import cv2  
import mediapipe  
import numpy  
import autopy  
import time  
cap = cv2.VideoCapture(0)  
initHand = mediapipe.solutions.hands # Initializing mediapipe  
# Object of mediapipe with "arguments for the hands module"  
mainHand = initHand.Hands(min\_detection\_confidence=0.9, min\_tracking\_confidence=0.9)  
draw = mediapipe.solutions.drawing\_utils # Object to draw the connections between each finger index  
wScr, hScr = autopy.screen.size() # Outputs the high and width of the screen (1920 x 1080)  
pX, pY = 0, 0 # Previous x and y location  
cX, cY = 0, 0 # Current x and y location  
ptime=0  
  
  
def handLandmarks(colorImg):  
 landmarkList = [] # Default values if no landmarks are tracked  
  
 landmarkPositions = mainHand.process(colorImg) # Object for processing the video input  
 landmarkCheck = landmarkPositions.multi\_hand\_landmarks # Stores the out of the processing object (returns False on empty)  
 if landmarkCheck: # Checks if landmarks are tracked  
 for hand in landmarkCheck: # Landmarks for each hand  
 for index, landmark in enumerate(  
 hand.landmark): # Loops through the 21 indexes and outputs their landmark coordinates (x, y, & z)  
 draw.draw\_landmarks(img, hand,  
 initHand.HAND\_CONNECTIONS) # Draws each individual index on the hand with connections  
 h, w, c = img.shape # Height, width and channel on the image  
 centerX, centerY = int(landmark.x \* w), int(  
 landmark.y \* h) # Converts the decimal coordinates relative to the image for each index  
 landmarkList.append([index, centerX, centerY]) # Adding index and its coordinates to a list  
  
 return landmarkList  
  
  
def fingers(landmarks):  
 fingerTips = [] # To store 4 sets of 1s or 0s  
 tipIds = [4, 8, 12, 16, 20] # Indexes for the tips of each finger  
  
 # Check if thumb is up  
 if landmarks[tipIds[0]][1] > lmList[tipIds[0] - 1][1]:  
 fingerTips.append(1)  
 else:  
 fingerTips.append(0)  
  
 # Check if fingers are up except the thumb  
 for id in range(1, 5):  
 if landmarks[tipIds[id]][2] < landmarks[tipIds[id] - 3][  
 2]: # Checks to see if the tip of the finger is higher than the joint  
 fingerTips.append(1)  
 else:  
 fingerTips.append(0)  
  
 return fingerTips  
  
  
while True:  
 check, img = cap.read() # Reads frames from the camera  
 imgRGB = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # Changes the format of the frames from BGR to RGB  
 lmList = handLandmarks(imgRGB)  
 # cv2.rectangle(img, (75, 75), (640 - 75, 480 - 75), (255, 0, 255), 2)  
  
 if len(lmList) != 0:  
 x1, y1 = lmList[8][1:] # Gets index 8s x and y values (skips index value because it starts from 1)  
 x2, y2 = lmList[12][1:] # Gets index 12s x and y values (skips index value because it starts from 1)  
 finger = fingers(lmList) # Calling the fingers function to check which fingers are up  
  
 if finger[1] == 1 and finger[0] == 0: # Checks to see if the pointing finger is up and thumb finger is down  
 x3 = numpy.interp(x1, (75, 640 - 75),  
 (0, wScr)) # Converts the width of the window relative to the screen width  
 y3 = numpy.interp(y1, (75, 480 - 75),  
 (0, hScr)) # Converts the height of the window relative to the screen height  
  
 cX = pX + (x3 - pX) / 7 # Stores previous x locations to update current x location  
 cY = pY + (y3 - pY) / 7 # Stores previous y locations to update current y location  
  
 autopy.mouse.move(wScr - cX,  
 cY) # Function to move the mouse to the x3 and y3 values (wSrc inverts the direction)  
 pX, pY = cX, cY # Stores the current x and y location as previous x and y location for next loop  
  
  
  
  
 if finger[1] == 1 and finger[4] == 1: # Checks to see if the pointer finger is down and thumb finger is up  
 autopy.mouse.click() # Left click  
  
 if finger[1]==1 and finger[2]==1:  
 autopy.mouse.click() #double click  
 autopy.mouse.click()  
  
  
 ctime = time.time()  
 fps = 1 / (ctime - ptime)  
 ptime = ctime  
 cv2.putText(img, str(int(fps)), (20, 50), cv2.FONT\_HERSHEY\_PLAIN, 3,  
 (0, 255, 250), 3)  
  
  
 cv2.imshow("Webcam", img)  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break