual mouse.

Sung et al.[3] proposed a machine learning model to track hand skeleton using 21 landmark points and to recognize hand gestures in a video. This real time tracker reduces the complexity by reducing the image quality. This prediction of hand skeleton of a human from a RGB camera for AR/VR applications. Similarly Y.Li et al [4] proposed a neural network based architecture for keypoint detection using K-means cluster algorithms based on object keypoint similarity(OKS). It outperforms previous methods in both accuracy and speed. A single network based on pose anchors that can output the 2D hand pose directly from an entire image.

Horaitu-Stefan grif [7] had proposed a gesture recognition model based on colour detection from images. Here, the person controlling the computer wears a colour band in his hand and the system recognizes the index finger based on the colour codes.The gestures are recognized from the orientation of the index finger with respect to other fingers. This model gave a good accuracy in both low and high intensity light environments. Limitation of this model is that it could only interpret the images in 2D and could not be used for real time tracking.

CHAPTER 3

GESTURE BASED MOUSE CONTROL SYSTEM

3.1 INTRODUCTION

This chapter describes the proposed system model for Gesture recognition system for virtual mouse control, the working principle along with various modules used in the project. This section explains about the usage of the frameworks and its integration in our project.

3.2 BLOCK DIAGRAM

The block diagram of the proposed system is shown in Fig. 3.1, it depicts the algorithms and frameworks used in implementing Gesture recognition system for virtual mouse control.

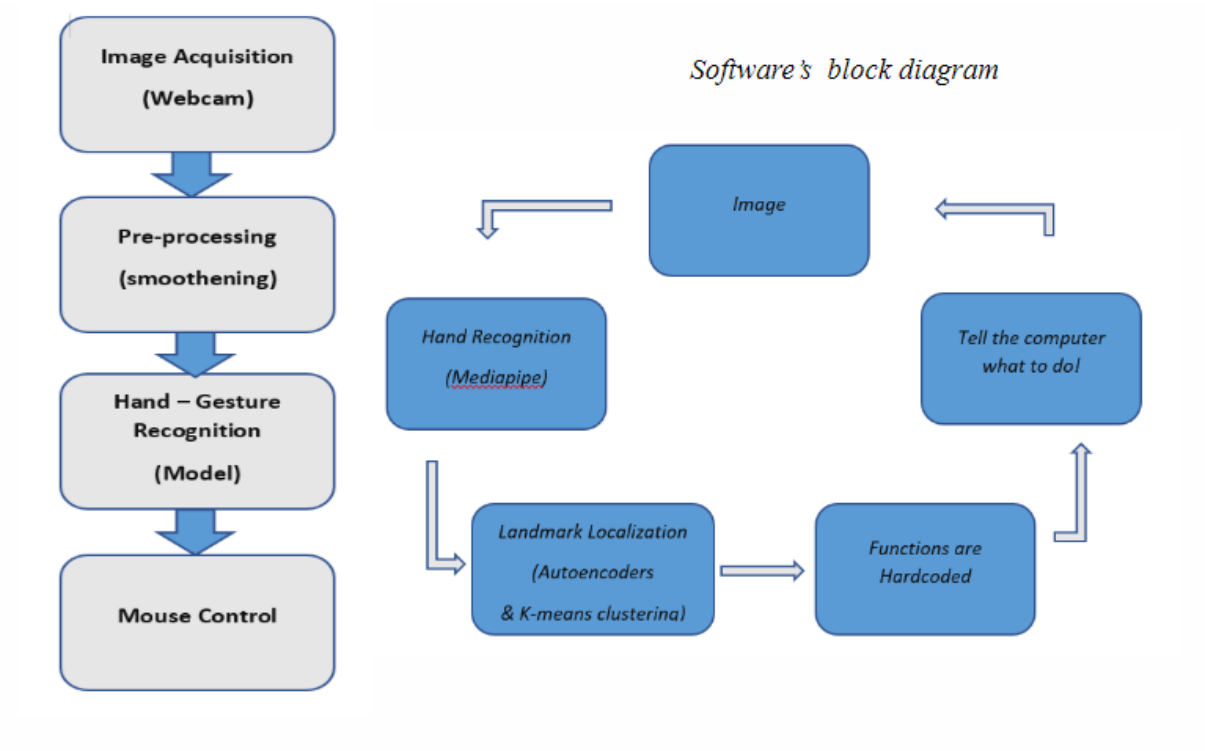


Fig 3.1 Block diagram of the proposed system.

3.3 FRAMEWORKS / ALGORITHMS

3.3.1 MEDIAPIPE

MediaPipe is a framework that is employed for applying a machine learning pipeline and is associated it's a Google open-source framework. Because the framework is built using statistical knowledge, the MediaPipe framework will help for cross-platform programming. The MediaPipe framework is multimodal, which means it can handle a wide range of audio and video formats. The MediaPipe framework is employed by the developers to create and analyse systems using graphs, as well as to create systems for application development. The steps involved in a MediaPipe-based system are specified area unit methods out in the pipeline configuration. The pipeline will be able to run on a variety of platforms, allowing for quantifiability on mobile and desktop devices.

Using traditional computer vision techniques, MediaPipe Box Tracking has been enabling real-time tracking in Motion Stills, YouTube's privacy blur, and Google Lens for a number of years.

The box tracking system takes image frames from a video or camera stream and computes the monitored box positions for each frame by starting box positions with timestamps, signalling 2D regions of interest to track. The starting box positions in this particular use case are determined by object detection, but the starting position can also be supplied directly by the user or by another system. Three key parts make up our solution: a motion analysis part, a flow packager part, and a box tracking part.

MediaPipe Box Tracking and ML inference can be combined to create useful and effective pipelines. For instance, a pipeline for object detection and tracking can be built by combining box tracking with ML-based object detection. In comparison to executing detection every frame (like MediaPipe Object Detection), this pipeline has the following benefits with tracking.

It offers instance-based tracking, meaning that the object ID is preserved throughout multiple frames. It's not necessary for detection to run every frame. As a result, it is possible to run more accurate detection models with larger detection models that are nevertheless lightweight and real-time on mobile devices.

With the use of tracking, object localisation is temporally consistent, resulting in less jitter being visible across frames.

3.3.2 OPENCV

To analyse and apply codes depending on a patient's policy coverage, the system may also read and assess an 834 form. One of the largest data dictionaries in the business and 50 million adjudicated claims were used to train the auto coder at first. MVP-1 had 89% accurate coding when it was delivered.

The system will become smarter as the product progresses through its MVP stages because it has been taught with 4 to 6 billion records and is anticipated to reach 98% coding accuracy.

3.3.3 AUTOENCODERS

The AI auto coder is an intelligent system of engagement that offers an answer to the combinatorial math problem. It is capable of analysing a claim using seven to ten factors, producing over 6.1 billion combinations of codes and modifiers, and then applying the appropriate ones for ICD-10, CPT, and HCPCS. Fig 3.2 depicts the architecture of the auto encoders.

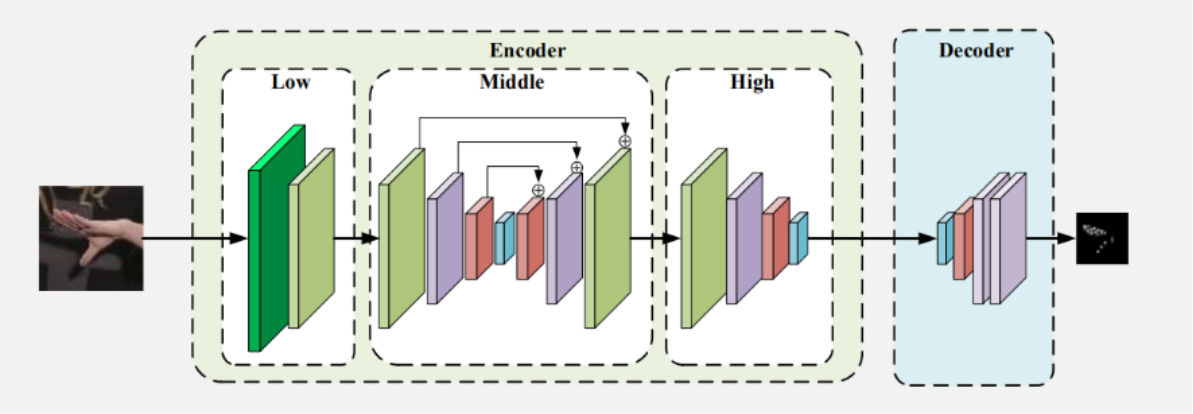


Fig 3.2 Autoencoder architecture

The appropriate claim form (such as 1500 or UB04) is then systematically applied with these codes, after which it may be routed internally for additional approvals or coding (such as charge master) or dropped off directly to the clearinghouse for quick processing of payments to payers.

3.4 PROPOSED MODEL

3.4.1 PALM DETECTION

The first stage of the proposed model is palm detection where the user's palm gets identified by the model. This is acheived by basic image detection and a pipeline created by using open CV and mediapipe framework.Usage of mediapipe framework in our project has facilitated the palm detection in the model. The pipelines deduced using the framework help detect the human palm on screen and also draw the closest bounding box around it.

To stumble on preliminary hand locations, we designed a single-shot detector version optimised for cellular real-time that makes use of in a way much like the face detection version in MediaPipe Face Mesh.

3.4.2 LANDMARK LOCALIZATION

The capacity to understand the form and movement of fingers may be a critical thing in enhancing the consumer's enjoyment throughout a whole lot of technological domain names and platforms. For example, it may shape the idea for signal language know-how and hand gesture control, and also can allow the overlay of virtual content material and facts on pinnacle of the bodily international in augmented reality. While coming evidently to people, sturdy real-time hand notion is a decidedly tough laptop imaginative and prescient task, as fingers regularly occlude themselves or every other (e.g. finger/palm occlusions and hand shakes) and absence excessive comparison patterns.

MediaPipe Hands is an excessive-constancy hand and finger monitoring solution. It employs system learning (ML) to deduce 21 three-D landmarks of a hand from only an unmarried frame. Whereas modern modern day strategies depend mostly on effective laptop environments for inference, our approach achieves real-time overall performance on a cell phone, or even scales to more than one finger. We hope that supplying this hand notion capability to the

broader studies and improvement network will bring about an emergence of innovative use cases, stimulating new programs and new studies avenues.

MediaPipe Hands makes use of an ML pipeline which include more than one fashions operating collectively: A palm detection version that operates on the entire photograph and returns an orientated hand bounding box. A hand landmark version that operates at the cropped photograph location described via means of the palm detector and returns high-constancy three-D hand keypoints. This method is much like that hired in our MediaPipe Face Mesh solution, which makes use of a face detector collectively with a face landmark version. Fig 3.3 shows the landmark localization provided by the model.

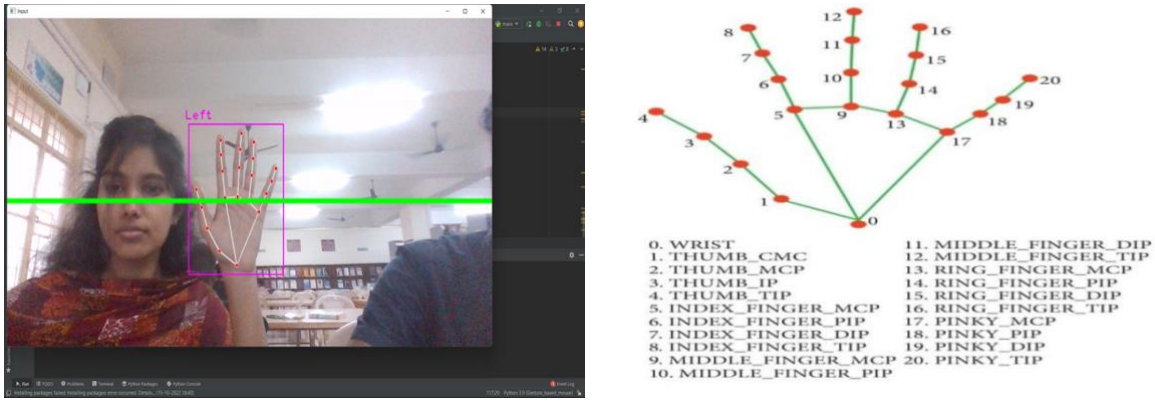


Fig 3.3 Landmark Localization

Providing the appropriately cropped hand photograph to the hand landmark version considerably reduces the want for statistics augmentation (e.g. rotations, translation and scale) and as an alternative permits the community to commit maximum of its ability toward coordinate prediction accuracy. In addition, in our pipeline the vegetation also can be generated primarily based totally at the hand landmarks recognized withinside the preceding frame, and most effective whilst the landmark version should not become aware of hand presence is palm detection invoked to relocalize the hand.

The pipeline is applied as a MediaPipe graph that makes use of a hand landmark monitoring subgraph from the hand landmark module, and renders the use of a dedicated hand renderer subgraph. The hand landmark monitoring subgraph internally makes use of a hand landmark subgraph from the equal module and a palm detection subgraph from the palm detection module.

After the palm detection over the complete photo our next hand landmark version plays unique keypoint localization of 21 3-D hand-knuckle coordinates in the detected hand areas thru regression, this is direct coordinate prediction. The version learns a regular inner hand pose illustration and is strong even to partly seen fingers and self-occlusions.

To gain floor reality data, we've manually annotated ~30K real-international snap shots with 21 3-D coordinates, as proven below (we take Z-cost from photo intensity map, if it exists consistent with corresponding coordinate). To better cowl the feasible hand poses and offer extra supervision on the character of hand geometry, we additionally render an excellent artificial hand version over diverse backgrounds and map it to the corresponding 3-D coordinates.

3.4.3 GESTURE DEVELOPMENT

Collection of detected/tracked hands, where each hand is represented as a list of 21 hand landmarks and each landmark is composed of x, y and z. x and y are normalised to [0.0, 1.0] by the image width and height respectively. z represents the landmark depth with the depth at the wrist being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.

Collection of detected/tracked hands, where each hand is represented as a list of 21 hand landmarks in world coordinates. Each landmark is composed of x, y and z: real-world 3D coordinates in metres with the origin at the hand's approximate geometric centre.

Collection of handedness of the detected/tracked hands (i.e. is it a left or right hand). Each hand is composed of a label and score. label is a string of values either "Left" or "Right". score is the estimated probability of the predicted handedness and is always greater than or equal to 0.5 (and the opposite handedness has an estimated probability of 1 - score).

3.5 SUMMARY

The gesture recognition system is done using an autoencoder based architecture and the mediapipe library. Autoencoder networks are unsupervised learning methods that are used to lay the landmark points on an image.

CHAPTER 4

EXPERIMENTAL RESULTS

This chapter presents the simulation results of the above work and explains the gestures the computer can recognize.

4.1 GESTURES USED

For the current project, there are four gestures that are created to perform necessary actions. The live actions can be performed live on a powerpoint presentation.

Major gestures are :

4.1.1 Pointer gesture

4.1.2 Left gesture

4.1.1 Right gesture

4.1.1 Pointer gesture

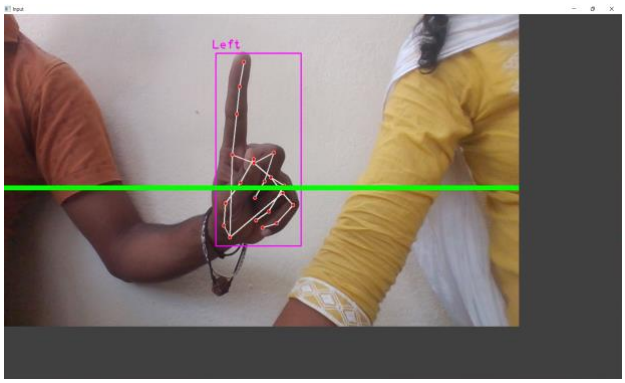
4.2 SIMULATION RESULTS

Palm detection model was built and run successfully with bounding boxes created every time the user's palm in infront of the camera.

The landmarks points (0 to 20) are localized and gestures are developed. There are a total 4 gestures in the model, each one signifying a certain cause.

4.2.1 POINTER GESTURE

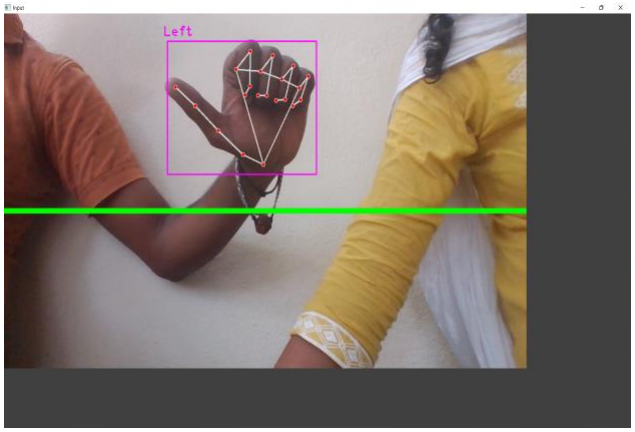
This gesture makes the user's pointing finger act like the mouse pointer. This gesture is not threshold limited as it works all across the screen.



4.2.1 Pointer gesture

4.2.2 LEFT GESTURE

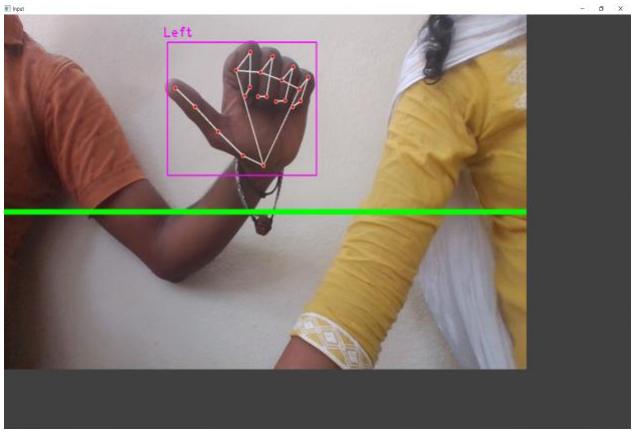
This gesture navigates the screen to the previous screen , for eg: it takes us to the last slide of the powerpoint.This gesture is threshold controlled in order to avoid mis signals, i.e the gesture is only recognized and the action following the gesture is performed if the palm is above the threshold line. The threshold line can be altered according to the need and purpose.



4.2.2 Left gesture

4.2.3 RIGHT GESTURE

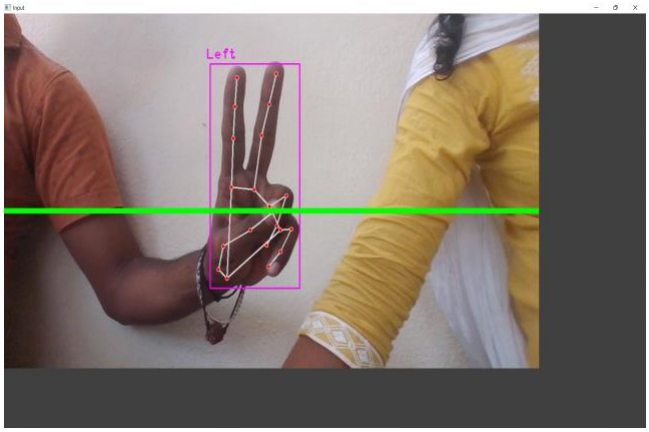
This gesture navigates the screen to the next screen , for eg: it takes us to the next slide of the powerpoint.This gesture is threshold controlled in order to avoid mis signals, i.e the gesture is only recognized and the action following the gesture is performed if the palm is above the threshold line. The threshold line can be altered according to the need and purpose.



4.2.3 Right gesture

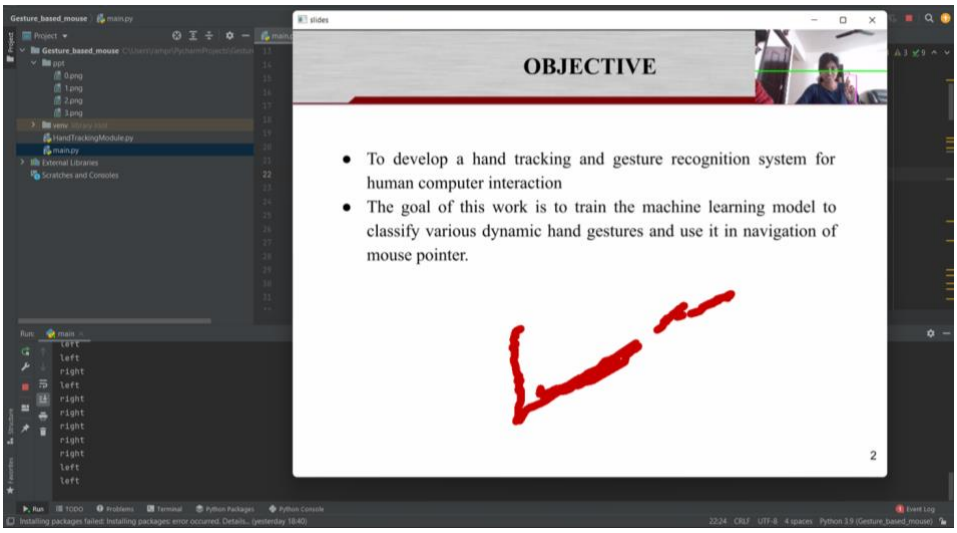
4.2.4 THE VIRTUAL PEN GESTURE

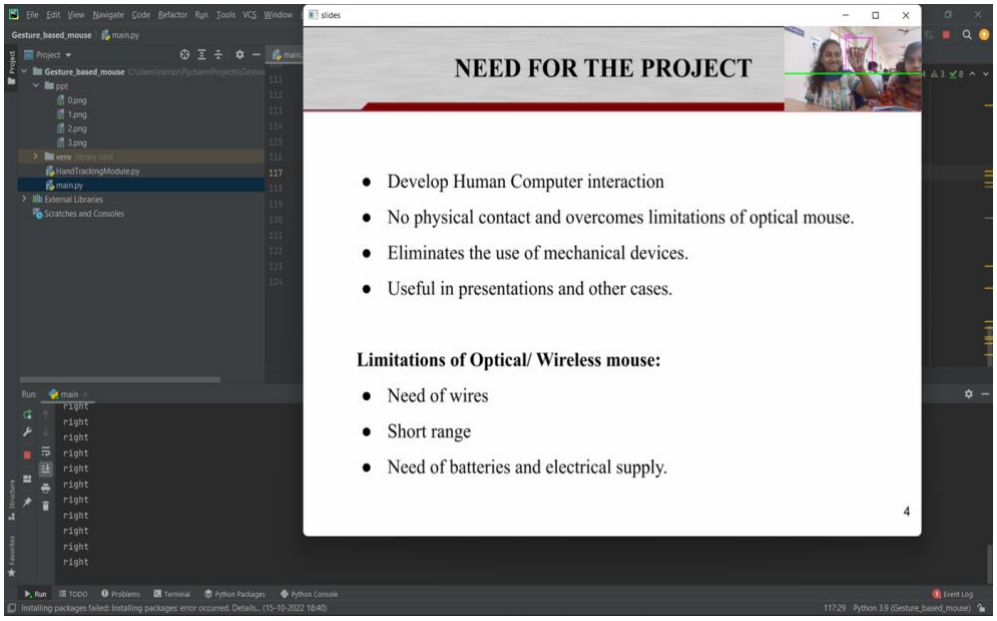
This gesture allows the user to use their hand as a virtual pen while they make short notes during the presentation.



4.2.4 Virtual pen gesture

The Results of the model and how the model interacts with the system is shown below in Fig.





4.2.4 Example of virtual gesture

4.3 SUMMARY

Based on the results and simulations obtained we can use this model in real time gesture tracking. Also this model could be used for various other applications or we could add more gestures. It reduces the complexity and computations as compared to the previous models.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

The main goal of the AI visual mouse system is to allow the user to control the mouse pointer function with a hand touch instead of manipulating things. Hand gestures and hand tips are captured and processed by the proposed system. You can access these systems through a webcam or built-in camera. The proposed model is so accurate that it can also be used in real-world applications. For

example, it can be used to reduce the spread of COVID19 and eliminate the need for wearable devices. Some of the limitations are the model could not detect hand on a low light environment. As a result, we will try to address these limitations in the future by developing fingerprint capture methods that provide more accurate results.

This model could also be used in Home automation i.e using hand gestures to control the appliances that could provide useful in the field of IoT.

REFERENCES

[1] Riza Sande, Neha Marathe, Neha Bhegade, AkankshaLugade, Prof. S. S. Jogdand “virtual mouse using hand gesture” 2021 International Journal of Advanced Research in Science, Engineering and Technology (IJARSET) pp17124-17132 2021.

[2] S. Thakur, R. Mehra, and B. Prakash, " Vision-based computer mouse control using hand gestures," 2015 International Conference on Soft Computing Techniques and Implementations (ICSCTI), Faridabad, 2015, pp. 85-87, 2015.

[3] Y. Li, X. Wang, W. Liu and B. Feng, "Pose Anchor: A Single-Stage Hand Keypoint Detection Network," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 7, pp. 2104-2113, July 2020, doi: 10.1109/TCSVT.2019.2912620.

[4] M. S. Ghute, M. Anjum and K. P. Kamble, "Gesture Based Mouse Control," 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), pp. 710-713, 2018, doi: 10.1109/ICECA.2018.8474905.

[5] S. Hazra and A. Santra, "Robust Gesture Recognition Using Millimetric-Wave Radar System," IEEE Sensors Letters, vol. 2, no. 4, pp. 1-4, Dec. 2018, Art no. 7001804, doi: 10.1109/LSENS.2018.2882642.

[6] Horatiu-Stefan Grif, Cornel Cristian Farcas, “Mouse Cursor Control System Based on Hand Gesture”, Procedia Technology, Volume 22, pp. 657-661, 2016, ISSN 2212-0173, doi:10.1016/j.protcy.2016.01.137.

[7] S. Thakur, R. Mehra and B. Prakash, "Vision based computer mouse control using hand gestures," 2015 International Conference on Soft Computing Techniques and Implementations (ICSCTI),pp. 85-89, 2015, doi: 10.1109/ICSCTI.2015.7489570.

[8] Aksaç, A., Öztürk, O., & Özyer, T. (2011, December). Real-time multi-objective hand posture/gesture recognition by using distance classifiers and finite state machine for virtual mouse operations. In: 2011 7th International Conference on Electrical and Electronics Engineering (ELECO), pp. II-457, IEEE.

[9] Shibly, K. H., Dey, S. K., Islam, M. A., & Showrav, S. I. (2019, May). Design and development of hand gesture based virtual mouse. In 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), pp. 1-5, IEEE.

[10] Farooq, J., & Ali, M. B. (2014, April). Real time hand gesture recognition for computer interaction. In 2014 International Conference on Robotics and Emerging Allied Technologies in Engineering (iCREATE) (pp. 73-77). IEEE.

Source Code:

Hand Tracker.py

```
import cv2
import mediapipe as mp
import math


class HandDetector:
    """
    Finds Hands using the mediapipe library. Exports the landmarks
    in pixel format. Adds extra functionalities like finding how
    many fingers are up or the distance between two fingers. Also
    provides bounding box info of the hand found.
    """


    def __init__(self, mode=False, maxHands=2, detectionCon=0.5, minTrackCon=0.5):
        """
        :param mode: In static mode, detection is done on each image: slower
        :param maxHands: Maximum number of hands to detect
        :param detectionCon: Minimum Detection Confidence Threshold
        :param minTrackCon: Minimum Tracking Confidence Threshold
        """
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.minTrackCon = minTrackCon
```

```
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(static_image_mode=self.mode, max_num_hands=self.maxHands,
                                        min_detection_confidence=self.detectionCon,
                                        min_tracking_confidence=self.minTrackCon)
        self.mpDraw = mp.solutions.drawing_utils
        self.tipIds = [4, 8, 12, 16, 20]
        self.fingers = []
        self.lmList = []


    def findHands(self, img, draw=True, flipType=True):
        """
        Finds hands in a BGR image.
        :param img: Image to find the hands in.
        :param draw: Flag to draw the output on the image.
        :return: Image with or without drawings
        """
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
        allHands = []
        h, w, c = img.shape
        if self.results.multi_hand_landmarks:
            for handType, handLms in zip(self.results.multi_handedness, self.results.multi_hand_landmarks):
                myHand = {}
                ## lmList
```

```
mylmList = []
xList = []
yList = []
for id, lm in enumerate(handLms.landmark):
    px, py, pz = int(lm.x * w), int(lm.y * h), int(lm.z * w)
    mylmList.append([px, py, pz])
    xList.append(px)
    yList.append(py)

## bbox
xmin, xmax = min(xList), max(xList)
ymin, ymax = min(yList), max(yList)
boxW, boxH = xmax - xmin, ymax - ymin
bbox = xmin, ymin, boxW, boxH
cx, cy = bbox[0] + (bbox[2] // 2), \
         bbox[1] + (bbox[3] // 2)

myHand["lmList"] = mylmList
myHand["bbox"] = bbox
myHand["center"] = (cx, cy)

if flipType:
    if handType.classification[0].label == "Right":
        myHand["type"] = "Left"
    else:
        myHand["type"] = "Right"
```

```
            else:
                myHand["type"] = handType.classification[0].label
            allHands.append(myHand)

            ## draw
            if draw:
                self.mpDraw.draw_landmarks(img, handLms,
                                           self.mpHands.HAND_CONNECTIONS)
                cv2.rectangle(img, (bbox[0] - 20, bbox[1] - 20),
                              (bbox[0] + bbox[2] + 20, bbox[1] + bbox[3] + 20),
                              (255, 0, 255), 2)
                cv2.putText(img, myHand["type"], (bbox[0] - 30, bbox[1] - 30), cv2.FONT_HERSHEY_PLAIN,
                            2, (255, 0, 255), 2)
        if draw:
            return allHands, img
        else:
            return allHands


    def fingersUp(self, myHand):
        """
        Finds how many fingers are open and returns in a list.
        Considers left and right hands separately
        :return: List of which fingers are up
        """
        myHandType = myHand["type"]
        myLmList = myHand["lmList"]
```

```
    if self.results.multi_hand_landmarks:
        fingers = []
        # Thumb
        if myHandType == "Right":
            if myLmList[self.tipIds[0]][0] > myLmList[self.tipIds[0] - 1][0]:
                fingers.append(1)
            else:
                fingers.append(0)
        else:
            if myLmList[self.tipIds[0]][0] < myLmList[self.tipIds[0] - 1][0]:
                fingers.append(1)
            else:
                fingers.append(0)

        # 4 Fingers
        for id in range(1, 5):
            if myLmList[self.tipIds[id]][1] < myLmList[self.tipIds[id] - 2][1]:
                fingers.append(1)
            else:
                fingers.append(0)
    return fingers

def findDistance(self, p1, p2, img=None):
    """
    Find the distance between two landmarks based on their
    index numbers.
```

```
        :param p1: Point1
        :param p2: Point2
        :param img: Image to draw on.
        :param draw: Flag to draw the output on the image.
        :return: Distance between the points
                 Image with output drawn
                 Line information
        """

        x1, y1 = p1
        x2, y2 = p2
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
        length = math.hypot(x2 - x1, y2 - y1)
        info = (x1, y1, x2, y2, cx, cy)
        if img is not None:
            cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
            cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
            cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)
            return length, info, img
        else:
            return length, info


def main():
    cap = cv2.VideoCapture(0)
```

```
detector = HandDetector(detectionCon=0.8, maxHands=2)
while True:
    # Get image frame
    success, img = cap.read()
    # Find the hand and its landmarks
    hands, img = detector.findHands(img)  # with draw
    # hands = detector.findHands(img, draw=False)  # without draw

    if hands:
        # Hand 1
        hand1 = hands[0]
        lmList1 = hand1["lmList"]  # List of 21 Landmark point
        bbox1 = hand1["bbox"]  # Bounding box info x,y,w,h
        centerPoint1 = hand1['center']  # center of the hand cx,cy
        handType1 = hand1["type"]  # Handtype Left or Right

        fingers1 = detector.fingersUp(hand1)

        if len(hands) == 2:
            # Hand 2
            hand2 = hands[1]
            lmList2 = hand2["lmList"]  # List of 21 Landmark points
            bbox2 = hand2["bbox"]  # Bounding box info x,y,w,h
            centerPoint2 = hand2['center']  # center of the hand cx,cy
            handType2 = hand2["type"]  # Hand Type "Left" or "Right"
```

```
            fingers2 = detector.fingersUp(hand2)


            # Find Distance between two Landmarks. Could be same hand or different hands
            length, info, img = detector.findDistance(lmList1[8][0:2], lmList2[8][0:2], img)  # with draw
            # length, info = detector.findDistance(lmList1[8], lmList2[8])  # with draw
        # Display
        cv2.imshow("Image", img)
        cv2.waitKey(1)



if __name__ == "__main__":
    main()
```

Main.py

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import os
import numpy as np

# var
width, height = 1280, 720
folderpath = "review ppt"
```

```
# camera
cap = cv2.VideoCapture(0)
cap.set(3, width)
cap.set(4, height)

# list of ppt
pathimages = sorted(os.listdir(folderpath), key=len)
# print(pathimages)

# variables
imgnumber = 0
hs, ws = 120, 213
gesture_threshold = 400
button_press = False
button_counter = 0
button_delay = 20
annotations = [[]]
annotation_number = -1
annotation_start = False


# hand detector
detector = HandDetector(detectionCon=0.8, maxHands=1)

# Check if the webcam is opened correctly
if not cap.isOpened():
```

```
    raise IOError("Cannot open webcam")

while True:
    # importing images.
    success, img = cap.read()
    img = cv2.flip(img, 1)
    pathfullimage = os.path.join(folderpath, pathimages[imgnumber])
    imgcurrent = cv2.imread(pathfullimage)

    hands, img = detector.findHands(img)
    cv2.line(img, (0, gesture_threshold), (width, gesture_threshold), (0, 255, 0),
10)

    if hands and button_press == False:
        hand = hands[0]
        fingers = detector.fingersUp(hand)
        cx, cy = hand['center']

        lmlist = hand['lmList']

        # constraining window
        xVal = int(np.interp(lmlist[8][0], [width//2, width], [0, width]))
        yVal = int(np.interp(lmlist[8][1], [150, height-150], [0, height]))
        index_finger = xVal, yVal  # index finger

        # print(fingers)
```

```
if cy <= gesture_threshold: # hand should be above line
    # gesture 1 - left slide
    if fingers == [1, 0, 0, 0, 0]:
        print("left")
        if imgnumber > 0:
            button_press = True
            annotations = [[]]
            annotation_number = -1
            annotation_start = False
            imgnumber -= 1
    # gesture 2 - right slide
    if fingers == [0, 0, 0, 0, 1]:
        print("right")
        if imgnumber < len(pathimages)-1:
            button_press = True
            annotations = [[]]
            annotation_number = -1
            annotation_start = False
            imgnumber += 1

# gesture 3 - Show pointer
if fingers == [0, 1, 1, 0, 0]:
    cv2.circle(imgcurrent, index_finger, 12, (0, 0, 255), cv2.FILLED)
    # annotation_start = False

# gesture 4 - draw
```

```
    if fingers == [0, 1, 0, 0, 0]:
        if annotation_start is False:
            annotation_start = True
            annotation_number += 1
            annotations.append([])
        annotations[annotation_number].append(index_finger)
        cv2.circle(imgcurrent, index_finger, 12, (0, 0, 255), cv2.FILLED)

    else:
        annotation_start = False

    # Gesture 5 - erase
    if fingers == [0, 1, 1, 1, 0]:
        if annotations:
            annotations.pop(-1)
            annotation_number -= 1
            button_press = True
else:
    annotation_start = False

# button iter
if button_press:
    button_counter += 1
    if button_counter > button_delay:
        button_counter = 0
        button_press = False
```

```
    # Drawing part
    for i in range(len(annotations)):
        for j in range(len(annotations[i])):
            if j != 0:
                cv2.line(imgcurrent, annotations[i][j-1], annotations[i][j], (0, 0, 200),
12)

    # adding webcam image on the slides.
    imgsmall = cv2.resize(img, (ws, hs))
    h, w, _ = imgcurrent.shape
    imgcurrent[0:hs, w-ws:w] = imgsmall

    cv2.imshow("slides", imgcurrent)
    cv2.imshow('Input', img)

    c = cv2.waitKey(1)
    if c == 27:
        break

cap.release()
cv2.destroyAllWindows()
```