AIR MOUSE USING OPENCV

INTRODUCTION:

Employing a hand as a virtual mouse can perform everything that a mouse does without barely involving your system. Users can use the webcam of their system to detect hands. It will subsequently produce a frame around the hand and focus on two fingers: The forefinger and the middle finger. The forefinger will act as a cursor and moving it around, we will be moving the cursor around. Now, to successfully click using hand tracking, it is detecting the distance between the forefinger and the middle finger. If they are joined together, then it will perform a click.

PURPOSE:

The main purpose of an AI virtual mouse system is to develop an alternative to the regular and traditional mouse system to perform and control the mouse functions, and this can be achieved with the help of a web camera that captures the hand gestures and hand tip.

PREREQUISITES:

To complete this project you must have the following software versions and packages.

- PyCharm (Download: https://www.jetbrains.com/pycharm/)
- Python 3.8.0 (Download: https://www.python.org/downloads/release/python-380/)
- AutoPy (https://pypi.org/project/autopy/)
- Mediapipe 0.8.3.1 (https://pypi.org/project/mediapipe/0.8.3.1/)
- Time, Math and Numpy.

The system should have a webcam or external camera. In the latest version of python autopy and mediapipe may not support it. So install the versions mentioned above.

PROJECT OBJECTIVE:

By the end of this project you will:

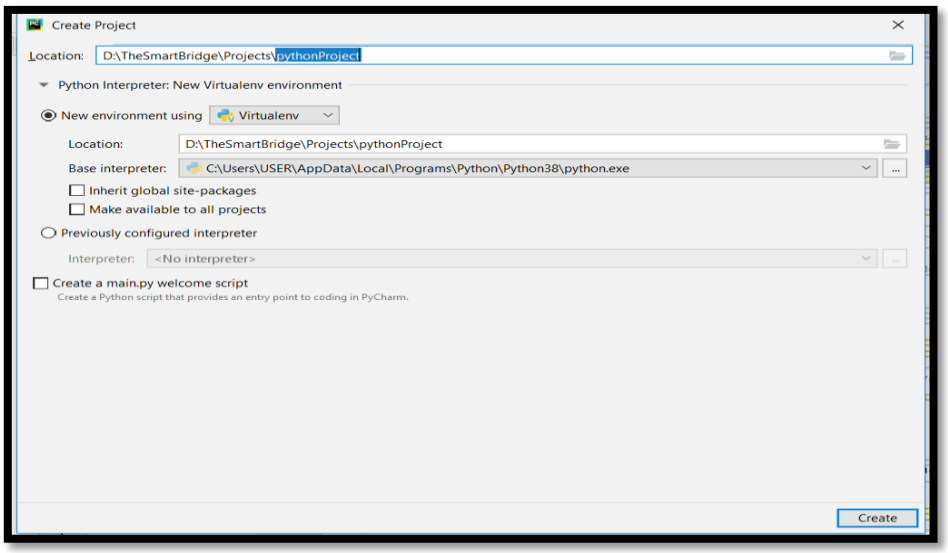
- Know fundamental concepts and techniques of computer vision(openCV).
- Gain a broad understanding of mediapipe package.

PROJECT STRUCTURE:

Milestone – 1: Creating environment for virtual mouse project

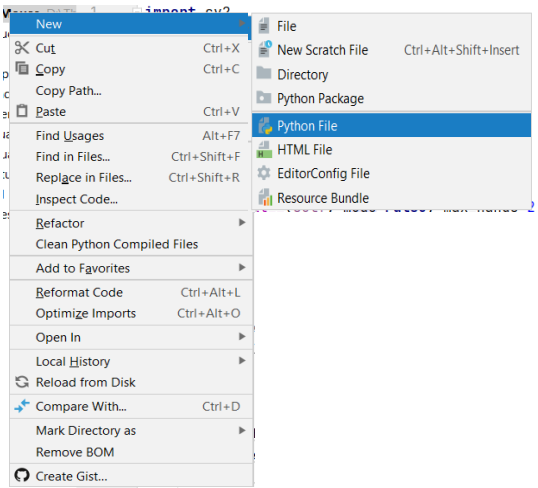
➢ Activity – 1: Creating a project

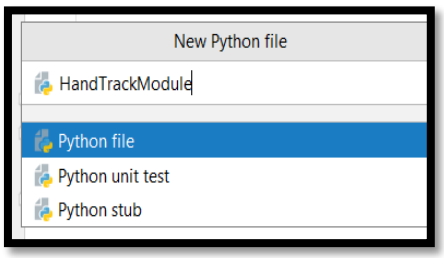
Click the PyCharm community edition desktop app from start menu. Click the new project tab and give the name of your project. Make sure the base interpreter dropdown should be python38. Click on create button.



Milestone – 2: Creating HandTrackModule.py file

Right click on your project title and click new->python file. And now give the name for the python file as HandTrackModule.





➢ Activity 1: Installing packages

Go to file -> settings -> python interpreter. Add the packages to be installed (opencv-python, autopy). Then, go to terminal and type (pip install mediapipe==0.8.3.1).

➢ Activity 2: Importing the required packages

Import the required packages.

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

➢ Activity 3: Create a class called HandDetector

All classes have a function called __init__(), which is always executed when the class is being initiated. Use the __init__() function to assign values to object properties. Now create a separate variable for mp.solutions.hands and mp.solutions.drawing_utils. To visualize the hand landmarks we use mp.solutions.hands.Hands() and mp.solutions.drawing_utils is used to join those landmarks. If you have any doubt on values assigned in __init__ function hold ctrl and click Hands() object you can see the default params. Finger tip landmarks are tipIds.

```
class HandDetector():   # Initializing
    def __init__(self, mode=False, max_hands=2, min_detection_confidence=0.5, min_tracking_confidence=0.5):
        self.mode = mode
        self.max_hands = max_hands
        self.min_detection_confidence = min_detection_confidence
        self.min_tracking_confidence = min_tracking_confidence
        self.mp_hand = mp.solutions.hands   # mandatory for using this module.
        self.hands = self.mp_hand.Hands(self.mode, self.max_hands,
                                        self.min_detection_confidence, self.min_tracking_confidence)
        # creating a hands variable with default Hands() object parameters.
        # To view parameter press ctrl + object.
        # This hands object reads only RGB coloured images.
        self.mp_draw = mp.solutions.drawing_utils
        self.tipIds = [4, 8, 12, 16, 20]
```

➢ Activity 4: Create a function FindHands

Normally, the image read by cv2 will be in BGR (Blue, Green, Red). But for using Hands() object the input should be in RGB (Red, Green, Blue). So converting the image and storing in new variable and processing it. By printing the result.multi_hand_landmarks, you can visualize the x,y & z values of landmarks. By using the loop concept if the hand is present in webcam it will draw lines to join the doted points (landmarks). RGB image is only used in result variable for processing purpose other than that we use normal image only.

```
def FindHands(self, image):
    image_RGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # converting BGR to RGB.
    self.result = self.hands.process(image_RGB)  # hands object processing the image.
    # print(result.multi_hand_landmarks)
    if self.result.multi_hand_landmarks:
        for hand in self.result.multi_hand_landmarks:
            self.mp_draw.draw_landmarks(image, hand, self.mp_hand.HAND_CONNECTIONS)
    return image
```

➢ Activity 5: Creating FindPosition function

To take landmarks of the current hand, hand_no=0 is used. Creating an empty list to store the index and position of landmarks. As we discused before, result.multi_hand_landmarks prints the decimal value of x, y & z. But we want the values to be in pixels. So, multiply the image width and height with x and y values and save it with new variable cx, cy. Now append the id, cx & cy to the empty list. This function is used to find the position of hand landmarks.

```
def FindPosition(self,image,hand_no=0):
    self.lmlist=[]
    if self.result.multi_hand_landmarks:
        one_hand = self.result.multi_hand_landmarks[hand_no]
        for id, landmarks in enumerate(one_hand.landmark):
            #print(id, landmarks)
            height, width, channel = image.shape
            cx, cy = int(landmarks.x * width), int(landmarks.y * height)
            self.lmlist.append([id,cx,cy])
            cv2.circle(image,(cx,cy), 4, (0,0,255), cv2.FILLED)
    return self.lmlist
        #print(id, cx, cy)
```

➢ Activity 6: Create FingersUp function

If the fingers are closed infront of the webcam, it prints 0 and if the fingers are opened it prints 1 for respective fingers. This function is used for moving mode on mouse setting.

```
def FingersUp(self):
    fingers = []
    if self.lmlist[self.tipIds[0]][1] > self.lmlist[self.tipIds[0] -1][1]:
        fingers.append(1)
    else:
        fingers.append(0)
    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
```

➢ Activity 7: Create FindDistance function

This function is used to find the distance between two fingers. x1, y1 & x2, y2 are the finger tip coordinates. To find the euclidean math.hypot() is used.

```
def FindDistance(self,p1,p2,image,draw=True,r=10,t=3):
    x1,y1 = self.lmlist[p1][1:]
    x2,y2 = self.lmlist[p2][1:]
    cx,cy = (x1+x2) // 2 ,(y1+y2) //2

    if draw:
        cv2.line(image, (x1,y1), (x2,y2), (255,0,0), t)
        cv2.circle(image, (x1,y1), r, (0,0,255), cv2.FILLED)
        cv2.circle(image, (x2,y2), r, (0,0,255), cv2.FILLED)
        cv2.circle(image, (cx,cy), r, (255,0,0), cv2.FILLED)
    length = math.hypot(x2-x1, y2-y1)
    return length, image, [x1,y1,x2,y2,cx,cy]
```

Milestone - 3: Creating virtual_mouse.py file

Right click on your project title and click new->python file. And now give the name for the python file as virtual_mouse.

➢ Activity 1: Import the required packages

Set the webcam frame (width & height) and call the object by creating a new variable to it (detector). VideoCapture(0) is used to open webcam. If you have single cam use 0 or if you have external camera use its respective no. as(1,2,…).

```
import cv2
import numpy as np
import mediapipe as mp
import autopy
import time
import HandTrackModule as htm

cam_width, cam_height = 640, 480
screen_width, screen_height = autopy.screen.size()
fr = 100 # Frame reduction
smooth = 7
plocx, plocy = 0, 0 # previous location
clocx, clocy = 0, 0 # current location

previous_time = 0
detector = htm.HandDetector(max_hands=1)

capture = cv2.VideoCapture(0)
capture.set(3, cam_width) # Set the cam frame width
capture.set(4, cam_height) # set the cam frame height
```

- **Activity 2: Finding the hand landmarks**

FindHands & FindPosition functions are already created. For your reference kindly check above. Pass the image to the functions.

```
# Step-1: Finding the hand landmarks
image = detector.FindHands(image)
lmlist = detector.FindPosition(image) # lmlist is list of landmarks
```

- **Activity 3: Getting the tip of the index and middle finger**

The landmark point for index finger tip is 8 and the landmark point for the middle finger tip is 12.

```
# Step-2: Getting the tip of the index and middle finger
if len(lmlist) != 0:
    x1,y1 = lmlist[8][1:]
    x2,y2 = lmlist[12][1:]
```

- **Activity 4: Finding the fingers which are opened**

Create a variable fingers to call the FingersUp function. A rectangle is drawn inside the 640 x 480 frame. Now this rectangle act as a frame.

```
# Step-3: Find the fingers which are opened
fingers = detector.FingersUp()
#cv2.rectangle(image, (100, 100), (cam_width - 100, screen_height - 100),
              #(255, 0, 255), 2)
#print(fingers)
cv2.rectangle(image, (fr, fr), (cam_width - fr, cam_height - fr), (255, 0, 255), 2)
```

- **Activity 5: Moving mode**

If the index finger is opened then it act as a moving mode operation.

- **Activity 6: Convert coordinates**

np.interp() is used to convert the coordinates.

➢ Activity 7: Smoothening the values

Virtual mouse can be done without smoothening. But the shake on mouse pointer will be high. To overcome this issue, values are smoothened.

➢ Activity 8: Moving mouse pointer

By using autopy module we can access the mouse. If x3 and y3 is passed on autopy.mouse.move() then, if you move your index finger to right side it moves left. To overcome this issue we have to flip the x coordinate by screen_width – x3. If you want the smooth moving mouse pointer, replace x3 & y3 with clocx, clocy.

```
# Step-4: Moving mode [if index finger is up then it is moving
if fingers[1] == 1 and fingers[2] == 0:

    # Step-5: Convert coordinates
    x3 = np.interp(x1, (fr,cam_width-fr), (0,screen_width))
    y3 = np.interp(y1, (fr,cam_height-fr), (0,screen_height))

    # Step-6: Smoothening values
    clocx = plocx + (x3 - plocx) / smooth
    clocy = plocy + (y3 - plocy) / smooth

    # Step-7: Moving mouse
    autopy.mouse.move(screen_width - clocx, clocy)
    cv2.circle(image, (x1,y1), 10, (255,0,255),cv2.FILLED)
    plocx, plocy = clocx, clocy
```

➢ Activity 9: Clicking mode

If both index finger and middle finger are joined it should be in clicking mode. On FindDistance function 8 (index finger tip), 12 (middle finger tip) are passed. If the distance between two finger is less than 40 it will be in clicking mode.

```
# Step-8: Clicking mode: Both index and middle fingers are up
if fingers[1] == 1 and fingers[2] == 1:
    # Step-9: Finding the distance between fingers
    length,image,info = detector.FindDistance(8,12,image)
    #print(length)
    # Step-10: Virtual clicking
    if length < 40:
        autopy.mouse.click()
```

- **Activity 10: Visualizing FPS on the frame**

By using time.time() we can get the current time. FPS formula is given below. To visualize the FPS on frame putText function is used. To display the frame imshow() is used.

```
# Step-11: Visualizing Frame rate
current_time = time.time()
fps = 1 / (current_time - previous_time)
previous_time = current_time
cv2.putText(image, str(int(fps)), (20,50), cv2.FONT_HERSHEY_PLAIN, 3, (255,0,255), 3)

# Step-12: Display
cv2.imshow('img', image)
cv2.waitKey(1)
```

Milestone 4 – Running and terminating project

- **Activity 1: Run the virtual_mouse.py file**

Now everything is done. Press run button or press ctrl + shift + f10.

- **Activity 2: Terminate virtual_mouse.py file**

Then press stop for terminate the webcam.

Final output:

OUTPUT: