AIR MOUSE:CONTROLLING MOUSE USING GESTURES

INTRODUCTION :

The emergence of gesture-based interaction has revolutionized the way humans interact with technology, offering a natural and intuitive approach to control devices and interfaces. Among various gesture recognition systems, hand gesture control stands out as a promising technology that allows users to interact with digital interfaces using hand gestures in the air. By leveraging computer vision and machine learning techniques, hand gesture control systems can accurately detect and interpret hand movements captured by a webcam or other sensors, enabling seamless navigation and interaction without physical touch or input devices.

Installation:

It will be best idea to install these tools on virtual environment.

- pip install mediapipe for installing mediapipe.
- pip install mouse for installing mouse package.

PRELIMINARY TASKS :

Import the libraries:

In [1]:
```
import mediapipe as mp
import cv2
import mouse
import numpy as np
import tkinter as tk
```

Get Screen Size:

The use of tkinter is only to find screen size

In [2]:
```
root = tk.Tk()

screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()

ssize = (screen_height, screen_width)
ssize
```

Out[2]: (768, 1366)

Write Basic Functions

- We need to convert the landmark position from the frame world to our screen world thus the method frame_pos2screen_pos is written.

- We will be working with Euclidean Distance to make some sense about gestures.

```
def frame_pos2screen_pos(frame_size=(480, 640), screen_size=(768, 1366), frame_pos=None):

    x,y = screen_size[1]/frame_size[0], screen_size[0]/frame_size[1]

    screen_pos = [frame_pos[0]*x, frame_pos[1]*y]

    return screen_pos

def euclidean(pt1, pt2):
    d = np.sqrt((pt1[0]-pt2[0])**2+(pt1[1]-pt2[1])**2)
    return d
euclidean((4, 3), (0, 0))
```

```
5.0
```

The below code utilizes OpenCV and MediaPipe for hand gesture recognition and control of mouse events. It starts by initializing a webcam capture (cam) and setting up parameters such as frame size, region of interest (ROI), and event detection frequency (check_every). It then creates instances for drawing utilities (mp_drawing) and hand tracking (mp_hands).

Within the main loop, it continuously reads frames from the webcam, processes them to detect hand landmarks using MediaPipe, and maps hand gestures to mouse events. Specifically, it identifies the positions of finger landmarks (e.g., index finger tip, thumb tip) relative to the defined ROI, calculates distances between finger points to determine gestures like single click, double click, right click, and drag, and translates these gestures into corresponding mouse actions using the mouse object. The script also handles event timing and updates the last_event variable accordingly to maintain gesture continuity.

```
cam = cv2.VideoCapture(0)
fsize = (520, 720)

left,top,right,bottom=200, 100, 500, 300

mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands


check_every = 10
check_cnt = 0

last_event = None
events = ["sclick", "dclick", "rclick", "drag"]

with mp_hands.Hands(
static_image_mode=True,
max_num_hands = 1,
min_detection_confidence=0.7) as hands:
    while cam.isOpened():
        ret, frame = cam.read()
        if not ret:
            continue
        frame = cv2.flip(frame, 1)
        frame = cv2.resize(frame, (fsize[1], fsize[0]))

        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 1)

        h, w,_ = frame.shape

        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        rgb.flags.writeable = False

        res = hands.process(rgb)
        #cv2.imshow("roi", roi)
        rgb.flags.writeable = True


        if res.multi_hand_landmarks:
            for hand_landmarks in res.multi_hand_landmarks:

                index_dip = mp_drawing._normalized_to_pixel_coordinates(
```

```
if res.multi_hand_landmarks:
    for hand_landmarks in res.multi_hand_landmarks:

        index_dip = mp_drawing._normalized_to_pixel_coordinates(
            hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_DIP].x,
            hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_DIP].y,
            w, h)

        index_tip = mp_drawing._normalized_to_pixel_coordinates(
            hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].x,
            hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].y,
            w, h)

        index_pip = np.array(mp_drawing._normalized_to_pixel_coordinates(
            hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_PIP].x,
            hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_PIP].y,
            w, h))

        thumb_tip = mp_drawing._normalized_to_pixel_coordinates(
            hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].x,
            hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].y,
            w, h)

        middle_tip = mp_drawing._normalized_to_pixel_coordinates(
            hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP].x,
            hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP].y,
            w, h)

        if index_pip is not None:
            if check_cnt==check_every:
                if thumb_tip is not None and index_tip is not None and middle_tip is not None:

                    #print(euclidean(index_tip, middle_tip))
                    if euclidean(index_tip, middle_tip)<60: # 60 should be relative to the height of frame
                        last_event = "dclick"
                    else:
                        if last_event=="dclick":
                            last_event=None
```

```
        if thumb_tip is not None and index_tip is not None:
            #print(euclidean(thumb_tip, index_pip))
            if euclidean(thumb_tip, index_pip) < 60: # 60 should be relative to height/width of frame
                last_event = "click"
            else:
                if last_event=="click":
                    last_event=None

            if euclidean(thumb_tip, index_tip) < 40:
                last_event="press"
            else:
                if last_event == "press":
                    last_event = "release"

        if thumb_tip is not None and index_tip is not None and middle_tip is not None:

            #print(euclidean(index_tip, middle_tip))
            if euclidean(thumb_tip, middle_tip)<60: # 60 should be relative to the height of frame
                last_event = "rclick"
            else:
                if last_event=="rclick":
                    last_event=None



        check_cnt=0

    #print(index_pip)
    index_pip[0] = np.clip(index_pip[0], left, right)
    index_pip[1] = np.clip(index_pip[1], top, bottom)

    # normalize the pip values
    index_pip[0] = (index_pip[0]-left)*fsize[0]/(right-left)
    index_pip[1] = (index_pip[1]-top)*fsize[1]/(bottom-top)


    screen_pos = frame_pos2screen_pos(fsize, wsize, index_pip)

    mouse.move(str(int(screen_pos[0])), str(int(screen_pos[1])))

    if check_cnt==0:
        if last_event=="click":
            mouse.click()
        elif last_event=="dclick":
            mouse.double_click()
        elif last_event=="press":
            mouse.press()
        elif last_event=="rclick":
            mouse.right_click()
        else:
            mouse.release()
        #print(last_event)
```

```
                check_cnt+=1

#                 cv2.putText(frame, last_event, (10, 50),
#                                         cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
                mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)
        cv2.imshow("Controller Window", frame)

        if cv2.waitKey(1)&0xFF == 27:
            break
cam.release()
cv2.destroyAllWindows()
```

```
cam = cv2.VideoCapture(0)

fsize = (520, 720)

mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

left, top, right, bottom = (100, 100, 500, 400)

events = ["sclick", "dclick", "rclick", "drag", "release"]

check_every = 15
check_cnt = 0
last_event = None

out = cv2.VideoWriter("out.avi", cv2.VideoWriter_fourcc('M','J','P','G'), 30, (fsize[1], fsize[0]))

with mp_hands.Hands(static_image_mode=True,
                    max_num_hands = 1,
                    min_detection_confidence=0.5) as hands:
    while cam.isOpened():
        ret, frame = cam.read()

        if not ret:
            continue

        frame = cv2.flip(frame, 1)
        frame = cv2.resize(frame, (fsize[1], fsize[0]))

        h, w, _ = frame.shape

        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        res = hands.process(rgb)

        if res.multi_hand_landmarks:
            for hand_landmarks in res.multi_hand_landmarks:
                index_tip = mp_drawing._normalized_to_pixel_coordinates(
                    hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].x,
                    hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].y,
                    w, h)

                index_dip = mp_drawing._normalized_to_pixel_coordinates(
                    hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_DIP].x,
                    hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_DIP].y,
                    w, h)


                index_pip = np.array(mp_drawing._normalized_to_pixel_coordinates(
                    hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_PIP].x,
                    hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_PIP].y,
                    w, h))

                thumb_tip = mp_drawing._normalized_to_pixel_coordinates(
                    hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].x,
```

t

When we run the above snap code, we observe various outputs that demonstrate its functionality

The below snaps are the outputs

(a)

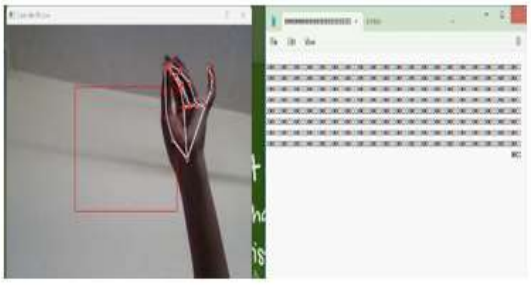
Fig(a) Dragging

(b)

Fig.(b) Single click

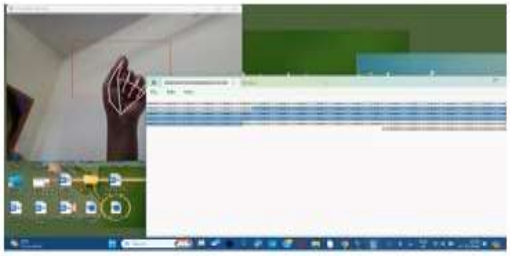
(c)

Fig.(c) Screen adjust



(d)

Fig.(d) press and release



(e)
Fig. (e) Right click

(f)
Fig. (f) double click

CONCLUSION:

It demonstrates the implementation of a hand gesture recognition system using a webcam for mouse control. By leveraging computer vision techniques and libraries such as Media Pipe and Open CV, the code captures real-time video frames from the webcam, detects hand landmarks, and recognizes specific gestures performed by the user. These gestures are then translated into corresponding mouse movements and actions, allowing for intuitive interaction with digital interfaces without the need for physical touch or input devices.