

Session 2 - MediaPipe

TOC :

1. Overview of MediaPipe and its Capabilities
2. Comparison with other computer vision libraries and frameworks
3. Installation and Setup
4. MediaPipe Hand tracking and Gesture recognition
5. MediaPipe Face Detection and Tracking

1. Overview of MediaPipe and its Capabilities

MediaPipe is an open-source cross-platform framework developed by Google for building multimodal machine learning applications. It provides a wide range of pre-built modules for tasks like face detection, pose estimation, hand tracking, object detection, and more. MediaPipe is built using C++ and Python and can be used on multiple platforms like Android, iOS, Windows, and Linux. MediaPipe also provides APIs for integrating custom machine learning models into the pipeline.

Mediapipe is a powerful library that provides a wide range of computer vision and machine learning solutions. Here are some of the things that can be done using Mediapipe:

- Object detection and tracking: Detecting and tracking objects in video or images.
- Face detection and recognition: Detecting faces and recognizing faces in video or images.
- Hand tracking and gesture recognition: Detecting and tracking hands and recognizing gestures in real-time video or images.
- Pose estimation: Estimating the human body pose from images or video streams.
- Segmentation: Segmenting objects from images or video streams. Image and video processing: Various image and video processing operations such as resizing, cropping, rotation, filtering, and blending.
- Audio processing: Processing and analyzing audio signals for various applications such as speech recognition, speaker identification, and emotion detection.
- Natural Language Processing (NLP): Natural language processing tasks such as sentiment analysis, text classification, and speech-to-text conversion. These are just a few examples of the many things that can be done using Mediapipe. The library is constantly evolving and new features are added frequently, making it an extremely versatile and powerful tool for computer vision and machine learning applications.

2. Comparison with other computer vision libraries and frameworks

MediaPipe provides a unique combination of machine learning-based approaches and traditional computer vision techniques that make it stand out from other computer vision libraries like OpenCV and frameworks like TensorFlow. MediaPipe provides a pipeline for building multimodal applications that integrate multiple machine learning and computer vision techniques. It also provides pre-built modules that can be used out of the box, reducing the need for complex code development.

3. Installation and Setup

MediaPipe can be installed using pip, the Python package manager, as follows:

```
pip install mediapipe
```

```
In [ ]: !pip install mediapipe
```

```

Requirement already satisfied: mediapipe in c:\users\omolp091\anaconda3\lib\site-packages (0.10.5)
Requirement already satisfied: numpy in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (1.23.5)
Requirement already satisfied: attrs>=19.1.0 in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (22.1.0)
Requirement already satisfied: opencv-contrib-python in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (4.8.1.78)
Requirement already satisfied: matplotlib in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (3.7.0)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (23.5.26)
Requirement already satisfied: sounddevice>=0.4.4 in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (0.4.6)
Requirement already satisfied: protobuf<4,>=3.11 in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (3.20.3)
Requirement already satisfied: absl-py in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (2.0.0)
Requirement already satisfied: CFFI>=1.0 in c:\users\omolp091\anaconda3\lib\site-packages (from sounddevice>=0.4.4->mediapipe) (1.15.1)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (2.8.2)
Requirement already satisfied: packaging>=20.0 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (22.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (1.4.4)
Requirement already satisfied: pillow>=6.2.0 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (9.4.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (4.25.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (3.0.9)
Requirement already satisfied: pycparser in c:\users\omolp091\anaconda3\lib\site-packages (from CFFI>=1.0->sounddevice>=0.4.4->mediapipe) (2.21)
Requirement already satisfied: six>=1.5 in c:\users\omolp091\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib->mediapipe) (1.16.0)

```

4. MediaPipe Hand tracking and Gesture recognition

One of the most popular features of MediaPipe is the hand landmark detection module, which allows for real-time and accurate detection of 21 key points (landmarks) on a person's hand.

The hand landmark detection module uses a deep neural network to analyze an input image or video frame and predict the 3D coordinates of the 21 hand landmarks. These landmarks correspond to various points on the hand, such as the tips of the fingers, the base of the thumb, and the center of the palm.

The Mediapipe hand landmark detection pipeline is composed of several stages, including:

- Hand detection: The first step involves detecting the presence of a hand in the input image or video frame. This is done using a machine learning model that has been trained to recognize the shape and structure of a hand.
- Hand localization: Once a hand has been detected, the next step involves localizing the hand and aligning it to a canonical coordinate system. This is important for ensuring that the hand landmarks are consistently detected across different orientations and positions of the hand.
- Hand landmark estimation: The final stage involves estimating the 3D coordinates of the 21 hand landmarks. This is done using a deep neural network that has been trained on a large dataset of hand images and corresponding landmark annotations.

Once the hand landmarks have been detected, they can be used for a wide range of applications, such as gesture recognition, hand tracking, and virtual try-on. The Mediapipe hand landmark detection module is highly optimized for real-time performance and can be easily integrated into Python applications using the Mediapipe Python API.

```
In [ ]: import cv2
import mediapipe as mp

# Load the Mediapipe hand landmark model
mp_hands = mp.solutions.hands.Hands(
    static_image_mode=False,
    max_num_hands=2,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5)

# Initialize the drawing module for hands
mp_drawing = mp.solutions.drawing_utils

# Initialize the video capture object
cap = cv2.VideoCapture(0)

while True:
    # Read a new frame from the video capture object
    ret, frame = cap.read()

    # Convert the color space from BGR to RGB
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

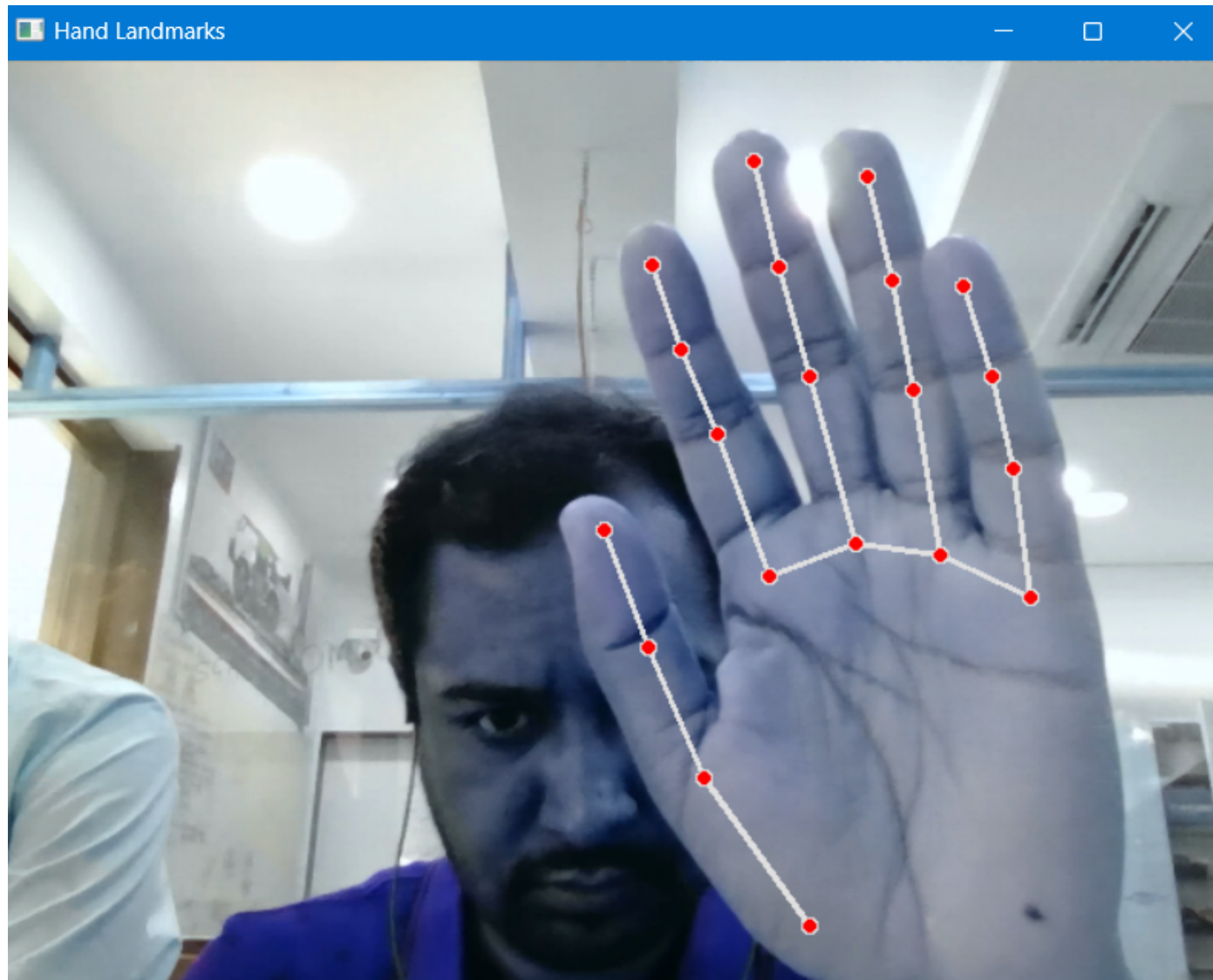
    # Detect the hand landmarks in the current frame
    results = mp_hands.process(frame)
```

```
# Draw the hand Landmarks and connections on the current frame
if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(
            frame, hand_landmarks, mp.solutions.hands.HAND_CONNECTIONS)

# Display the current frame in a window
cv2.imshow('Hand Landmarks', frame)

# Check for a key event and exit if 'q' is pressed
if cv2.waitKey(1) == ord('q'):
    break

# Release the video capture object and destroy all windows
cap.release()
cv2.destroyAllWindows()
```



5. MediaPipe Face Detection and Tracking

- MediaPipe Face Detection and Tracking is a pre-built computer vision pipeline developed by Google that uses machine learning to detect and track faces in real-time video streams or image sequences. It is based on a deep neural network trained on a large dataset of images and is capable of detecting and tracking multiple faces simultaneously.

- The MediaPipe Face Detection and Tracking pipeline consists of two main components: a face detection model and a face tracking model. The face detection model is responsible for detecting faces in the input video frames or images, while the face tracking model is responsible for tracking the detected faces across frames and maintaining their identities.
- The face detection model is based on the Single Shot Detector (SSD) architecture, which is a popular object detection algorithm that uses a single neural network to predict object bounding boxes and class probabilities in an input image. The SSD architecture is trained on a large dataset of annotated images of faces and is capable of detecting faces in various orientations and lighting conditions.
- The MediaPipe Face Detection and Tracking pipeline can be used for a wide range of applications, including video conferencing, virtual makeup try-on, and emotion detection. It is also highly customizable, allowing developers to fine-tune the pipeline for specific use cases and integrate it into their own applications.

Here's an example code snippet to detect and track faces using MediaPipe :

```
In [ ]: import cv2
import mediapipe as mp

# Initialize the MediaPipe face detection module
mp_face_detection = mp.solutions.face_detection

# Initialize the MediaPipe drawing module
mp_draw = mp.solutions.drawing_utils

# Initialize the VideoCapture object
cap = cv2.VideoCapture(0)

# Loop over the frames
while True:
    # Read the frame from the camera
    success, img = cap.read()
    if not success:
        break

    # Convert the image to RGB
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

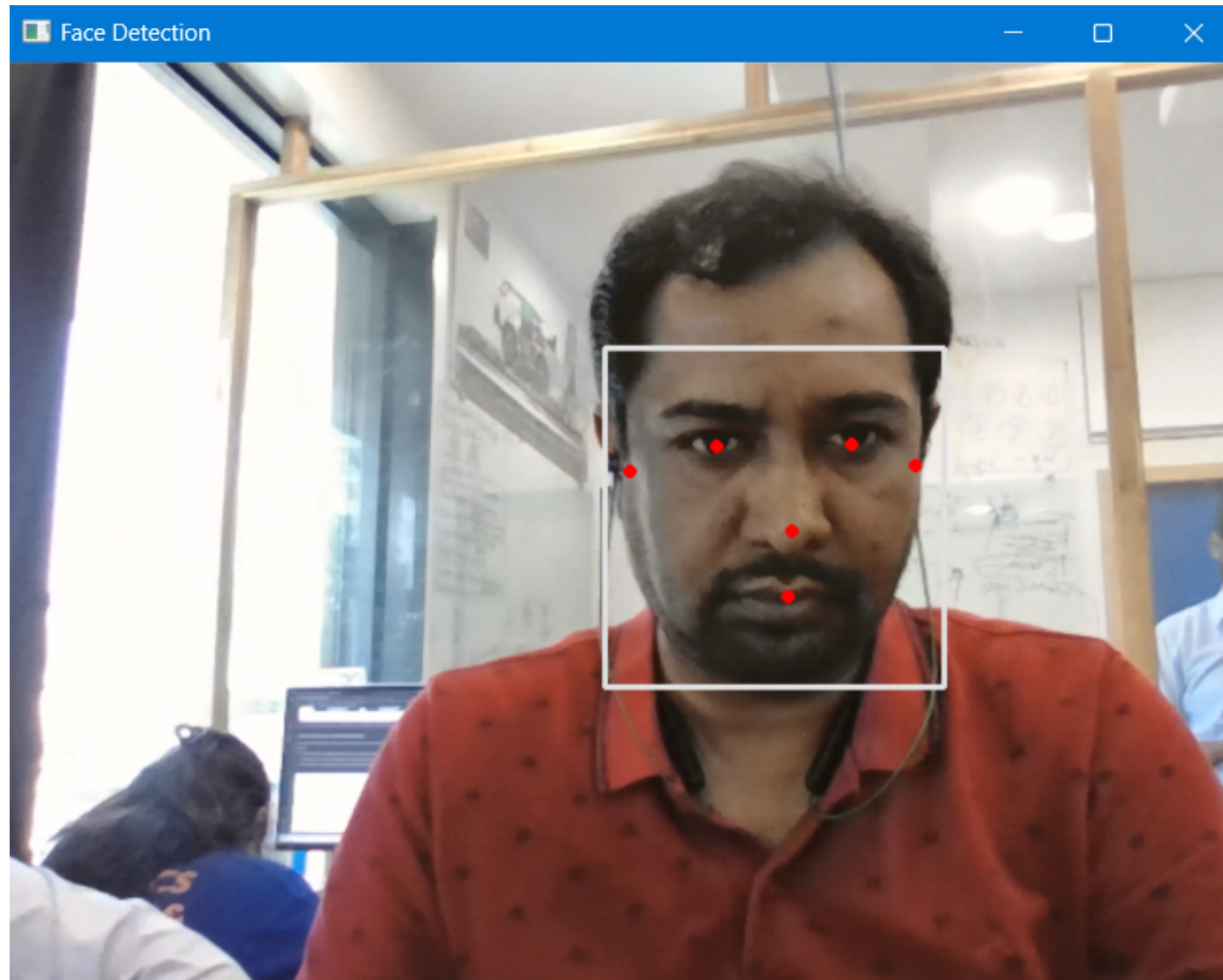
    # Detect faces in the image
    with mp_face_detection.FaceDetection(model_selection=0, min_detection_confidence=0.5) as face_detection:
        results = face_detection.process(img_rgb)
        if results.detections:
```

```
        for detection in results.detections:
            # Draw the bounding box around the face
            mp_draw.draw_detection(img, detection)

    # Display the image
    cv2.imshow("Face Detection", img)

    # Wait for a key press
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the VideoCapture object and destroy the windows
cap.release()
cv2.destroyAllWindows()
```

6. MediaPipe Hand Gesture Recognition on Automatic Volume Control Project

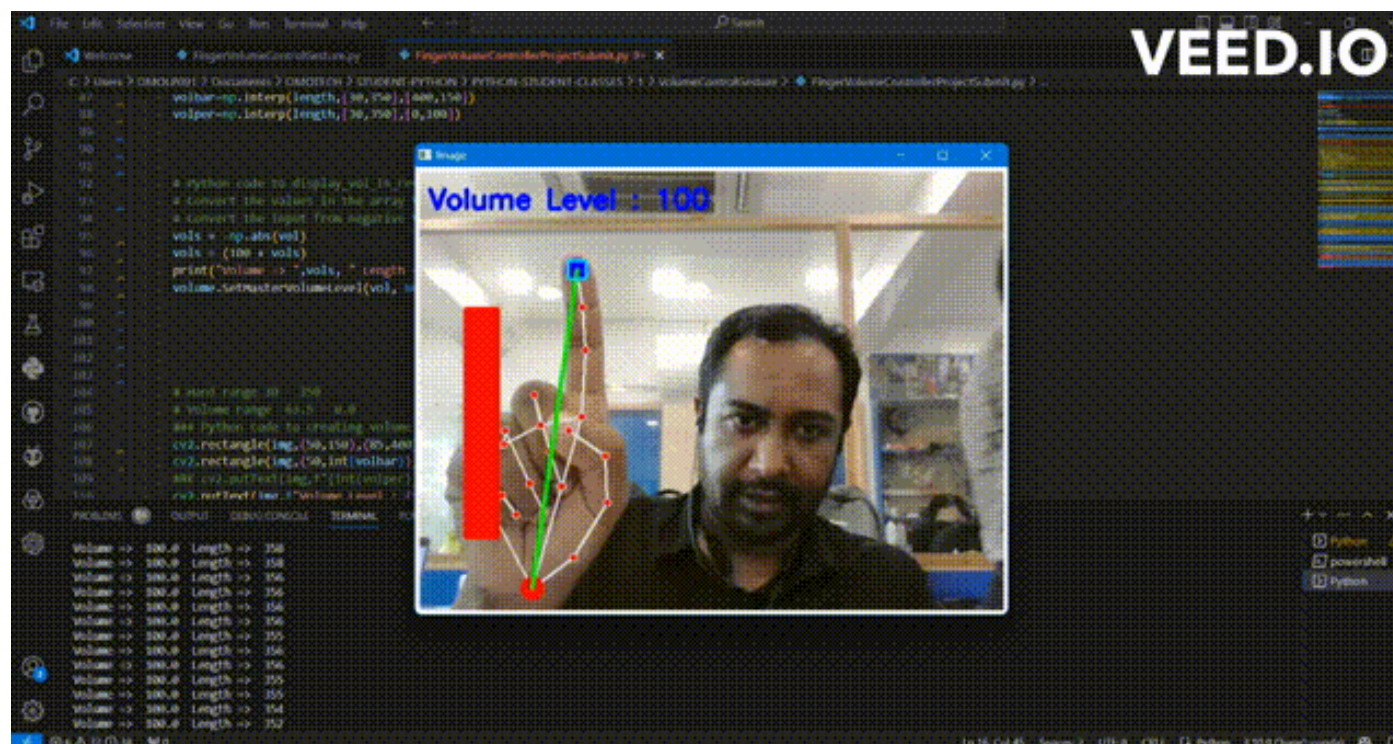
In this Python project, we are trying to process a video so that we can control volume of device with help of webcam camera using the tip our index finger.

Gesture recognition helps computers to understand human body language. This helps to build a more potent link between humans and machines, rather than just the basic text user interfaces or graphical user interfaces (GUIs). In this project for gesture recognition, the human body's motions are read by computer camera. The computer then makes use of this data as input to handle applications. The objective of this project is to develop an interface which will capture human hand gesture dynamically and will control the volume level.

1) NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

2) PyCaw : Python Audio Control Library

3) Mediapipe is an open-source machine learning library of Google, which has some solutions for face recognition and gesture recognition, and provides encapsulation of python, js and other languages. MediaPipe Hands is a high-fidelity hand and finger tracking solution. It uses machine learning (ML) to infer 21 key 3D hand information from just one frame. We can use it to extract the coordinates of the key points of the hand.



The following pip install is required for the code you provided: `pip install mediapipe pycaw numpy comtypes` OR

```
pip install mediapipe -upgrade
```

```
pip install pycau -upgrade
```

```
pip install comtypes -upgrade
```

```
In [ ]: !pip install comtypes mediapipe pycau numpy
```

Requirement already satisfied: comtypes in c:\users\omolp091\anaconda3\lib\site-packages (1.2.0)
 Requirement already satisfied: mediapipe in c:\users\omolp091\anaconda3\lib\site-packages (0.10.5)
 Requirement already satisfied: pycaw in c:\users\omolp091\anaconda3\lib\site-packages (20230407)
 Requirement already satisfied: numpy in c:\users\omolp091\anaconda3\lib\site-packages (1.23.5)
 Requirement already satisfied: absl-py in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (2.0.0)
 Requirement already satisfied: flatbuffers>=2.0 in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (23.5.26)
 Requirement already satisfied: opencv-contrib-python in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (4.8.1.78)
 Requirement already satisfied: sounddevice>=0.4.4 in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (0.4.6)
 Requirement already satisfied: attrs>=19.1.0 in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (22.1.0)
 Requirement already satisfied: matplotlib in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (3.7.0)
 Requirement already satisfied: protobuf<4,>=3.11 in c:\users\omolp091\anaconda3\lib\site-packages (from mediapipe) (3.20.3)
 Requirement already satisfied: psutil in c:\users\omolp091\anaconda3\lib\site-packages (from pycaw) (5.9.0)
 Requirement already satisfied: CFFI>=1.0 in c:\users\omolp091\anaconda3\lib\site-packages (from sounddevice>=0.4.4->mediapipe) (1.15.1)
 Requirement already satisfied: cyclor>=0.10 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (0.11.0)
 Requirement already satisfied: pillow>=6.2.0 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (9.4.0)
 Requirement already satisfied: pyparsing>=2.3.1 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (3.0.9)
 Requirement already satisfied: fonttools>=4.22.0 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (4.25.0)
 Requirement already satisfied: packaging>=20.0 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (22.0)
 Requirement already satisfied: python-dateutil>=2.7 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (2.8.2)
 Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (1.4.4)
 Requirement already satisfied: contourpy>=1.0.1 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib->mediapipe) (1.0.5)
 Requirement already satisfied: pycparser in c:\users\omolp091\anaconda3\lib\site-packages (from CFFI>=1.0->sounddevice>=0.4.4->mediapipe) (2.21)
 Requirement already satisfied: six>=1.5 in c:\users\omolp091\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib->mediapipe) (1.16.0)

```
In [ ]: ##### Import necessary libraries
import cv2
import mediapipe as mp
from math import hypot
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
import numpy as np
```

```

##### Capture video from webcam
cap = cv2.VideoCapture(0)

##### Initialize MediaPipe Hands
mpHands = mp.solutions.hands
hands = mpHands.Hands()
##### Initialize MediaPipe Drawing Utilities
mpDraw = mp.solutions.drawing_utils

### Python code To access speaker through the Library pycaw
##### Access speaker through the library pycaw
devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))

##### Set initial volume bar position and percentage
volbar = 400
volper = 0

volMin, volMax = volume.GetVolumeRange()[ :2]

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

    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    ### Python code to Collection of gesture information
    results = hands.process(imgRGB)

    lmList = []
    ##### If hands are detected
    if results.multi_hand_landmarks:
        ##### Iterate over all detected hands
        for handlandmark in results.multi_hand_landmarks:
            ##### Iterate over all landmarks in the hand
            for id, lm in enumerate(handlandmark.landmark):
                ### Python code to Get finger joint points
                h, w, _ = img.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                ##### Add Landmark to List
                lmList.append([id, cx, cy])

```



```

##### Draw Landmarks on image
mpDraw.draw_landmarks(img, handlandmark, mpHands.HAND_CONNECTIONS)

##### If any Landmarks were found
if lmList != []:
    ##### Get coordinates of palm base and index finger tip
    #getting the value at a point
        #x        #y
    x1,y1 = lmList[0][1],lmList[0][2] #palm
    x2,y2 = lmList[8][1],lmList[8][2] #index finger

    ### Python code to creating circle at the tips of thumb and index finger
    cv2.circle(img,(x1,y1),13,(0,0,255),cv2.FILLED) #image #fingers #radius #rgb
    cv2.circle(img,(x2,y2),13,(255,0,0),cv2.FILLED) #image #fingers #radius #rgb
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),3) #create a line b/w tips of index finger and thumb

    ### Python code to add a LIGHT BLUE box for INDEX FINGER MEDIAPIPE hand Landmarks +8
    # Get the index finger landmark.
    index_finger_landmark = lmList[8]

    ### Python code to Calculate the top-left and bottom-right coordinates of the box.
    box_top_left = (index_finger_landmark[1] - 10, index_finger_landmark[2] - 10)
    box_bottom_right = (index_finger_landmark[1] + 10, index_finger_landmark[2] + 10)

    # Draw the box.
    cv2.rectangle(img, box_top_left, box_bottom_right, (255, 255, 0), 2)

    ### Python code to Calculate the distance between palm base and index finger tip.
    length = hypot(x2-x1,y2-y1) #distance b/w tips using hypotenuse
    # from numpy we find our length,by converting hand range in terms of volume range ie b/w -63.5 to 0
    vol = np.interp(length,[30,350],[volMin,volMax])
    volbar=np.interp(length,[30,350],[400,150])
    volper=np.interp(length,[30,350],[0,100])

    # Python code to display_vol_in_reverse(vol)
    # Convert the values in the array to a range of 0 to 100, with 0 displayed as 100, 1 as 99, etc. till 100 as 0.
    # Convert the input from negative to positive and positive to negative.
    vols = -np.abs(vol)
    vols = (100 + vols)

```

```

print("Volume => ",vols, " Length => ",int(length))
volume.SetMasterVolumeLevel(vol, None)

# Hand range 30 - 350
# Volume range -63.5 - 0.0
### Python code to creating volume bar for volume level
cv2.rectangle(img,(50,150),(85,400),(0,0,255),4) # vid ,initial position ,ending position ,rgb ,thickness
cv2.rectangle(img,(50,int(volbar)),(85,400),(0,0,255),cv2.FILLED)
#RK cv2.putText(img,f"{int(volper)}%",(10,100),cv2.FONT_ITALIC,1,(0, 255, 98),3)
cv2.putText(img,f"Volume Level : {int(vols)}",(10,40),cv2.FONT_ITALIC,1,(255, 0, 0),3)

### Python code to tell the volume percentage ,location,font of text,length,rgb color,thickness
cv2.imshow('Image',img) #Show the video
if cv2.waitKey(1) & 0xff==ord(' '): #By using spacebar delay will stop
    break

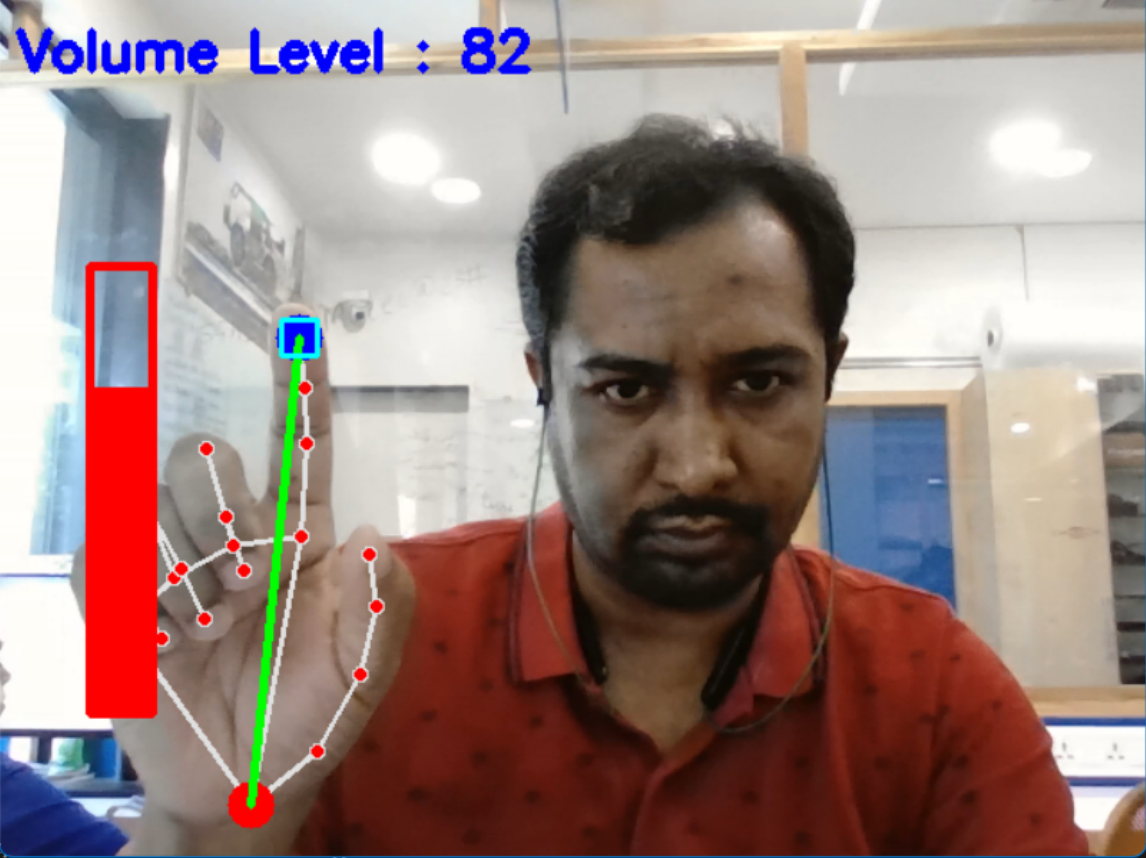
cap.release() #stop cam
cv2.destroyAllWindows() #close window

```

The Kernel crashed while executing code in the the current cell or a previous cell. Please review the code in the cell(s) to identify a possible cause of the failure. Click

Image

Volume Level : 82



[6] 30.9s

Volume =

Volume =

Volume =

Volume =

Volume =

Volume =

Volume =

Volume =

Volume => 79.61393263564318 Length => 247

Volume => 79.41651610047322 Length => 246

Volume => 79.45839792860356 Length => 246

Volume => 79.06393068542687 Length => 244

Volume => 79.30640514319515 Length => 245

Volume => 79.06393068542687 Length => 244

Volume => 79.0217080422763 Length => 244