MACHINE LEARNING

HAND GESTURE VOLUME CONTROL

Created by:Tanisha Sood

Project Summary:

The Hand Gesture Volume Controller project aims to provide a method for controlling a PC's volume using hand gestures. This project utilizes computer vision technology and Mediapipe (an open source by Google) to recognize and interpret specific hand movements and translate them into volume adjustments for audio playback.

Project Objectives:

- Develop a real-time hand recognition and detection system.
- Enable users to control volume levels through different hand gestures.
- Ensure robustness and accuracy in the project for a reliable user experience.

Requirements:

- opencv-python
- mediapipe
- comtypes

- numpy
- pycaw

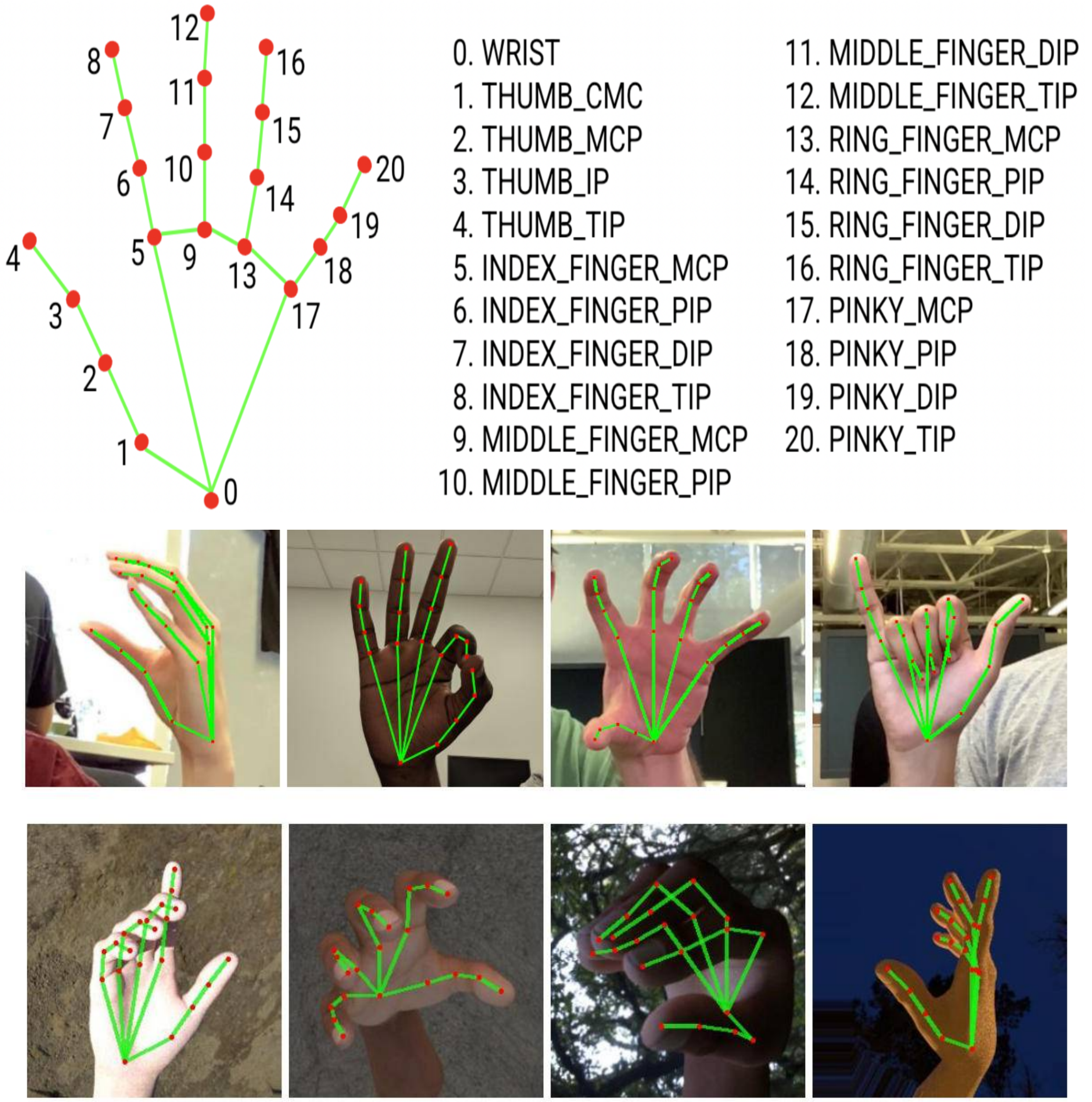
1. **openCV:** openCV stands for Open Source Computer Vision Library. It is a cross-platform library that includes hundreds of computer vision algorithms. It is used in image processing, video processing, object detection, face recognition, motion tracking, augmented reality etc.

2. **Mediapipe:** MediaPipe is an open-source framework for building pipelines to perform computer vision inference over arbitrary sensory data such as video or audio.

These libraries and resources provide the core functionality for each MediaPipe Solution:

- MediaPipe Tasks: Cross-platform APIs and libraries for deploying solutions.
- MediaPipe Models: Pre-trained, ready-to-run models for use with each solution.

In this project, the Hand Landmarks Detection Model of Mediapipe is used.



3. Comtypes: It is a lightweight Python COM package, based on the ctypes FFI library. Comtypes allows us to define, call, and implement custom and dispatch-based COM interfaces in pure Python. This package works on Windows only.

4. Numpy: Numpy contains a multi-dimensional array and matrix data structures. It can be utilized to perform many mathematical operations on arrays.

5. Pycaw: Python Core Audio Windows Library, working for both Python2 and Python3.It was created by Andre Miras.
The Git repository link for pycaw is given below:
https://github.com/AndreMiras/pycaw.git

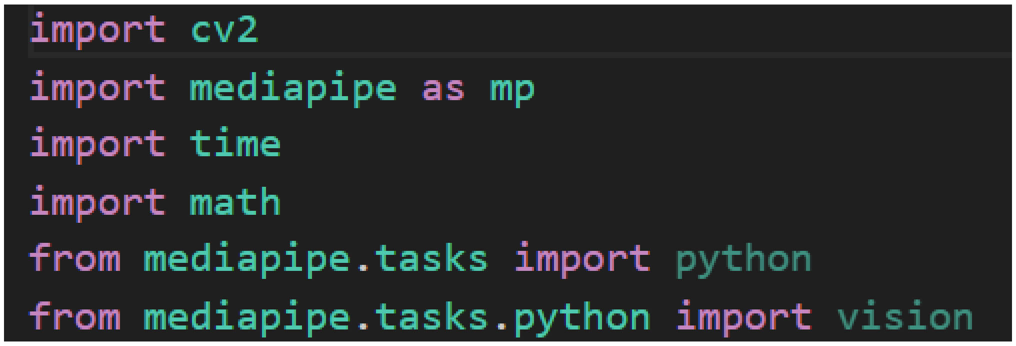
Implementation:

- Computer vision algorithm: Computer vision algorithms analyze certain criteria in images and videos, and then apply interpretations to predictive or decision-making tasks.
- Gesture to volume mapping: Establishes a mapping between recognized gestures and corresponding volume adjustments.
- User interface: Displays adjusted volume level and personalized messages. Personalized gestures for program termination.
- Link for the project: https://drive.google.com/drive/folders/1y2dH3fEiC9-hCFxbtskdEfTJwkUqT3px?usp=sharing

Code Explanation:

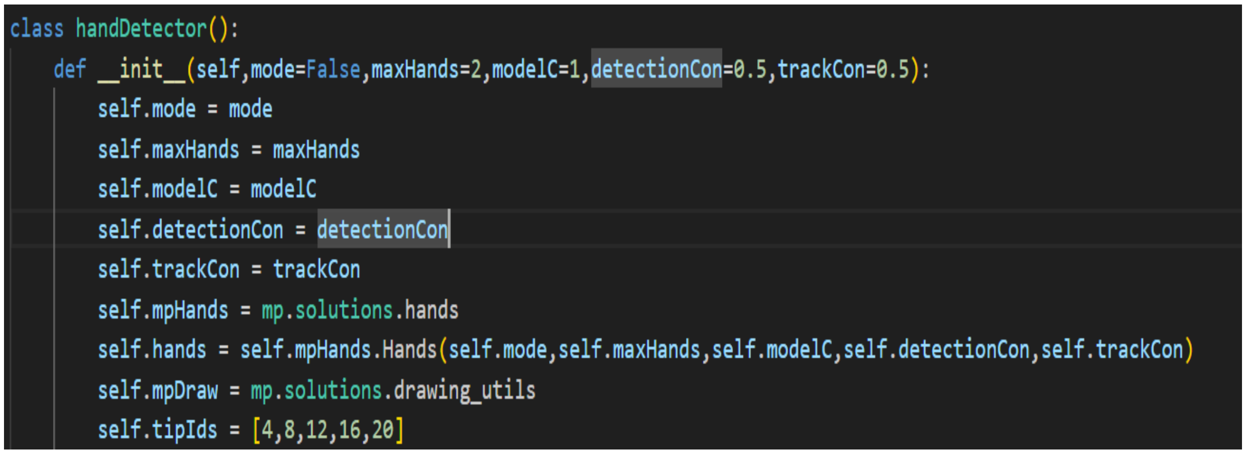
1. Creating Hand tracking module:

- Importing libraries:



```
import cv2
import mediapipe as mp
import time
import math
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
```

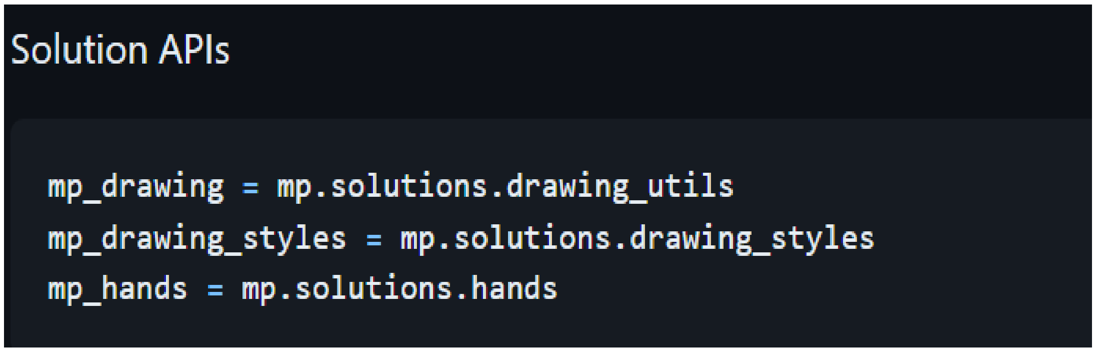
- Creating a class and _init_function(an instance method that initializes a newly created object.) and solution APIs



```
class handDetector():
    def __init__(self,mode=False,maxHands=2,modelC=1,detectionCon=0.5,trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.modelC = modelC
        self.detectionCon = detectionCon
        self.trackCon = trackCon
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode,self.maxHands,self.modelC,self.detectionCon,self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils
        self.tipIds = [4,8,12,16,20]
```

➔ Static_Image_Mode: If set to false, the solution treats the input images as a video stream. It will try to detect hands in the first input images, and upon a successful detection further localizes the hand landmarks. In subsequent images, once all max_num_hands hands are detected and the corresponding hand landmarks are localized, it simply tracks those landmarks without invoking another detection until it loses track of any of the hands. This reduces latency and is ideal for processing video frames. If set to true, hand detection runs on every input image, ideal for processing a batch of static, possibly unrelated, images. Default to false.

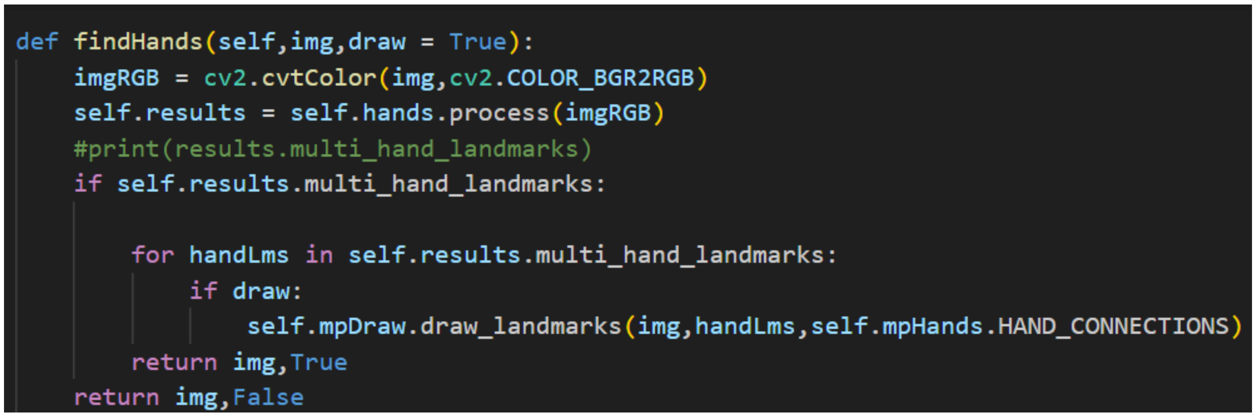
- ➔ MAX_NUM_HANDS: Maximum number of hands to detect. Default to 2.
- ➔ MODEL_COMPLEXITY: Complexity of the hand landmark model: 0 or 1. Landmark accuracy as well as inference latency generally go up with the model complexity. Default to 1.
- ➔ MIN_DETECTION_CONFIDENCE: Minimum confidence value ([0.0, 1.0]) from the hand detection model for the detection to be considered successful. Default to 0.5.
- ➔ MIN_TRACKING_CONFIDENCE: Minimum confidence value ([0.0, 1.0]) from the landmark-tracking model for the hand landmarks to be considered tracked successfully, or otherwise hand detection will be invoked automatically on the next input image. Setting it to a higher value can increase the robustness of the solution, at the expense of higher latency. Ignored if static_image_mode is true, where hand detection simply runs on every image. Default to 0.5.



Solution APIs

```
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands
```

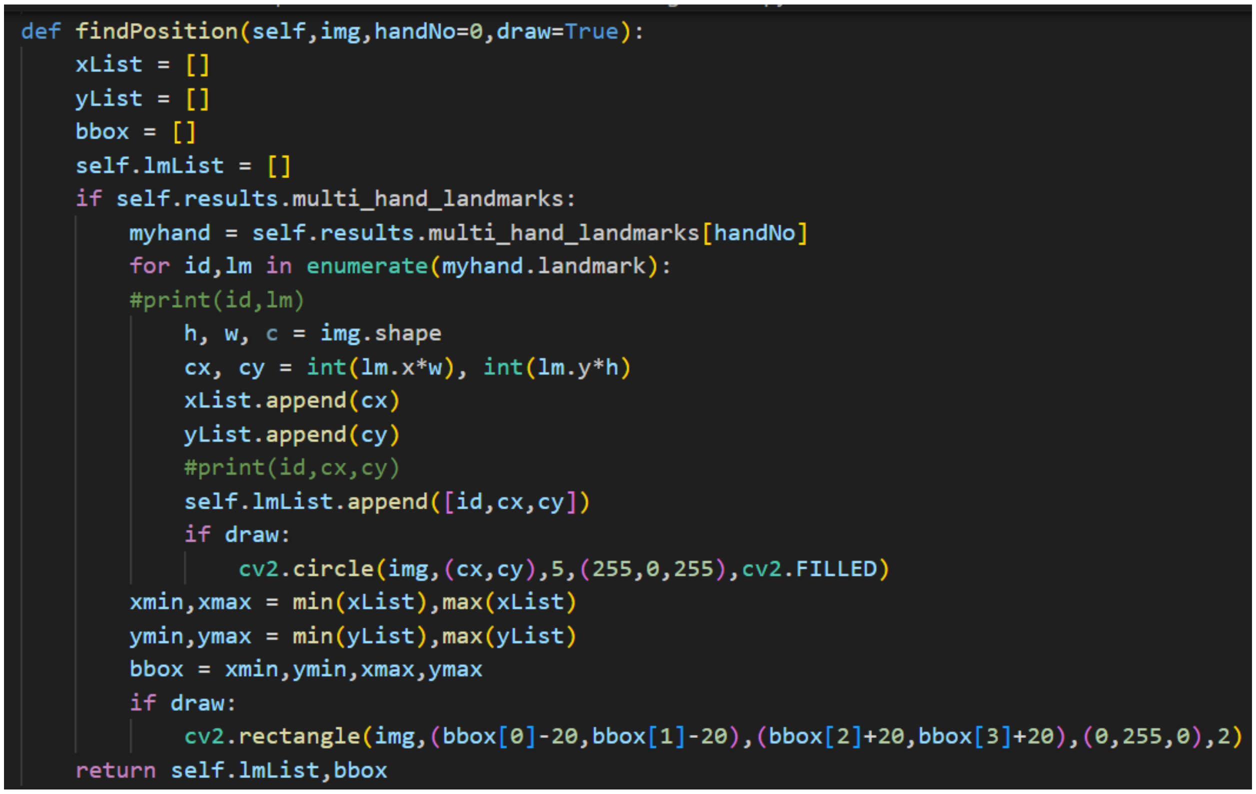
- Create a function to find hands using mediapipe:



```python
def findHands(self,img,draw = True):
    imgRGB = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    self.results = self.hands.process(imgRGB)
    #print(results.multi_hand_landmarks)
    if self.results.multi_hand_landmarks:

        for handLms in self.results.multi_hand_landmarks:
            if draw:
                self.mpDraw.draw_landmarks(img,handLms,self.mpHands.HAND_CONNECTIONS)
        return img,True
    return img,False
```

- Create a function to find the position of the image and creating a bounding box across the hand detected.



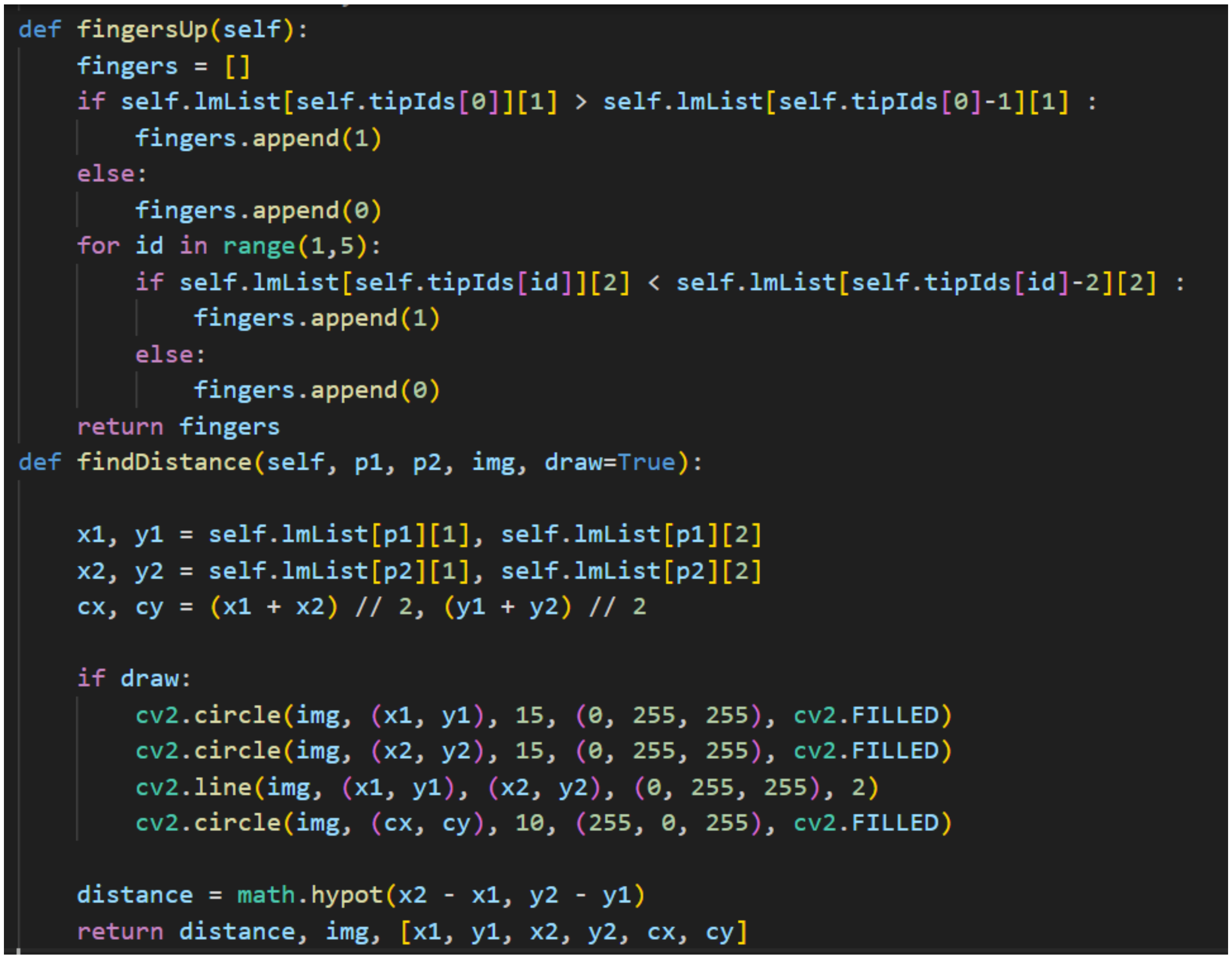
```python
def findPosition(self,img,handNo=0,draw=True):
    xList = []
    yList = []
    bbox = []
    self.lmList = []
    if self.results.multi_hand_landmarks:
        myhand = self.results.multi_hand_landmarks[handNo]
        for id,lm in enumerate(myhand.landmark):
        #print(id,lm)
            h, w, c = img.shape
            cx, cy = int(lm.x*w), int(lm.y*h)
            xList.append(cx)
            yList.append(cy)
            #print(id,cx,cy)
            self.lmList.append([id,cx,cy])
            if draw:
                cv2.circle(img,(cx,cy),5,(255,0,255),cv2.FILLED)
        xmin,xmax = min(xList),max(xList)
        ymin,ymax = min(yList),max(yList)
        bbox = xmin,ymin,xmax,ymax
        if draw:
            cv2.rectangle(img,(bbox[0]-20,bbox[1]-20),(bbox[2]+20,bbox[3]+20),(0,255,0),2)
    return self.lmList,bbox
```

- Create a function to find which finger is up using tipIds[4,8,12,16,20] and a function to find the distance between the tip of index finger and the tip of the thumb.

➔ We will map this distance with the volume range. We will assume that with increasing distance the volume will also increase i.e. direct proportionality.

➔ Circle will be drawn on the hand landmarks as well as the tips of index finger and thumb.A connecting line will also be drawn between desired landmarks.

➔ Distance is calculated using math library.



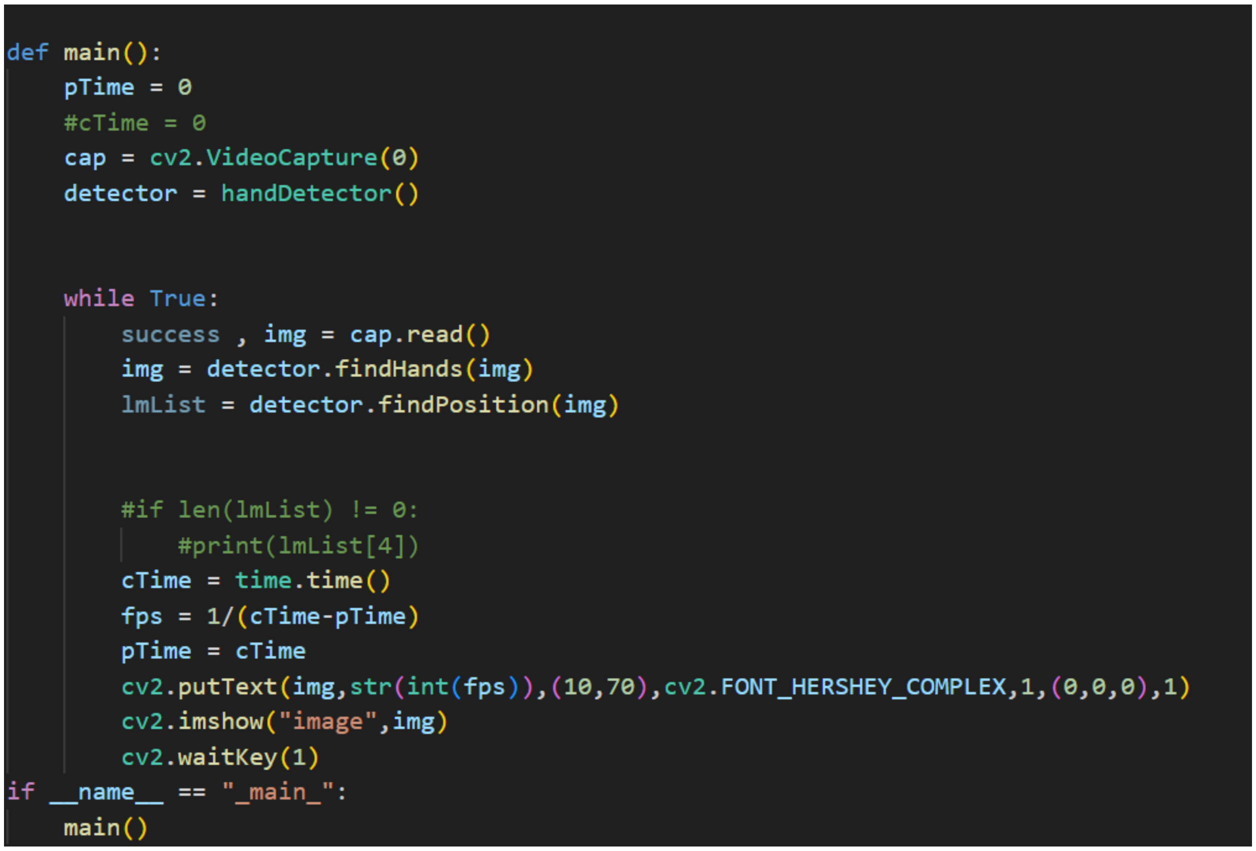
```
def fingersUp(self):
    fingers = []
    if self.lmList[self.tipIds[0]][1] > self.lmList[self.tipIds[0]-1][1] :
        fingers.append(1)
    else:
        fingers.append(0)
    for id in range(1,5):
        if self.lmList[self.tipIds[id]][2] < self.lmList[self.tipIds[id]-2][2] :
            fingers.append(1)
        else:
            fingers.append(0)
    return fingers
def findDistance(self, p1, p2, img, draw=True):

    x1, y1 = self.lmList[p1][1], self.lmList[p1][2]
    x2, y2 = self.lmList[p2][1], self.lmList[p2][2]
    cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

    if draw:
        cv2.circle(img, (x1, y1), 15, (0, 255, 255), cv2.FILLED)
        cv2.circle(img, (x2, y2), 15, (0, 255, 255), cv2.FILLED)
        cv2.line(img, (x1, y1), (x2, y2), (0, 255, 255), 2)
        cv2.circle(img, (cx, cy), 10, (255, 0, 255), cv2.FILLED)

    distance = math.hypot(x2 - x1, y2 - y1)
    return distance, img, [x1, y1, x2, y2, cx, cy]
```

- Creating and calling the main function:
  - ➔ Initializing device’s camera to capture video.Read the image from the camera and call the required functions.
  - ➔ Create FPS.
  - ➔ Display the image.
  - ➔ Create terminating condition.



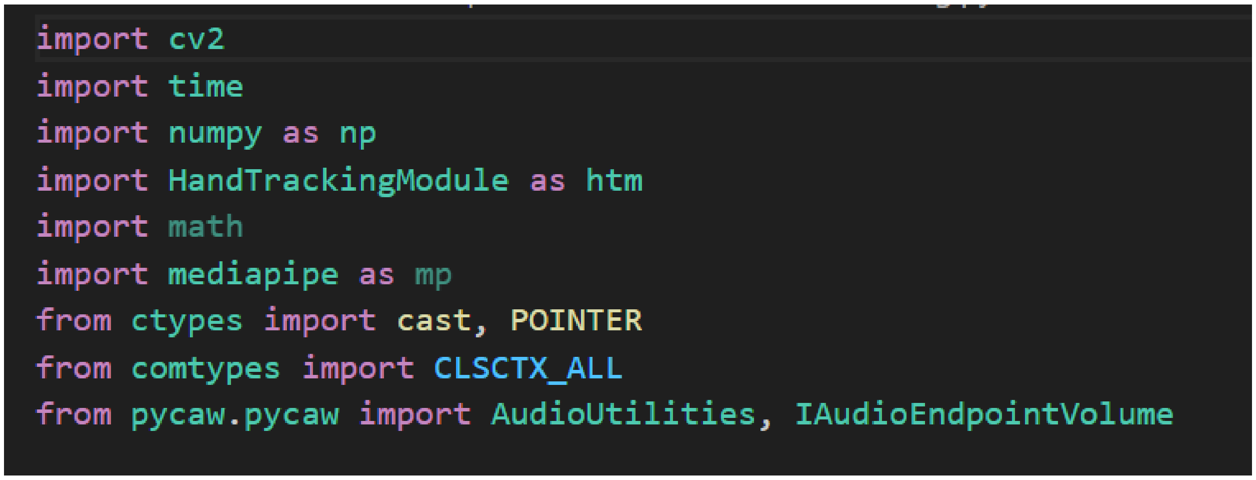
```
def main():
    pTime = 0
    #cTime = 0
    cap = cv2.VideoCapture(0)
    detector = handDetector()


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


        #if len(lmList) != 0:
            #print(lmList[4])
        cTime = time.time()
        fps = 1/(cTime-pTime)
        pTime = cTime
        cv2.putText(img,str(int(fps)),(10,70),cv2.FONT_HERSHEY_COMPLEX,1,(0,0,0),1)
        cv2.imshow("image",img)
        cv2.waitKey(1)
if __name__ == "_main_":
    main()
```

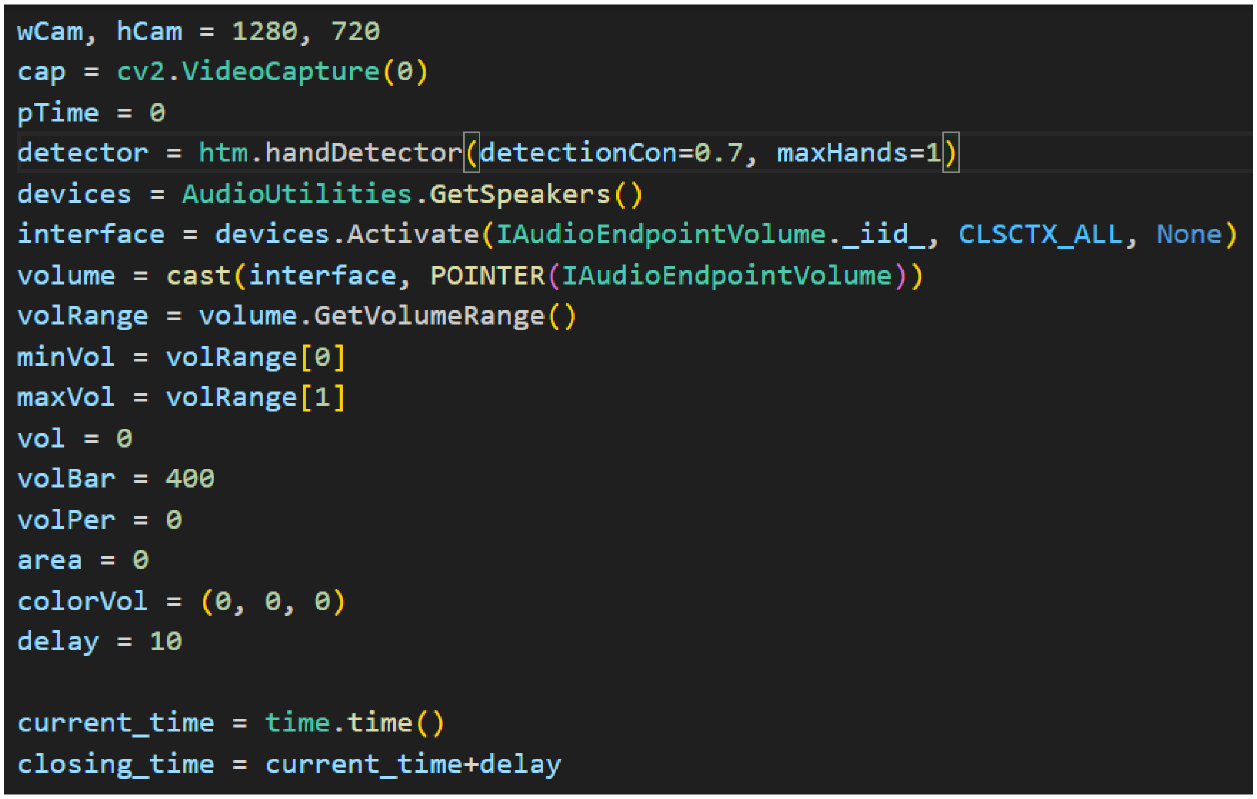
2. ImplementingVolume adjustment :

- Import required libraries:



```
import cv2
import time
import numpy as np
import HandTrackingModule as htm
import math
import mediapipe as mp
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
```

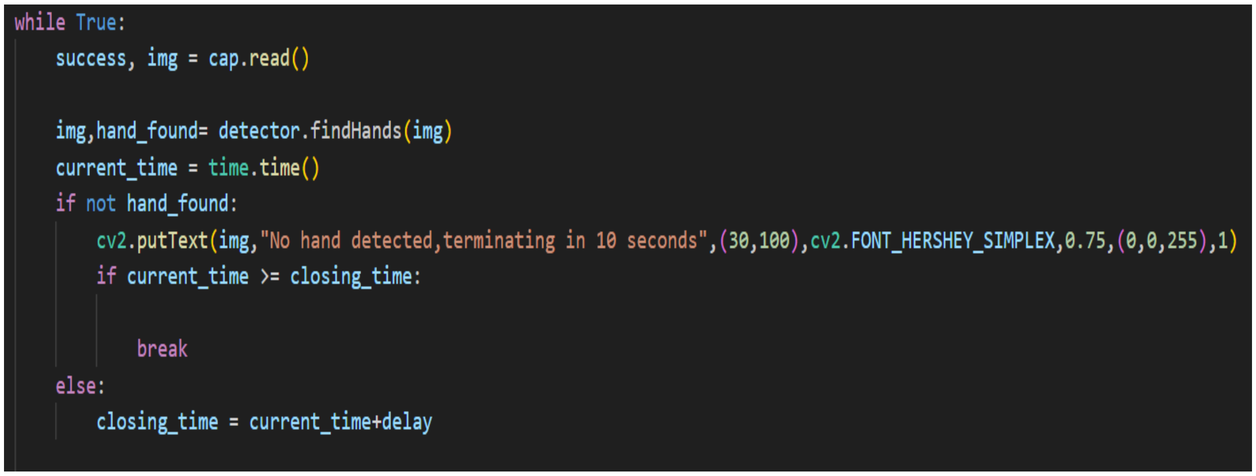
- Setting up the device's camera and declaring the variables use.



```python
wCam, hCam = 1280, 720
cap = cv2.VideoCapture(0)
pTime = 0
detector = htm.handDetector(detectionCon=0.7, maxHands=1)
devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
volRange = volume.GetVolumeRange()
minVol = volRange[0]
maxVol = volRange[1]
vol = 0
volBar = 400
volPer = 0
area = 0
colorVol = (0, 0, 0)
delay = 10

current_time = time.time()
closing_time = current_time+delay
```

- Creating a condition in case of no hand detection.The program will be terminated if no hand is detected in the camera for 10 seconds.



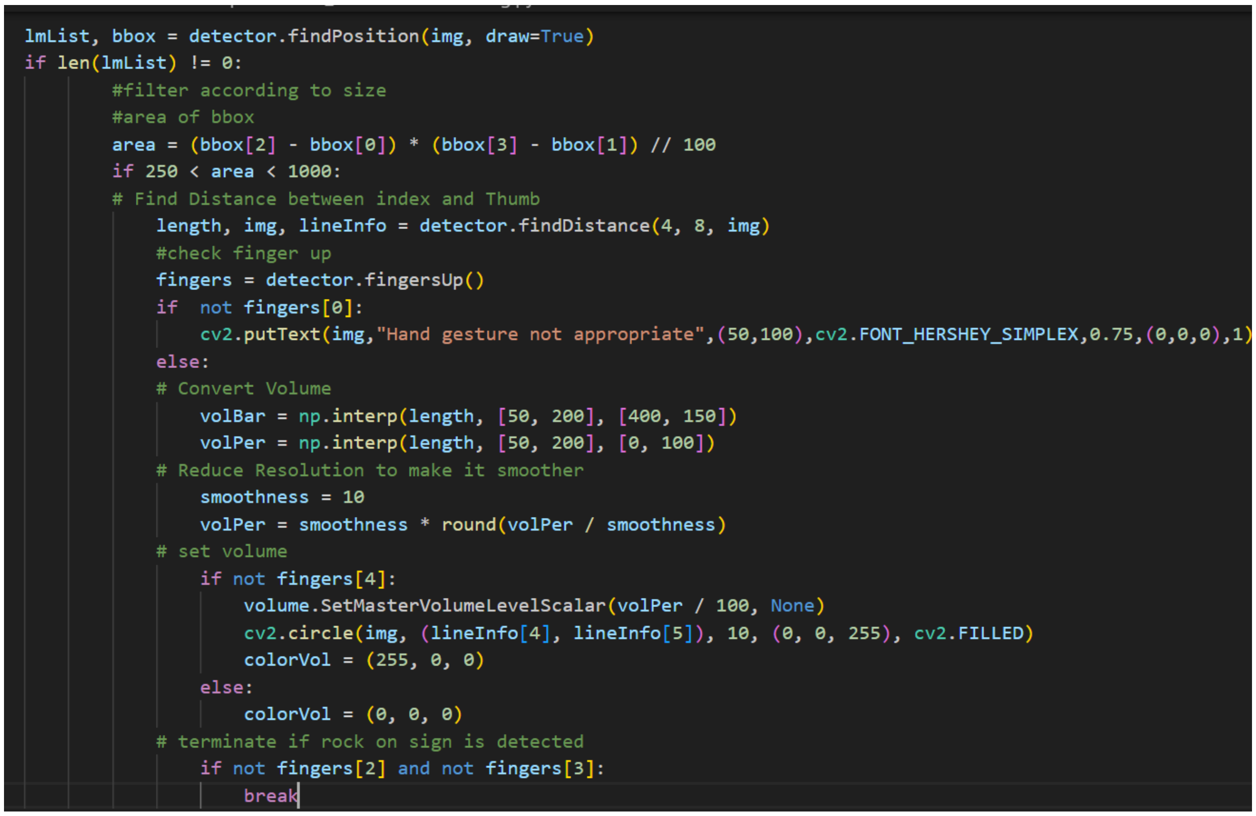
```python
while True:
    success, img = cap.read()

    img,hand_found= detector.findHands(img)
    current_time = time.time()
    if not hand_found:
        cv2.putText(img,"No hand detected,terminating in 10 seconds",(30,100),cv2.FONT_HERSHEY_SIMPLEX,0.75,(0,0,255),1)
        if current_time >= closing_time:

            break
    else:
        closing_time = current_time+delay
```

- Applying the main logic for volume adjustment.
  - ➔ Filter the images received from camera during the live streaming according to a particular size for convenience and better accura y.

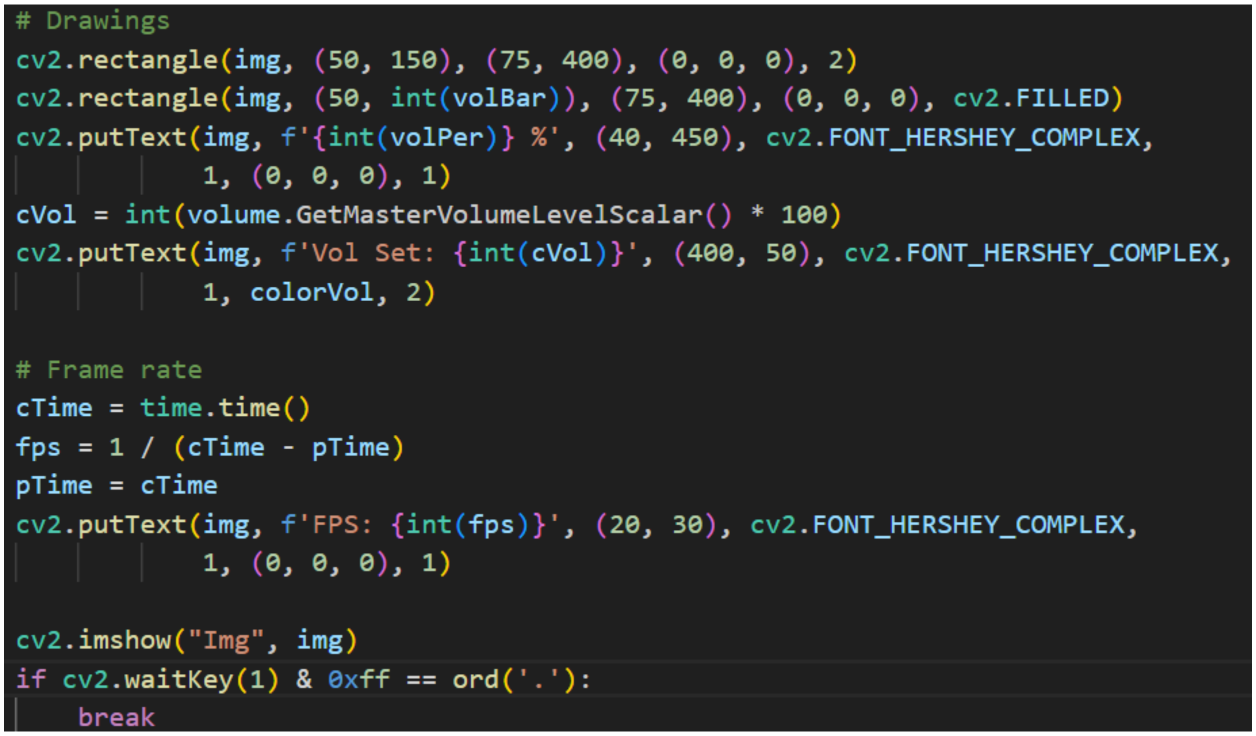
→ Finding the distance between index finger and thumb.
→ Checking the state of the fingers i.e. whether they are up or down.
→ Warning message will be displayed if hand gesture is not appropriate .
→ If the hand gesture is appropriate,the distance will be converted to numerical values for volume. This is done using np.interp().
→ Reducing resolution for better smoothness.
→ Set the volume if pinky finger is down.
→ Terminate program if rock-on sign is detected i.e. middle and ring finger are down.



```
lmList, bbox = detector.findPosition(img, draw=True)
if len(lmList) != 0:
        #filter according to size
        #area of bbox
        area = (bbox[2] - bbox[0]) * (bbox[3] - bbox[1]) // 100
        if 250 < area < 1000:
        # Find Distance between index and Thumb
            length, img, lineInfo = detector.findDistance(4, 8, img)
            #check finger up
            fingers = detector.fingersUp()
            if  not fingers[0]:
                cv2.putText(img,"Hand gesture not appropriate",(50,100),cv2.FONT_HERSHEY_SIMPLEX,0.75,(0,0,0),1)
            else:
            # Convert Volume
                volBar = np.interp(length, [50, 200], [400, 150])
                volPer = np.interp(length, [50, 200], [0, 100])
            # Reduce Resolution to make it smoother
                smoothness = 10
                volPer = smoothness * round(volPer / smoothness)
            # set volume
                if not fingers[4]:
                    volume.SetMasterVolumeLevelScalar(volPer / 100, None)
                    cv2.circle(img, (lineInfo[4], lineInfo[5]), 10, (0, 0, 255), cv2.FILLED)
                    colorVol = (255, 0, 0)
                else:
                    colorVol = (0, 0, 0)
            # terminate if rock on sign is detected
                if not fingers[2] and not fingers[3]:
                    break
```

- Creating all the necessary drawings.
  → Rectangles for the volume bar and volume percentage.
  → Normalizing volume range.

➔ Creating frame rate and terminating conditions.



```
# Drawings
cv2.rectangle(img, (50, 150), (75, 400), (0, 0, 0), 2)
cv2.rectangle(img, (50, int(volBar)), (75, 400), (0, 0, 0), cv2.FILLED)
cv2.putText(img, f'{int(volPer)} %', (40, 450), cv2.FONT_HERSHEY_COMPLEX,
            1, (0, 0, 0), 1)
cVol = int(volume.GetMasterVolumeLevelScalar() * 100)
cv2.putText(img, f'Vol Set: {int(cVol)}', (400, 50), cv2.FONT_HERSHEY_COMPLEX,
            1, colorVol, 2)

# Frame rate
cTime = time.time()
fps = 1 / (cTime - pTime)
pTime = cTime
cv2.putText(img, f'FPS: {int(fps)}', (20, 30), cv2.FONT_HERSHEY_COMPLEX,
            1, (0, 0, 0), 1)

cv2.imshow("Img", img)
if cv2.waitKey(1) & 0xff == ord('.'):
    break
```

- Closing web cam using cam.release().