VRTUAL MOUSE

A PROJECT REPORT

Submitted By

SHUBHAM MAHIND (21BAI10407)

SUPRATIK PAL (21BAI10354)

VINAYAK SINGH (21BAI10350)

PRATHAMESH GOLE (21BAI10346)

PRASHANT KUMAR MISHRA (21BAI10335)

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING WITH SPECIALIZATION IN ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
VIT BHOPAL UNIVERSITY
KOTHRIKALAN, SEHORE
MADHYA PRADESH - 466114

OCT 2022

VIT BHOPAL UNIVERSITY, KOTHRIKALAN, SEHORE
MADHYA PRADESH – 466114

BONAFIDE CERTIFICATE

Certified that this project report titled **"VIRTUAL MOUSE"** is the bonafide work of "**SHUBHAM MAHIND (21BAI10407)**, **SUPRATIK PAL (21BAI10354)**, **VINAYAK SINGH (21BAI10350)**, **PRATHAMESH GOLE (21BAI10346)** and **PRASHANT KUMAR MISHRA (21BAI10335)"** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

PROGRAM CHAIR	**PROJECT GUIDE**
Dr. Suthir Sriram	Dr. Nilamadhab Mishra
School of Computer Science and Engineering VIT BHOPAL UNIVERSITY	School of Computer Science and Engineering VIT BHOPAL UNIVERSITY

The Project Exhibition I Examination is held on 30/07/2022.

ACKNOWLEDGEMENT

First and foremost, I would like to thank the Lord Almighty for His presence and immense blessings throughout the project work.

I wish to express my heartfelt gratitude to Dr Dr. Suthir Sriram, Head of the Department, School of Computer Science for much of his valuable support encouragement in carrying out this work.

I would like to thank my internal guide Dr Nilamadhab Mishra, for continually guiding and actively participating in my project, giving valuable suggestions to complete the project work.

I would like to thank all the technical and teaching staff of the School of Aeronautical Science, who extended directly or indirectly all support.

Last, but not least, I am deeply indebted to my parents who have been the greatest support while I worked day and night for the project to make it a success.

ABSTRACT

The python based virtual mouse uses hand gestures to control the cursor and also to left click. We are creating this mouse so as to create a touchless interface to interact with our computer. Here we are tracking the hand movements using computer vision to perform the functions. We are implementing the code such that when we point our index finger and move our hand the cursor also moves along with it and to perform left click operation, our index and middle finger should touch each other.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	Abstract	4
1	**CHAPTER-1:** **PROJECT DESCRIPTION AND OUTLINE** 1.1 Introduction 1.2 Motivation for the work 1.3 Problem Statement 1.4 Objective of the work	6
2	**CHAPTER-2:** **RELATED WORK INVESTIGATION** 2.1 Existing Approaches/Methods 2.2 Cons of the stated Approaches/ Methods>	8
3	**CHAPTER-3:** **REQUIREMENT ARTIFACTS** 3.1 Hardware requirements 3.2 Software requirements	9
4	**CHAPTER-4:** **DESIGN METHODOLOGY AND ITS NOVELTY** 4.1 Setting up our environment 4.2 Creating a hand tracking program 4.3 Coding 4.4 Creating a module from our code 4.5 Result 4.6 Conclusion	10
5	**CHAPTER-5:** **TECHNICAL IMPLEMENTATION & ANALYSIS** 5.1 Code 5.2 Operations	19
7	**CHAPTER-6:** **CONCLUSIONS AND RECOMMENDATION** 6.1 Future Enhancements 6.2 Inference	23
	References	25

PROJECT DESCRIPTION AND OUTLINE

1.1 Introduction

One of the marvels of Human-Computer Interaction (HCI) technology is the mouse. Since a wireless mouse or Bluetooth mouse still need a battery for power and a dongle to connect it to the PC, they are not entirely device-free at this time. This restriction can be removed from the proposed AI virtual mouse system by using a webcam or a built-in camera to record hand motions and recognise hand tips using computer vision. The machine learning algorithm is utilised by the system's algorithm. Without using a real mouse, the computer can be virtually controlled using hand gestures to accomplish left-click, right-click, scrolling, and computer cursor tasks.

1.2 Motivation for the work

It is fair to say that the Virtual Mouse will soon to be substituting the traditional physical mouse in the near future, as people are aiming towards the lifestyle where that every technological device can be controlled and interacted remotely without using any peripheral devices such as the remote, keyboards, etc. it doesn't just provide convenience, but it's cost effective as well.

1.3 Problem Statement

It's no surprised that all technological devices have its own limitations, especially when it comes to computer devices. After the review of various type of the physical mouse, the problems are identified and generalized. The following describes the general problem that the current physical mouse suffers:

- Physical mouse is subjected to mechanical wear and tear.
- Physical mouse requires special hardware and surface to operate.
- Physical mouse is not easily adaptable to different environments and its performance varies depending on the environment.
- Mouse has limited functions even in present operational environments.
- All wired mouse and wireless mouse have its own lifespan.

1.4 Objective of the work

The purpose of this project is to develop a Virtual Mouse application that targets a few aspects of significant development. For starters, this project aims to eliminate the needs of having a

physical mouse while able to interact with the computer system through webcam by using various image processing techniques. Other than that, this project aims to develop a Virtual Mouse application that can be operational on all kind of surfaces and environment.

The following describes the overall objectives of this project:

- To design to operate with the help of a webcam. The Virtual Mouse application will be operational with the help of a webcam, as the webcam are responsible to capture the images in real time. The application would not work if there are no webcam detected.
- To design a virtual input that can operate on all surface. The Virtual Mouse application will be operational on all surface and indoor environment, as long the users are facing the webcam while doing the motion gesture.
- To convert hand gesture/motion into mouse input that will be set to a particular screen position.

RELATED WORK INVESTIGATION

2.1 Existing Approaches and Methods

A Computer Mouse is an input device that helps to point and to interact with whatever that is being pointed. There are so many types of mice in the current trend, there's the mechanical mouse that consists of a single rubber ball which can rotate in any direction and the movement of the pointer is determined by the motion of that rubber ball. Later the mechanical mouse is replaced by the Optical Mouse.

Optical Mouse consists of a led sensor to detect the movement of the pointer. Years Later the laser mouse was introduced to improve the accuracy and to overcome the drawbacks of the Optical Mouse. Later as the Technology has been increased drastically wireless mouse was introduced so as to enable hassle free movement of the mouse and to improve the accuracy.

2.2 Cons of the approach

No Matter how much the accuracy of the mouse increases but there will always be limitations of the mouse as the mouse is a hardware input device and there can be some problems like mouse click not functioning properly ad etc., as the mouse is a hardware device like any other physical object even the mouse will have a durability time within which is functional and after its durability time, we have to change the mouse Disadvantages:

- There will always be limitations of the mouse as the mouse is a hardware input device and there can be some problems like mouse click not functioning properly.
- The mouse is a hardware device like any other physical object even the mouse will have a durability time within which is functional and
- After its durability time we have to change the mouse.

REQUIREMENT ARTIFACTS

3.1 Hardware Requirements

- Computer System
- Webcam

3.2 Software Requirements

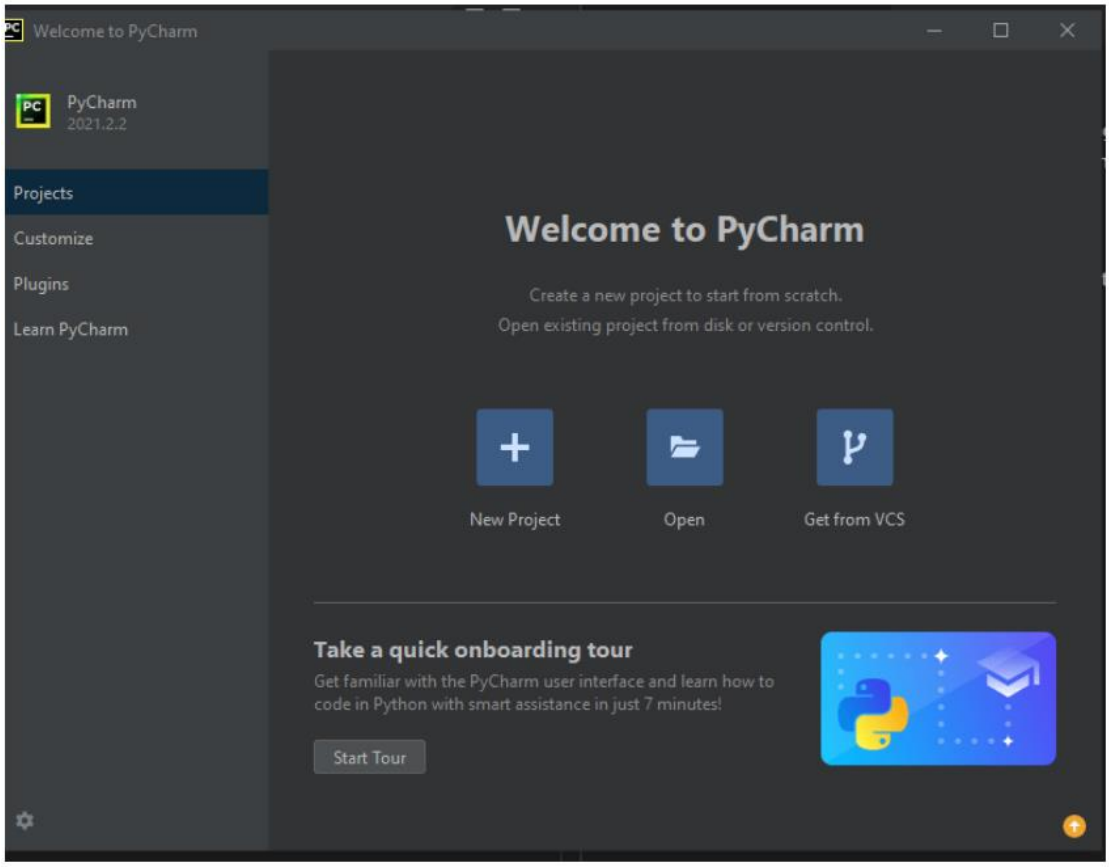
- Python 3.8.0
- Python modules- Mediapipe, OpenCV, Autopy, NumPy, Time, Math
- Pycharm IDE or Visual Studio Code for writing code

DESIGN METHODOLOGY AND ITS NOVELTY

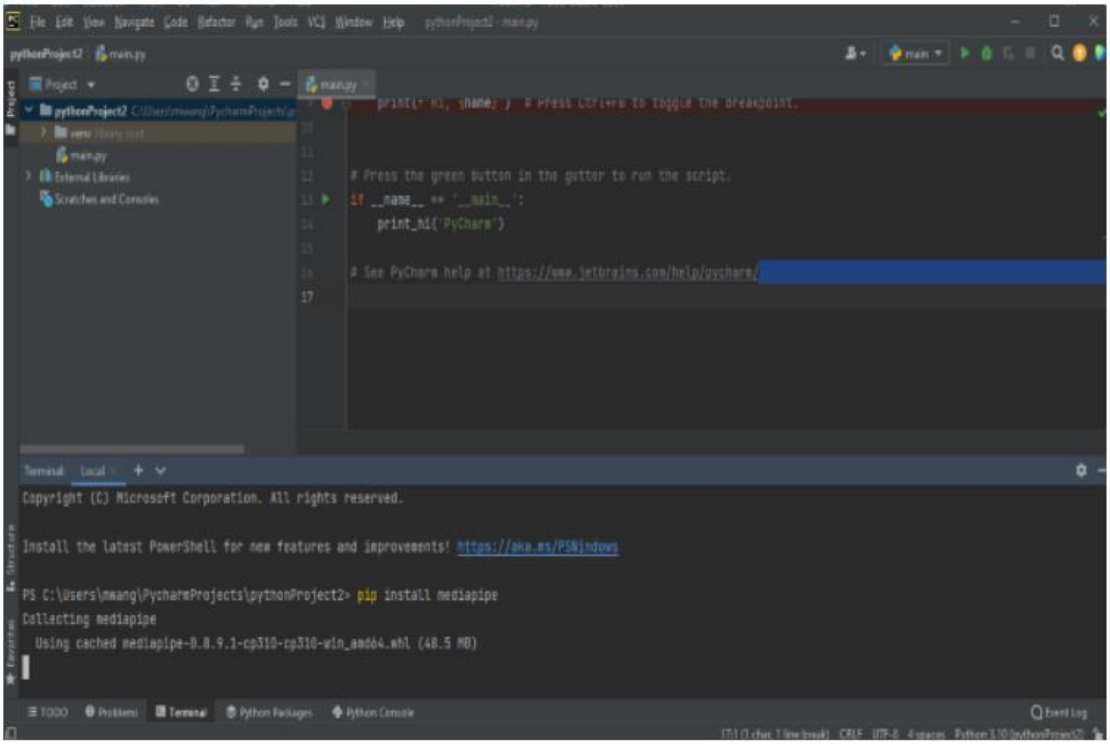
4.1 Setting up our environment

Launch the Pycharm app and do the following:

1. Click create a new project on the first window that is displayed. This is shown in the following screenshot:



2. On the window that appears next, click on create.
3. Install the two python libraries we discussed. To do so, open the terminal as shown in the screenshot below then follow the steps below it.



- Type the following command in the terminal to install MediaPipe:

```
pip install mediapipe
```

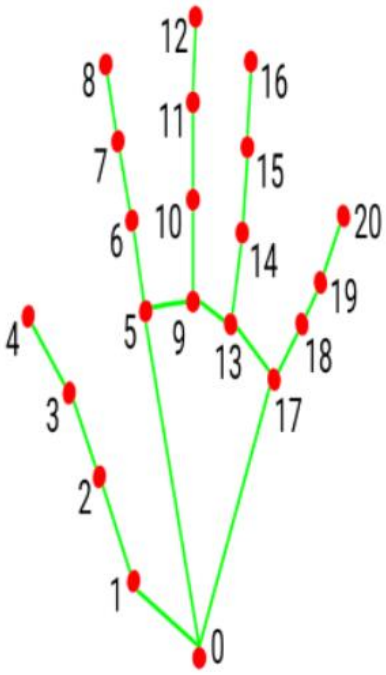
- To install openCV, use the following command:

```
pip install opencv-python
```

4.2 Creating a hand tracking program

- **Palm detection** - MediaPipe works on the complete input image and provides a cropped image of the hand.
- **Hand landmarks identification** - MediaPipe finds the 21 hand landmarks on the cropped image of the hand.

The 21 hand points that MediaPipe identifies are shown in the image below:



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

The image above shows the hand landmarks that MediaPipe uses to identify the hand. The numbered parts are the hand points.

4.3 Coding

Pycharm creates a main.py file for you automatically after you create a new project.

Step 1 - Importations and initializations

We start by importing the two libraries we discussed. Importing the libraries enables us to use its dependencies

We will then create an object cap for video capturing. We require the other three objects to manipulate our input using MediaPipe.

```
import cv2
import mediapipe as mp

cap = cv2.VideoCapture(0)
mpHands = mp.solutions.hands
hands = mpHands.Hands()
mpDraw = mp.solutions.drawing_utils
```

Step 2 - Capturing an image input and processing it

The code below takes the image input from the webcam. It then converts the image from BGR to RGB. This is because MediaPipe only works with RGB images, not BGR.

It then processes the RGB image to identify the hands in the image:

```
while True:
    success, img = cap.read()
    imageRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = hands.process(imageRGB)
```

Step 3 - Working with each hand

```
# checking whether a hand is detected
    if results.multi_hand_landmarks:
        for handLms in results.multi_hand_landmarks: # working
with each hand
            for id, lm in enumerate(handLms.landmark):
                h, w, c = image.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
```

In the code above, we use the if statement to check whether a hand is detected. We then use the first for loop to enable us work with one hand at a time.

The second for loop helps us get the hand landmark information which will give us the x and y coordinates of each listed point in the hand landmark diagram. This loop will also give us the id of each point.

We will then find the height, width, and channel of our image using the image.shape function. We finally get the central positions of the identified hand points.

Step 4 - Drawing the hand landmarks and hand connections on the hand image

```
if id == 20 :
                    cv2.circle(image, (cx, cy), 25, (255, 0,
255), cv2.FILLED)

            mpDraw.draw_landmarks(image, handLms,
mpHands.HAND_CONNECTIONS)
```

In the code above, we circle the hand point number 20. This is the tip of the pinkie finger.

Feel free to use the number of the hand point you want to circle as they are listed on the hand landmark diagram. We then draw the hand landmarks and the connections between them on the input image.

Step 5 - Displaying the output

```
cv2.imshow("Output", image)
    cv2.waitKey(1)
```

We use the code above to display the output to the user. The output is a real-time video of the user. It has the user's hands tracked, hand landmarks, and connections drawn on the hands.

4.4 Creating a module from our code

Create a new file and name it pyhandTrackingModule. Feel free to name it whatever name you want. Let us now create a module from the code above by following the steps below:

Step 1 - Importing the required libraries

We first import the Python libraries that we need in our project. This will enable us to use its dependencies:

```
import cv2
import mediapipe as mp
```

Step 2 - Creating a class that we will use for hand detection

```
class handTracker():
    def __init__(self, mode=False, maxHands=2,
detectionCon=0.5,modelComplexity=1,trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.modelComplex = modelComplexity
        self.trackCon = trackCon
        self.mpHands = mp.solutions.hands
```

```
        self.hands = self.mpHands.Hands(self.mode,
self.maxHands,self.modelComplex,
                                        self.detectionCon,
self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils
```

In the code above, we create a class that we will use for tracking. We then key in the basic parameters that we require for the hands function to work. MediaPipe provides these parameters in the hands function.

Afterwards, we provide all the initializations that we need for our class. These are the parameters above and the MediaPipe initializations.

We put self before every object to allow access to the methods and the attributes of that object. This in turn allows each object to possess its own attributes and methods.

Step 3 - Creating a method that will track the hands in our input image

```
def handsFinder(self,image,draw=True):
        imageRGB = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imageRGB)

        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:

                if draw:
                    self.mpDraw.draw_landmarks(image, handLms,
self.mpHands.HAND_CONNECTIONS)
        return image
```

In the code above, we created a method that we will use to specifically track the hands in our input image. The code that goes in this method is the one that converts the image to RGB and processes the RGB image to locate the hands.

It also draws the hand landmarks on the image and finally draws the hand connections.

Step 4 - Creating a method to find the 'x' and 'y' Co-ordinates of each hand point

```
def positionFinder(self,image, handNo=0, draw=True):
        lmlist = []
        if self.results.multi_hand_landmarks:
            Hand = self.results.multi_hand_landmarks[handNo]
            for id, lm in enumerate(Hand.landmark):
                h,w,c = image.shape
                cx,cy = int(lm.x*w), int(lm.y*h)
                lmlist.append([id,cx,cy])
            if draw:
                cv2.circle(image,(cx,cy), 15 , (255,0,255),
cv2.FILLED)

        return lmlist
```

In the code above, we created a method that we will use to find the x and y coordinates of each of the 21 hand points. We also created a list that we will use to store the values of these coordinates.

The code that goes in this method is the one we use to find the id and hand landmark information of each hand point. We also put in the code that we will use to circle the hand-point that we want to use.

Step 5 - Creating the main method

The code below represents the dummy code that we will use to showcase what the module can do. In our case, it can identify and track hands. It uses the image and lmlist object. This code appears in the main method.

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

    while True:
        success,image = cap.read()
        image = tracker.handsFinder(image)
        lmList = tracker.positionFinder(image)
        if len(lmList) != 0:
            print(lmList[4])

        cv2.imshow("Video",image)
        cv2.waitKey(1)
```

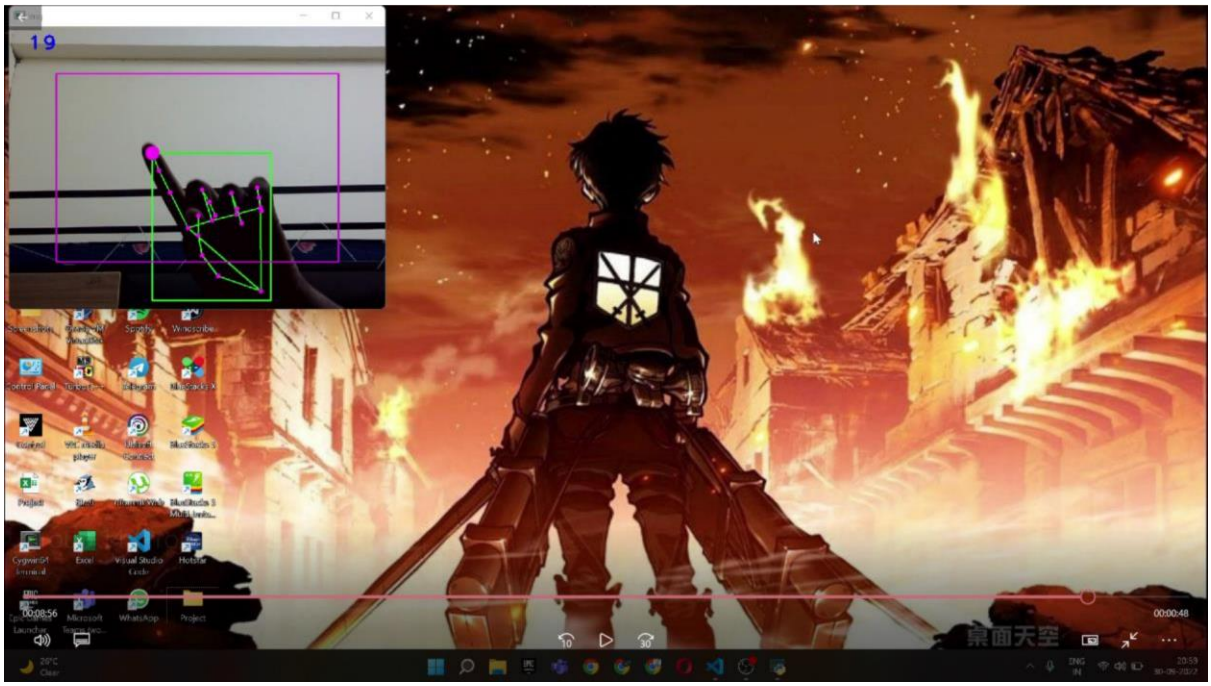
Step 6 - Executing the main method

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

This code implies that, if we are running the module script, then execute the main method.

4.5 Results

The output of the program and the module will be identical. When each of them has run into completion without any errors, the output will be as shown below:



4.6 Conclusion

You now understand and possess all the skills needed to create a program that performs hand tracking. You also have the skills required to convert the code into a module.

Go ahead and import the module into any Python project that requires hand tracking and watch the module perform its magic.

TECHNICAL IMPLEMENTATION & ANALYSIS

5.1 Code

pyHandTrackingModule

```
import cv2
import mediapipe as mp
import time
import math

class handDetector():
    def __init__(self, mode=False, maxHands=2, detectionCon=0.5,
trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.trackCon = trackCon

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands,
                                        self.detectionCon, self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils
        self.tipIds = [4, 8, 12, 16, 20]

    def findHands(self, img, draw=True):
        imgRGB = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
        #print(results.multi_hand_landmarks)

        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms,
                                               self.mpHands.HAND_CONNECTIONS)

        return img

    def findPosition(self, img, handNo=0, draw= True ):
        xList = []
        yList = []
        bbox = []
        self.lmList = []
        if self.results.multi_hand_landmarks:
            myHand = self.results.multi_hand_landmarks[handNo]
            for id, lm in enumerate(myHand.landmark):
                # print(id,lm)
```

```
                h, w, c = img.shape
                cx, cy = int(lm.x*w), int(lm.y*h)
                xList.append(cx)
                yList.append(cy)
                # print(id, cx, cy)
                self.lmList.append([id, cx, cy])
                if draw:
                    cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)
            xmin, xmax = min(xList), max(xList)
            ymin, ymax = min(yList), max(yList)
            bbox = xmin, ymin, xmax, ymax

            if draw:
                cv2.rectangle(img, (bbox[0],bbox[1]),
                              (bbox[2]+20, bbox[3]+20), (0, 255, 0), 2)

        return self.lmList, bbox

    def findDistance(self, p1, p2, img, draw=True):

        x1, y1 = self.lmList[p1][1], self.lmList[p1][2]
        x2, y2 = self.lmList[p2][1], self.lmList[p2][2]
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

        if draw:
            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)

        length = math.hypot(x2 - x1, y2 - y1)
        return length,img, [x1, y1, x2, y2, cx, cy]

    def fingersUp(self):
        fingers = []

        # Thumb
        if self.lmList[self.tipIds[0]][1] > self.lmList[self.tipIds[0] -
1][1]:
            fingers.append(1)
        else:
            fingers.append(0)

        # 4 fingers
        for id in range(1, 5):
            if self.lmList[self.tipIds[id]][2] < self.lmList[self.tipIds[id] -
2][2]:
                fingers.append(1)
```

```
            else:
                fingers.append(0)
        return fingers

def main():
    pTime = 0
    cTime = 0
    cap = cv2.VideoCapture(0)# (0) in VideoCapture is used to connect to your
computer's default camera
    detector = handDetector()
    while True:
        success, img = cap.read()
        img = detector.findHands(img)
        lmList = detector.findPosition(img)
        if len(lmList) !=0:
            print(lmList[4])

        cTime = time.time()
        fps = 1 / (cTime - pTime)
        pTime = cTime

        cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3,
                    (255, 0, 255), 3)

        cv2.imshow("Image", img)
        cv2.waitKey(1)


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

aimouse

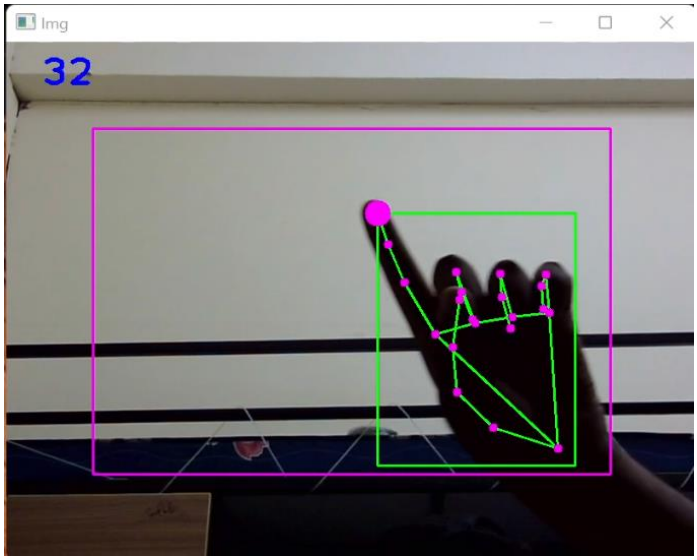
```
import cv2
import autopy
import time
import numpy as np
import pyHandTrackingModule as htm

##############################
wCam, hCam = 800, 600
frameR = 100 # Frame Reduction
smoothening = 7
##############################

pTime = 0
plocX, plocY = 0, 0
clocX, clocY = 0, 0
```

```
cap = cv2.VideoCapture(0)# (0) in VideoCapture is used to connect to your
computer's default camera
cap.set(3, wCam)
cap.set(4, hCam)
detector = htm.handDetector(maxHands=1)
wScr, hScr = autopy.screen.size()
#print(wScr, hScr)

while True:
    success, img = cap.read()
    img = detector.findHands(img)
    lmList, bbox = detector.findPosition(img)

    if len(lmList)!=0:
        x1, y1 = lmList[8][1:]
        x2, y2 = lmList[12][1:]

        #print(x1, y1, x2, y2)

        fingers = detector.fingersUp()
        #print(fingers)
        cv2.rectangle(img, (frameR, frameR), (wCam - frameR, hCam - frameR),
                      (255, 0, 255), 2)
        # 4. Only index Finger :Moving Mode
        if fingers[1] == 1 and fingers[2] == 0:
            # 5. convert coordinates

            x3 = np.interp(x1, (frameR, wCam-frameR), (0, wScr))
            y3 = np.interp(y1, (frameR, hCam-frameR), (0, hScr))
            # 6. Smoothen Values
            clocX = plocX +(x3 -plocX) /smoothening
            clocY = plocY + (y3 - plocY) / smoothening

            # 7. Move Mouse
            autopy.mouse.move(wScr-clocX, clocY)
            cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
            plocX, plocY = clocX, clocY

        # 8. Both Index and Middle Finger are up : Clicking Mode
        if fingers[1] == 1 and fingers[2] == 1:
            length, img, lineInfo = detector.findDistance(8, 12, img)
            print(length)
            if length < 39:
                cv2.circle(img, (lineInfo[4], lineInfo[5]),
                           15, (0, 255, 0), cv2.FILLED)
                autopy.mouse.click()
        # 9. Find Distance between Fingers
```

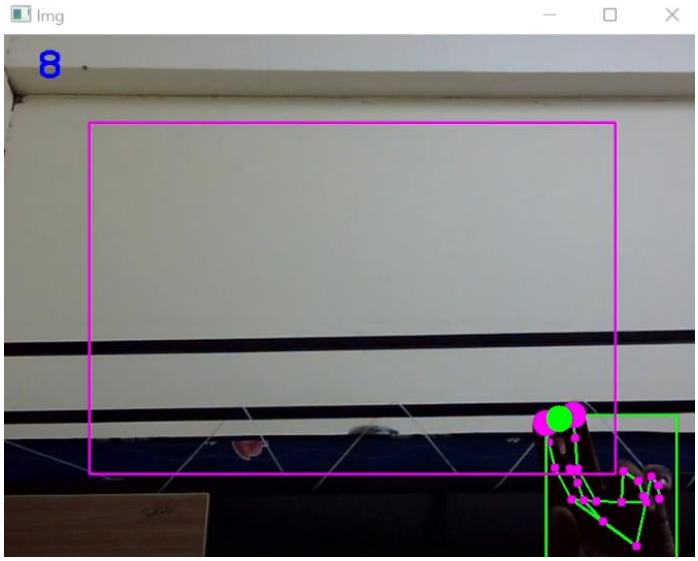
```
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    cv2.putText(img, str(int(fps)), (40, 50), cv2.FONT_HERSHEY_PLAIN, 3,
                (255, 0, 0), 3)

    cv2.imshow("Img", img)
    cv2.waitKey(1)
```

5.2 Operations

To move cursor use only index finger of any hand



To left click use both index and middle finger



CONCLUSIONS AND RECOMMENDATION

6.1 Future Enhancements

We are going to add some additional functionalities to enhance the virtual mouse-

- **Right Click-** There are many functionalities that require right click. So, there should be a hand gesture to add right click operation.
- **Scroll-** Scrolling is an important part of interaction provided by a mouse. Without it we will losing a very essential functionality of a mouse.
- **Zoom-** Add zooming operation will further enhance the system and make it easier to zoom in and out.
- **More Devices-** We can program to make our code compatible with more than one type device.
- **Flexible-** We can also give an option to the user to add their custom functionality which they want also to change which hand gesture controls which operation.

6.2 Inference

This paper presented a brandnew AI virtual mouse method using OpenCv, autopy and mediapipe by the help of fingertip movement interacted with the computer in front of camera without using any physical device. The approach demonstrated high accuracy and highly accurate gesture eliminates in practical applications. The proposed method overcomes the limitations of already existing virtual mouse systems. It has many advantages, e.g., working well in low light as well as changing light condition with complex background, and fingertip tracking of different shape and size of finger with same accuracy. The experimental results shows that the approaching system can perform very well in realtime applications. We also intend to add new gesture on it to handle the system more easily and interact with other smart systems. It is possible to enrich the tracking system by using machine learning algorithm like Open pose. It is also possible to including body, hand and facial key points for different gestures.

REFERENCES

- https://ijisrt.com/assets/upload/files/IJISRT21NOV703.pdf
- https://www.youtube.com/watch?v=kYxpQvqePyQ&t=8s
- https://www.youtube.com/watch?v=8tng9RsbXoU&t=59s
- https://www.youtube.com/watch?v=xb5Xe8eE_Jw&t=1s
- https://www.researchgate.net/publication/324206388_Human_hand_gesture_based_system_for_mouse_cursor_control