AI Virtual Mouse

VIDEO :- https://www.youtube.com/watch?v=NZde8Xt78Iw
CODE:
WEBCAM ACCESS:-

```
import cv2
import mediapipe as mp
import time

cap = cv2.VideoCapture(0)
while True:
   success, img = cap.read()

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

EXPLANATION:-

The first line initializes the VideoCapture object by passing in the camera index. In this case, 0 is passed as the camera index, which is the default camera on most systems.
The while loop runs continuously, and in each iteration, it reads the next frame from the camera using the read() method of the VideoCapture object. This method returns two values: a boolean success that indicates whether the frame was successfully captured, and the img object which represents the captured frame.
The imshow() method is used to display the captured image in a window titled "Image". The waitKey() method waits for a key event for a specified amount of time (in milliseconds). In this case, it waits for 1 millisecond. The loop then continues to the next iteration and reads the next frame. The loop runs indefinitely until the user closes the window or the program is terminated manually.

CREATING OBJECT :-

CODE:-

```
mpHands = mp.solutions.hands
hands = mpHands.Hands()
```

EXPLANATION:-

The code mpHands = mp.solutions.hands imports the hands module from the mediapipe library, which is a popular computer vision library used for various tasks such as object detection, facial recognition, and hand tracking.
The code hands = mpHands.Hands() creates an instance of the Hands class from the mpHands module. The Hands class is a pre-trained machine learning model for detecting and tracking hand landmarks in images and videos. The model uses a set of 21 key points to represent the hand and fingers, which can be used for various applications such as gesture recognition and virtual reality hand tracking.
By creating an instance of the Hands class, we can use the pre-trained hand detection model to process image frames and obtain the landmarks of hands present in the image.

CONVERTING IMAGE TO RGB:-

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

EXPLANATION:-

These two lines of code perform the following operations:

cv2.cvtColor(img,cv2.COLOR_BGR2RGB): The cv2.cvtColor() function from OpenCV is used to convert the image img from the BGR color space to the RGB color space. The reason for doing this is that the mpHands module expects input images in RGB format.
hands.process(imgRGB): The process() method of the Hands object created using mpHands.Hands() is used to process the image imgRGB. This method takes the RGB image as input and returns a HandLandmark object, which contains the detected hand landmarks such as the positions of the fingertips, palm, etc.

HAND IS DETECTED OR NOT:-

```
print(results.multi_hand_landmarks)
```

EXPLANATION:-

To know weather a hand is detected or not.

MULTIPLE HAND PRESENT OR NOT:-

```
mpDraw = mp.solutions.drawing_utils
```
Before while loop then,

```
#print(results.multi_hand_landmarks)
if results.multi_hand_landmarks:
   for handLms in results.multi_hand_landmarks:
       mpDraw.draw_landmarks(img,handLms)
```

EXPLANATION :-

This code snippet checks if there are any hands detected in the image frame by looking for the presence of multi_hand_landmarks in the results object returned by the hands.process() method.
If one or more hands are detected, it then loops through all the hand landmarks using a for loop, and uses the mpDraw.draw_landmarks() method to draw the landmarks on the original image.
This helps in visualizing the hand landmarks on the image and allows us to track and analyze hand movements.

FOR CONNECTION:-

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

FRAME RATE ON SCREEN:-

```
pTime = 0
cTime = 0

#Before While loop

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)
```

EXPLANATION:-

In this part of the code, we are calculating and displaying the Frames Per Second (FPS) on the image.
cTime is the current time at which this block of code is executed.
fps is calculated as the reciprocal of the time difference between the current time (cTime) and the previous time (pTime).
pTime is then set to cTime, as the current time becomes the previous time for the next iteration.
Then, we use the cv2.putText() function to display the FPS value on the image. The parameters for cv2.putText() function are:
img is the image on which we want to display the FPS value.
str(int(fps)) is the text to be displayed, which is the FPS value converted to an integer and then to a string.
(10,70) is the position where we want to display the text on the image, in pixel coordinates.
cv2.FONT_HERSHEY_PLAIN specifies the font style to be used for the text.
3 is the font size.
(255,0,255) is the color of the text, which is in BGR format.
3 is the thickness of the text.

GETTING THE ID & LANDMARK:-

After for handlms…..

```
for id , lm in enumerate(handLms.landmark):
   print(id,lm)
```

EXPLANATION:-

This will give us the information about the landmark and the id.

FINDING WIDTH, HEIGHT, CHANNEL AND THE PIXELS:-

```
#print(id,lm)
h, w, c = img.shape
cx , cy = int(lm.x*w), int(lm.y*h)
print(id , cx ,cy)
```

EXPLANATION:-

This will give us the id with respect to every value of x and y co-ordinates .

GETTING VALUE :-

```
#if id == 4: --------For any particular id if we want
cv2.circle(img, (cx,cy), 12 , (255,0,255), cv2.FILLED)
```

WE WILL CREATE A MODULE FOR THAT AT END :-

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

WE WILL DEFINE def main():-

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

```
  while True:
    success, img = cap.read()
    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)
```

CREATE A CLASS AT TOP:-

```
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
```

DEFINING ANOTHER CLASS OF FINDHANDS:-

```
import cv2
import mediapipe as mp
import time

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

  def findHands(self,img, draw=True):
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = hands.process(imgRGB)
    # print(results.multi_hand_landmarks)
    if results.multi_hand_landmarks:
      for handLms in results.multi_hand_landmarks:
        if draw:
          self.mpDraw.draw_landmarks(img, handLms, self.mpHands.HAND_CONNECTIONS)

    return img



        #for id, lm in enumerate(handLms.landmark):
          # print(id,lm)
```

```
                #h, w, c = img.shape
                #cx, cy = int(lm.x * w), int(lm.y * h)
                #print(id, cx, cy)
                # if id == 4:
                #cv2.circle(img, (cx, cy), 12, (255, 0, 255), cv2.FILLED)


def main():
  pTime = 0
  cTime = 0
  cap = cv2.VideoCapture(0)
  detector = handDetector()


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

     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()
```

OUR CODE LOOKS LIKE THIS NOW: