PYTHON CODE FOR AIR MOUSE CONTROLLING

# Imports

import cv2

import mediapipe as mp

import pyautogui

import math

from enum import IntEnum

from ctypes import cast, POINTER

from comtypes import CLSCTX\_ALL

from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

from google.protobuf.json\_format import MessageToDict

import screen\_brightness\_control as sbcontrol

pyautogui.FAILSAFE = False

mp\_drawing = mp.solutions.drawing\_utils

mp\_hands = mp.solutions.hands

# Gesture Encodings

class Gest(IntEnum):

# Binary Encoded

"""

Enum for mapping all hand gesture to binary number.

"""

FIST = 0

PINKY = 1

RING = 2

MID = 4

LAST3 = 7

INDEX = 8

FIRST2 = 12

LAST4 = 15

THUMB = 16

PALM = 31

# Extra Mappings

V\_GEST = 33

TWO\_FINGER\_CLOSED = 34

PINCH\_MAJOR = 35

PINCH\_MINOR = 36

# Multi-handedness Labels

class HLabel(IntEnum):

MINOR = 0

MAJOR = 1

# Convert Mediapipe Landmarks to recognizable Gestures

class HandRecog:

"""

Convert Mediapipe Landmarks to recognizable Gestures.

"""

def \_\_init\_\_(self, hand\_label):

"""

Constructs all the necessary attributes for the HandRecog object.

Parameters

----------

finger : int

Represent gesture corresponding to Enum 'Gest',

stores computed gesture for current frame.

ori\_gesture : int

Represent gesture corresponding to Enum 'Gest',

stores gesture being used.

prev\_gesture : int

Represent gesture corresponding to Enum 'Gest',

stores gesture computed for previous frame.

frame\_count : int

total no. of frames since 'ori\_gesture' is updated.

hand\_result : Object

Landmarks obtained from mediapipe.

hand\_label : int

Represents multi-handedness corresponding to Enum 'HLabel'.

"""

self.finger = 0

self.ori\_gesture = Gest.PALM

self.prev\_gesture = Gest.PALM

self.frame\_count = 0

self.hand\_result = None

self.hand\_label = hand\_label

def update\_hand\_result(self, hand\_result):

self.hand\_result = hand\_result

def get\_signed\_dist(self, point):

"""

returns signed euclidean distance between 'point'.

Parameters

----------

point : list contaning two elements of type list/tuple which represents

landmark point.

Returns

-------

float

"""

sign = -1

if self.hand\_result.landmark[point[0]].y < self.hand\_result.landmark[point[1]].y:

sign = 1

dist = (self.hand\_result.landmark[point[0]].x - self.hand\_result.landmark[point[1]].x)\*\*2

dist += (self.hand\_result.landmark[point[0]].y - self.hand\_result.landmark[point[1]].y)\*\*2

dist = math.sqrt(dist)

return dist\*sign

def get\_dist(self, point):

"""

returns euclidean distance between 'point'.

Parameters

----------

point : list contaning two elements of type list/tuple which represents

landmark point.

Returns

-------

float

"""

dist = (self.hand\_result.landmark[point[0]].x - self.hand\_result.landmark[point[1]].x)\*\*2

dist += (self.hand\_result.landmark[point[0]].y - self.hand\_result.landmark[point[1]].y)\*\*2

dist = math.sqrt(dist)

return dist

def get\_dz(self,point):

"""

returns absolute difference on z-axis between 'point'.

Parameters

----------

point : list contaning two elements of type list/tuple which represents

landmark point.

Returns

-------

float

"""

return abs(self.hand\_result.landmark[point[0]].z - self.hand\_result.landmark[point[1]].z)

# Function to find Gesture Encoding using current finger\_state.

# Finger\_state: 1 if finger is open, else 0

def set\_finger\_state(self):

"""

set 'finger' by computing ratio of distance between finger tip

, middle knuckle, base knuckle.

Returns

-------

None

"""

if self.hand\_result == None:

return

points = [[8,5,0],[12,9,0],[16,13,0],[20,17,0]]

self.finger = 0

self.finger = self.finger | 0 #thumb

for idx,point in enumerate(points):

dist = self.get\_signed\_dist(point[:2])

dist2 = self.get\_signed\_dist(point[1:])

try:

ratio = round(dist/dist2,1)

except:

ratio = round(dist1/0.01,1)

self.finger = self.finger << 1

if ratio > 0.5 :

self.finger = self.finger | 1

# Handling Fluctations due to noise

def get\_gesture(self):

"""

returns int representing gesture corresponding to Enum 'Gest'.

sets 'frame\_count', 'ori\_gesture', 'prev\_gesture',

handles fluctations due to noise.

Returns

-------

int

"""

if self.hand\_result == None:

return Gest.PALM

current\_gesture = Gest.PALM

if self.finger in [Gest.LAST3,Gest.LAST4] and self.get\_dist([8,4]) < 0.05:

if self.hand\_label == HLabel.MINOR :

current\_gesture = Gest.PINCH\_MINOR

else:

current\_gesture = Gest.PINCH\_MAJOR

elif Gest.FIRST2 == self.finger :

point = [[8,12],[5,9]]

dist1 = self.get\_dist(point[0])

dist2 = self.get\_dist(point[1])

ratio = dist1/dist2

if ratio > 1.7:

current\_gesture = Gest.V\_GEST

else:

if self.get\_dz([8,12]) < 0.1:

current\_gesture = Gest.TWO\_FINGER\_CLOSED

else:

current\_gesture = Gest.MID

else:

current\_gesture = self.finger

if current\_gesture == self.prev\_gesture:

self.frame\_count += 1

else:

self.frame\_count = 0

self.prev\_gesture = current\_gesture

if self.frame\_count > 4 :

self.ori\_gesture = current\_gesture

return self.ori\_gesture

# Executes commands according to detected gestures

class Controller:

"""

Executes commands according to detected gestures.

Attributes

----------

tx\_old : int

previous mouse location x coordinate

ty\_old : int

previous mouse location y coordinate

flag : bool

true if V gesture is detected

grabflag : bool

true if FIST gesture is detected

pinchmajorflag : bool

true if PINCH gesture is detected through MAJOR hand,

on x-axis 'Controller.changesystembrightness',

on y-axis 'Controller.changesystemvolume'.

pinchminorflag : bool

true if PINCH gesture is detected through MINOR hand,

on x-axis 'Controller.scrollHorizontal',

on y-axis 'Controller.scrollVertical'.

pinchstartxcoord : int

x coordinate of hand landmark when pinch gesture is started.

pinchstartycoord : int

y coordinate of hand landmark when pinch gesture is started.

pinchdirectionflag : bool

true if pinch gesture movment is along x-axis,

otherwise false

prevpinchlv : int

stores quantized magnitued of prev pinch gesture displacment, from

starting position

pinchlv : int

stores quantized magnitued of pinch gesture displacment, from

starting position

framecount : int

stores no. of frames since 'pinchlv' is updated.

prev\_hand : tuple

stores (x, y) coordinates of hand in previous frame.

pinch\_threshold : float

step size for quantization of 'pinchlv'.

"""

tx\_old = 0

ty\_old = 0

trial = True

flag = False

grabflag = False

pinchmajorflag = False

pinchminorflag = False

pinchstartxcoord = None

pinchstartycoord = None

pinchdirectionflag = None

prevpinchlv = 0

pinchlv = 0

framecount = 0

prev\_hand = None

pinch\_threshold = 0.3

def getpinchylv(hand\_result):

"""returns distance beween starting pinch y coord and current hand position y coord."""

dist = round((Controller.pinchstartycoord - hand\_result.landmark[8].y)\*10,1)

return dist

def getpinchxlv(hand\_result):

"""returns distance beween starting pinch x coord and current hand position x coord."""

dist = round((hand\_result.landmark[8].x - Controller.pinchstartxcoord)\*10,1)

return dist

def changesystembrightness():

"""sets system brightness based on 'Controller.pinchlv'."""

currentBrightnessLv = sbcontrol.get\_brightness(display=0)/100.0

currentBrightnessLv += Controller.pinchlv/50.0

if currentBrightnessLv > 1.0:

currentBrightnessLv = 1.0

elif currentBrightnessLv < 0.0:

currentBrightnessLv = 0.0

sbcontrol.fade\_brightness(int(100\*currentBrightnessLv) , start = sbcontrol.get\_brightness(display=0))

def changesystemvolume():

"""sets system volume based on 'Controller.pinchlv'."""

devices = AudioUtilities.GetSpeakers()

interface = devices.Activate(IAudioEndpointVolume.\_iid\_, CLSCTX\_ALL, None)

volume = cast(interface, POINTER(IAudioEndpointVolume))

currentVolumeLv = volume.GetMasterVolumeLevelScalar()

currentVolumeLv += Controller.pinchlv/50.0

if currentVolumeLv > 1.0:

currentVolumeLv = 1.0

elif currentVolumeLv < 0.0:

currentVolumeLv = 0.0

volume.SetMasterVolumeLevelScalar(currentVolumeLv, None)

def scrollVertical():

"""scrolls on screen vertically."""

pyautogui.scroll(120 if Controller.pinchlv>0.0 else -120)

def scrollHorizontal():

"""scrolls on screen horizontally."""

pyautogui.keyDown('shift')

pyautogui.keyDown('ctrl')

pyautogui.scroll(-120 if Controller.pinchlv>0.0 else 120)

pyautogui.keyUp('ctrl')

pyautogui.keyUp('shift')

# Locate Hand to get Cursor Position

# Stabilize cursor by Dampening

def get\_position(hand\_result):

"""

returns coordinates of current hand position.

Locates hand to get cursor position also stabilize cursor by

dampening jerky motion of hand.

Returns

-------

tuple(float, float)

"""

point = 9

position = [hand\_result.landmark[point].x ,hand\_result.landmark[point].y]

sx,sy = pyautogui.size()

x\_old,y\_old = pyautogui.position()

x = int(position[0]\*sx)

y = int(position[1]\*sy)

if Controller.prev\_hand is None:

Controller.prev\_hand = x,y

delta\_x = x - Controller.prev\_hand[0]

delta\_y = y - Controller.prev\_hand[1]

distsq = delta\_x\*\*2 + delta\_y\*\*2

ratio = 1

Controller.prev\_hand = [x,y]

if distsq <= 25:

ratio = 0

elif distsq <= 900:

ratio = 0.07 \* (distsq \*\* (1/2))

else:

ratio = 2.1

x , y = x\_old + delta\_x\*ratio , y\_old + delta\_y\*ratio

return (x,y)

def pinch\_control\_init(hand\_result):

"""Initializes attributes for pinch gesture."""

Controller.pinchstartxcoord = hand\_result.landmark[8].x

Controller.pinchstartycoord = hand\_result.landmark[8].y

Controller.pinchlv = 0

Controller.prevpinchlv = 0

Controller.framecount = 0

# Hold final position for 5 frames to change status

def pinch\_control(hand\_result, controlHorizontal, controlVertical):

"""

calls 'controlHorizontal' or 'controlVertical' based on pinch flags,

'framecount' and sets 'pinchlv'.

Parameters

----------

hand\_result : Object

Landmarks obtained from mediapipe.

controlHorizontal : callback function assosiated with horizontal

pinch gesture.

controlVertical : callback function assosiated with vertical

pinch gesture.

Returns

-------

None

"""

if Controller.framecount == 5:

Controller.framecount = 0

Controller.pinchlv = Controller.prevpinchlv

if Controller.pinchdirectionflag == True:

controlHorizontal() #x

elif Controller.pinchdirectionflag == False:

controlVertical() #y

lvx = Controller.getpinchxlv(hand\_result)

lvy = Controller.getpinchylv(hand\_result)

if abs(lvy) > abs(lvx) and abs(lvy) > Controller.pinch\_threshold:

Controller.pinchdirectionflag = False

if abs(Controller.prevpinchlv - lvy) < Controller.pinch\_threshold:

Controller.framecount += 1

else:

Controller.prevpinchlv = lvy

Controller.framecount = 0

elif abs(lvx) > Controller.pinch\_threshold:

Controller.pinchdirectionflag = True

if abs(Controller.prevpinchlv - lvx) < Controller.pinch\_threshold:

Controller.framecount += 1

else:

Controller.prevpinchlv = lvx

Controller.framecount = 0

def handle\_controls(gesture, hand\_result):

"""Impliments all gesture functionality."""

x,y = None,None

if gesture != Gest.PALM :

x,y = Controller.get\_position(hand\_result)

# flag reset

if gesture != Gest.FIST and Controller.grabflag:

Controller.grabflag = False

pyautogui.mouseUp(button = "left")

if gesture != Gest.PINCH\_MAJOR and Controller.pinchmajorflag:

Controller.pinchmajorflag = False

if gesture != Gest.PINCH\_MINOR and Controller.pinchminorflag:

Controller.pinchminorflag = False

# implementation

if gesture == Gest.V\_GEST:

Controller.flag = True

pyautogui.moveTo(x, y, duration = 0.1)

elif gesture == Gest.FIST:

if not Controller.grabflag :

Controller.grabflag = True

pyautogui.mouseDown(button = "left")

pyautogui.moveTo(x, y, duration = 0.1)

elif gesture == Gest.MID and Controller.flag:

pyautogui.click()

Controller.flag = False

elif gesture == Gest.INDEX and Controller.flag:

pyautogui.click(button='right')

Controller.flag = False

elif gesture == Gest.TWO\_FINGER\_CLOSED and Controller.flag:

pyautogui.doubleClick()

Controller.flag = False

elif gesture == Gest.PINCH\_MINOR:

if Controller.pinchminorflag == False:

Controller.pinch\_control\_init(hand\_result)

Controller.pinchminorflag = True

Controller.pinch\_control(hand\_result,Controller.scrollHorizontal, Controller.scrollVertical)

elif gesture == Gest.PINCH\_MAJOR:

if Controller.pinchmajorflag == False:

Controller.pinch\_control\_init(hand\_result)

Controller.pinchmajorflag = True

Controller.pinch\_control(hand\_result,Controller.changesystembrightness, Controller.changesystemvolume)

'''

---------------------------------------- Main Class ----------------------------------------

Entry point of Gesture Controller

'''

class GestureController:

"""

Handles camera, obtain landmarks from mediapipe, entry point

for whole program.

Attributes

----------

gc\_mode : int

indicates weather gesture controller is running or not,

1 if running, otherwise 0.

cap : Object

object obtained from cv2, for capturing video frame.

CAM\_HEIGHT : int

highet in pixels of obtained frame from camera.

CAM\_WIDTH : int

width in pixels of obtained frame from camera.

hr\_major : Object of 'HandRecog'

object representing major hand.

hr\_minor : Object of 'HandRecog'

object representing minor hand.

dom\_hand : bool

True if right hand is domaniant hand, otherwise False.

default True.

"""

gc\_mode = 0

cap = None

CAM\_HEIGHT = None

CAM\_WIDTH = None

hr\_major = None # Right Hand by default

hr\_minor = None # Left hand by default

dom\_hand = True

def \_\_init\_\_(self):

"""Initilaizes attributes."""

GestureController.gc\_mode = 1

GestureController.cap = cv2.VideoCapture(0)

GestureController.CAM\_HEIGHT = GestureController.cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT)

GestureController.CAM\_WIDTH = GestureController.cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH)

def classify\_hands(results):

"""

sets 'hr\_major', 'hr\_minor' based on classification(left, right) of

hand obtained from mediapipe, uses 'dom\_hand' to decide major and

minor hand.

"""

left , right = None,None

try:

handedness\_dict = MessageToDict(results.multi\_handedness[0])

if handedness\_dict['classification'][0]['label'] == 'Right':

right = results.multi\_hand\_landmarks[0]

else :

left = results.multi\_hand\_landmarks[0]

except:

pass

try:

handedness\_dict = MessageToDict(results.multi\_handedness[1])

if handedness\_dict['classification'][0]['label'] == 'Right':

right = results.multi\_hand\_landmarks[1]

else :

left = results.multi\_hand\_landmarks[1]

except:

pass

if GestureController.dom\_hand == True:

GestureController.hr\_major = right

GestureController.hr\_minor = left

else :

GestureController.hr\_major = left

GestureController.hr\_minor = right

def start(self):

"""

Entry point of whole programm, caputres video frame and passes, obtains

landmark from mediapipe and passes it to 'handmajor' and 'handminor' for

controlling.

"""

handmajor = HandRecog(HLabel.MAJOR)

handminor = HandRecog(HLabel.MINOR)

with mp\_hands.Hands(max\_num\_hands = 2,min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as hands:

while GestureController.cap.isOpened() and GestureController.gc\_mode:

success, image = GestureController.cap.read()

if not success:

print("Ignoring empty camera frame.")

continue

image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR\_BGR2RGB)

image.flags.writeable = False

results = hands.process(image)

image.flags.writeable = True

image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)

if results.multi\_hand\_landmarks:

GestureController.classify\_hands(results)

handmajor.update\_hand\_result(GestureController.hr\_major)

handminor.update\_hand\_result(GestureController.hr\_minor)

handmajor.set\_finger\_state()

handminor.set\_finger\_state()

gest\_name = handminor.get\_gesture()

if gest\_name == Gest.PINCH\_MINOR:

Controller.handle\_controls(gest\_name, handminor.hand\_result)

else:

gest\_name = handmajor.get\_gesture()

Controller.handle\_controls(gest\_name, handmajor.hand\_result)

for hand\_landmarks in results.multi\_hand\_landmarks:

mp\_drawing.draw\_landmarks(image, hand\_landmarks, mp\_hands.HAND\_CONNECTIONS)

else:

Controller.prev\_hand = None

cv2.imshow('Gesture Controller', image)

if cv2.waitKey(5) & 0xFF == 13:

break

GestureController.cap.release()

cv2.destroyAllWindows()

# uncomment to run directly

# gc1 = GestureController()

# gc1.start()